TopoShot: Uncovering Ethereum's Network Topology Leveraging Replacement Transactions

Kai Li
kli111@syr.edu
Syracuse University
Syracuse, NY, USA

Yuzhe Tang ✉
ytang100@syr.edu
Syracuse University
Syracuse, NY, USA

Jiaqi Chen
jchen217@syr.edu
Syracuse University
Syracuse, NY, USA

Yibo Wang
ywang349@syr.edu
Syracuse University
Syracuse, NY, USA

Xianghong Liu
xliu317@syr.edu
Syracuse University
Syracuse, NY, USA

ABSTRACT

Ethereum relies on a peer-to-peer overlay network to propagate information. The knowledge of Ethereum network topology holds the key to understanding Ethereum's security, availability, and user anonymity. However, an Ethereum network's topology is stored in individual nodes' internal routing tables, measuring which poses challenges and remains an open research problem in the existing literature.

This paper presents TopoShot, a new method uniquely repurposing Ethereum's transaction replacement/eviction policies for topology measurement. TopoShot can be configured to support Geth, Parity and other major Ethereum clients. As validated on local nodes, TopoShot achieves 100% measurement precision and high recall (88% ∼ 97%). To efficiently measure the large Ethereum networks in the wild, we propose a non-trivial schedule to run pair-wise measurements in parallel. To enable ethical measurement on Ethereum mainnet, we propose workload-adaptive configurations of TopoShot to minimize the service interruption to target nodes/network.

We systematically measure a variety of Ethereum networks and obtain new knowledge including the full-network topology in major testnets (Ropsten, Rinkeby and Goerli) and critical sub-network topology in the mainnet. The results on testnets show interesting graph-theoretic properties, such as all testnets exhibit graph modularity significantly lower than random graphs, implying resilience to network partitions. The mainnet results show biased neighbor selection strategies adopted by critical Ethereum services such as mining pools and transaction relays, implying a degree of centralization in real Ethereum networks.

* ✉ Yuzhe Tang is the corresponding author.

IMC '21, November 2–4, 2021, Virtual Event, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9129-0/21/11...$15.00
https://doi.org/10.1145/3487552.3487814

CCS CONCEPTS

• **Networks** → **Peer-to-peer protocols**; **Peer-to-peer networks**.

KEYWORDS

Blockchain, Overlay networks, Network measurements, Ethereum transactions

ACM Reference Format:
Kai Li, Yuzhe Tang, Jiaqi Chen, Yibo Wang, and Xianghong Liu. 2021. TopoShot: Uncovering Ethereum's Network Topology Leveraging Replacement Transactions. In *ACM Internet Measurement Conference (IMC '21), November 2–4, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3487552.3487814

1 INTRODUCTION

A blockchain system relies on an underlying peer-to-peer (P2P) network to propagate information including recent transactions and blocks. The topology of the P2P network is foundational to the blockchain's availability under network partitions, its security against a variety of attacks (e.g., eclipsing targeted nodes [29], denial of specific node service [34, 35], and deanonymization of transaction senders [20, 33]), and its performance (e.g., mining power utilization [27] and the quality of RPC services [4, 7, 14]). Details are in § 3. This value has motivated a line of measurement studies on the network topology of popular blockchains including Bitcoin [24, 28] and Monero [22]. However, although Ethereum is the second largest blockchain network (after Bitcoin) and the biggest smart-contract platform, measuring Ethereum's network topology remains an open research problem. The existing Ethereum measurement studies [31, 32] focus on profiling individual peer nodes, but not the connections among them.

Research goals: Specifically, the operational Ethereum P2P network today runs tens of thousands nodes and host multiple overlays: 1) an underlying P2P overlay, called platform overlay, which forms a structured DHT network by following Kademlia's protocols [36] for peer discovery (RLPx) and session establishment (DevP2P) [32], and 2) a number of application-specific overlays [11, 32], among which the dominant ones are Ethereum blockchains for information propagation. In particular, the Ethereum P2P network hosts multiple blockchain overlays with different "networkIDs" including the mainnet and various testnets, such as Ropsten [16], Rinkeby [15]

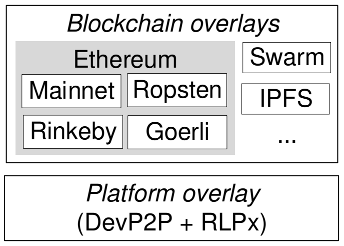


Figure 1: P2P overlay networks on Ethereum. Shaded is the measurement target of this work.

and Goerli [9]. This multi-layer view of Ethereum's P2P network is depicted in Figure 1. In the P2P network, each Ethereum node maintains "peer" connections at these two layers: 1) On the blockchain overlay, a node maintains a list of *active* neighbors through which local information is propagated. 2) On the platform overlay, a node stores the *inactive* neighbors in a DHT routing table, from which live nodes are promoted to active neighbors in the future.

This work aims at measuring the Ethereum P2P network's blockchain overlay and its *active* links.[1] In practice, it is the blockchain overlay's active links, instead of platform overlay's inactive ones, that capture the exact flow of information propagation and are more informative. For instance, a node running the Geth client [8] (which is the most popular Ethereum client and is deployed on more than 80% nodes in the mainnet [6]) maintains 272 inactive neighbors and around 50 active neighbors, by default. Knowing what these 50 neighbors are is helpful to understand the node's resilience to eclipse attacks (as information is propagated through the 50 active neighbors, not the 272 inactive ones, and an attacker only needs to disable the 50 active neighbors to block information propagation). Also, knowing whether the 50 neighbors contain nodes from top mining pools, such as Sparkpool [17] (or popular transaction relay service, such as infura.io [4]), is useful to estimate the timeliness and quality of the blocks (or transactions) received on the node, as well as understanding the centralization of the blockchain network.

Measuring Ethereum network's active links is an open research problem. In the existing literature, 1) the related works that measure Ethereum networks focus on profiling individual nodes [31, 32] or detecting inactive links [26, 41], but not the active connections. Compared to the inactive links that are exposed in Ethereum peer discovery messages (i.e., RLPx's `FIND_NODE` packets) and can be directly measured as in [26, 41], active links are hidden information inside remote Ethereum nodes, directly measuring which without inference is impossible as we thoroughly examine Ethereum protocol's messages. 2) The other related works explore the topology of non-Ethereum blockchains including Bitcoin [24, 28] and Monero [22]. Their measurement approaches exploit features specific to Bitcoin/Monero and are inapplicable to Ethereum as will be detailed in § 4. Notably, TxProbe's approach [24] to infer Bitcoin's topology cannot be applied to measuring Ethereum topology, as these two blockchains differ in transaction model (account-based versus UTXO-based) and propagation model (direct propagation versus announcement), as will be further explained in § 4.1.

[1]In this paper, we use terms "links", "connections" and "edges", interchangeably.

Measurement methods: In this work, we propose TopoShot to measure an Ethereum blockchain overlay by repurposing Ethereum's transaction replacement and eviction policies. Briefly, an Ethereum node buffers unconfirmed transactions (prior to mining) in a local data structure named `mempool`, where an unconfirmed transaction can be replaced or evicted by a subsequent transaction at a sufficiently higher Gas price.[2] Transaction replacement and eviction are standard Ethereum features, widely supported by Ethereum clients (including Geth [8], Parity [13] and others [1, 10, 12]), and highly desirable by real-world applications. For instance, a common practice in blockchain-based decentralized applications is that a user having sent a transaction can posthumously speed up its inclusion into the blockchain by sending replacement transactions at higher price per computation unit (or the so-called Gas price). Leveraging these features, TopoShot runs a measurement node M to detect the connection between two remote nodes A and B. In TopoShot, node M propagates a high-priced transaction tx_A on target node A, a low-priced transaction tx_B to target node B, and a medium-priced transaction tx_C propagated to all other nodes in the same network. It then observes tx_A's presence on node B and, if so, draws the conclusion that node A is actively connected to node B. To ensure the accurate measurement, when node A is not linked to node B, measurement transaction tx_A should not be propagated and do not reach node B (the so-called "isolation" property [24]). One of the key insights in this work is that Ethereum's transaction replacement policy can be repurposed to enforce isolation property for accurate link measurement. Intuitively, the isolation is ensured by the fact that TopoShot's high-priced tx_A can replace the low-priced tx_B on node B but not the medium-priced tx_C on other nodes, through which tx_A cannot be propagated to reach node B.

To set up the measurement as above, TopoShot further leverages Ethereum's support of transaction eviction and future transactions, that is, to evict an existing unconfirmed transaction on a node by incoming future transactions (the concept of future transaction in Ethereum is similar to orphan transactions in Bitcoin). Specifically, when using TopoShot to measure the connectivity between Nodes A and B, the measurement node M first needs to connect to both nodes, propagates tx_C to all nodes, then sends future transactions to evict tx_C (with other existing transactions) on node A and B before sending tx_A and tx_B to node A and B, respectively. This method can be applied to measuring the connectivity among all possible pairs of nodes by the standard approach of launching a "supernode" connecting to all other nodes in the network [31, 32].

The basic TopoShot achieves 100% result precision but not 100% recall, which can be attributed to non-default settings of target node. We further propose a pre-processing phase retrofittable with TopoShot to profile the actual settings of target node and to improve the result recall, proactively.

For large-scale measurement on real Ethereum networks, we propose a non-trivial method to parallelize multiple pair-wise measurements, reducing the rounds and overall time of measurement.

Measurement results: We systematically evaluate the validity of TopoShot and conduct measurement studies on both testnets and

[2]The difference between transaction replacement and eviction is that a transaction tx is replaced by another transaction from the same sender account with tx, and tx is evicted by another transaction from a different sender from tx when the mempool is full.

the mainnet. The measurement results uncover, for the first time, the full network topology of Ethereum's major testnets (including Ropsten, Rinkeby and Goerli) and the inter- and intra-connections among the mainnet's mining-pools and transaction relay services.

First, we validate the TopoShot's correctness in terms of recall and precision. We set up a local node under control and a remote node in a testnet, and we use TopoShot to measure the connection between the two nodes. By comparing against the ground-truth of node connection (via querying the local node's state), we confirm that TopoShot achieves the perfect precision (100%) and a high recall (up to 97%).

Second, we use TopoShot to measure, for the first time, the network topology of major Ethereum testnets including Ropsten, Goerli and Rinkeby. We also analyze the captured graphs which reveal a number of graph-theoretical properties including degree distribution, distances, assortativity, clustering and community structures. Our comparative analysis shows that the measured Ethereum networks have particularly lower modularity than classic random graphs [19, 25, 40], implying a better resilience against attacks to partition the networks.

Third, we propose enhanced TopoShot configurations to allow lightweight yet effective measurement on the mainnet without ethical concerns. The TopoShot enhancement minimizes the impacts on the target nodes being measured, and particularly ensures that the set of transactions included in the blockchain do not change under measurement. Using the approach, we measure a critical substructure of Ethereum's mainnet overlay. The result reveals biased neighbor selection strategy commonly practiced by critical Ethereum services such as mining pools and transaction relays who prioritize to connect other critical nodes over average nodes. We acknowledge the high cost of our method and avoid measuring the topology of entire mainnet network which would otherwise cost 60 million USD at the Ether price as of May 2021.

Contributions: This work makes the following contributions:

• *Novel methods*: We propose a novel method, named TopoShot, to measure Ethereum network links and topology. TopoShot takes a unique approach by exploiting Ethereum's handling of unconfirmed transactions (i.e., transaction replacement and eviction). TopoShot is generic and supports all Ethereum clients (including Geth and Parity). TopoShot is effective and achieves 100% result precision and high recall (88% ∼ 97%).

• *Large-scale measurements*: We address the scalability and ethical challenges raised in measuring large-scale, real Ethereum networks. We propose to schedule pair-wise measurements in parallel for efficiency. We propose workload-adaptive mechanisms to configure TopoShot for minimal service interruption on the target nodes/network.

• *New systematic results*: Without TopoShot, an Ethereum network's topology remains hidden information inside blackbox Ethereum nodes, measuring which stays an open research problem. By systematically conducting measurements against a variety of Ethereum networks, we obtain a series of new knowledge on network topology and its graph-theoretic statistics, ranging from full-network topology in popular testnets (Ropsten, Rinkeby and Goerli) and critical sub-network topology in the mainnet.

The source code of TopoShot is on https://github.com/syracuse-fullstacksecurity/Toposhot.

Roadmap: The paper is organized in the following order: § 2 presents the preliminary knowledge. Motivation of this work is presented in § 3. § 4 surveys the related works and their (in)applicability to measuring Ethereum's topology. § 5 presents TopoShot's measurement primitive, parallel schedule, as well as correctness analysis. § 6 presents the measurement results of Ethereum testnets and the mainnet. The ethical aspects of this work are discussed in § 7, and the conclusion is in § 8.

2 PRELIMINARY

Ethereum transactions: To begin with, we describe the transaction model used in Ethereum. An Ethereum transaction binds a sender account to a receiver account. Each transaction is associated with a *nonce*, which is a monotonically increasing counter per sender. An Ethereum transaction is associated with *Gas price*, that is, the amount of Ether the sender is willing to pay to the miner for each unit of computation carried out by the miner to validate the transaction.

Unconfirmed transaction buffer (`mempool`): Each Ethereum node stores unconfirmed transactions in a local data structure, named `mempool`. In a `mempool`, a transaction tx is *pending*, if its nonce equals one plus the maximal nonce of the transactions of the same sender in the `mempool` (i.e., equal to $n+1$). Otherwise, if tx's nonce is strictly larger than $n+1$, tx is a *future transaction*.

When a transaction tx propagated from other nodes arrives at a node N, node N determines whether to *admit* tx into its `mempool`. Admitting a transaction tx may trigger two more `mempool` events: 1a) *eviction* of an existing transaction tx' by tx where tx and tx' are of different accounts or nonces, and 1b) *replacement* of an existing transaction tx' by tx where tx and tx' are of the same sender and nonce.

Transaction propagation: When admitting a pending transaction to its `mempool`, an Ethereum node propagates the transaction to its active neighbors. If an incoming transaction is not admitted or the admitted transaction is a future transaction, it will not be propagated.

Normally, an Ethereum node directly *pushes* a pending transaction to its neighbors. That is, it sends a message to its neighbors encoding the transactions it wants to propagate. It may be the case that the propagated transactions are already received on the neighbors. This is the default transaction propagation protocol supported widely in Geth, Parity and other clients.

Some Ethereum clients (e.g., Geth with version later than 1.9.11) support *announcements* as an optional transaction-propagation protocol. It works in three messages: 1) The node announces its local pending transactions by their hash and propagates the hash to its neighbors. 2) Then within the next 5 seconds, its neighbors will respond with requests if they want to receive the transaction. Within these 5 seconds, the neighbors will not respond to other announcements of the same transaction. 3) The node propagates the transactions to all requesting neighbors. While this is similar to Bitcoin's transaction announcement as exploited in TxProbe [24], there is an important distinction: Ethereum's transaction announcement has to *co-exist* with transaction pushing, and Ethereum's pushing can bypass the blocking of an announcement.

3 MOTIVATION: SIGNIFICANCE OF KNOWING BLOCKCHAIN TOPOLOGY

The motivation of this work is that a blockchain network's topology is foundational to the blockchain's security and performance. In this section, we present a non-exhaustive list of "use cases" of blockchain topology knowledge in the hope of justifying its significance.

3.1 Implication to Blockchain Security

The knowledge of blockchain network topology is crucial to understanding its security against various attack vectors, with examples listed below.

Use case 1: Targeted eclipse attacks. In the network topology, if a blockchain node is found to be of a low degree (i.e., few neighbors), such a node is particularly vulnerable under a targeted eclipse attack [29]. That is, such an eclipse attacker can concentrate her attack payload to the few neighbors to disable the information propagation and to isolate the victim node from the rest of the network at low costs.

Use case 2: Single point of failure. The blockchain's network topology may reveal the centralization of network connection, leading to a single point of failure. Specifically, there may be supernodes that connect to all other nodes, "bridge" nodes that control the connection to the backend of critical services, and topology-critical nodes removing which may lead to partitioned networks. Directing denial-of-service attacks onto these critical nodes, using attack vectors recently discovered [34, 35], can lead to consequences such as crippled blockchain services and the censorship of individual transactions.

Use case 3: Deanonymizing transaction senders. With the knowledge of the network topology, if nodes' neighbors are distinguishing (i.e., node X's neighbors are distinct from another node Y's neighbors), the neighbor set can be used to identify/fingerprint nodes and can be further used to facilitate the deanonymization of transaction senders. Specifically, in the deanonymization attack [20], a blockchain "client" node (i.e., a node behind the NAT) is identified by its "server"-node neighbors (a server node is of public IPs, is not behind the NAT and accepts incoming connections). An attacker then simply monitors the transaction traffic on all server nodes in the blockchain (e.g., a Bitcoin network contains much fewer server nodes than the client nodes, thus lowering the attacker's costs). The attacker can link a transaction sender's blockchain address (her public key) to a client node's IP address, which can be further linked to a real-world identity, thus deanonymizing the blockchain address.

3.2 Implication to Blockchain Performance

Blockchain network topology is essential to achieving its performance promises and matters to both miners and client users.

Use case 4: Mining efficiency and mining pools' QoS (quality of service). In a blockchain, the time to propagate a recently found block from its miner to the entire network is critically important: If it takes too long to propagate miner X's block, her block may arrive after another miner Y's block, leading to X's block unable to be included in the blockchain and X's loss of revenue. Thus, a blockchain's network topology that affects propagation delay can influence a miner node's revenue and mining-power utilization (i.e., how much mining power spent is useful and is reflected in the main chain's blocks). Thus, it is unfavorable to have a minor with limited connectivity and incur long propagation delays.

For a client interested in joining a mining pool, she may want to access the knowledge of blockchain topology and make an informed decision to choose the mining pool with better connectivity and lower propagation delay to ensure high mining revenue.

Use case 5: RPC service's QoS. For a client sending transactions through RPC services (e.g., infura.io), she may want to choose a service with better connectivity so that her transaction can be relayed on a timely basis.

In summary, the knowledge of blockchain network topology is critical to understanding its security, performance, and decentralization. Given the high market capitalization of today's blockchains (e.g., $4106 billion USD for Ethereum as of Sep. 2021 [18]), we believe measuring blockchains' network topology is valuable and worthy even it may cost as much as 60 million USD, estimated in § 6.3.

4 RELATED WORK

In this section, we present the existing measurement studies on public blockchain networks. Existing works can be categorized into three classes: W1) Measuring blockchain nodes, W2) measuring blockchain inactive edges, and W3) measuring blockchain active edges.

Measuring blockchain nodes (W1): Kim et al. [32] propose a passive method to characterize the Ethereum mainnet by launching a "supernode" to connect all reachable mainnet nodes and collecting messages exchanged. The result reveals node-wise characteristics including network size, node geo-distribution, clients' age and freshness, and others.

Neudecker et al. (2019) [38] is a passive measurement study that last four years to characterize the behavior of individual Bitcoin peers and their operators. Their approach is by launching "supernodes" and passively collecting transaction traffic, a method similar to [32].

Measuring blockchain inactive edges (W2): Ethereum's peer discovery protocol (RLPx) has a `FIND_NODE` message through which a node can discover another node's current routing-table entries (inactive neighbors). Recent research works [26, 41] directly measure Ethereum's inactive links by sending `FIND_NODE` messages to all nodes in an Ethereum network. This method cannot distinguish a node's (50) active neighbors from its (272) inactive ones and does not reveal the exact topology information as TopoShot does.

Henningsen et al. [30] measure the Kademlia network topology in IPFS by sending crafted peer-discovery queries. Despite other findings, this work reveals IPFS's network combines a structured Kademlia DHT and an unstructured P2P overlay.

Inference of blockchains' active edges (W3): Coinscope [37] targets Bitcoin's network topology and infers the links by leveraging the expiration timestamps of Bitcoin's `ADDR` messages.

TxProbe [24] infers Bitcoin's network topology by exploiting Bitcoin's support of orphan transactions and announcement-based transaction propagation. We will describe how TxProbe works, with more details in § 4.1, to understand its applicability to measuring Ethereum networks.

Grundmann et al. [28] present two Bitcoin-topology inference approaches among which the more accurate one exploits Bitcoin's behavior of dropping double-spending transactions. Neudecker et al. (2016) [39] conducts a timing analysis of Bitcoin transaction propagation to infer the network topology. Despite the optimization, both works are limited in terms of low accuracy.

Daniel et al. [23] propose to exploit block relay mechanisms to passively infer connections among mining nodes and their direct neighbors in the ZCash network.

Cao et al. [22] measure Monero's P2P network topology by exploiting the timing of neighbors' liveness probes. Specifically, a Monero node maintains the liveness of its neighbors (the `last_seen` label) by periodically discovering its hop-2 neighbors, probing their liveness by sending PING messages, and selectively promoting them to be hop-1 neighbors. Topology inference methods are proposed to exploit the timing difference of neighbor nodes' `last_seen` labels. This method is specific to Monero's liveness-check protocol.

Table 1: Existing works on blockchain topology measurement and TopoShot's distinction

Research work	Blockchain	Measurement target
[38]	Bitcoin	Nodes
TxProbe [24] & others [28, 37, 39]	Bitcoin	Active edges (W3)
[22]	Monero	Active edges (W3)
[23]	ZCash	Active edges (W3)
[32]	Ethereum	Nodes (W1)
[26, 41]	Ethereum	Inactive edges (W2)
[30]	IPFS	Inactive edges (W2)
TopoShot	Ethereum	Active edges (W3)

The existing blockchain measurement studies are summarized in Table 1. In general, existing techniques on W1 and W2 directly measure the target (as the target information of nodes and inactive edges is exposed in collected messages), while measuring active edges (W3) requires inference. Existing topology-inference techniques focus on non-Ethereum blockchains and exploit blockchain specific features (e.g., Monero's timing of liveness probes and Bitcoin's announcement-based propagation) that are absent in Ethereum.

4.1 TxProbe's Applicability to Ethereum

To understand how TxProbe works and its (in)applicability to measuring Ethereum network, we first describe the following measurement framework: Suppose a measurement node M is to detect the connection between a pair of target nodes, say A and B. Node M can propagate a transaction tx_A to node A and observe tx_A's presence on node B. If present, nodes A and B are actively linked. The success of this method depends on the so-called isolation property. That is, when node A and B are not actively linked, tx_A should not be propagated to node B. In other words, there is no alternative routing path beside the direct link between A and B that tx_A can take to reach node B.

TxProbe [24, 28] materializes this framework to measure active links in Bitcoin and ensures the isolation property by Bitcoin's transaction announcement mechanism. Briefly, Bitcoin's transaction announcement works as follows: a Bitcoin node propagates a transaction to its neighbor by first sending a transaction announcement (i.e., a hash value) to the neighbor and, upon neighbor's acknowledgment, then sending the actual transaction. Bitcoin has a policy that the neighbor node receiving an announcement will ignore the subsequent announcements of the same transaction from other nodes for 120 seconds. TxProbe exploits this policy to ensure the isolation during the 120-second period. This is done by having Node M to announce tx_A to all nodes besides B so that these nodes will not relay tx_A when Node A starts to propagate tx_A to B, ensuring the isolation property.

However, TxProbe's method is inapplicable to measuring Ethereum. Ethereum's transaction propagation only *partially* depends on announcement, that is, a transaction is announced to some neighbors and is directly sent to other neighbors without announcement. The existence of direct propagation, no matter how small portion it plays, negates the isolation property, as measurement transaction tx_A can be propagated through the nodes taking direct propagation as the alternative path to reach node B, introducing false positives to the measurement results.

In addition, TxProbe relies on Bitcoin's UTXO model, which differs from Ethereum's account model. Directly applying TxProbe to Ethereum risks incorrect measurement, as analyzed in Appendix A.

5 TOPOSHOT MEASUREMENT METHODS

We first present our observation on real Ethereum clients' behavior in transaction replacement and eviction, which lays the foundation of TopoShot measurement method (§ 5.1). We then describe the measurement primitive in TopoShot that detects just one link between two nodes (§ 5.2). We will then describe how to use this primitive to measure a network of a large number of links (§ 5.3).

5.1 Profiling Ethereum Clients' Behavior

We first describe a parameterized model for `mempool` and then our measurement study that reveals the `mempool` parameters of real Ethereum clients.

`mempool` model: Recall that transaction eviction (replacement) is a `mempool` process that takes as input the initial state of `mempool` and an incoming transaction tx_1 and produces as output the end state of the `mempool` and possibly, a transaction tx_2 that is of the same (different) sender with tx_1 and that is evicted (replaced) from the `mempool`. To formally describe the process, suppose the initial state is a full `mempool` consisting of l pending transactions and $L-l$ future transactions, where L is the capacity of the `mempool` (denoted in Table 2). The incoming transaction tx_1 is a future transaction with Gas price higher than any transactions currently in the `mempool`. There are u transactions currently in the `mempool` that are of the same sender with tx_1.

When there is another transaction tx_2 in the `mempool` that has the same sender and nonce with tx_1, admitting tx_1 to the `mempool` triggers the replacement of tx_2. The generic transaction replacement strategy is that `mempool` *decides to replace* tx_2 *by* tx_1*, if* tx_1*'s Gas price is* $1+R$ *of* tx_2*'s Gas price.*

Otherwise (i.e., when there is no transaction of the same sender and nonce with tx_1), admitting tx_1 may trigger transaction eviction.

For transaction eviction, the situation of interest to us is the eviction victim tx_2 being a pending transaction. Under this situation, the transaction eviction strategy generally follows the template: `mempool` *decides to evict a pending transaction tx_2 by tx_1, if 1) tx_1's Gas price is higher than tx_2's Gas price, and 2) there are more than P pending transactions existing in the* `mempool`*, and 3) there are fewer than U existing transactions of the same sender with tx_1*. The three `mempool` parameters, namely R, U and P, and their meanings are presented in Table 2.

Table 2: Notations

Notation	Meaning
R	Minimal Gas price difference for an incoming transaction (tx) to replace an existing tx in `mempool`
U	Max number of future txs sent from the same account that can be admitted to a node's `mempool`
P	Minimal number of pending txs buffered in a node to allow eviction by future txs
L	Maximal number of txs allowed to store in a `mempool` (`mempool` capacity)

Table 3: Profiling different Ethereum clients in terms of transaction eviction and replacement policies The second column refers to the percentage of mainnet nodes running a specific client [6].

Ethereum clients	Deployment (mainnet)	Replacement behavior	Eviction behavior		
		R	U	P	L
Geth	83.24%	10%	4096	0	5120
Parity	14.57%	12.5%	81	2000	8192
Nethermind	1.53%	0%	17	0	2048
Besu	0.52%	10%	∞	0	4096
Aleth	0%	0%	1	0	2048

`mempool` tests: The measurement is set up with 1) a measurement node M running the test and 2) a target node T running the Ethereum client to be measured. For each test, node T's initial state of `mempool` contains l future transactions and $L - l$ pending transactions.

We design the first set of tests to trigger transaction replacement and measure R. Specifically, tx_1 has an identical sender and nonce with an existing transaction tx_2 in `mempool`. In each unit test, Node M sends tx_1 of a certain Gas price to node T, and observes if node T replaces tx_2 by tx_1. We run a series of unit tests with varying tx_1's Gas prices, in order to observe the minimal Gas price that triggers the replacement, from which we calculate and report R.

We design the second set of tests to trigger transaction eviction and measure U and P. Specifically, the `mempool` contains $L-l$ future transactions and l pending transactions, among which there are u transactions sent from the same account with future transaction tx_1. As before, in each unit test, node M sends to node T tx_1 at a Gas price higher than any transactions in node T's `mempool`. We run a series of unit tests with varying l and u. We observe the maximal value of u that triggers a successful eviction by tx_1 and report such value by U. We observe the minimal value of l that triggers a successful eviction by tx_1 and report such value by P.

Test results on Ethereum clients: We conduct the tests on two local nodes: We first set up a local measurement node M running tests on an instrumented Geth client and a local target node T. The statically instrumented Geth client allows node M to bypass local checks and to propagate future transactions to node T. We run the two sets of tests against target node T running five different Ethereum clients: Geth (Go), OpenEthereum/Parity (Rust), Nethermind (.net), Besu (Java) and Aleth (C++). Here, we discard the Python client (i.e., Trinity) as the incomplete implementation. The distribution of mainnet nodes running the five Ethereum clients is presented in the second column of Table 3, where Geth (83%) and Parity (15%) are the dominant clients on the mainnet.

The measurement results are reported in Table 3. The `mempool` model and measurement results will guide the design of TopoShot's method and the configuration of the measurement on different Ethereum clients. Noteworthy here is that Aleth's and Nethermind's R values are both zero (0%), which renders our TopoShot unable to work (as will be seen, it requires a non-zero R to enforce the isolation property). Thus, TopoShot currently does not work with Aleth and Nethermind clients. On the other hand, we deem a zero-value R is a flawed setting that can be exploited to construct low-cost denial of service or flooding. For instance, an attacker can send multiple replacing transactions at almost the same Gas price, consuming network resources by propagating multiple transactions yet without paying additional Ether. We sent bug reports to Ethereum Foundation's bug bounty program [5], and further updates, if any, will be documented on a private link [2].

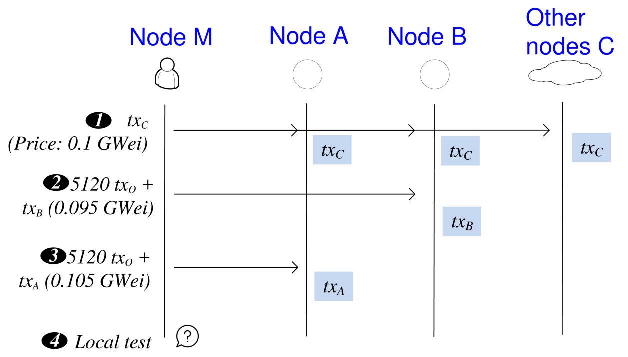
5.2 Measurement Primitive

We consider the basic system model consisting of a measurement node M, target node A, target node B and the rest of Ethereum network denoted by node(s) C. The measurement primitive's goal is to detect one link, that is, whether Node A and B are actively connected in the Ethereum network. Note that this model assumes a strongly connected Ethereum network without network partition.

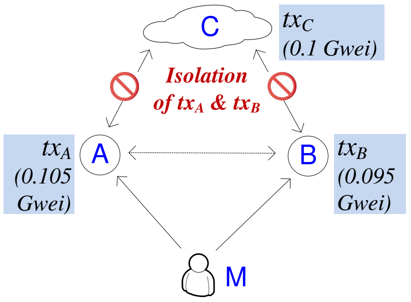
Mechanism: We denote our measurement primitive by *measureOneLink*(A, B, X, Y, Z, R, U), which are parameterized by target nodes A and B, target nodes' `mempool` behavior R/U (recall Table 2) and $X/Y/Z$ that will be described below. As depicted in Figure 2a, the measurement primitive works in four steps:

1. Node M sends a pending transaction tx_C with Gas price Y Gwei[3] to A, and waits for X seconds (e.g., $X = 10$ in our study as will be described) for tx_C to be propagated to other nodes including node B. Setting Y at a low Gas price is intended to slow down or even prevent the inclusion of tx_C in the next block (recall Ethereum nodes decide which transactions to be included in the next block based on Gas/Gas price).
2. Node M sends to Node B Z future transactions $\{tx_{O1}, tx_{O2}, ...tx_{OZ}\}$ at Gas price $(1 + R) \cdot Y$ Gwei. These future transactions are uniformly sent from $\frac{Z}{U}$ accounts (i.e., there are U future transactions sent from each account). Immediately after that, Node M sends a transaction tx_B at Gas price $(1 - 0.5R) \cdot Y$ Gwei to Node B. Transaction tx_B has the same nonce with tx_C.
3. Node M sends to Node A Z future transactions $\{tx_{O1}, tx_{O2}, ...tx_{OZ}\}$ which are at Gas price $(1 + R) \cdot Y$ Gwei and sent from $\frac{Z}{U}$ accounts. Immediately after that, Node M sends a transaction tx_A at Gas price $(1 + 0.5R) \cdot Y$ Gwei to Node A. Transaction tx_A has the same nonce with tx_C.

[3]One Gwei equals 10^{-9} Ether.



(a) **Workflow: Shaded are transactions in the nodes'** `mempool`**. Step 2 would evict tx_C and add tx_B on node B's** `mempool`**, and Step 3 evicts tx_C and adds tx_A on node A's** `mempool`**.**



(b) **Snapshot right before Step 4**

Figure 2: TopoShot's measurement primitive: Running *measureOneLink* with $Y = 0.1$ Gwei, $Z = 5120$, $R = 10\%$, $U = 1$

The purpose of the future transactions is to fill up the `mempool` on Nodes A (and B), to evict tx_C there, and to make room for tx_A (tx_B) of the same nonce to tx_C.

❹ Node M checks if it receives tx_A from Node B. If so, it draws the conclusion that Node A is a neighbor of Node B, as will be analyzed in § 5.2.1.

Configuration of R/U: Parameters of the *measureOneLink* primitive are configured as follows: On a target Ethereum client, parameters R/U will be set at the client's value as in Table 3. Here, note that Nethermind and Aleth are not measure-able by TopoShot due to their zero-value R which is also flawed as explained before. Besu has an infinite large value of U, and Geth has a fairly large U. In these two cases, only one account is used to send the future transactions $\{tx_O\}$. Geth/Parity have non-zero P, which are fairly small compared with their `mempool` capability L. The working of measureOneLink requires the following condition: *The number of pending transactions in the measured* `mempool` *should remain larger than* P *in the entire process of measurement.* We verify that this condition holds on the mainnet for all Ethereum clients' P and L.

Configuration of X: Parameter X, which is the time period that Step ❶ waits, is set to be large enough so that transaction tx_C is propagated to all nodes in the network. In order to obtain a proper value of X in an Ethereum network, we conduct a test by running several local nodes (e.g., 11 nodes in our study) and joining them to the Ethereum network. Among the 11 nodes, there are no direct connections. During the test, we send a transaction through one node, wait for X' seconds and observe the presence of the transaction on the other 10 nodes. We conduct a series of such tests with varying X'es to obtain such a $X' = X$ that with 99.9% chances, the transaction is present on the 10 nodes after X seconds.

The four steps occur in order. That is, Step ❶ occurs X seconds before Step ❷, which finishes before Step ❸ starts, which is before Step ❹. Timing and ordering are essential to the success of our measurement method, as is analyzed below.

5.2.1 *Correctness Analysis.*

We analyze the correctness of the measurement primitive (*measureOneLink*):

10 seconds after Step ❶, Transaction tx_C is propagated to the entire Ethereum network and it is stored in all nodes' `mempools` including Nodes A and B.

During ❷, when Node B receives Z future transactions tx_Os, its `mempool` becomes full. Based on the eviction policy in Table 3, adding a new transaction to a full `mempool` triggers evicting the transaction with the lowest Gas price. Assuming Gas price Y Gwei is low enough (we will describe how to set Y next), transaction tx_C at Y Gwei will be evicted on Node B. Then, without tx_C, transaction tx_B is stored in Node B's `mempool`. In other words, Step ❷ replaces tx_C with tx_B on Node B. Note that in the process, future transactions $\{tx_O\}$ are not propagated, thus C still stores tx_C.

Note that after the arrival of $\{tx_O\}$ but before tx_B, there are chances that certain Nodes C can propagate tx_C back to Node B, which, if occurs, would invalidate the efforts of $\{tx_O\}$ and leave tx_B unable to replace (the re-propagated) tx_C on B. In TopoShot, the actual chance of this event is very low and the reason is two-fold: 1) ❷ waits long enough (10 seconds) after ❶ to start and 2) tx_B is propagated immediately after $\{tx_O\}$. In addition, in our local validation experiment (in § 6.1), we don't observe the occurrence of the event.

By a similar analysis, Step ❸ can replace tx_C with tx_A on Node A.

Now, we have established that after Steps ❶,❷ and ❸, Node A stores tx_A, Node B stores tx_B and Nodes C store tx_C. The snapshot of our measurement system at this timing is illustrated in Figure 2b.

We consider two cases: Case 1) A and B are directly connected. In this case, A will propagate tx_A to B, which will replace tx_B because of tx_A's R (e.g., 10% for Geth) higher Gas price than tx_B. In this case, A will also propagate tx_A to C, which however will not replace tx_C as tx_A's Gas price is lower than R (e.g., 10%) of tx_C's price. The property that tx_A is stored only on Node A and cannot be propagated through Nodes C is called isolation. That is, tx_A is isolated on Node A. Thus, after a sufficient delay for propagation from A to B, M can receive tx_A from Node B.

Case 2) A and B are not connected. In this case, A propagates tx_A only to Node C. As analyzed, tx_A cannot replace tx_C on Node C because of insufficient Gas price. Also, Node C's tx_C cannot replace tx_B on Node B. Thus, tx_B stays on Node B, and M does not receive tx_A from Node B.

To ensure correctness, TopoShot requires that the `mempool` on the two measured nodes, namely nodes A and B, are full. This condition holds quite commonly in Ethereum mainnet, as observed in our measurement study (99% of the time during our mainnet measurement, the measurement node's `mempool` is full).

Configuration of Y/Z: Pending transactions like tx_C should stay in the `mempool` of Nodes C, in such a way that they are not included in the next block or be evicted. To do so, the Gas price of tx_C should be low enough so that it will not be included in the next block, and at the same time, be high enough to avoid eviction by incoming transactions. To estimate a proper Gas price in the presence of current transactions, we rank all pending transactions in the `mempool` of Node M by their Gas prices, and use the median Gas price for tx_C. In actual measurement studies, the value of Y varies from testnets and at different times. We apply the estimation method before every measurement study and obtain Y dynamically.

5.2.2 *Cost Analysis.* The cost of running *measureOneLink* comes from the pending transactions sent (i.e., either tx_A or tx_C), assuming their inclusion in the blockchain. In practice, whether these two transactions are included is not deterministic and depends on the state of the miners' `mempool`. Also, note that the future transactions tx_O sent during the measurement are guaranteed not to be included in the testnets and mainnet, thus incurring no costs.

5.2.3 *Improving Result Recall.* Based on the above analysis, the TopoShot guarantees that any tested connection is a true positive (i.e., no false positives) but may miss the detection of a connection (i.e., false negative may exist). In other words, the 100% result precision is guaranteed by the protocol but not for the recall. Note that 100% precision/recall means no false positive/no true negative in the measurement result. In the following, we present several heuristics to improve the result recalls in practice.

A passive method to improve the result recall is to repeat the measurements multiple times and use the union of the results. This passive method has limited applicability if the false negative is caused by the non-default setting on the remote Geth node being measured. In the following, we propose a proactive method to improve the recall.

Handling node-specific configurations by pre-processing: In Ethereum networks, client configurations (e.g., on `mempool`) are specific to nodes. This is evident in our field experience where the `mempool` capacities (i.e., L) differ across nodes. Using the same value of L when measuring different nodes can lead to incorrect results.

To solve the problem, we add a pre-processing phase: Before the measurement, we can launch a speculative B' node locally and use it to connect all other nodes in the network. For each other node, say A', we then run TopoShot between A' and B'. Because the local node B' is under our control and its actual neighbors can be known (by sending `peer_list` RPC queries), we compare the measurement result with the ground truth. If there is a false negative, it implies the remote node A' has some non-default setting on its node (e.g., use a `mempool` larger than the default Z). We then increase the `mempool` size in additional pre-processing measurements to discover a proper setting of the `mempool`. The result of the pre-processing can help guide the actual measurement to use a "right" parameter on the connections involving node A'.

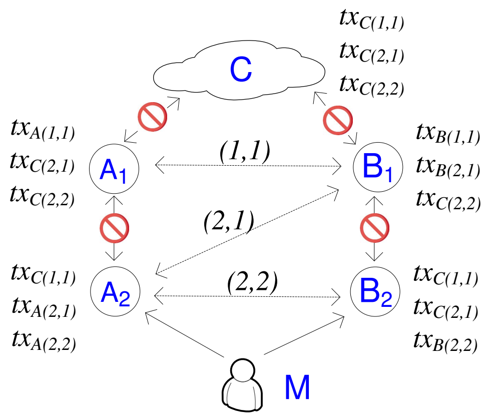
5.3 Parallel Measurement Framework

We previously described the primitive of measuring one connection between a source and a sink node. To measure a network, a native schedule is to serially run the pair-wise primitive over all possible pairs, which however incurs a long measurement time in the case of large networks and is not a scalable method. For time efficiency, we propose a parallel schedule that decomposes the set of all possible pairwise connections into subsets and measures all connections within each subset in parallel. In the following, we first describe the parallel measurement primitive (§ 5.3.1) and then the schedule that measures the entire network in parallel by repeatedly using the primitives (§ 5.3.2).

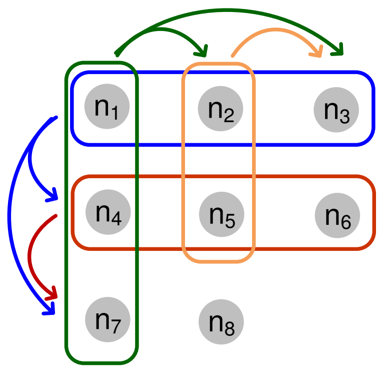
5.3.1 *Parallel Measurement Primitive.* We consider a pair of nodes whose connectivity is measured consist of a source node and a sink node. For instance, in Figure 2b, node A is a source node and node B is a sink node. In a parallel measurement, we consider measuring the connectivity between not one pair of source and sink nodes, but multiple such pairs. Specifically, suppose there are p "source" nodes $A_1, A_2, ...A_k, ...A_p$ and q "sink" nodes $B_1, B_2, ..., B_l, ..., B_q$; note k (l) is the index of a source (sink) node. In this bipartite graph, there are a total of $p \cdot q$ possible edges from a source to a sink. The objective here is to measure r specified edges out of the $p \cdot q$ ones. We denote by $sink(k, j)$ a sink node which is the j-th neighbor of a source node A_k. Then, the edge between A_k and $sink(k, j)$ is "indexed" by (k, j). Initially, assume there are sufficient funds set up in r Externally Owned Accounts (or EOAs).

- **p1** Node M sends a total of r transactions $\{tx_{C(k,l)}\}$ and propagates them to the Ethereum network. Any two different transactions are sent from different EOAs.
- **p2** To each Node A_k, Node M 1) first sends Z (e.g., 5120 for Geth) future transactions tx_F's followed immediately by 2) sending $\{..., tx_{C(k-1,q_{k-1})}, tx_{C(k+1,1)}, ...\}$. 3) It then sends $\{tx_{A(k,1)}, ...tx_{A(k,q_k)}\}$. Here, $tx_{A(k,i)}$ spends the same account with $tx_{C(k,i)}$ and its Gas is priced at $1.05Y$ Gwei. After **p2**, $tx_{C(k,i)}$ on Node Ai is replaced by $tx_{A(k,i)}$, while other tx_C's stay.
  It is noteworthy that after **p2**, Node M checks whether $tx_{A(k,\cdot)}$ are actually stored on Node A_k. It proceeds only if the checked condition holds. Node M carries out the check by observing if $tx_{C(k,\cdot)}$ is propagated from Node A_k before waiting for a timeout.
- **p3** Node M sends to each Node B_l Z future transactions tx_F's followed immediately by r transactions where the i-th transaction is a tx_B transaction (whose Gas is 95% of that of a tx_C transaction) if the i-th edge's sink is Node B_l, and otherwise, is a tx_c.
- **p4** For edge connecting A_k and B_l, Node M checks if $tx_{A(k,j)}$ (note that $sink(k, j) = l$) is present on Node B_l. If so, A_k and B_l are neighbors.

Note that Ethereum clients, including both Geth and Parity, limit the number of future transactions in their `mempool`. In our parallel measurements, we ensure the group size is much smaller than the limit of future transactions, which further ensures the measurement correctness, since all measurement transactions will be admitted and stored on the participant nodes.



(a) Network snapshot before Step p4



(b) Parallel-measurement schedule with an example network of 8 nodes

Figure 3: TopoShot's parallel measurement protocol; in Figure 3b, the four colors represent four *measurePar* iterations where a rectangle with rounded angles is the selected nodes A's in the measurement and the arcs are the collection of edges being measured. For instance, the blue iteration is $measurePar(\{n_1, n_2, n_3\}, \{n_4, n_5, n_6, n_7, n_8\}, \{\emptyset\})$.

Example: We use an example to illustrate the parallel measurement protocol. Among two sources A_1, A_2 and two sinks B_1, B_2, assume it measures the connections on three edges, that is, $\langle A_1, B_1\rangle, \langle A_2, B_1\rangle, \langle A_2, B_2\rangle$. Figure 3a depicts the snapshot of exercising our parallel measure method right after p3.

Ensuring isolation: As in the case of measurement primitive, isolation is critical to the success of our measurement method. In the parallel setting, a Node A needs to prevent propagating the tx_A transactions to Nodes B's via not only Nodes C's but also other Nodes A's and Nodes B's. For instance, in the example above, when measuring the connection between Node A_2 and B_1, it needs to ensure that $tx_A(2, 1)$ is not propagated to Node B_1 via Node A_1 or B_2. This is guaranteed by our measurement method because Nodes B_2 and A_1 store tx_C transactions and can be treated as a C node when measuring the connection between Node A_2 and B_1.

5.3.2 Parallel Measurement Schedule. Given a network of nodes $\{n_1, n_2, ..., n_N\}$, we partition the nodes into N/K groups where each group is of K distinct nodes; for instance, the i-th group (i starting from 0) is of nodes $\{n_{i*K+1}, n_{i*K+2}, ...n_{i*K+K-1}\}$.

We schedule the network measurement in the two rounds: The first round runs N/K iterations, where each iteration measures the edges between group i and the rest of the network. The second round measures the edges within a group.

To be more specific, we denote the parallel measurement primitive described in § 5.3.1 by $measurePar(\{A_i\}, \{B_i\}, \{C\})$. 1) Given the i-th group, the first round calls $measurePar(\{n_{i*K+1}, n_{i*K+2}, ...n_{i*K+K-1}\}, \{n_1, ..., n_{i*K-1}, n_{i*K+K}, ...n_N\}, \{\emptyset\})$, where A is the i-th group, B is the rest of the blockchain network, and C is empty. Each of these iterations sets a goal to measure $K \cdot (N-K)$ possible edges.

2) The second round measures the edges within groups. Specifically, given a group, it maps the first half of nodes as A and the other half as Nodes B. An iteration measures intra-group edges for all groups. It then applies the same splitting respectively for the first and second half of the group. In other words, the next iteration measures the intra-group edges for half of the original groups. This process repeats until the group size reaches 2.

Example: Suppose $N = 8$ and $K = 3$. The parallel schedule is of two rounds, each of two iterations, as illustrated by the four curved rectangles (with different colors) in Figure 3b.

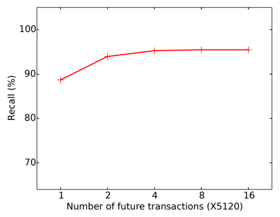
The first round runs the following two iterations: $measurePar(\{n_1, n_2, n_3\}, \{n_4, n_5, n_6, n_7, n_8\}, \{\emptyset\})$ which measures the $3 * 5 = 15$ edges across node group $\{n_1, n_2, n_3\}$ and group $\{n_4, n_5, n_6, n_7, n_8\}$. This is visualized by the horizontal rectangle in blue in the figure. The second iteration is $measurePar(\{n_4, n_5, n_6\}, \{n_7, n_8\}, \{n_1, n_2, n_3\})$ which measures all $3 * 2 = 6$ edges and is visualized by the horizontal rectangle in red in the figure.

The second round runs another two iterations: $measurePar(\{n_1, n_4, n_7\}, \{n_2, n_3, n_5, n_6, n_8\}, \{\emptyset\})$ which measures 5 edges across groups (i.e., edges $(n_1, n_2), (n_1, n_3), (n_4, n_5), (n_4, n_6), (n_7, n_8)$) by the vertical rectangle in green, and $measurePar(\{n_2, n_5\}, \{n_3, n_6\}, \{n_1, n_4, n_7, n_8\})$ which measures 2 edges (i.e., $(n_2, n_3), (n_5, n_6)$) by the vertical rectangle in orange.

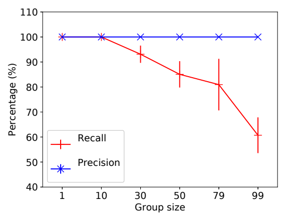
Complexity Analysis and Configuration of K: On the measurement of a network of N nodes with a group of size K, the total number of iterations is $\frac{N}{K} + \log K$ where the first round runs $\frac{N}{K}$ iterations and the second round runs $\log K$ iterations. Roughly, the number of iterations decreases with increasing K. However, making the value of K too large would lead to the overflow of `mempool` as it generates $K * (N - K)$ transactions in each iteration. In practice, an Ethereum node's `mempool` has a capacity of 5120 transactions and to bound the interference, we only use no more than 2000 transaction slots in the `mempool`. For an Ethereum network of 500 nodes, such as Ropsten, we use $K = 2000/500 = 4$ which results in a total of $500/4 + \log 4$=127 iterations.

6 MEASUREMENT RESULTS

Initially, we run a measurement node M that joins an Ethereum network, such as the Ropsten testnet. The measurement node M is



(a) Recall with TopoShot sending increasing number of future transactions.



(b) Precision and recall with increasing group size in parallel measurement.

Figure 4: Measurement validation results

set up without bounds on its neighbors, so it can be connected to the majority of the network.

6.1 Measurement Validation

The correct functioning of TopoShot relies on several factors that may vary in a deployed Ethereum network. For instance, TopoShot assumes the default `mempool` size on Geth nodes (i.e., 5120) that may not hold if an Ethereum node is configured with a different `mempool` size. The variance would introduce false negatives into TopoShot results and affect the recall. In this subsection, we validate the TopoShot results by evaluating/estimating the result recall. Here, we use the "external" experiment setup; in Appendix B, we use a fully local setup to conduct additional validation study.

Experiment setup: In addition to the measurement node M, we set up a local machine to play node B; the node joins the testnet being measured (e.g., Ropsten) and is configured with a number (e.g., 5000) larger than the size of testnet. After staying online for 12 hours in Ropsten, node B connects to 520 nodes, among which 471 nodes run Geth clients. The setup here is external as nodes A and B join a remote Ethereum network.

Validating measurement primitive (*measureOneLink*): We then iterate through the 471 nodes, selecting each node as node A to measure the connection between B and A using an unmodified TopoShot. In each iteration, the connection is measured three times. When running the measurement primitive, we verify that tx_C is evicted from nodes A and B. This is done by turning on the RPC interface and sending an `eth_getTransactionByHash` query to it. The final result is positive (i.e., there is a connection) if any of the three measurements is positive. For each unit experiment, we report the number of positive connections TopoShot can detect and from there calculate the recall.

We increase the number of future transactions sent in TopoShot and measure the recall using the validation method above. The results are shown in Figure 4a. With the increasing number of future transactions, the recall of TopoShot grows from 84% to 97%. An implication here is that even with a large number of future transactions, TopoShot does not reach 100% recall. We suspect the following culprits: 1) The remote node is configured with a custom `mempool` size much larger than the default 5120. 2) The node is configured with a custom Gas price threshold other than the default 10%; this threshold determines the `mempool`'s transaction replacement policy. 3) There are nodes who join Ropsten testnet but do not participate in forwarding transactions, preventing tx_A being propagated.

Validating parallel method (*measurePar*: In the same experiment we then validate TopoShot's parallel measurement method.

Recall parallel TopoShot is parameterized with p and q. In this experiment, we use $q = 1$ and vary p (referred to as the group size), that is, a node B' and p nodes A's in a parallel measurement. p is varied between 1 to 99.

Specifically, we set up a new node B' with the default 50 active neighbors and join the Ropsten testnet. It turned out its 35 active neighbors run Geth. We then serially measure the 35 neighbors, which successfully detects 29 neighbors. When running validation of the parallel method, we need to choose p nodes A's. When $p \leq 29$, we choose a subset of the 29 active neighbors of node B' to play nodes A's. When $p > 29$, we choose the 29 neighbors of node B', as well as the nodes that do not have connections with node B', to be nodes A's.

For each group size, we run the parallel measurement three times and report a positive result if any of the three returns a positive result. The results are presented in Figure 4b. The precision is always 100%. The recall is initially 100% until the group size is larger than 29. It then decreases as the group grows larger. For a group of 99 nodes, the recall is about 60%. The reason for a non-100% recall under a large group is that TopoShot does not guarantee isolation among nodes $\{A\}$, and a larger group increases the chance of non-isolation/interference among nodes $\{A\}$.

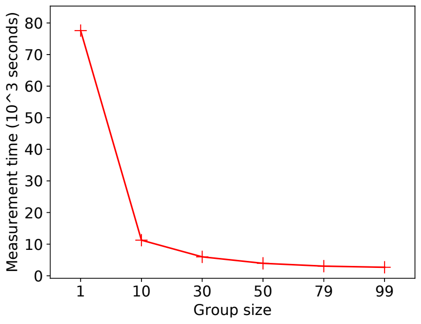


Figure 5: Speedup of TopoShot's parallel measurement over the serial measurement

Measurement speedup of the parallel method: We also report the time of measuring the same group of nodes with varying group size, with the purpose to evaluate possible speedup by the parallel measurement over the serial one. In a similar experiment setup, the measurement target is a group of 100 nodes. With about 4950 edges detected, the measurement times are reported in Figure 5. It can be seen that as the group size K increases, the time to measure the same group of nodes (as in the previous experiment) decreases significantly. For instance, with a group size $K = 30$, the measurement time is reduced by an order of magnitude (about 10× times).

6.2 Testnet Measurement Results

6.2.1 Ropsten Results.

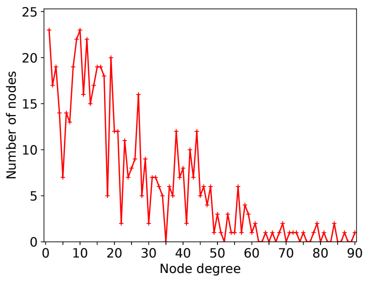


Figure 6: Node degree distribution in Ropsten

Table 4: Graph properties of the Ropsten testnet

	Measured Ropsten	ER ($n = 588$, $m = 7496$)	CM	BA ($n = 588$, $l' = 26$)
Diameter	5	3.0	5.2	3.0
Periphery size	36	293.5	24.9	509.4
Radius	3	3.0	3.0	2.0
Center size	36	293.5	51.7	78.6
Eccentricity	4.037	3.0	3.98	2.87
Clustering coefficient	0.207	0.044	0.139	0.159
Transitivity	0.127	0.044	0.122	0.156
Degree assortativity	-0.1517	0.0026	-0.0664	-0.0181
Clique number	60.75	250.3	557.4	50.6
Modularity	0.0605	0.161	0.152	0.102

Table 5: Detected communities in Ropsten testnet

Community index	No. of nodes	Intra-comm. edges (density)	Inter-comm. edges
1	92	423 (10%)	1547
2	142	603 (6%)	1612
3	107	548 (9.7%)	1827
4	84	391 (11%)	1505
5	75	379 (14%)	1704
6	51	127 (10%)	773
7	37	121 (18%)	840

We first conducted a measurement study on Ropsten testnet. We use parallel measurement method with parameter $K = 60$. In particular, the testnet is under loaded and there are not sufficient "background" transactions in mempools. We tried to apply TopoShot, as is, to measure Ropsten and found that however low Gas price we set for tx_C (recall Step ❶ in TopoShot), the transaction will always be included in the next block, leaving no time for accurate measurement. To overcome this problem, we launch another node that sends a number of "background" transactions (from a different account than tx_C). This effectively populate an operating mempool and helped tx_C stay in a mempool for long enough during the measurement period. We encounter the same situation when measuring Goerli and use the same trick here. Note that more than 95% of peer nodes our supernodes initially connect to stay connected throughout the measurement period.

In the testnet, a target node may run a non-default setting in which they forward future transactions, invalidating the assumption made in TopoShot. Such a custom node is avoided in our measurement study as follows: In the pre-processing, one can launch an additional monitor node (to the measurement node) to connect to the target node one. The measurement node then sends a future transaction to the target node. If the monitor node observes the future transaction from the target node, the target node is removed from the measurement. Besides, the pre-processing phase in TopoShot also avoids unresponsive nodes.

We present a snapshot of the Ropsten testnet taken on Oct. 13, 2020. The precision of the measurement result is 100% and recall is 88% (under group size $K = 60$), using a validation method described above. The network contains 588 (Geth) nodes and 7496 edges among them. This result has the test node and its edges excluded. The degree distribution is plotted in Figure 6. Most nodes have a degree between 1 and 60: Particularly, 4% of nodes have degree 10, another 4% have degree 1 and another 4% have degree 12. Omitted in the figure are ten nodes with degree between 90 and 200. This result shows that degrees by active links are much smaller than the default number of inactive neighbors (250).

Table 4 summarizes the characteristics of the measured testnet in terms of distances, assortativity, clustering and community structure. 1) For distances, the network diameter, defined as the maximal distance between any pair of nodes, is 5, and the radius is 3. The number of center nodes and periphery nodes, defined respectively as the nodes with eccentricity equal to radius and diameter, are both 36. 2) Degree assortativity, which measures how likely a node connects to a similar node, is -0.1517. 3) The clustering coefficient, which shows how well nodes in a graph tend to form cliques together, is 0.207. The transitivity, which considers the clustering of particular 3-node substructure, is 0.127. 4) There are 60.748 unique cliques detected in the testnet. The modularity of the testnet, which measures the easiness of partitioning the graph into modules, is 0.0605.

As a baseline for comparison, we generate a random graph following the Erdos-Renyi [25] (ER) model which generates an edge between each pair of nodes using the same probability, independently. It follows a binomial degree distribution and is commonly used as the network-analysis baselines. We use the same number of vertices and edges with the measured Ropsten network (that is, $n = 588$ and $m = 7496$) when generating the Erdos-Renyi random graph. We run the graph generation algorithms for 10 times and report the average properties of these random graphs in Table 5. Particularly, the density is calculated by the number of measured intra-community edges divided by the number of total possible edges in that community. For instance, the density of a community of 92 nodes and 423 intra-community edges is $423/\binom{8,000}{2} = 0.10$. Besides, Table 5 shows other two random graphs, namely configuration model [40] (CM), and Barabasi-Alber [19] (BA). The former is generated using the same sequence of node degrees with the measured testnet, and the latter is generated using the same number of nodes ($n = 588$) and same average node degree ($l' = 26$).

Compared with the ER random graph, the measured Ropsten network has a much larger diameter, a smaller center size, a larger clustering coefficient, and more importantly, fewer cliques and lower modularity. This is similarly the case when comparing Ropsten with CM (except for CM's comparable diameter) and BA (except for BA's comparable number of cliques). The implication is that a Ropsten network is much more resilient to network-partition attacks (e.g., eclipse and other DoS attacks) than these random graphs.

We also detect the communities in the Ropsten testnet, using the NetworkX tool [3] implementing the Louvain method described in [21]. The results are in Table 5. There are seven communities detected. The biggest community is community number two with 22% of the nodes of the network. The average degree in the community is 19, and 9% of the nodes (i.e., 13 out of 142 nodes) only have a degree of 1. By comparison, community number five contains 12.7% of the nodes with the largest average degree 32.8.

6.2.2 Summary of Rinkeby & Goerli Results. We conduct similar measurements on two other major Ethereum testnets, Rinkeby and Goerli. Here, we summarize what's noteworthy in the results while deferring the full presentation to Appendix D. From our measurement results, Rinkeby has smaller node degrees than Ropsten. Many Geth nodes in Rinkeby are with degrees smaller than 15, and between degrees 15 and 180 the nodes are evenly distributed. In terms of graph statistics, Rinkeby's modularity (0.0106) is much lower than Goerli's (0.048) which is comparable with Ropsten's modularity (0.0605); this result implies that Rinkeby's the most resilient against network partitioning.

Explaining the results: In the measurement results, we consistently observe smaller modularity in testnets than that in random graphs. Full explainability of the measurement result is challenging and out of the scope of this paper. We take preliminary efforts to explain the measured results as follows.

We suspect the measurement results, particularly the discrepancy to the properties of random graphs and the much lower modularity, are due to the way Ethereum nodes choose/promote active links and the scale of the networks measured. Briefly, in the Ethereum protocol, a node maintains a "buffer" of inactive neighbors from which 50 active neighbors are selected in the case of existing active neighbors go offline. At the first glimpse, the presence of this buffer localizes the selection of active neighbors in a smaller candidate set than all the nodes as in the random graph, and it should facilitate forming the network of higher modularity. However, by looking more closely at the Ethereum protocol, a node N's candidate buffer consists of node N's inactive neighbors and node N's inactive neighbors' inactive neighbors. For instance, with each node of 272 inactive neighbors by default, the buffer size is $272 * 272 = 73984$ which is larger than the size of the testnets we measured. Thus, the effect of localization is not reflected in the testnet results. In fact, the deduplication of active neighbors (i.e., Ethereum clients, such as Geth, check if a recently promoted neighbor is already an active neighbor) may contribute to the much lower modularity in the measured testnets. While here we explain the measurement results by qualitative analysis, we leave it to the future work the quantitative modeling and analysis of Ethereum network-connection protocols.

6.3 Mainnet Measurement Results

Measuring the mainnet's topology raises new challenges: 1) Due to ethical concerns, the measurement should not interfere the normal operation of live mainnet nodes. 2) Due to mainnet's large scale (about 8,000 nodes and $\binom{8,000}{2} = \frac{1}{2} \cdot 8,000 \cdot (8,000 - 1)$ possible links) and the high price of Ether, measuring the entire network of mainnet incurs high cost, estimated to be more than 60 million USD as will be analyzed.

To tackle the ethical challenge, we propose a TopoShot extension to additionally verify certain conditions and ensure the non-interference to the service of target mainnet nodes. To bypass the high-cost challenge, in this work, we focus on measuring the topology of a small but critical subnetwork, instead of the entire mainnet. We conduct the measurement study on the mainnet on May 11th, 2021 and have spent 0.05858 Ether (amount to 197.94 USD at the price as of Aug 2021).

Non-interference extension of TopoShot: Consider a measurement node M runs TopoShot against a target node S in the Ethereum network C (S can be either A or B in our measurement primitive as in Figure 2b). Suppose the measurement starts at time t_1 and ends at t_2. Node M sets tx_C's Gas price at $Y = Y_0$ and monitors the following two conditions. Only when both conditions are met, the measurement proceeds.

V1) All blocks produced in $[t_1, t_2 + e]$ are full in the sense that the Gas limit of each block is filled. e denotes the expiration time of an unconfirmed transaction buffered in an Ethereum node. For instance, a Geth node would drop an unconfirmed transaction $e = 3$ hours after it is submitted to the node, if it is not mined.

V2) In the blocks produced during $[t_1, t_2 + e]$, the included transactions have Gas prices higher than Y_0.

This extended TopoShot achieves the following non-interference property: *The Ethereum blocks produced with the measurement turned on include the same set of transactions with the blocks produced with the measurement turned off.* We formally state the property and prove it in Appendix C.

Goal: mainnet's critical subnetwork: With the above pricing strategy, measuring a pair of nodes on the mainnet costs $7.1 \cdot 10^{-4}$ Ether or 1.91 USD (at the Ether price as of May, 2021). Thus, for the mainnet that consists of more than 8,000 nodes, measuring all $\frac{1}{2} \cdot 8,000 \cdot (8,000 - 1)$ possible links would cost $22.845 \cdot 10^3$ Ether or more than 60 million USD. We thus refrain from directly measuring the entire mainnet in this work.

Instead, we choose a smaller but critical subnetwork of the mainnet to measure. Our observation is that in today's blockchain networks, essential transaction activities are centralized to few popular "services" that account for a small portion of the nodes in the network, such as popular transaction relay service (e.g., SrvR1[4] that relays 63% of Ethereum transactions on the mainnet) and mining pools.

We aim to answer the following research question: *RQ: Do Ethereum mainnet nodes prioritize the critical service nodes as their active neighbors?*

To address the research, we design a measurement study on the mainnet that 1) discovers Ethereum nodes running behind the known popular services (including transaction relay and mining pools) and then 2) uses TopoShot to measure the pair-wise connections among the discovered service-backend nodes.

Step 1: Discovering critical nodes: We discover the mainnet nodes on the backend of critical services. We use the approach described in the existing work [35]. Briefly, the approach is to obtain the client version of the backend nodes by submitting the standard Ethereum RPC query (i.e., `web3_clientVersion`) through

[4]We anonymize the names of critical services tested to protect their service.

the service frontend and to match the version against the ones in the Ethereum handshake messages received on a local "supernode" joining the mainnet, which is similar to the existing measurement work [32].

Using the above method, we discover the following mining-pool nodes on the mainnet: 59 SrvM1 nodes, 8 SrvM2 nodes, 6 SrvM3 nodes, 2 SrvM4 nodes, 2 SrvM5 nodes, and 1 SrvM6 node. We also discover the following transaction-relay nodes on the mainnet: 48 SrvR1 nodes and 1 SrvR2 node. When discovering the nodes, we use the codename revealed through the `web3_clientVersion` query. The measurement result is consistent with [35].

Table 6: Connections among critical nodes

Type	Conn.	Type	Conn.
SrvR1- SrvM1	✓	SrvM1- SrvM1	✗
SrvR1- SrvM2	✓	SrvM1- SrvM2	✓
SrvR1- SrvM3	✓	SrvM1- SrvM4	✓
SrvR1- SrvM4	✓	SrvM1- SrvM3	✓
SrvR2- SrvM1	✗	SrvM2- SrvM2	✓
SrvR2- SrvM2	✗	SrvM2- SrvM3	✓
SrvR2- SrvM3	✗	SrvM2- SrvM4	✓
SrvR2- SrvM4	✗	SrvM3- SrvM4	✓
SrvR2- SrvR1	✗	SrvR1- SrvR1	✓

Step 2: Measuring topology among critical nodes: We run the extended TopoShot to detect whether critical nodes discovered as above are connected with each other. We consider three possible connection types: the inter-connection between a mining-pool node and a relay-service node, the connection between two mining-pool nodes and the connection between two relay-service node. For each case, we select random nodes from each service and measure all possible links. For measuring the connection between "SrvR1-SrvM1", for instance, we select two random SrvR1 nodes and two random SrvM1 nodes, and measure the four combinations of links. In addition, we select two nodes for SrvM2 and select one node for each of the services: SrvR2, SrvM3, SrvM4. In total, we choose 9 mainnet nodes.

We report the result in Table 6. We make the following observation: 1) A node behind relay service SrvR1 connects to all tested mining pools and other SrvR1 nodes. It does not connect to other relay services such as SrvR2. 2) The single node behind relay service SrvR2 does not connect to any mining pools or other relay service. Here, SrvR2's node may randomly choose neighbors as vanilla Ethereum clients do. 3) Nodes behind all mining pools connect to nodes of the same pool and other pools. They also connect to SrvR1. The only exception is that SrvM1 nodes do not connect to other SrvM1 node.

Explaining the results: There are two possible explanations of the results: a) SrvR1 and all mining pools run supernodes internally, which connect to all other nodes. SrvR2 runs a regular node that declines incoming connection requests once its active neighbors are full. b) An SrvR1 node prioritizes the connections to other SrvR1 nodes and mining-pool nodes. It does not prioritize connecting to other RPC-service nodes like SrvR2. So are the mining pool nodes.

6.4 Summary of Measurement Costs/Time

We summarize the measurement costs/time in Table 7, which reports the actual Ether cost spent for measuring the testnets and the estimated cost of measuring the full topology of mainnet. The mainnet cost is estimated by multiplying the pairwise-measurement cost by the number of possible edges in the network (as mentioned before). Note that in the mainnet, the measurement transactions' Gas prices are set to be higher than at least 10% of the pending transactions in the `mempool` (for estimation purposes, we assume the target node's `mempool` has the same content as the measurement node's mempool).

Table 7: Summary of measurement studies on the testnets/-mainnet. # refers to "number".

Network	Size (# of nodes)	Cost (Ether)	Date	Duration (hours)
Ropsten	588	0.067	Oct. 30, 2020	12
Rinkeby	446	2.10	Nov. 15, 2020	10
Goerli	1025	0.62	Oct. 20, 2020	20
mainnet	9	0.05858	May. 15, 2021	0.5

7 ETHICAL DISCUSSION

In this work, we use TopoShot to measure testnets. While the approach is active measurement (to refill underwhelmed `mempool` in the testnet), the testnets do not run business, and the possible service interruption to the testnets will have limited impacts. We also measure a limited sub-network on the Ethereum mainnet. As analyzed before in § 6.3, the presence of measurement using the TopoShot extension does not affect what set of transactions are included in the blockchain. A more formal statement is in Theorem C.2 which is proven in Appendix C.2. We believe TopoShot's impact to normal transactions when measuring the mainnet is small.

8 CONCLUSION

This work presents TopoShot, a measurement study that uncovers Ethereum's network topology by exploiting transaction replacement and eviction policies. TopoShot achieves the perfect precision and high recall. A parallel schedule is proposed to apply the pairwise measurement to large-scale networks. TopoShot uncovers the topology of three major Ethereum testnets, which show their difference to random graphs and high resilience to network partitioning. We also use TopoShot to measure critical service interconnection in the mainnet which reveals biased neighbor selection strategies by top mining pools and relay service nodes.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers in ACM IMC'21, SIGCOMM'21, and SIGMETRICS'21. The authors are partially supported by the National Science Foundation under Grants CNS1815814 and DGE2104532.

REFERENCES

[1] Retrieved May, 2021. Aleth – Ethereum C++ client, tools and libraries. https://github.com/ethereum/aleth.

[2] Retrieved May, 2021. Bug disclosure and authors' conversations. Privately shared link: https://shorturl.at/kuET6.

[3] Retrieved May, 2021. Community detection for NetworkX's documentation. https://python-louvain.readthedocs.io/en/latest/.

[4] Retrieved May, 2021. Ethereum & IPFS APIs. Develop now on Web 3.0. https://infura.io/.

[5] Retrieved May, 2021. Ethereum Bounty Program. https://bounty.ethereum.org/.

[6] Retrieved May, 2021. Ethereum Mainnet Statistics. https://www.ethernodes.org/.

[7] Retrieved May, 2021. Etherscan: Ethereum (ETH) Blockchain Explorer. https://etherscan.io/.

[8] Retrieved May, 2021. Geth: the Go Client for Ethereum. https://www.ethereum.org/cli#geth.

[9] Retrieved May, 2021. The Goerli testnet of Ethereum. https://goerli.etherscan.io.

[10] Retrieved May, 2021. Hyperledger Besu. https://www.hyperledger.org/use/besu.

[11] Retrieved May, 2021. Introduction to Swarm. https://swarm-guide.readthedocs.io/en/latest/introduction.html.

[12] Retrieved May, 2021. Nethermind Ethereum client. https://nethermind.io/client.

[13] Retrieved May, 2021. Parity Ethereum is now OpenEthereum: Fast and feature-rich multi-network Ethereum client. https://www.parity.io/ethereum/.

[14] Retrieved May, 2021. Quiknode: Blockchain Infrastructure Cloud. https://www.quiknode.io/.

[15] Retrieved May, 2021. The Rinkeby testnet of Ethereum. https://rinkeby.etherscan.io.

[16] Retrieved May, 2021. The Ropsten testnet of Ethereum. https://ropsten.etherscan.io.

[17] Retrieved May, 2021. Sparkpool: Crypto Mining and Staking Pool. https://www.sparkpool.com/en/.

[18] Retrieved Sep., 2021. ETH live marketcap. https://coinmarketcap.com/currencies/ethereum/.

[19] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. Reviews of modern physics 74, 1 (2002), 47.

[20] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. 2014. Deanonymisation of Clients in Bitcoin P2P Network. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM, 15–29. https://doi.org/10.1145/2660267.2660379

[21] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment 2008, 10 (2008), P10008.

[22] Tong Cao, Jiangshan Yu, Jérémie Decouchant, Xiapu Luo, and Paulo Veríssimo. 2020. Exploring the Monero Peer-to-Peer Network. In Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12059), Joseph Bonneau and Nadia Heninger (Eds.). Springer, 578–594. https://doi.org/10.1007/978-3-030-51280-$4_3$1

[23] Erik Daniel, Elias Rohrer, and Florian Tschorsch. 2019. Map-Z: Exposing the Zcash Network in Times of Transition. In 44th IEEE Conference on Local Computer Networks, LCN 2019, Osnabrueck, Germany, October 14-17, 2019, Karl Andersson, Hwee-Pink Tan, and Sharief Oteafy (Eds.). IEEE, 84–92. https://doi.org/10.1109/LCN44214.2019.8990796

[24] 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 Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers. 550–566. https://doi.org/10.1007/978-3-030-32101-$7_3$2

[25] P. Erdos and A Renyi. 1960. On the Evolution of Random Graphs. In PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES. 17–61.

[26] Yue Gao, Jinqiao Shi, Xuebin Wang, Qingfeng Tan, Can Zhao, and Zelin Yin. 2019. Topology Measurement and Analysis on Ethereum P2P Network. In 2019 IEEE Symposium on Computers and Communications, ISCC 2019, Barcelona, Spain, June 29 - July 3, 2019. IEEE, 1–7. https://doi.org/10.1109/ISCC47284.2019.8969695

[27] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. 2018. Decentralization in Bitcoin and Ethereum Networks. In Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 - March 2, 2018, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10957), Sarah Meiklejohn and Kazue Sako (Eds.). Springer, 439–457. https://doi.org/10.1007/978-3-662-58387-$6_2$4

[28] Matthias Grundmann, Till Neudecker, and Hannes Hartenstein. 2018. Exploiting Transaction Accumulation and Double Spends for Topology Inference in Bitcoin. In Financial Cryptography and Data Security - FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10958), Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala (Eds.). Springer, 113–126. https://doi.org/10.1007/978-3-662-58820-8_9

[29] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In USENIX Security 2015, Washington, D.C., USA, Jaeyeon Jung and Thorsten Holz (Eds.). USENIX Association, 129–144. https://www.usenix.org/conference/usenixsecurity15

[30] Sebastian A. Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann. 2020. Mapping the Interplanetary Filesystem. In 2020 IFIP Networking Conference, Networking 2020, Paris, France, June 22-26, 2020. IEEE, 289–297. https://ieeexplore.ieee.org/document/9142766

[31] Lucianna Kiffer, Asad Salman, Dave Levin, Alan Mislove, and Cristina Nita-Rotaru. [n.d.]. Under the Hood of the Ethereum Gossip Protocol. ([n. d.]).

[32] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. 2018. Measuring Ethereum Network Peers. In Proceedings of IMC 2018. 91–104. https://dl.acm.org/citation.cfm?id=3278542

[33] Philip Koshy, Diana Koshy, and Patrick D. McDaniel. 2014. An Analysis of Anonymity in Bitcoin Using P2P Network Traffic. In Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 8437), Nicolas Christin and Reihaneh Safavi-Naini (Eds.). Springer, 469–485. https://doi.org/10.1007/978-3-662-45472-$5_3$0

[34] Kai Li, Jiaqi Chen, Xianghong Liu, Yuzhe Richard Tang, XiaoFeng Wang, and Xiapu Luo. 2021. As Strong As Its Weakest Link: How to Break Blockchain DApps at RPC Service. In 28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021. The Internet Society. https://www.ndss-symposium.org/ndss-paper/as-strong-as-its-weakest-link-how-to-break-blockchain-dapps-at-rpc-service/

[35] Kai Li, Yibo Wang, and Yuzhe Tang. 2021. DETER: Denial of Ethereum Txpool sERvices. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15 - 19, 2021, Virtual Event, Republic of Korea.

[36] Petar Maymounkov and David Mazières. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers (Lecture Notes in Computer Science, Vol. 2429), Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron (Eds.). Springer, 53–65. https://doi.org/10.1007/3-540-45748-8_5

[37] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2015. Discovering bitcoin's network topology and influential nodes. University of Maryland, Tech. Rep (2015).

[38] Till Neudecker. 2019. Characterization of the Bitcoin Peer-to-Peer Network (2015-2018). Technical Report 1. Karlsruher Institut für Technologie (KIT). https://doi.org/10.5445/IR/1000091933

[39] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. 2016. Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network. In 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France, July 18-21, 2016. IEEE Computer Society, 358–367. https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070

[40] Mark EJ Newman. 2003. The structure and function of complex networks. SIAM review 45, 2 (2003), 167–256.

[41] Aristodemos Paphitis, Nicolas Kourtellis, and Michael Sirivianos. 2021. A First Look into the Structural Properties and Resilience of Blockchain Overlays. CoRR abs/2104.03044 (2021). arXiv:2104.03044 https://arxiv.org/abs/2104.03044

Appendices

A TXPROBE'S APPLICABILITY TO ETHEREUM: ADDITIONAL DETAILS

TxProbe is inapplicable to measuring the topology of Ethereum network, due to two distinct features in Ethereum: 1) Ethereum's propagation model where transactions can be directly propagated without announcement, and 2) Ethereum's account-based model where the balance state can be arbitrarily recharged while a Bitcoin balance can only transit one way, from unspent to spent (as in its UTXO model). In the main text, we explain why TxProbe's inapplicable in Ethereum due to 2). Here, we explain TxProbe's inapplicable in Ethereum due to 1).

Briefly, before sending tx_A, TxProbe sends two double-spending transactions respectively to node A and B such that tx_A on node B will become an orphan transaction that is not propagated further. However, with Ethereum's account model, tx_A on node B would not necessarily be an orphan transaction (or equivalently a future transaction in Ethereum's jargon). It can be an overdraft transaction that is propagated. More detailed description is below:

TxProbe [24] actively measures Bitcoin network topology, by exploiting its handling of orphan/double-spending transactions in transaction propagation. Other works [28] measure the Bitcoin topology using a similar approach. In the following, we describe the working of TxProbe [24] in detail, with the purpose to discuss its applicability to measuring Ethereum topology. Suppose using TxProbe to measure the connection between nodes A and B. The measurement node first sends to nodes A and B two double-spending transactions, say tx'_A and tx'_B. It then sends the third transaction tx_A spending tx'_A to node A. It observes the presence of the transaction tx_A on node B. If it is present, there is a connection between nodes A and B.

Applying TxProbe to measuring Ethereum topology is unfeasible due to Ethereum's account model: Ethereum adopts a different model to store ledger states than Bitcoin, and the definition of orphan transactions in Ethereum is different than that in Bitcoin. Specifically, Ethereum stores ledger state (how much cryptocurrency an address/account has) in per-account balances, namely the account model, while Bitcoin stores the balance state in per-transaction UTXO, namely the UTXO model. Under UTXO, an orphan transaction is a transaction that spends an input whose state is yet to be determined. This makes the third transaction in the TxProbe protocol an orphan on Node B, which does not propagate. However, under the account model, an orphan transaction (or so-called future transaction) is one with a noncontinuous nonce to any previous transaction, where a nonce is per a sender account. Thus, the third transaction that spends a double-spending transaction is not necessarily an orphan, as it may be an over-drafting transaction with a valid nonce (in which case, the transaction will be propagated by node B).

B LOCAL VALIDATIONS

B.1 Local Validation of Serial Measurement

Local evaluation: In this experiment, we set up three local Ethereum nodes, mutually connected and without the communication to any external Ethereum nodes. The three nodes represent nodes M, A and B in the TopoShot protocol. In this local environment, we aim at evaluating TopoShot's correctness with respect to varying `mempool` sizes on node A.

On node A, we vary node A's `mempool` sizes from 3120 to 9120. In the experiments, we also populate the three-node network with a varying number of pending transactions tx_O's, such as 1, 1000, 2000, and 3000 such transactions.

We use the measurement results from TopoShot and compare it against the ground-truth to report the recall in Figure 7. The result shows that given `mempool` size X and the number of pending transactions X', TopoShot achieves 100% recall when $X - X' <= 5120$. Otherwise, the recall drops to 0%.

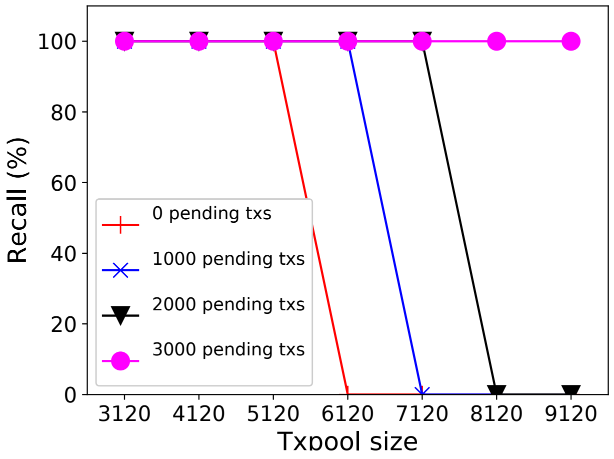


Figure 7: Recall with increasing `mempool` size

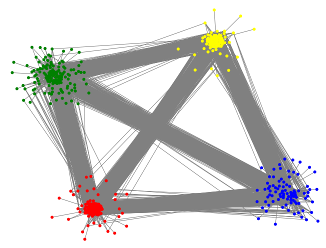
This validation study implies that matching the number of pending transactions to the actual `mempool` size is crucial to achieving 100% measurement recall.

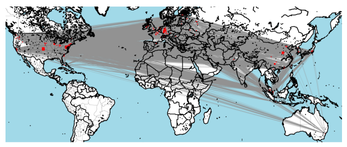
B.1.1 *Local Validation of Parallel Measurement.*

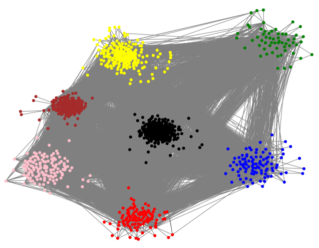
We conduct a validation of the parallel measurement method in a local environment without connection to a remote Ethereum network. Here, the measurement node M, two nodes A_1 and A_2, as well as node B, are run on local machines under our control.

In terms of the connections among A_1, A_2 and B, there are two permutations of three possibilities, that is, $P(3, 2) = 8$. Given the symmetry (e.g., $\langle A_2, B\rangle$ is equivalent to $\langle A_1, B\rangle$), we consider six possibilities as listed below. For each possibility, we use TopoShot to conduct measurements for 100 times. The final result is positive if any of the 100 measurements returns a positive result (i.e., there is a connection). Then by comparing the measurement results and the ground-truth, we obtain the precision and recall of the measurement results.

As can be seen from table 8, all results are with 100% recall and precision, even when there is a connection between A_1 and A_2. The theoretic measurement inaccuracy when A_1 connects to A_2 seems unlikely to occur in practice.


(a) Community in Rinkeby


(b) Geo distribution of Rinkeby


(c) Community in Goerli

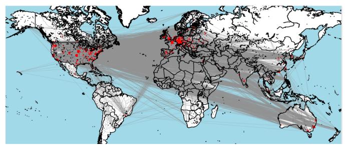

(d) Geo distribution of Goerli

Figure 8: Visualization of measured Rinkeby and Goerli networks

Table 8: Recall and precision of using TopoShot on local nodes

	Recall	Precision
$\langle A_1, A_2\rangle$, $\langle A_1, B\rangle$, $\langle A_2, B\rangle$	100%	100%
$\langle A_1, A_2\rangle$, $\langle A_1, B\rangle$ $\langle A_1, A_2\rangle$, $\langle A_2, B\rangle$	100%	100%
$\langle A_1, A_2\rangle$	100%	100%
$\langle A_1, B\rangle$, $\langle A_2, B\rangle$	100%	100%
$\langle A_1, B\rangle$ (also $\langle A_2, B\rangle$)	100%	100%
Null	100%	100%

C MEASUREMENT EXTENSION FOR NON-INTERFERENCE

C.1 Extending TopoShot with Non-interference Verification

Goals: When running our TopoShot against an operational Ethereum network, notably the mainnet, it is required that the measurement does not interfere with the normal operations of the network; for instance, the TopoShot should not evict any transactions that are otherwise included in the blockchain. This non-interference property is formally described in our analysis framework in § C.2.

Design rationale: To design TopoShot extension for assurance of non-interference, one possible approach is to set a low Gas price (i.e., TopoShot's parameter Y) and prove the non-interference, *a priori*, by considering the theoretically worst case that could occur after the measurement starts. Our initial design follows this approach, but just to find it is unfeasible with the current Ethereum-node settings.[5]

Instead of proving non-interference a priori, we aim at verifying the non-interference a posteriori. That is, the measurement node M initially sets a conservatively low Gas price (e.g., based on heuristics) and conducts the measurement. Meanwhile the node monitors several conditions on the tested network during and after the measurement, in order to establish non-interference *posterior*.

Measurement extension for verification: Consider a measurement node M runs TopoShot against a subject node S in the Ethereum network C (S can be either Node A or B as in our serial-measurement model in Figure 2b). The measurement starts at time t_1 and ends at t_2. Node M sets a low Gas price at $Y = Y_0$ and monitors the blockchain on the following conditions:

[5]To be specific, Geth's default `mempool` length 5120 is too small to feed all the blocks produced in the three-hour span (expiration time), by considering the worst case that no new transactions are submitted to the Ethereum network after the measurement.

V1) All blocks produced in $[t_1, t_2 + e]$ are full in the sense that the Gas limit of each block is filled. Here, e denotes the expiration time of a transaction in Ethereum-node `mempool`, for instance, $e = 3$ hours in Geth by default.

V2) In the blocks produced in $[t_1, t_2 + e]$, all transactions' Gas prices are higher than the preset Gas price Y_0.

C.2 Non-interference Analysis

In this subsection, we first define what the measurement interference means. We then prove that verified Conditions V1 and V2 ensure non-interference on the measured nodes.

Intuitively, non-interference means the action of measurement does not affect what blocks are produced by the Ethereum network being tested. In other words, with and without the measurement P, the blocks produced by the Ethereum network should be the same.

Formally, we consider a node M runs a measurement process against a subject node S, which is connected to the rest of an Ethereum network C. In the case that TopoShot is used to detect the link between A and B, S can be either node A or B. The measurement process starts at time t_1 and ends at time t_2.

Definition C.1. Consider a measurement process parameterized by $P(M, S, C, t_1, t_2)$. Denote by $\{b_i\}$ the sequence of blocks produced by the Ethereum network (S, C) in period $[t_1, t_2 + e]$.

Now consider a hypothetical world in which the measurement did not occur at t_1 and the Ethereum network produces the sequence of block headers with $\{b'_i\}$. The hypothetical world is deterministic in the sense that it produces the same block from the same miner at the same time with the actual world with measurement, that is, block b'_i has the same timing with b_i.

$P(M, S, C, t_1, t_2)$ does not interfere with the measured Ethereum network (S, C), if and only if the transactions included in each block b_i (i.e., the block at the index i of the sequence) in the actual world with measurement are identical to those included in block b'_i in the hypothetical world without measurement.

Theorem C.2. *Consider a measurement $P(M, S, C, t_1, t_2)$ is conducted using the method of* TopoShot. *If Conditions V1 and V2 hold, P does not interfere with the Ethereum network (S, C).*

Proof. Generally speaking, blocks can be produced by node S or other nodes in Ethereum network C. Because TopoShot will not evict transactions on nodes besides S, the measurement will not affect the block produced by nodes other than S. Thus, we consider in this proof the "worst case" that all blocks $\{b_i\}$ are produced by node S.

Due to the design of TopoShot, the measurement process will evict only the transactions in S's `mempool` (as late as of time t_2) whose Gas prices are lower than Y_0. For other transactions whose Gas prices are higher than Y_0 and transactions submitted after t_2, measurement process P will not affect them.

Now, we are ready to prove the theorem by contradiction: Assuming there is interference under Conditions V1 and V2, our goal of the proof is to find contradictory. That is, with V1 and V2, there is at least one transaction included in a block produced hypothetically without measurement, say b'_i, and that is not included in the corresponding actual block b_i. We name this transaction by tx_l. Since the measurement will only affect the transactions with Gas prices lower than Y_0, tx_l's Gas price must be lower than Y_0.

Because of $V1$, b_i must be full. Thus, there must be a "victim" transaction in the hypothetical world, say tx_h, that is included in b_i but is replaced by tx_l in b'_i. Because tx_h's Gas price is higher than Y_0 and is not affected by the presence of measurement, tx_h must reside in the node's `mempool` in the hypothetical world without measurement.

Now, we can constitute a scenario in the hypothetical world that Miner S is faced with two transactions in its `mempool`, tx_l and tx_h. To make tx_l in the blockchain, S must prioritize tx_l (with Gas price lower than Y_0) over tx_h (with Gas price higher than Y_0) to mine. This *contradicts* with the property in Ethereum client implementations (both Geth and Parity) that transactions of higher Gas price have higher priority to be mined than those of lower Gas price. Note that here both tx_l and tx_h have small enough Gas and can fit into block b_i under the block Gas limit. □

D MEASUREMENT RESULTS OF RINKEBY & GOERLI

Table 9: Graph properties of the Rinkeby testnet

	Measured Rinkeby	ER (n=446, m=15380)	CM	BA (n=446, l=69)
Diameter	4	2.7	4.6	2.0
Periphery size	203	512.0	76.1	446.0
Radius	3	2.0	3.0	2.0
Center size	243	442.4	233.3	446.0
Eccentricity	3.455	2.008	3.4953	2.0
Clustering coefficient	0.4375	0.1548	0.3407	0.3592
Transitivity	0.4981	0.1548	0.3589	0.3513
Degree assortativity	-0.03202	-0.001536	-0.03275	-0.04555
Clique number	274775.0	150.6	383.2	82.5
Modularity	0.01063	0.08198	0.07332	0.05310

We similarly apply the TopoShot method to measure the Rinkeby testnet. Compared with Ropsten, Rinkeby is more heavily used and the `mempools` there contain more transactions. For instance, on our local node M connected to Rinkeby, it is not uncommon that the `mempool` has more than 4500 transactions. We thus estimate the median Gas price in the `mempool` (using the method described in § 5.2.1) and use it as tx_C's Gas price.

Noteworthy is that during this measurement, we found when our measurement node M sends future transactions (as in Step ❷) to certain nodes in Rinkeby, these nodes return the same future transactions back to node M. To avoid overloading M with the future transactions bounced back, we modify the Geth client running on M to discard figure transactions received from other nodes.

Table 10: Graph properties of the Goerli testnet

	Measured Goerli	ER (n=1025, m=18530)	CM	BA (n=1025, l=36)
Diameter	5	3.0	5.1	3.0
Periphery size	23	1025.0	31.3	866.3
Radius	3	3.0	3.0	2.0
Center size	115	1025.0	154.7	158.7
Eccentricity	3.775	3.0	3.911	2.845
Clustering coefficient	0.0354	0.0355	0.1281	0.1380
Transitivity	0.09616	0.0354	0.1052	0.1374
Degree assortativity	-0.1573	-0.0036	-0.0742	-0.0050
Clique number	134.49	416.8	1007.2	63.4
Modularity	0.048	0.132	0.125	0.084

We present the similar measurement metrics of Rinkeby with Ropsten. The node degree distribution is in Figure 9 where node degrees are distributed from 1 to 180. There are many nodes with degree smaller than 15, and between degrees 15 and 180 the nodes are evenly distributed. Graph statistics of Rinkeby, in comparison with the three random graphs, are presented in Table 9, where the measured testnet similarly shows most traits, such as with much lower modularity, which implies the testnet's higher resilience to network partitioning. Particularly, there are many more cliques found on Rinkeby than on the random graphs, which corroborates the low modularity of the testnet and hardness to partition its topology.

Compared with Ropsten, Rinkeby has a much larger center size (more nodes in the center of the graph), a higher transitivity (more likely the adjacent nodes are connected) and a lower-level modularity (harder to partition the graph into densely connected modules).

We conducted a similar measurement study on Goerli, another Ethereum testnet, and present results in node degree distribution in Figure 10 and Figure 11, and graph statistical properties in Table 10. Notably, there are nodes in the Goerli network that are globally connected and are with very high degrees (e.g., more than 700 neighbors). It has a very low clustering coefficient (0.0354) compared with those of Rinkeby (0.4375) and Ropsten (0.207). In terms of modularity, Goerli (0.048) is comparably lower than Ropsten (0.0605), and is much higher than Rinkeby (0.0106). This implies that Rinkeby is the most resilient to network partitioning (in terms of low graph modularity), and Ropsten is the least partitioning resilient.

D.1 Visualization of Rinkeby & Goerli

Using the NetworkX tool [3], we detect the communities of the Rinkeby and Goerli, respectively in Figures 8a and 8c. In Rinkeby, there are four communities detected, and the biggest one (in green) are of 33.9% of the nodes of the network. The average degree in the community is 52.3, and 5.3% of the nodes (i.e., 8 out of 151 nodes) only have a degree of 1. In Goerli, there are seven communities detected, and the biggest one (in black) are of 24.6% of the nodes of the network. The average degree in the community is 40.5, and 2% of the nodes (i.e., 5 out of 252 nodes) only have a degree of 1.

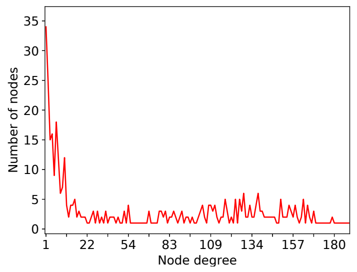


Figure 9: Degree distribution in Rinkeby

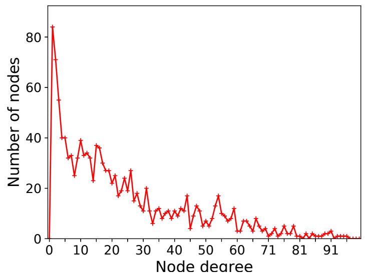


Figure 10: Node degree distribution in Goerli

Figure 11: Nodes of large degree in Goerli

Degree range	Count	Degree range	Count
100-150	12	150-200	3
200-300	4	300-500	3
697	1	711	1

We plot the geographical distributions of the Rinkeby and Goerli testnets using GeoLite2 database service[6]. Figures 8b and 8d respectively show the Rinkeby and Goerli nodes in a world map and their connections. In both testnets, most nodes are located in the United States, Europe, and East Asia. Particularly in Goerli, there are a significant portion of nodes located in Australia.

E DISCUSSION ON THE IMPACTS OF EIP1559

In EIP1559, there are three fee components: a base fee that is automatically set by the blockchain based on the recent block utilization, a priority fee set by the sender and a max fee also set by the sender. A transaction included in the blockchain always pays the base fee (burnt) and pays the priority fees to the miner. It is also ensured that the sum of the base fee and priority fee is lower than the max fee.

Under EIP1559, the `mempool` uses the max fee to make admission/eviction decisions. Noteworthy is that when a pending transaction's max fee is below the base fee (i.e., negative priority fee), the transaction becomes underpriced and is dropped. Thus, in EIP1559, as long as we ensure the max fee in measurement transactions (i.e., tx_A, tx_C, tx_O) is above the base fee, the measurement process is not affected by the presence of EIP1559.

[6](https://github.com/maxmind/GeoIP2-python