arXiv:1812.00942v1 [cs.CR] 3 Dec 2018

TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions

Sergi Delgado-Segura[1,*], Surya Bakshi[2], Cristina PÃľrez-SolÃă[3], James Litton[4], Andrew Pachulski[4], Andrew Miller[2,*], and Bobby Bhattacharjee[4]

[1] Universitat AutÃšnoma de Barcelona
[2] University of Illinois Urbana-Champaign
[3] Universitat Rovira i Virgili
[4] University of Maryland
* Corresponding Authors: s.delgado@ucl.ac.uk, soc1024@illinois.edu

Abstract. Bitcoin relies on a peer-to-peer overlay network to broadcast transactions and blocks. From the viewpoint of network measurement, we would like to observe this topology so we can characterize its performance, fairness and robustness. However, this is difficult because Bitcoin is deliberately designed to hide its topology from onlookers. Knowledge of the topology is not in itself a vulnerability, although it could conceivably help an attacker performing targeted eclipse attacks or to deanonymize transaction senders.

In this paper we present TxProbe, a novel technique for reconstructing the Bitcoin network topology. TxProbe makes use of peculiarities in how Bitcoin processes out of order, or "orphaned" transactions. We conducted experiments on Bitcoin testnet that suggest our technique reconstructs topology with precision and recall surpassing 90%. We also used TxProbe to take a snapshot of the Bitcoin testnet in just a few hours. TxProbe may be useful for future measurement campaigns of Bitcoin or other cryptocurrency networks.

1 Introduction

Bitcoin builds on top of a peer-to-peer (P2P) network to relay transactions and blocks in a decentralized manner. Broadcast is the routing scheme chosen to propagate transactions and blocks over the network, in order to spread the information as quick as possible and facilitate agreement on a common state. The topology of the Bitcoin network is unknown by design and it is built to mimic a random network. While knowing the topology of the network does not pose a threat by itself, it eases the performance of several network based attacks, such as eclipse attacks [8,12], or attacks on users anonymity [10,2].

On top of that, a study of the network topology may reveal to what extent the network is really decentralized, whether there exist supernodes, bridge nodes, potential points of failure, etc.

In this paper we present TxProbe, a technique to infer the topology of the publicly reachable Bitcoin network. Nodes of the non-reachable network, such as

nodes behind NAT or firewalls, or nodes not accepting incoming connections will not be inferred with our technique. Our work builds on prior work in exploiting Bitcoin network side channels as measurement techniques, but exploits a new side channel involving the handling of orphan transactions (transactions that arrive out of order).

To validate our technique, we have conducted an experiment in which our custom node is connected to our own ground truth nodes (running Bitcoin Core software). We then check whether we were able to get the connections of such nodes. On top of that, a scan of the entire live network has been performed resulting on a snapshot of the Bitcoin testnet. Finally, a comparative analysis of the obtained testnet graph against similar random graphs is provided to quantify whether or not the network resembles a random network.

TxProbe is an active measurement technique, and we have not conclusively ruled out that it could interfere with ordinary transactions. We have therefore limited our measurement and validate activities to the Bitcoin testnet. The technique could be used in the future to infer the topology of Bitcoin or any alt-coin sharing its network protocol, including Bitcoin Cash, Litecoin or Dogecoin.

2 Related work

Network topology inference is a topic that has been previously analysed in several other works. Biryukov et al. [2] showed how a node could be uniquely identified by a subset of its neighbourhood, and how the neighbourhood could be easily inferred by checking the address messages propagation throughout the network. Biryukov et al. [3] also showed how using Tor to guard against the aforementioned technique was not useful, and it could even ease the deanonymization process.

The use of address message propagation along with timestamp analysis was used by Miller et al. [11] to infer the topology of the Bitcoin network. The analysis highlighted how the network did not behave as a random graph but, instead, it was filled with several influential nodes representing a disproportionate amount of mining power. Their AddressProbe technique took advantage of the two-hour penalty applied to received address messages from connected peers. However, the two-hour penalty was removed from the Bitcoin Core nodes after 0.10.1 release [17,16], reducing the fingerprint left by address messages, and therefore, making AddressProbe no longer useful to infer the topology of the network.

Neudecker et al. [13] performed timing analyses of the transaction propagation to infer the topology of the Bitcoin network with a substantial precision and recall ($\sim 40\%$).

Network information from the P2P network has also been used, alongside with address clustering heuristics, to check whether such information could be useful in the deanonymization of Bitcoin users [14]. The study shows how while most of the network information cannot ease the address clustering process, a small number of users show correlations that may make them vulnerable to network based deanonymization attacks.

A recent proposal by Grundmann et al. [6] has shown how transaction accumulation of double-spending transactions can also be used to infer the neighbourhood of a targeted node with precision and recall as high as 95%.

Finally, Efe Gencer et al. [5] have presented a comparative analysis of the decentralization on two of the most popular cryptocurrencies to the date, Bitcoin and Ethereum, using application layer information obtained from the Falcon Network. Their results show how around 56% of Bitcoin nodes are run in datacenters. On top of that, their study highlights how the top four Bitcoin miners control more than the 54% of the mining power.

3 Background

In this section we give an overview of Bitcoin's transaction propagation behavior. Since our TxProbe technique relies on subtleties of this process, we go into detail on just the relevant parts.

3.1 Three-round transaction propagation

Bitcoin nodes propagate transactions by flooding, such that each node relays data about each transaction to every one of its peers. However, to minimize network traffic, nodes follow a three-step protocol, first sending just the transaction hash (32 bytes) and only sending the entire transaction (range from a few hundred bytes up to tens of kilobytes) if it is requested. This protocol is depicted in Figure 1. In more detail, the three steps are:

- **Inventory messages (inv)** are used to announce the knowledge of one or more transactions or blocks. When a node receives (or generates) a new transaction or block he announces it to his peers by creating an `inv` message containing the transaction hash. Those peers who do not know about the announced item will ask for it back using a `getdata` message. Furthermore, when a node receives an `inv` message asking for a certain item, and he requests it back using a `getdata` message, the requester will wait up to 2 minutes for the node offering the item to respond back with it. Any other request offering the same item will be queued and only responded, first in first out, if the first node fails to reply.
- **Get data messages (getdata)** are used by Bitcoin nodes to request transactions and blocks to their peers. Such messages are sent as a response to the aforementioned `inv` messages when the receiver of the latter is interested in any of the offered items.
- **Transaction messages (tx)** are used to send transactions between peers. They are usually sent as a response to a `getdata` message. In contrast to the previously introduced messages, `tx` messages always contain a single transaction.

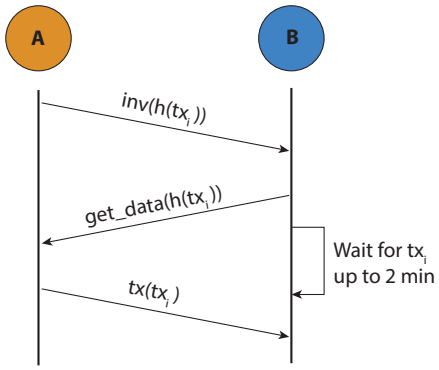


Fig. 1: Three-step protocol used to forward transactions in Bitcoin.

3.2 Mempool and Unspent Transaction Outputs (UTXOs)

A Bitcoin node validates each transaction it receives before relaying to its peers as described above. A valid transaction must have correct signatures, and must only spend existing and currently-unspent coins. Otherwise the transaction is discarded and not propagated further.

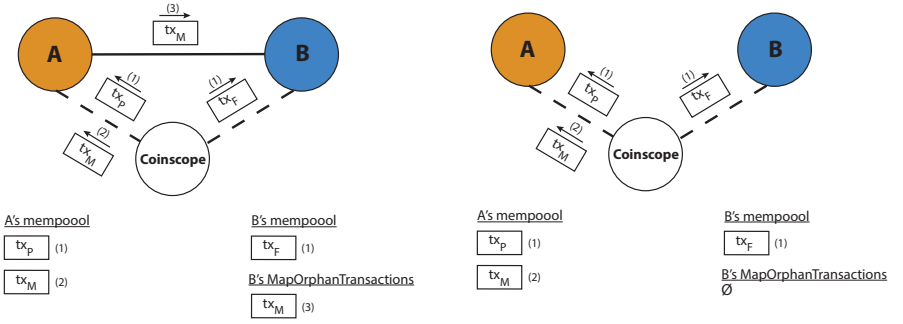
To aid in validation, each Bitcoin full node maintains a view of the current set of available coins (the `utxo set`). It also maintains a collection of pending transactions, called `mempool`, all of which have been validated against the `utxo set` and contain no double-spends amongst themselves.

Much of the complexity in the Bitcoin software, and the behavior we exploit in TxProbe, involves handling special cases during validation. Hence when a transaction is received, it is validated against the current `utxo set`. Since mempool is also kept free of double-spends, when a Bitcoin node receives a second transaction that spends the same coin as a transaction held in `mempool`, the second transaction is simply discarded.[5]

3.3 Handling orphan transactions

Sometimes a node receives transactions out of order. A transaction is considered an "orphan" if it is received prior to its direct ancestors, i.e. it spends a coin that is not yet part of the blockchain or in `mempool`. Since orphan transactions cannot be validated until the parent arrives, they are not immediately relayed to peers. Instead, orphan transactions are stored in a buffer, `MapOrphanTransactions` so that when the parent arrives it can be validated without re-requesting it from the network.

[5] There is a special case, called replace-by-fee (RBF) [7], in which a double-spending transaction replaces a previous transaction as long as the previous transaction is flagged to allow this and if the new transaction pays a larger fee. This does not affect the TxProbe technique.



(a) Basic positive edge inferring technique between two nodes.

(b) Basic negative edge inferring technique between two nodes.

Fig. 2: Basic edge inferring technique.

To point out a detail relevant to our TxProbe technique: If a node receives a notice about a transaction from a peer (an `inv` message), but that transaction has already been stored as an orphan, then that transaction will be omitted from subsequent `get_data` messages. Looking ahead, this behavior enables our measurement node to probe whether an orphan transaction has already been received or not. We discuss other details about `MapOrphanTransactions`, such as eviction policies, later on when discussing optimizations to TxProbe.

4 Inferring the Bitcoin network topology

In this section we explain our technique for inferring the topology of Bitcoin's reachable peer-to-peer network, making use of the subtleties of transaction propagation in Bitcoin as described earlier, and in particular conflicting transactions and orphan transactions. We start by introducing a basic edge inference technique that tests for an edge between a single pair of peers. Later we discuss how to scale the technique up to take network-wide snapshots efficiently.

4.1 Basic edge inferring technique

To explain the main idea behind our technique, we start by describing a scenario in which our Coinscope measurement node is connected to two nodes A and B, and we want to check if there exists an edge between them. Note that such a scenario is not realistic, but we will discuss real cases later on. First we create a pair of double spending transactions referred to as the parent (tx_P) and the flood (tx_F). We send tx_P to A and tx_F to B and assume that both transactions arrive to their destination at the same time, so A will reject tx_F if B sends it to him and vice versa. Now we create a third transaction, the marker (tx_M), that

spends from tx_P and we send it to A. On receiving tx_M, A will forward it to all his peers. If the edge between the two nodes exists, as depicted in Figure 2a, B will receive the transaction. It is worth noting that B does not know about tx_P, so tx_M will be flagged as orphan and not relayed any further. On the contrary, if the edge between nodes A and B does not exist, as depicted in Figure 2b, tx_M will never be sent to B.

At this point we can check if the connection between the two nodes exists. To do so, we ask B about tx_M by sending him an `inv` message containing tx_M's hash. If the connection between the two nodes exists, B will have tx_M stored in his `MapOrphanTransactions` pool, so he will not request it back. On the other hand, if the edge does not exist, B will respond with a `getdata` message containing tx_M's hash.

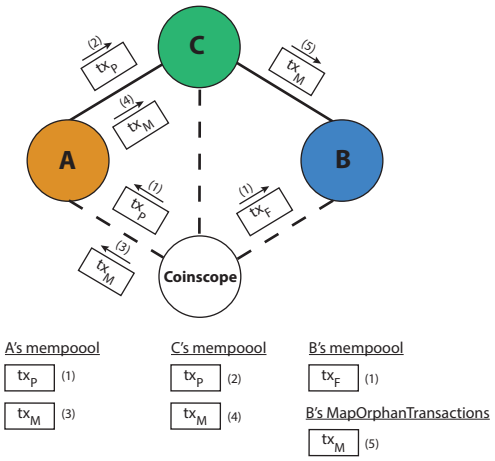


Fig. 3: Incorrect edge inferring with three nodes.

While this basic technique works in the most simple scenario, namely with two nodes potentially connected only between them, it can drastically fail if just one additional node is added to the picture. Let's see what happens if we connect one additional node C, as depicted in Figure 3, and repeat the procedure. Since A is connected to C, tx_P and tx_M will be forwarded to him, C will treat both transactions as regular ones, and forward them to B, who will reject tx_P as double spending of tx_F, but accept tx_M as orphan. Ultimately, we will ask B about tx_M, and infer a non-existing edge between him and A.

Such a basic example highlights the first main issue of the basic approach: **isolation**. We need to ensure that tx_P remains only in the node we have sent it to. Otherwise, we may end up inferring non-existent edges. Moreover, this basic technique also builds on top of another fragile property: **synchronicity**. If node A receives tx_P before B receives tx_F, A can forward tx_P to B, and the latter will end up rejecting tx_F upon reception, making the technique to

fail. Finally, the basic technique lacks **scalability**. Assuming we can sort out the two previous issues, for every three transactions created we will be able to infer at most the whole neighbourhood of a single node. Inferring the topology of the whole reachable network will require creating almost three transactions per node, namely $3 \cdot (n-1)$ where n is the number of reachable nodes in the network. In order to solve the three aforementioned problems, we have created a technique called TxProbe.

4.2 TxProbe

TxProbe is a topology inference technique that uses double spending and orphan transactions to check the existence of edges between a pair of nodes. TxProbe can be used to infer the topology of several cryptocurrency P2P networks, as long as they share the network protocol and orphan transactions handling with Bitcoin (i.e. Bitcoin Cash, Litecoin, ZCash, etc). In contrast to recently proposed techniques, such as [6], TxProbe is intended to perform full network topology inference, instead of targeted neighbourhood discovery, even though the latter can be also achieved. TxProbe builds on the aforementioned basic edge inferring technique solving its three main downsides:

Regarding **isolation** and **synchronicity**, TxProbe uses Coinscope, the observation and testing framework introduced by Miller et al. in [11], to maintain connections with all reachable nodes and performs the `invblock` technique (proposed also in [11]) to ensure that a target transaction will remain in a target node. With regard to **scalability**, TxProbe takes advantage of the `MapOrphanTransactions` pool management to perform multiple nodes neighbourhood discovery at the same time.

We now describe the main components of the TxProbe technique. In a single trial, we break the network nodes into two groups, the `source set` and the `sink set`, where we aim to infer all the connections between `source set` nodes and `sink set` nodes. The `source set` will be usually smaller than the `sink set`, and should at least be less than the size of the `mapOrphanTransactions` pool.

Setup

Create conflicting transactions: First, we need to create the set of conflicting transactions, namely the parents, markers, and flood transactions. This time we are not targeting a single node to infer his peers (as we did with A in the basic inferring technique), but all the nodes in the `source set`. Therefore instead of creating a single parent and the flood transaction spending from the same `utxo`, we will create $n+1$ distinct double spending transactions, n being the number of nodes in the `source set`: n of those transactions will be tagged as parents, while the remaining one will be the flood transaction. Finally we create a marker transaction from each of the parents, resulting in n parents, n markers, and the flood transaction. Figure 4 depicts a high level representation of the created transactions (spending from UTXO_1).

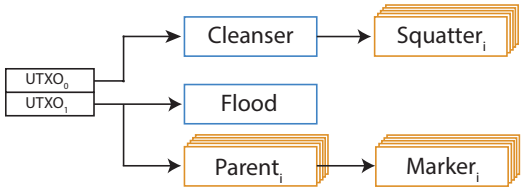


Fig. 4: High level representation of the transactions created in TxProbe

Invblock the network: Once all the parents, markers and the flood transaction have been created, it is time to ensure that the **isolation** property will hold during the experiment. It is worth noting that for the isolation property to hold there are two things we need to ensure: First, that the flood transaction (tx_F) remains within the `sink set`. Secondly, that each parent (tx_{Pi}) remains only in the `source set` node (N_i) it will be sent to. To ensure so, we will perform an `invblock` of tx_F and every tx_P. `invblock` consists of sending `inv` messages to all the nodes in the network with the transaction hashes we want to block the propagation. Recall that a node requesting a transaction with a `getdata` message in response to a `inv` message will wait up to two minutes until requesting such transaction to any other peer offering it. By sending multiple `inv` messages containing the same transaction hashes to a node it can be blocked to request those transactions to any other node for an arbitrary number of minutes, which gives enough time to send all the transactions without having to worry about the isolation property being broken. It is also worth noting that the network will not be blocked with the markers hashes, since their propagation from the `source set` to the `sink set` is what will allow us to infer edges between nodes.

Main protocol

Once we have set the proper conditions for the experiment to be run, we can start sending the transactions we created earlier.

Send transactions: First, the flood transaction is sent to all the nodes in the `sink set`. After waiting a few seconds for the flood transaction to propagate, we can send the proper transactions to the `source set`. We start by sending a different parent to each node in the set, wait a couple of seconds, and then send the corresponding marker to each node. Since we have `invblocked` the whole network with the flood and parents, at this point we are sure that, as long as the nodes behave properly, the flood transaction is only present in the `sink set` and each parent is only present in its respective node from the `source set`.

Requesting markers back: After waiting a few seconds for the propagation of the markers, we will request all of them back from every node in the `sink set`. Despite being orphans, markers are still considered known transactions by those `sink set` nodes. In that sense, as we have already seen in the basic inferring

technique, when an orphan transaction is requested as part of a `getdata` message the node holding it will not include it in their response. By sending an `inv` message containing all the markers to the `sink set` nodes we will receive back a request of only the subset of markers they have not heard of. [6] By mapping the markers that have not been sent back to the `source set` node we originally sent them to, we can infer edges between the `source set` and the respondent `sink set` nodes.

Permuting the sets: With all the aforementioned steps, we are able to infer the edges between a certain configuration of the network, that is, a specific set of nodes forming the `source set` and `sink set`. However, the technique cannot infer edges between nodes in the same set. In order to infer the whole topology, we need to run several rounds permuting the sets. Therefore, both the setup and the main protocol will be run until every pair of nodes have been in a different set at least once.

4.3 Making room for marker transactions.

The `MapOrphanTransactions` pool is not allowed to grow unbounded. In fact, it has a small default limit of only 100 transactions at a time. When this limit is exceeded, orphan transactions are evicted. The eviction mechanism works as follows: First, it generates a random hash *randomhash*. Next, it selects the transaction in the pool with the closest hash higher than *randomhash* and evicts it from the pool. The eviction mechanism repeats until the pool size is within limits.

Eviction poses a problem for scaling up the TxProbe technique. Marker transactions must not be evicted until they are read back at the end of a measurement trial. However, the eviction policy has a design flaw, which enables us to make preferential transactions that are hard to evict. By crafting transactions for which their hashes lay between a small fixed range (e.g: by re-signing the transaction), and since the *randomhash* hash used in the eviction mechanism picks values over an uniform distribution, we can bound the odds of our transaction being evicted depending on how small the range is set.

Cleansing the orphan pool: When we were performing the basic inferring technique there was no need to worry about transactions being evicted from the `MapOrphanTransactions` pool since we were only creating a single marker. However, now up to n markers would need to be stored by a single node. In order to ensure that there is enough space to store all the markers, we will empty the `MapOrphanTransactions` pool of all nodes in the `sink set`. We start by creating a transaction we call the cleanser, and spending from it we create 100 distinct double spending transactions we call the squatters.

Next, we send all the squatters to every single node in the `sink set` aiming to full the orphan pool. Finally, we send the cleanser to every `sink set` node.

[6] Notice that some times the subset will be the actual set.

Upon reception of the cleanser, all transactions in the orphan transactions pool will no longer be orphans. One of the squatters will be flagged as valid, whereas the rest will be discarded as double-spending transactions. Regardless of which squatter is accepted by each node, the `MapOrphanTransactions` pool of each `sink set node` will be emptied. Figure 4 depicts a high level representation of the orphans and cleanser transaction creation (spending from UTXO_0).

5 Costs of topology inferrence

In this section we discuss the costs of running TxProbe both in terms of time and transaction fees.

5.1 Time costs

How long it takes to infer the topology of a network using TxProbe directly depends on the number of reachable nodes r_n in the network. As we have already seen, the size of our `source set`s is bound by the `MapOrphanTransactions` pool size, which is 100 by default. Our set partitioning algorithm works as a grid, in order to separate nodes in two sets we create a grid of width $w = min(\lceil\sqrt{r_n}\rceil, 100)$ and length $h = \lceil\frac{r_n}{w}\rceil$, and we traverse the grid by rows and columns, being the selected row/column in iteration i our `source set` for the i-th round of the experiment, and the rest of nodes our `sink set`.[7] The total number of different `source sets`, and therefore, the total number of rounds required to run an experiment will then be:

$$t_r = \begin{cases} h + w - 2, \text{for } h \leq w \\ h - 1 + \lceil\frac{h}{w}\rceil \cdot w, \text{for } h > w \end{cases}$$

Each round of the TxProbe can be run in about 2.5 minutes, resulting in $2.5 \cdot t_r$ minutes to run TxProbe over a network of r_n nodes. Inferring the topology of a network like Bitcoin testnet ($\sim$ 1000 nodes) requires, therefore, about 2.6 hours, whereas inferring the topology of Bitcoin mainnet ($\sim$ 10000 nodes) requires about 8.25 hours. The partitioning algorithm can be found depicted in Figure 5.

5.2 Transaction fee costs

The costs of running TxProbe directly depend on the number of rounds of the experiment t_r and the fee rate to be paid in order to get our transactions relayed by the network. For every round we will perform an orphan cleansing, resulting in two standard 1-1 P2PKH transactions (only the cleanser and one squatter will be accepted, the rest will be eventually flagged as double-spends). Moreover, at

[7] Notice that when traversing columns the number of elements in the set can be higher than w, in which case the algorithm will create $\lceil h/w \rceil$ sets per column.

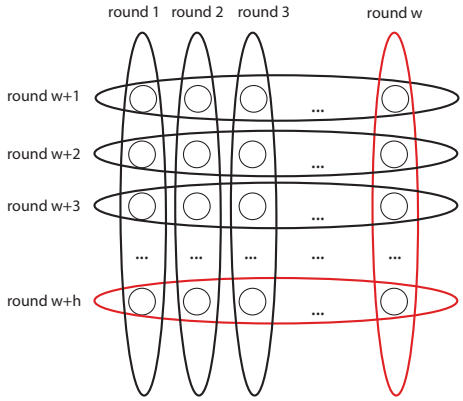


Fig. 5: Set-partitioning algorithm used in TxProbe. Sets corresponding to the last row and column (marked in red) can be skipped since they will be already counted by the rest of sets.

every round either the flood will be accepted (1-1 P2PKH transaction) or a parent-marker pair will be accepted (two 1-1 P2PKH transactions).

The size of a 1-1 P2PKH transaction using compressed public keys and assuming a signature of maximum length (73-byte signature) is 193 bytes long. Putting all together, the cost of running TxProbe in a network of r_n nodes ranges between $3 \cdot 193 \cdot fee\_rate \cdot t_r$ and $4 \cdot 193 \cdot fee\_rate \cdot t_r$.

At a fee rate of 5 $sat/byte$ [8], running the experiment in a network like Bitcoin mainnet will cost between 573210 and 764280 $satoshi$.

Impact of TxProbe measurement. We say a few words about the feasibility of using TxProbe to do ethical measurement. The TxProbe measurement involves sending many kinds of abnormal transactions, and thus it can only be used ethically if we ensure it does not harm or burden the network we are measuring.

To start with, although TxProbe transactions are unusual in that they are multi-way conflicting double-spends, they are not relayed and thus do not increase network or storage utilization compared to ordinary transactions.

The TxProbe experiment can have a destructive effect, however, on the `MapOrphanTransactions` data structure. As we discussed earlier, if the orphan transactions pool is full, then adding new orphan transactions (such as the marker transactions in TxProbe) can evict others.

We could not rule out the potential that our measurement would add to this congestion (i.e., over an 8+ hours for a scan of the entire network) and could adversely affect real transactions.

[8] Fee to get transactions confirmed between 1-2 blocks on 27th August 2018 according to `https://bitcoinfees.earn.com/`.

6 Experiments and results

We conducted experiments using the Bitcoin testnet in order to evaluate our topology inference technique. We first conducted ground truth experiments to quantify its precision and recall, and then took a snapshot measurement of testnet to demonstrate its usefulness for network-wide scans.

6.1 Validation

In order to validate our results we run 5 local Bitcoin nodes as ground truth. The ground truth nodes are included as part of the `source set` in each round of the experiment. This means that, if our results are correct, at the end of the experiment we would have inferred every edge between the ground truth nodes and the `sink set` nodes.

For every run of the experiment there are always nodes that do not behave according to the default client, for example by ignoring `invblock` and therefore, sending transactions without receiving `getdata` messages. In order to detect such nodes, an `invblock` test is performed before every experiment using two Coinscope instances: the first instance crafts a random 32-byte hash and offers it to the whole network using `inv` messages. The second instance offers the exact same hash within the next two minutes and collects all the `getdata` responses. All those nodes who responded the second instance are flagged as unblockable nodes and taken out of the experiment.

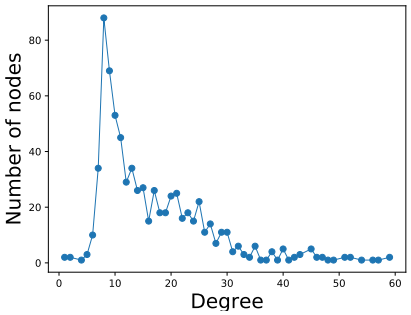
Transitory edges (i.e. edges that have been there for a short amount of time) are also removed from the inferred results, as well as nodes who know about transactions they are not supposed to (nodes who missed a parent/marker when it has been sent to them, nodes holding the flood transaction when they were supposed to hold a parent transactions, etc). Finally, disconnecting nodes (nodes that have disconnected from Coinscope while the experiment was running) are also removed, as well as all inferred edges referring them.

Our validation reported a precision of 100% and recall between 93.86% and 95.45% with a 95% confidence over 40 runs of the experiment.

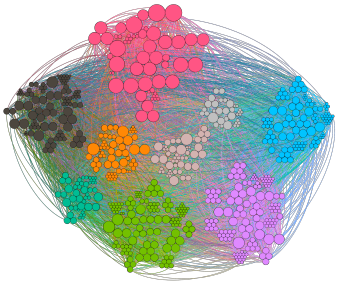
6.2 Analysis of the inferred network

This section includes a description of a testnet network snapshot taken on 21st February, 2018, as obtained using our technique, which reported a precision of 100% with a recall of 97.40%.

The observed network has 733 nodes and 6090 edges, with an average degree of 16.6. The degree distribution of the network is far from uniform (Figure 6a), with most of the nodes having between 7 and 14 neighbors. The most common degree observed in the network is 8 (shown by 12% of the nodes), a value that matches the default maximum number of outgoing nodes of the Bitcoin Core



(a) Degree distribution of nodes in the testnet snapshot.



(b) Communities detected in the testnet snapshot.

client.[9] The maximum degree is 59, less than half of the maximum number of default peer connections of Bitcoin Core.[10]

Table 1 provides a summary of basic properties of the network regarding clustering, distances, assortativity, and community structure, comparing the observed values with those obtained over random graphs with similar characteristics. That is, for each property, we create 100 random graphs that *resemble* the obtained testnet graph, compute the property over the random graphs, and provide both the average of the results over the random graphs and the percentage of times the value over the random graph is higher than the observed in the testnet snapshot. We have considered three different models for generating random graphs: Erdős-Rényi [4] (ER), configuration model [15] (CM), and Barabási-Albert [1] (BA). The Erdős-Rényi model generates graphs where each pair of nodes may have an edge with the same probability, and independently of the other edges of the network. ER generates graphs with a binomial degree distribution, and it is commonly used as baseline to analyse networks. However, the observed testnet graph does not seem to have an ER-like degree distribution (recall Figure 6a). Therefore, we also create random graphs using the configuration model, that allows creating networks with a chosen degree distribution. Finally, since many real world computer networks have been reported to be preferential attachment networks, we also include Barabási-Albert model. BA generates scale-free networks, that have power-law degree distributions. The random graphs we create *resemble* the observed network: ER graphs are created with the same number of nodes and edges, CM graphs have the same degree distribution (and, therefore, they implicitly also have the same number of nodes and edges), BA graphs have the same number of nodes and a similar number of edges (by adjusting the number of new edges created at each step of the graph

[9] `https://github.com/bitcoin/bitcoin/blob/v0.16.2/src/net.h#L59`

[10] The maximum number of default connections is set to 125: `https://github.com/bitcoin/bitcoin/blob/v0.16.2/src/net.h#L73`

generation algorithm, the number of edges is adjusted to be as close as possible to the observed one).

Metric	Testnet	ER		CM		BA	
Diameter	5	4	(0%)	4.93	(1%)	4	(0%)
Periphery size	6	612.9	(100%)	21.2	(80%)	379.6	(100%)
Radius	3	3	(0%)	3	(0%)	3	(0%)
Center size	45	120.7	(100%)	57.9	(70%)	362.9	(100%)
Eccentricity	3.946	3.827	(0%)	3.979	(70%)	3.528	(0%)
Clustering coefficient	0.052	0.023	(0%)	0.036	(0%)	0.066	(100%)
Transitivity	0.128	0.023	(0%)	0.036	(0%)	0.057	(0%)
Degree assortativity	0.291	−0.001	(0%)	−0.008	(0%)	−0.043	(0%)
Country assortativity	0.007	−0.001	(0%)	−0.001	(0%)	−0.002	(0%)
Clique number	24	3.73	(0%)	4.05	(0%)	6.58	(0%)
Modularity	0.270	0.220	(0%)	0.216	(0%)	0.214	(0%)

Table 1: Network properties. For random graphs, the 100-run average is provided, with the percentage of times the property over the random graph is higher than the observed in the testnet snapshot in parenthesis.

Distances are the properties analyzed in the testnet graph that most approximate those obtained in random graphs. For instance, the radius of the testnet snapshot is 3, exactly the same value observed in all the generated random graphs. The diameter (the maximum distance between any pair of nodes) of the testnet graph is 5, which is higher than most of the random graphs, but very close to their diameters. Moreover, by removing just 3 of the lowest degree nodes of the testnet graph, its diameter becomes 4 (removing the degrees from the sequence in the CM model has the same effect). On the contrary, the number of nodes in the center and in the periphery (i.e. nodes with eccentricity equal to the radius and the diameter, respectively) differs largely from random graphs.

With respect to clustering, the testnet graph exhibits a higher average clustering coefficient than ER and CM graphs, but less than BA graphs. However, observed transitivity is higher than any of the random graphs. Clustering coefficient analyses how well connected the neighborhood of a node is (taking into account the neighborhood regardless of its size), whereas transitivity is focused on studying 3-node substructures.

The testnet snapshot shows higher assortativity than the expected for random graphs, that is, nodes in the testnet tend to connect to other nodes that are similar to themselves more often than what ER, CM and BA random graphs exhibit. Specifically, nodes tend to connect to other nodes with the same degree and, to a less extent but also in a significant manner, to nodes in the same country.

Regarding community structure, we have computed the modularity over the best partition found using the Louvain method. The modularity of the testnet graph is higher than any of the random graphs, regardless of the chosen model.

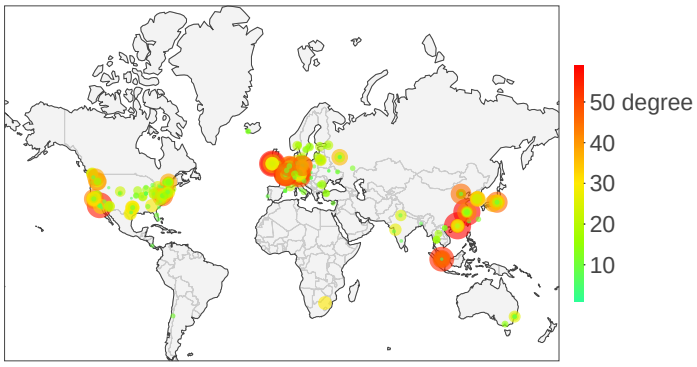


Fig. 6: Geographical location of nodes.

That is, the network shows more community structure than what should be expected for a random graph. Figure 6b depicts a visualization of the communities found in the testnet snapshot, with the color of the node denoting the community it belongs to. There are nine communities, with the biggest two (purple and green in the image) having 37% of the nodes of the network. Notably, there is one community (colored in pink) that contains only 7.5% of the nodes but includes the 25 highest degree nodes of the network. This is consistent with the high degree assortativity reported in the network. Remarkably, the testnet graph contains a clique (a fully connected graph) of 24 nodes. This clique is found inside the high-degree community (depicted in pink in the visualization). In contrast, the largest clique formed by nodes of any other community has a size of just 6 nodes.

We have also used an IP Geolocation API [11] to obtain the geographical location of the nodes in the testnet snapshot. Figure 6 shows a map with the node's locations, where both the size and color of the nodes are used to denote nodes' degree. Most nodes are located in the United States, Central Europe, and East Asia.

7 Conclusions

We set out to design an effective measurement technique that can reconstruct the Bitcoin network topology. We validated TxProbe to show that it is accurate and can indeed be scaled up to snapshot the entire network with reasonably low fees. However, we decided not to carry out a measurement of the main network because we could not rule out its potential to delay the propagation of real users'

[11] `http://ip-api.com/`

transactions. We consider it an open question whether this technique (or analysis thereof) can be improved so it can be used less invasively.

We did, however, take network measurement snapshots of the Bitcoin test network topology (over 700 nodes). Our analysis of the Bitcoin testnet reveals significant non-random structure, including several communities, as well as a clique of high-degree nodes. Although our findings over testnet cannot be applied to the mainnet, it demonstrates that the technique is viable not only in Bitcoin, but in any other cryptocurrency sharing the network protocol and orphan transaction handling with it.

Like other measurement techniques, TxProbe makes use of implementation-specific behaviors in the Bitcoin software. While cryptocurrencies have not made topology-hiding an explicit design requirement, in the past software changes that improve user anonymity have also had the effect of closing off measurement avenues. Viewed in this light, TxProbe is the next step in a tacit arms race between measurement efforts and privacy enhancing design. We make the following suggestions to the cryptocurrency community to avoid this cycle: First, determine whether network topology or other metrics should be an explicit design goal, in which case effort can be focused on achieving it robustly. Second, follow the Tor project [9] for example, in deploying measurement-supporting mechanisms into the software itself, that balances the positive goals of network measurement (such as quantifying decentralization, detecting weaknesses or attacks, etc.) with the privacy goals of users.

Acknowledgments This work was sponsored in part by a gift from the DTR foundation, and a grant from by IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR) - a research collaboration as part of the IBM AI Horizons Network.

References

1. Albert, R., Barabási, A.: Statistical mechanics of complex networks. CoRR cond-mat/0106096 (2001)
2. Biryukov, A., Khovratovich, D., Pustogarov, I.: Deanonymisation of clients in bitcoin p2p network. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 15–29. CCS '14, ACM, New York, NY, USA (2014)
3. Biryukov, A., Pustogarov, I.: Bitcoin over tor isn't a good idea. In: Proceedings of the 2015 IEEE Symposium on Security and Privacy. pp. 122–134. SP '15, IEEE Computer Society, Washington, DC, USA (2015), https://doi.org/10.1109/SP.2015.15
4. Erdös, P., Rényi, A.: On the evolution of random graphs. In: Math. Inst. Hungar. Acad. Sci. pp. 17–61 (1960)
5. Gencer, A.E., Basu, S., Eyal, I., van Renesse, R., Sirer, E.G.: Decentralization in bitcoin and ethereum networks (2018)
6. Grundmann, M., Neudecker, T., Hartenstein, H.: Exploiting transaction accumulation and double spends for topology inference in bitcoin. In: Financial Cryptography and Data Security. Springer International Publishing (2018)

7. Harding, D.A., Todd, P.: Opt-in Full Replace-by-Fee Signaling. https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki (2015)
8. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 129–144. USENIX Association, Washington, D.C. (2015)
9. Jansen, R., Johnson, A.: Safely measuring tor. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1553–1567. ACM (2016)
10. Koshy, P., Koshy, D., McDaniel, P.: An analysis of anonymity in bitcoin using p2p network traffic. In: Christin, N., Safavi-Naini, R. (eds.) Financial Cryptography and Data Security. pp. 469–485. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
11. Miller, A., Litton, J., Pachulski, A., Gupta, N., Levin, D., Spring, N., Bhattacharjee, B.: Discovering bitcoin's public topology and influential nodes (2015)
12. Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In: 2016 IEEE European Symposium on Security and Privacy (EuroS P). pp. 305–320 (March 2016)
13. Neudecker, T., Andelfinger, P., Hartenstein, H.: 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). pp. 358–367 (July 2016)
14. Neudecker, T., Hartenstein, H.: Could network information facilitate address clustering in bitcoin? In: Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P.Y., Teague, V., Bracciali, A., Sala, M., Pintore, F., Jakobsson, M. (eds.) Financial Cryptography and Data Security. pp. 155–169. Springer International Publishing, Cham (2017)
15. Newman, M.E.: The structure and function of complex networks. SIAM review 45(2), 167–256 (2003)
16. Nick, J.: Guessing bitcoin's p2p connections. https://jonasnick.github.io/blog/2015/03/06/guessing-bitcoins-p2p-connections/ (2015)
17. The Bitcoin Core developers: Bitcoin core 0.10.1 release notes. https://github.com/bitcoin/bitcoin/blob/v0.10.1/doc/release-notes.md (april 2015)