

Discovering Bitcoin’s Public Topology and Influential Nodes

Andrew Miller* James Litton* Andrew Pachulski* Neal Gupta* Dave Levin*
Neil Spring* Bobby Bhattacharjee *

Abstract

The Bitcoin network relies on peer-to-peer broadcast to distribute pending transactions and confirmed blocks. The topology over which this broadcast is distributed affects which nodes have advantages and whether some attacks are feasible. As such, it is particularly important to understand not just which nodes participate in the Bitcoin network, but how they are connected.

In this paper, we introduce AddressProbe, a technique that discovers peer-to-peer links in Bitcoin, and apply this to the live topology. To support AddressProbe and other tools, we develop CoinScope, an infrastructure to manage short, but large-scale experiments in Bitcoin. We analyze the measured topology to discover both high-degree nodes and a well connected giant component. Yet, efficient propagation over the Bitcoin backbone does not necessarily result in a transaction being accepted into the block chain. We introduce a “decloaking” method to find influential nodes in the topology that are well connected to a mining pool. Our results find that in contrast to Bitcoin’s idealized vision of spreading mining responsibility to each node, mining pools are prevalent and hidden: roughly 2% of the (influential) nodes represent three-quarters of the mining power.

1 Introduction

Bitcoin communication is built upon peer-to-peer broadcast, which carries transactions, blocks, and other global protocol state. Through broadcast, Bitcoin achieves eventual consistency: information about all transactions and blocks is relayed to all peers, and despite inconsistent ordering and partial incompleteness, all honest peers eventually “agree” on a globally consistent state of committed transactions and blocks.

The underlying peer topology over which protocol messages are exchanged is of critical importance: broadcast over this topology is the only mechanism by which peers learn and inform each other of transactions and blocks. Understanding how information propagates throughout the Bitcoin ecosystem is therefore integral to being able to reason about attacks and manipulation in the Bitcoin network. For instance, initial studies have revealed that “advantages” within the broadcast network

can be parlayed directly into gains in coins mined [10]. Furthermore, a broadcast advantage enables an attacker to pull off forms of double-spending attacks against fast-payments processors [14].

In this paper, we describe mechanisms to discover two features of Bitcoin’s topological structure: first, we map the public topology consisting of the edges that comprise the peer-to-peer network. Next, within the discovered topology, we find “influential” nodes that appear to directly interface with a hidden topology that consists of mining pools that are otherwise not connected to the public Bitcoin network. (We are unable to map the hidden intra-pool topology, since these are private networks operating using potentially proprietary protocols.)

To map the public topology, we introduce a probing mechanism called *AddressProbe* that efficiently discovers peer links. Using AddressProbe, we can efficiently map the *connectable* Bitcoin network, i.e., AddressProbe will find links between x and y iff x and y are connected, and permit incoming connections. AddressProbe can also find links made by non-connectable nodes (e.g., nodes that are behind a NAT), as long as such a node initiates connections to a probe host. Our techniques also identify a set of artificially high-degree nodes that attempt to connect to many peers, potentially to reduce latency in learning about and propagating new blocks and transactions. We map these nodes to various services, including mining pools and wallet services.

Although AddressProbe is able to map the entire connectable Bitcoin network within minutes, discovering the public topology alone is insufficient to account for the Bitcoin ecosystem’s mining power. The original Bitcoin paper [22] proposed a democratic world-view (“one-cpu-one-vote”) where peers participating in the broadcast would also mine for new coins. As Bitcoin has become popular and financially relevant, coin mining has become the domain of specialized *miners* operating special-purpose hardware around the world organized into “mining pools.” Miners often do not connect directly to the Bitcoin broadcast network. Instead, pool operators deploy gateway hosts that transfer transactions and blocks between the Bitcoin network and pool members. Pools may choose to remain clandestine about their gateway because they may be targeted by attackers, such as other competing pools [13, 34], and it is not clear

*University of Maryland, College Park. {amiller, litton, ajp, ngupta, dml, nspring, bobby}@cs.umd.edu

how or where these mining pools connect to the Bitcoin broadcast network.

Furthermore, because of the well-known attacks on Bitcoin if a single principal gains more than 50% of the mining power, large pools may prefer to disguise some of their power. In fact, while a malicious principal with a majority of the hash power can subvert Bitcoin’s most basic security goals (i.e., revise the transaction history arbitrarily), prior work has shown that even a third is sufficient to unbalance the incentive structure and reward scheme [10]. Thus, large enough pools may choose to mine anonymously, hiding their true mining power by paying out to different keys, without disclosing their gateway(s).

A primary contribution of our work is uncovering *influential* nodes within the public topology that provide connectivity to mining pools. In particular, we find that *efficient propagation over the Bitcoin network does not necessarily result in a transaction being accepted into the block chain or a block being extended*. Instead, there are a few (approx. 100) hitherto unidentified nodes that act as “front-ends” to mining pools, and it is far more important that these nodes receive a transaction or block more efficiently than others. We introduce “decloaking” techniques to identify these influential nodes, and analyze how these nodes map to different mining pools. Our analysis cannot reveal if an influential node is merely well connected to a pool or is a gateway run by the pool operator; instead, our techniques allow us to map specific nodes to blocks that are claimed by a different pools.

In summary, our contributions are as follows:

- We introduce the AddressProbe technique, which can find the connectable Bitcoin topology, within minutes, and can discover other peers over time. Using AddressProbe, we show that the deployed Bitcoin topology is not a random graph.
- We introduce decloaking techniques to find influential nodes that skew broadcast fairness. First, a randomized technique efficiently finds candidate influential nodes, and a related technique validates these candidates.
- All of our measurements use a new software infrastructure, CoinScope. This paper presents results of running AddressProbe and decloaking using CoinScope over the live Bitcoin network.

The rest of the paper is structured as follows: In Section 2, we present a background on the pertinent details of Bitcoin, along with related work on mapping the Bitcoin network. Section 3 introduces the design and implementation of our AddressProbe technique, which we apply in Section 4 to analyze the live Bitcoin network. We present our techniques for finding influential Bit-

coin nodes in Section 5, and analyze the most influential nodes in Section 6. Finally, we conclude in Section 7.

2 Background and Related Work

One of Bitcoin’s salient features, especially in contrast to digital currencies that have preceded it (e.g., Digi-Cash [32]), is that it runs on a decentralized peer-to-peer network and has no formally designated administrators. Instead, participation in Bitcoin is open, and the network largely self-organizes. Most of the novel aspects of Bitcoin’s design, and indeed its peculiarities, arise from the challenge of converging on a global and coherent state.

In this section we provide a brief introduction to the basic operation of the Bitcoin network, with a focus on how it achieves globally consistent state (for more general information on Bitcoin, please see [5]). We also discuss the role that Bitcoin’s peer-to-peer broadcast plays in ensuring consistency and fairness. Finally, we close this section by reviewing related work on measuring and analyzing Bitcoin’s broadcast topology.

2.1 The Bitcoin protocol

The Bitcoin protocol is built around the creation and dissemination of a public global *log* of the state of all bitcoins in the system. Each entry in this log is a *transaction*, which represents the transfer of virtual currency from one “account” to another. Transactions consist of inputs and outputs: a transaction “spends” a set of transaction inputs and “creates” a set of transaction outputs. In general, each transaction input contains a reference to (i.e., the hash of) an output of a previous transaction—in this sense, the log is append-only, and extending it requires access to the latest log entry. Each transaction output contains a value representing the quantity of bitcoin currency, as well as information describing which user “owns” those coins. Transactions are structured as a directed graph, which facilitates maintaining invariants about the transaction log, (for example, that users cannot spend more money than they have).

The protocol guarantees each transaction output can be spent by at most one subsequent transaction. *Conflicting transactions* are a pair of distinct transactions that spend a common transaction input. A *valid* transaction is one that has valid signatures for each of the transaction inputs; additionally, the sum of the values of the outputs must not exceed the sum of inputs (the difference is treated as a transaction processing fee, as we’ll describe shortly). The central tenet behind Bitcoin is that if everyone has access to the log of transactions, then anyone can verify who has the right to spend what bitcoins.

2.2 The role of broadcast in Bitcoin

The Bitcoin protocol’s primary goal is to converge on an *eventually consistent* sequential log of transactions. For the system to succeed, users must be able to submit

transactions for timely inclusion in the directed transaction graph, and the network should converge *quickly* to a single valid (prefix of a) graph. If an attacker could prevent transactions from entering this graph or delay agreement, it would deny service to users. Alternatively, if an attacker could revert an agreed-upon graph, he could effectively steal funds by *double-spending*.

Bitcoin achieves this consistency through the use of a peer-to-peer broadcast topology. Unfortunately, little is standardized about how exactly this broadcast operates beyond the format of the various messages; we describe here what the reference implementation (“Satoshi client”) does.¹ Every Bitcoin peer maintains a database, called the *addrMan*, of peers it has heard about. A peer first learns about a set of peers by contacting bootstrap DNS nodes; peers subsequently exchange *addrMan* information with one another to learn about new peers. Every Bitcoin peer initiates a connection to up to eight others, and maintains a maximum of 125 total connections (incoming and outgoing), rejecting any connection request that would push it beyond this capacity. Its total connections constitute that peer’s *neighbors*.

Bitcoin’s peer-to-peer broadcast is based on flooding neighbors’ links in a gossip-like manner. At a high level, when a peer learns of a new transaction or block, it informs its neighbors by sending a *INV* message containing the item’s hash to each of its neighbors. In response, if a given neighbor does not yet have that item, it requests it with a *GETDATA* message. The original peer responds with a *TX* or *BLOCK* message containing the relevant data. Finally, because this new neighbor has learned about a new transaction or block, the entire process continues recursively until all reachable peers receive it.

This ad hoc broadcast protocol forms the entire basis of Bitcoin’s global, eventually consistent log, and is therefore of utmost importance to its correct and fair operation. If a data item does not spread throughout the network *quickly* then the system risks reaching an inconsistent state. Moreover, if a peer were somehow able to have their messages spread *more* quickly than others’, it could help that peer gain disproportionate profits from deviating from the protocol [10]. Bitcoin’s network formation procedure is intended to induce a random graph topology that should propagate information efficiently; however, as the topology has not been studied, it is unknown whether this ideal is actually attained. Thus a quantitative, thorough measurement and analysis of the Bitcoin peer-to-peer network is of critical importance to understanding and evaluating its properties.

¹Other Bitcoin clients and variants appear to adopt the same protocol details.

2.3 Miners and mining pools

A novel and unusual aspect of Bitcoin’s design is the “mining” mechanism by which the network robustly reaches global agreement on the set of committed transactions. Some nodes on the network, called “miners,” opt in to attempt to solve proof-of-work puzzles [2]. As these proofs of work are based on finding inputs to hashes that yield digests with many leading zero bits, they are solvable only through brute force. The difficulty of the puzzle is set so that on average, some miner on the network finds a solution every 10 minutes. Each puzzle identifier contains the hash of a previous puzzle solution as well as a new batch of transactions to append to the log, together called a “block”; the proof-of-work solutions therefore form a “blockchain”. Miners always work to extend the longest such blockchain they know of.² Upon producing a block, the miner propagates it to the rest of the network using the same broadcast mechanism as that for transactions. Honest miners who receive this block accept it and begin working to append it.

Although it is possible during ordinary operation for two different puzzle solutions extending equal-length chains to be found at approximately the same time, this happens infrequently since the average time between blocks is relatively slow compared to network latency; [9, 19] any such event is quickly resolved when one “fork” or the other gets extended and pulls ahead.

Mining is expensive; participation is encouraged through use of an incentive mechanism. Upon producing a valid block, the winning miner is rewarded with bitcoins in two forms: first, any difference between the input value and output value of a transaction included in the block is rewarded as a “fee”; second, a “block creation bonus” of newly minted coins. A miner claims these rewards by including a single transaction with no inputs, called the “coinbase” transaction.

These rewards also provide incentives for other, interesting behaviors. In particular, miners have incentive to garner as much computational power as possible; the more CPUs they have, the greater then chance they will be able to extend the block before anyone else. Additionally, *miners have incentive to collude* by pooling their resources together and splitting the profits rather than competing for them. This has led to so-called *mining pools*, who recruit other miners to collude.

On the one hand, mining pools require some degree of exposure (being able to advertise high win rates can be a useful tool for recruiting other members). However, they also have incentive not to appear to have grown too large: if any one entity approached a majority of the mining power, they could possibly prevent the rest of the network from globally converging on a growing transaction

²More accurately, the blockchain with the greatest cumulative puzzle difficulty. These can differ when the difficulty is adjusted.

log [11, 19, 22]. It is therefore of critical importance to develop the ability to investigate the true extent of mining pools’ collusion.

2.4 Related Work

Previous empirical studies have examined facets of Bitcoin’s surrounding ecosystem, such as online currency exchanges (and their tendency to collapse) [21] and illicit marketplaces [8, 18], and botnets that supplement their income through Bitcoin mining [12, 26]. A major focus has been on evaluating user privacy. While users can interact with Bitcoin using only pseudonyms, it has been demonstrated that Bitcoin transactions can be linked across pseudonyms to effectively “deanonymize” users [6, 15, 18, 24, 28, 30].

Relatively less work has examined the structure of the Bitcoin communication network itself. Decker et al. [9] measured the rate of information propagation throughout the network, and proposed modifications to accelerate it. Babaioff et al. [1] pointed out that peers have incentives *not* to participate in broadcast, but this could be remedied by paying fees to relaying peers. The effectiveness of the broadcast mechanism is essential to Bitcoin’s operation for two main reasons. First, it determines the potential security of “fast payments,” by which transactions are accepted based on their apparent propagation through the network, even before being ratified through inclusion in proof-of-work blocks [4, 14]. Second, an advantage in the broadcast network can be leveraged by a non-compliant miner to gain disproportionate rewards from mining [3, 10]. Prior work had focused on strategies non-compliant miners could use to gain reward if they had a broadcast advantage. Our work studies the underlying network topology, and shows that indeed, substantial broadcast advantage can be gained on the deployed network.

Our AddressProbe technique is directly related to previous work on network structure detection and analysis [20, 27, 29]. Biryukov et al. [6] disclose a technique that also uses Bitcoin address propagation messages to detect peer links. Their technique, however, is somewhat invasive since it involves polluting each node’s table of potential peers with fake entries. In contrast, AddressProbe only gathers and analyzes information that nodes readily provide, and is better suited to network-wide mapping.

While Bitcoin mining was initially performed using commodity computer equipment (i.e., “one-cpu-one-vote” [22]), the mining landscape has evolved according to two main trends: first, mining pools [31] have allowed users to pool their resources and share the rewards; and second, customized Bitcoin-specific mining hardware have been developed that vastly outperform general purpose computers [33]. There have been several game-

theoretic analyses of Bitcoin mining coalitions [10, 16] but, to our knowledge, no systematic empirical analysis that identifies mining entities. However, speculation about miner activities abounds [23], and we believe our work (Section 5) provides the first systematic mechanism to locate nodes correlated with mining pools in the wild.

3 Mapping the Broadcast Topology

In this section we describe an efficient method for discovering peer links in the Bitcoin network. We validate our approach, AddressProbe, using ground-truth data, and present an analysis of the broadcast topology.

3.1 Using timestamps to infer links

Recall that new Bitcoin nodes find initial network peers by querying a set of hard-coded DNS servers. The DNS servers provide joining nodes with their initial peer list to try to connect to. Each node maintains a local database called the `addrMan`, which it tries to populate with the addresses of other peers. Nodes exchange address information using two protocol messages: `GETADDR` and `ADDR`. `GETADDR` messages are requests and `ADDR` messages are replies that contain address information.

Upon initiating a new connection a Bitcoin node (say x) requests address information by issuing a `GETADDR` request (say to a peer y). Node y will reply with up to 1000 entries from its `addrMan` database, chosen uniformly at random.³ For each chosen entry, the reply `ADDR` message includes the (IP address, port) pair, a timestamp, and a list of services offered. The vast majority of entries in `addrMan` do not correspond to active connections but to addresses that the node has learned from `ADDR` messages of others. Bitcoin includes a somewhat unintuitive way of updating the timestamp corresponding to an address, which we exploit in formulating the AddressProbe technique. (Appendix A gives a more detailed walkthrough of the actual code).

- For outgoing connections, i.e., ones that it initiates, a node updates the timestamp (corresponding to the peer IP address and port in `addrMan`) each time it receives a message from the peer. Therefore, timestamps for outgoing connections are updated frequently, on the order of a few minutes maximum.
- For incoming connections, the timestamp is set to when the connection was created, but it is *not* updated as the peers exchange messages. Thus, if an incoming connection is long-lived, the `ADDR` message does not provide information to distinguish it as live.
- For all other (address, port) pairs that a node learns of (from `ADDR` messages sent by others), the node

³This is a simplification that describes the general behavior. For full details see Algorithm 1 in the appendix

“ages” the address by adding a two hour penalty before adding the address to its `addrMan`.

Finally, we note that a node can send unsolicited ADDR messages in two cases: first, upon receiving an incoming connection from n , a node x sends an ADDR message, to a randomly chosen peer, containing only x with the timestamp set to the current local time. Nodes keep state about which addresses their neighbors know of (because they received/sent information about these nodes from/to their neighbor). When nodes receive a ADDR messages with fewer than 10 entries, either as the result of a GETADDR response or a new connection, they choose two peers at random and relay the same ADDR information (without updating any timestamps), as long as the node believes the neighbors don’t already have this address. Nodes purge all information about what addresses their neighbors know every twenty-four hours. Thus when a node with a hitherto unknown address joins the network, the relay messages containing the node’s IP address eventually flood the entire network. Whenever an existing node x makes a new connection, its address propagates in the network; how far depends on how many nodes already knew about x . Finally, a node will update `addrMan` upon receiving an ADDR message with a newer timestamp for an address. The two-hour aging penalty is applied to the new timestamp learned.

We illustrate how timestamps update with a simplified example: consider a scenario where node x initiates a connection to node y at time t . Suppose that node y relays the information about the connection to neighbor n , and the randomized protocol further relays information about the connection to nodes r_0, r_1, \dots . Finally, assume node l learns about node x from node r_0 at a later time (not during the initial relay but as a reply to a GETADDR).

- As long as this connection is active, node x will report a recent timestamp for node y .
- Node y will not update its timestamp t for x , regardless of how long the connection stays open, unless it hears about x with a more recent timestamp than $t - 20m$ from a third node.
- Similarly, node y ’s neighbor n and the relay nodes r_i will initially also report the same timestamp t for node x . If this single relay manages to flood the network, then all nodes except those with outgoing connections to x will report timestamp t for x . They may update their timestamp for x later if they hear of a more recent timestamp or initiate a connection to x .
- Suppose node l received timestamp t' for node x . Node l will report $t' - 2h$, since it will age the timestamp by two hours. Any node that learns of x through l will further age the timestamp by two hours, and so on.

A’s ts of B	B’s ts of A		
	ts \geq 2hr	Unique & ts < 2hr	Not unique & ts < 2hr
ts \geq 2hr	\nexists edge	\exists edge $B \rightarrow A$	Unclear $A \not\rightarrow B$
Unique and ts < 2hr	\exists edge $A \rightarrow B$	\exists edge $A \leftrightarrow B$	\exists edge $A \rightarrow B$
Not unique and ts < 2hr	Unclear $B \not\rightarrow A$	\exists edge $B \rightarrow A$	Unclear

Table 1: Connection Inference rules for AddressProbe: Timestamps for nodes A and B as reported by repeated GETADDR requests. Here, $A \rightarrow B$ denotes that we can infer that A initiated the connection to B .

Therefore, by issuing GETADDR messages to all nodes in the network to whom we can connect, and analyzing the timestamps, we can obtain a map of connections, and potentially even when connections are made. By analyzing discrete two hour differences in timestamps, we can infer how nodes learn about each other. We summarize the inferences based on timestamps obtained by issuing GETADDR messages in Table 1.

Unfortunately, the description above simplifies true behavior in a few ways, which can lead to both false positive and false negative inferences.

False Positives The `addrMan` database is not updated when connections break or peers depart. Hence, a recently (less than two hours) broken outgoing connection leads to an inferred edge that no longer exists.

Simply receiving a “recent” timestamp is not an indication of an active outgoing connection, since relay nodes (r_i in our example) also respond with the connection genesis timestamp (t in our example). Thus, for new connections, all relay nodes (potentially the entire network) will respond with a recent and identical timestamp, which can lead to a false positive diagnosis. This is why Table 1 requires that a timestamp be both recent and unique in inferring outgoing connections.

False Negatives The `addrMan` structure is finite, and nodes may evict addresses, including those of actively connected peers. Addresses included in replies to GETADDR messages are chosen randomly, and it is possible that multiple GETADDR messages may “miss” an address, possibly of an outgoing connection. Both these scenarios will cause the inference to miss existing edges.

3.2 AddressProbe Implementation

Our approach to measuring the Bitcoin topology relies on short bursts of message activity to create a snapshot of the network. Yet, to achieve both a swift measurement and a wide one requires addressing a few technical challenges.

First, connections take time to establish, which means that an experimental platform must be long-running. To

GT Node	Num. true pos.	Num. false pos.	Num. false neg.
A	15.12 ± 1.84	0.08 ± 0.03	5.02 ± 0.69
B	8.29 ± 1.10	0.41 ± 0.17	2.13 ± 0.36
C	8.28 ± 1.13	0.29 ± 0.14	2.22 ± 0.37
D	7.63 ± 2.12	0.02 ± 0.04	2.92 ± 0.95
E	6.52 ± 0.81	0.20 ± 0.13	1.64 ± 0.27

Figure 1: Ground truth validation of AddressProbe, using runs spanning October 20–November 7 with five ground-truth nodes. Values denote averages with 95% confidence intervals.

connect to some hosts requires patience, as they may be temporarily “full” of connections and refuse more. Incoming connections, such as from hosts behind NATs and firewalls, accumulate slowly, as such hosts must first learn about our node through ADDR advertisements and then choose to connect to us. Once established, connections must be kept active to be maintained.

To provide a long-running platform, we isolate typical, base Bitcoin protocol functionality that accepts, creates, and maintains connections. This forms the CoinScope “connector”.

Second, the measurement techniques we apply are diverse and rely on wide distribution of messages. For example, requests for addresses may be sent to all connected hosts or inventory messages (Section 5) sent to a select set. Fortunately, techniques do not always need to process responses on-line.

We design the interface for CoinScope clients so that they issue commands to the connector using a library that connects via a control channel, and handle responses in a separate path. Typical commands include requests to connect to an address, list connected peers, or to broadcast GETADDR messages. Multiple CoinScope clients can connect to the control channel simultaneously to support concurrent (non-conflicting) experiments. CoinScope is efficient: it can saturate a 1Gbps network connection with Bitcoin protocol messages without significant CPU overhead. In our experiments, CoinScope is throttled such that messages to the entire network, such as GETADDR requests, are transmitted over one minute.

Third, developing measurement techniques can require substantial reinterpretation of data as a model of protocol behavior is refined. For example, our approach to interpreting ADDR messages based on their size has evolved. This encourages persistent storage of responses at the most primitive level—connection events as they occur and messages as they are received—so that these results can be reinterpreted.

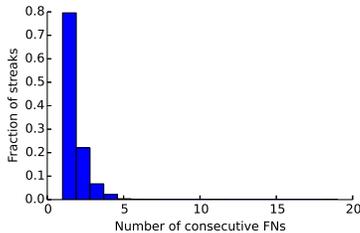


Figure 2: False negatives typically do not persist across more than one or two AddressProbe experiments, thus they are due to under-scraping peers’ addrMan.

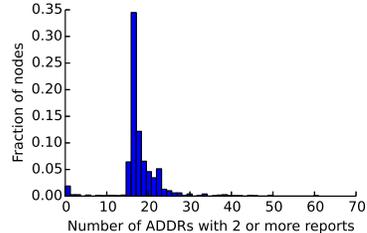


Figure 3: The vast majority of ADDR messages containing two or more addresses arrive within at most 24 GETADDR queries, thus we rarely under-scrape addrMan.

To provide both persistent storage and the feedback required by some CoinScope clients, we apply a logserver that marshals tagged messages from the connector to subscribers. A “verbatim” subscriber collects all events and writes them to disk for archival. A CoinScope client may subscribe to only relevant log messages, for example, to only those associated with ADDR messages to determine when to stop requesting from a given host. We have found that the tagged logserver architecture simplifies the combined task of archiving data while supporting on-line experiments.

In sum, the CoinScope architecture permits a wide view of the network by maintaining long-lived connections; supports concurrent, short-lived experiments by allowing clients to issue requests to the connector via a control channel; and transparently stores measurement results persistently by marshaling all responses through a logging system.

3.3 Validation using Ground Truth

To validate AddressProbe’s accuracy, we operated five “ground-truth nodes” throughout our experiments (from October 20, 2014 to November 7, 2014). Each of our ground-truth nodes was a mainline Satoshi client. Every two minutes, we collected a list of all active connections each peer has (as reflected in the PeerInfo data structure). Unfortunately, not all ground-truth nodes remained up the entire 18 days; in the results that follow, we average only over the experiments when a given node was available.

For the purposes of comparing ground-truth edges to edges inferred from AddressProbe, we distinguish between what we call *stable* and *transient* edges. We define an edge to be “stable” with respect to a given experiment if it appears in all PeerInfo snapshots within four hours before and four hours after the experiment. If AddressProbe fails to detect a stable edge, then we treat that as a false negative—it is very likely that the edge exists during the experiment if our ground-truth says it

was consistently present before and after the experiment. Similarly, if an edge appears in at least one such snapshot but not all, then we call it a “transient” edge. Failing to detect a transient edge is less dire than for a stable edge: it might not have existed during the experiment. Thus, if AddressProbe detects a transient edge, we count it as a true positive, whereas if it fails to detect a transient edge, we do *not* count it as a false negative. Finally, we clarify that we only consider edges between our ground-truth nodes and other nodes to whom we could connect—in particular, we do not include NAT’ed nodes.

We present our ground-truth results in Figure 1. Our false positive rates are extremely low across all ground-truth nodes: that is, AddressProbe is very unlikely to ever assert that an edge exists when it does not. A false positive can occur when a *non-unique* time-stamp less than two hours old appears to us to be unique, for instance because the time-stamp did not propagate far into the network. These data reflect such a case to be rare.

Next, we turn to the false negative rates in Figure 1, that is, cases when AddressProbe fails to find an edge. Although higher than its near-zero false positive rates, AddressProbe’s false negative rates are still considerably less than its true positive rate. Broadly speaking, there are two possible causes for a false negative AddressProbe:

1. AddressProbe *under-scraped* a peer’s `addrMan` and simply did not send enough GETADDR messages to learn about all of the peer’s edges, or
2. the peer *evicted* the edge’s corresponding entry from its `addrMan`. This can occur in the mainline Satoshi client if its `addrMan` has too many entries.

Ideally, the more common cause would be under-scraping: if eviction was common, that would be problematic for AddressProbe, as it would violate our assumption that if an edge exists, then so does a time-stamp less than two hours in at least one of the peer’s `addrMan`. Because peers’ responses to GETADDR choose randomly from `addrMan`, we expect that if we were to under-scrape, then it would be very unlikely to obtain a false negative on the same edge for many consecutive experiments. To evaluate this, we plot in Figure 2 how often AddressProbe obtains the same false negative for multiple consecutive experiments. This shows that 70% of the false negatives are resolved in the subsequent experiment; nearly 90% are resolved within two experiments. This provides strong evidence that *under-scraping is the most likely cause for AddressProbe’s false negatives*.

Figure 2 also shows evidence of eviction. There was a single edge, for instance, which AddressProbe failed to detect over 19 consecutive experiments. Because it is astronomically unlikely for an edge not to be randomly chosen from a peer’s `addrMan` after so many trials, we

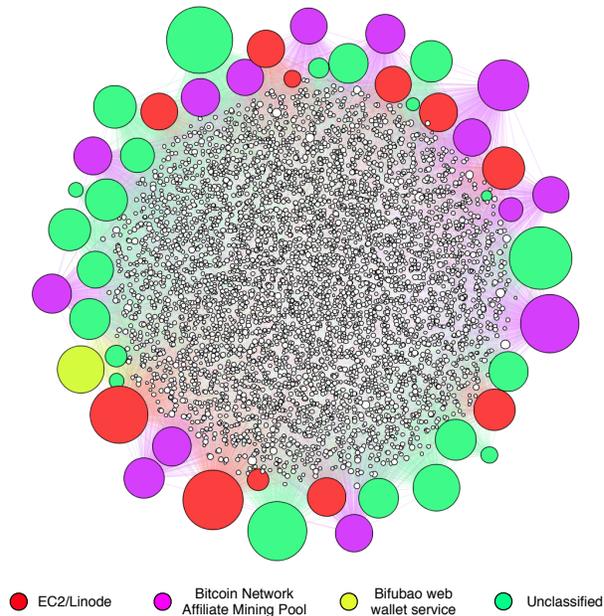


Figure 7: A snapshot of the (reachable) Bitcoin network discovered by AddressProbe on Nov. 5. The highest degree nodes (with degrees ranging 90–708) are colored.

believe this is an example of eviction. Fortunately, while the tail is long, it is also shallow, and thus we conclude that the root cause of false negatives is under-scraping.

We could of course improve AddressProbe’s false negative rate by simply scraping more. However, Figure 3 shows that we quickly reach a point of diminishing returns. In this figure, we show how many ADDR messages we received that contain two or more addresses (as these are the ADDR messages in response to our GETADDR queries). The x -axis represents how many GETADDR queries we sent until a peer stopped sending us ADDR responses with two or more addresses. The concentration at $x = 0$ corresponds to the set of nodes who had yet to populate their `addrMan` data structures. Most nodes’ `addrMan` were exhausted after sending 16 GETADDR requests, with a sharp decline after the 20th GETADDR. Guided by this, we instituted a cut-off at 24 GETADDR messages in order to balance between completeness of results and bandwidth preservation.

To summarize, AddressProbe is effective at accurately detecting edges, and its threats to validity (predominantly eviction) are extremely uncommon. Our main source of errors is due to our decision to trade off false negatives for increased bandwidth consumption, but future applications of AddressProbe need not make this trade-off.

4 The Bitcoin Topology

We begin our analysis of the Bitcoin topology by investigating the topological features of the peer-to-peer net-

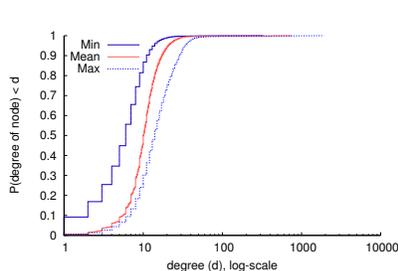


Figure 4: Degree distributions from AddressProbe runs on 10/20–11/7.

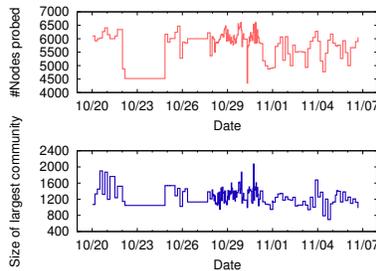


Figure 5: Nodes probed and largest community.

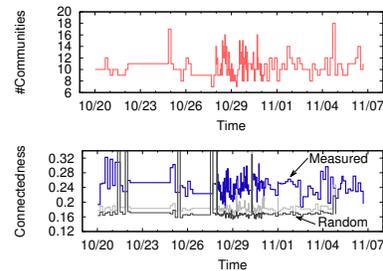


Figure 6: Community connectedness.

work using AddressProbe measurements over a period of 18 days. Most of our measurements span four consecutive days (10/28–10/31), allowing us to see if there are topological changes within a relatively short window of time. We also performed AddressProbe measurements one week before and one week after this period of time, though less frequently, so as to determine if these were representative or sensitive to longer-term diurnal patterns. In sum, we collected 133 snapshots of the network using AddressProbe.

We present a representative snapshot of the Bitcoin topology in Figure 7. Each node is sized proportionally to its degree. Two features immediately stand out: First, a handful of nodes have *far* greater degree than others in the system; in this section, we identify these high-degree nodes. Second, upon visual inspection, this graph appears to be random; we demonstrate in this section, however, that it exhibits properties that distinguish it from a random graph.

Node degree. Figure 4 shows the degree distributions averaged across all our network snapshots. We also plot the single minimal and single maximal distributions (as determined by the runs’ average degree). This demonstrates that the mean is representative, and that the shape is upheld across all 2.5 weeks of our experimentation.

On average, the majority of nodes to which we could connect have degree in the range of 8–12. This is consistent with the mainline Satoshi client implementation: unmodified peers seek to maintain eight outgoing connections, and permit incoming connections, as well. Because this is constrained only to the edges for which we could connect to both nodes, this result does not include any edges from NAT’ed nodes, and thus nodes are likely to have greater degrees than is plotted here. However, recall that peers cannot create *outgoing* connections to NAT’ed nodes—if AddressProbe is accurate, then it should be able to detect all outgoing edges. The fact that the degree distribution is so heavily concentrated near eight indicates that AddressProbe is indeed accurate at detecting these outgoing edges.

Another feature of the mainline Satoshi client is that

it permits a maximum of 125 active connections, but we consistently see nodes that far exceed this maximum, sometimes by a *factor of nearly 80*. This is not a singular phenomenon; we see these high degree nodes across all runs of AddressProbe. That is, *extremely high-degree nodes are persistent over time*.

Benign measurement studies seeking to understand the Bitcoin topology could appear to be high-degree nodes.⁴ In an effort to understand the root cause behind these high degrees, we tried to manually classify all nodes with degree at least 90 in the network snapshot of Figure 7. Of the nodes we could classify, over half of them were members of the Bitcoin Affiliate Network mining pool. Also, we identify one node from a Bitcoin wallet service. These results indicate that the long tail of high degree nodes is indeed not an anomaly of AddressProbe, but rather an accurate reflection of coordinated efforts to measure the network. We also identify many high-degree nodes running within cloud providers such as EC2, but have thus far been unable to classify them further.

Graph randomness. Visual inspection of Figure 7 seems to indicate that the Bitcoin network is random. To evaluate this more formally, we apply the so-called Louvain community detection algorithm [7] to each snapshot graph of the Bitcoin network AddressProbe returns. This algorithm returns a set of communities, with the property that nodes within a given community exhibit greater connectivity than two nodes in different communities. Figure 5 shows that the largest community in the network typically constitutes roughly 25% of the overall network.

For a given community C , let $E_{\text{intra}}(C)$ denote the set of intra-community edges (edges connecting two members within C) and let $E_{\text{all}}(C)$ denote all edges involving a member of C (inter- and intra-community edges). For each graph returned by AddressProbe, we compute what we call the graph’s *community connectedness*: the weighted average of communities’ fraction of intra-community edges. Concretely, if a graph has N nodes

⁴Acknowledging this, AddressProbe does not reply to GETADDR messages like other clients, so we expect that our experiments have not affected others.

and M communities (C_1, \dots, C_M) , then its community connectedness is:

$$\sum_{i=1}^M \frac{|C_i|}{N} \cdot \frac{|E_{\text{intra}}(C_i)|}{|E_{\text{all}}(C_i)|} \quad (1)$$

This definition of connectedness serves as a useful metric for determining how well the graph supports fast mixing and dispersion of data. Note that community connectedness is maximized with a value of one for graphs whose communities’ edges are *strictly* inside the community, that is, the graph would be fragmented. Broadly speaking, low values of connectedness indicate a healthier Bitcoin network.

Figure 6 shows the community connectedness (bottom) and overall number of communities (top) across all 133 snapshots of the Bitcoin network we obtained with AddressProbe. Note that the community connectedness remains within a relatively small bound over these 18 days, tightly centered around 0.24. This low variance indicates that there are few major changes to the inherent structure of Bitcoin’s network over time.

We use connectedness to answer the question of whether the Bitcoin network is truly a random graph. To lend perspective to the raw values of connectedness, we also generated random graphs with the same vertex and edge counts of each 133 networks and computed their connectedness. We do not model the high-degree nodes in our generation of these random graphs; because the high-degree nodes constitute a small fraction of the overall edges, we believe this not to affect the results. Figure 6 presents the average connectedness among ten random graphs, as well as the average plus two standard deviations. We observe that truly random graphs of the same size exhibit connectedness that is statistically significantly smaller than those of the Bitcoin network. In more than 98% of the runs, the measured graph’s connectedness was more than two standard deviations greater than that of the random graph. This indicates that *the Bitcoin network is not purely random*. We hypothesize that this deterministic structure is due to the process by which nodes join the Bitcoin network: recall that they connect to a small set of DNS nodes, and slowly percolate to more distal parts of the graph as they learn of new peers. Quantitatively evaluating the source of determinism is an area of future work.

Discussion. AddressProbe provides an accurate, repeatable snapshot of the entire Bitcoin network within tens of minutes. With this unprecedented view into the Bitcoin topology, we make two important observations:

1. There are peers who connect to $\sim 125 \times$ more than the typical peer; these almost exclusively correspond to mining pools and other measurements nodes.

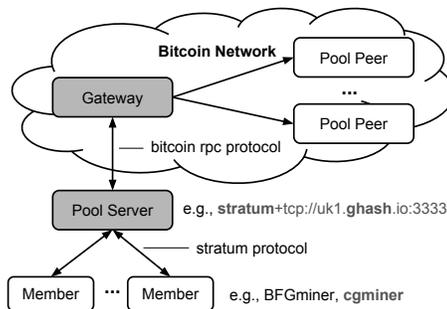


Figure 8: A typical mining pool architecture. The gateway and pool servers are administered by the pool operator. A pool server generally has a publicly known address, while the gateway does not. A gateway may accept inbound connections.

2. The Bitcoin graph is, for the most part, consistently well connected, but it exhibits properties that distinguish it from a truly random graph.

In small doses, these topological abnormalities are not detrimental to Bitcoin’s operation. However, it is important to note that the Bitcoin protocol does nothing to keep them from happening. Malicious or misguided behavior could drive the network to bad states, such as worse connectedness or, worse yet, a disconnected network. Regularly running AddressProbe to collect network-wide topology information can help the community more quickly detect and react to attacks and mistakes. We continue to run AddressProbe and will make our data publicly available for the Bitcoin and research communities.

5 Discovering Influential Nodes

AddressProbe can efficiently map the reachable Bitcoin topology, and our analysis has shown both the structure of the network, and how malicious nodes can affect broadcast. However, the vast majority of compute power that sustains Bitcoin and extends the block chain is not visible, but instead is concealed within *mining pools*. This is apparent from observing that these pools mine the vast majority of blocks and reap new coins. In this section, we describe the structure of these pools, and then introduce techniques that discover how pools interact and interface with the visible broadcast network.

5.1 Structure of Mining Pools

Commercial mining pools have a centralized administrative structure. The mining pool operator uses a command and control (C&C) system to coordinate the pool’s computation.

A prototypical mining pool (Figure 8) consists a pool server, one or more gateways, and the pool members. We describe each in turn next.

Pool Server The pool server assigns units of work to pool members. Specifically, the mining pool periodically assigns each member a Merkle root of a set of transactions, and a range of nonce-space to search through. Mining participants explore this search space looking for nonces that result in valid “shares”, which are partial proofs-of-work that are a superset of the ones that qualify as actual Bitcoin blocks. Mining pools support one of several protocols for communicating with members’ mining-rigs, the most popular of which is Stratum [25]. Generally, the IP address of a pool’s Stratum server is publicly known, and can be found in documentation for joining the pool.

Many pools operate multiple pool servers in different geographic regions to improve fault tolerance and latency. There are several open source implementations of pool servers, although mining pools may also use custom implementations.⁵

Gateways A mining pool typically connects its pool server to a local or trusted instance of a Bitcoin node—the gateway—to which the pool server communicates using the Bitcoin RPC protocol. Gateways interface pool resources with the global Bitcoin network, and must provide low latency broadcast for the following reasons:

- When a pool member finds a share that is a winning Bitcoin solution, the pool operator needs to claim the embedded coins and fees by broadcasting the new block to the rest of the network. This should be done as quickly as possible, since any delay increases the risk that a competing pool will find a competing block (we ignore for now block-withholding attacks, in which a pool may deliberately delay block propagation if it harms competing pools more than itself [10]).
- A pool must also learn about blocks and transactions broadcast by other nodes in the network quickly to minimize wasted work done by mining on an old block. The pool also has an incentive to learn about transactions so it can include them in its blocks and collect the transaction fees.

Pool Members Mining pool participants typically run specialized mining-rig control software, such as BFG-Miner, cgminer, and poclbm. One reason for the use of various software like this is the need to control various mining equipment, such as overclocking, monitoring temperature, and detecting errors. Mining pool participants need not run an ordinary Bitcoin node; the task of receiving, validating, and relaying transactions is delegated to the mining pool operator.

Tracking mining pool power On average, a mining pool wins a number of blocks in proportion to the total amount

of hash power the pool members contribute. Mining pools often claim credit for the blocks they produce by posting them on the pool’s website. (Presumably publicly claiming blocks help pools recruit new members.) Pools often use a longstanding public key, and mine blocks that pay the block reward to this key. It is possible for a pool to mine a block, forego the block reward, and reward another, since only the public key is necessary to pay the block reward to a key. Pools may also claim its blocks by including a short message in the coinbase transaction (effectively, a string that miners can use to convey arbitrary information); these can be more readily spoofed since it does not require the miner to forego the block reward.

Occasionally, large enough pools may have an incentive to conceal their hashpower. Since there are devastating attacks on Bitcoin (e.g., history revision attacks) that become feasible when a single entity wields more power than the rest of the network, when a pool grows very large in size it attracts the concern and suspicion of the Bitcoin community [17].

Several aggregator websites, including blockchain.info, maintain low-latency connections to a thousands of nodes and record the IP address of the first connected node to relay each block. This method is most effective when the aggregator has a direct connection to the gateways that initiate the broadcast of each block; in any case, this method is sensitive to transmission latency.

Several other websites such as <http://organofcorti.blogspot.com> and <http://mempool.info/> use the methods described along with other anecdotal evidence specifically to monitor the activity of large mining entities.

5.2 Influential Nodes

In the rest of this section, we describe a new technique for finding a small set of “influential” nodes in the Bitcoin broadcast network. We show that this small set of nodes (≈ 100 or so) seemingly account for over three-quarters of the hash power. We hypothesize that these nodes are either gateways to mining pools, or are otherwise connected with low-latency to gateways.

Interestingly, the set of influential nodes have entirely benign topological and protocol features: they don’t have exceptionally high degree, don’t form an exclusive community, don’t have a unique version string, or are even particularly long lived in the network, making it difficult to find and track these nodes. However, these nodes provide an exceptional network advantage for broadcast: as long as a transaction (block) reaches these nodes, it is far more likely to be included in a block (extended in the block chain).

Prior work [10] has shown that a selfish mining pool can gain advantage if it initially withholds blocks it finds, but then releases them as soon as a competing block is

⁵See <https://en.bitcoin.it/wiki/Poolservers> for a comparison of publicly available pool server software.

found. The advantage accrues proportional to the fraction of the rest of the mining pool that extends the block released by the selfish pool. Assuming honest pools broadcast using the standard protocol, and the selfish pool selectively creates low latency connections to the influential set, the latter can gain huge broadcast advantage. In the limit, the attacker can win every broadcast race, and therefore profits from selfish mining regardless of its fraction of hashpower. As the rest of the section will demonstrate, such an attack is not hypothetical, but indeed feasible on the current Bitcoin network.

5.3 Decloaking

We introduce a randomized protocol for identifying influential nodes. Our protocol has two phases: Candidate Selection (CS), which finds potential influential nodes, followed by Influence Validation (IV), which demonstrates the candidates do indeed represent disproportionate mining power. In practice, the CS and IV phases should be run concurrently, with IV validating the output of CS.

The CS and IV algorithms both use two low-level primitives, which we introduce first.

Coloring Nodes Each Bitcoin node maintains a set of transactions it has received in a data structure called memPool. Typical mining pool server implementations (e.g., CoiniumServ) use the memPool of the connected gateway node to determine which transactions to include in their blocks. Two Bitcoin transactions “conflict” if they both spend a common transaction input (i.e., represent a double spend). Bitcoin requires that among a pair of conflicting transactions, at most one may be included in a block. Nodes must guarantee that none of the transactions in their memPool are conflicting. The reference client always prefers the first transaction it receives, i.e., a node discards any transaction that conflicts with one already resident in its memPool.

Ideally, we could color each node with a different conflicting transaction, and upon repeating this step, we would find the influential nodes, because the transactions sent to them would “win”, or be included in a block, more often. This could potentially be accomplished by creating a unique conflicting transaction for each reachable node in the network, and delivering them to each node *simultaneously*. (Otherwise, once a node receives its transaction, following the protocol, it would forward to others, and the mapping of transaction to node would no longer remain one-to-one.) Unfortunately, due to varying network latencies and connections between nodes via other unreachable (NAT’ed) nodes, it is practically impossible to simultaneously send a distinct transaction to each node using simultaneous delivery. We describe a feasible technique for coloring next.

InvBlocking Bitcoin uses a flooding protocol to propagate transactions (and blocks) throughout the network. To reduce bandwidth, and indeed to curb uncontrolled flooding, a node does not simply broadcast any new transaction (or block) to all neighbors, but instead employs a three-round protocol. After a node accepts a new transaction into its memPool, it floods an INV message containing the transaction *hash only* to each neighbor. Neighbors may choose to pull the transaction using a GETDATA message with the corresponding hash.

As a broadcast makes its way through the network, a node may receive the same INV from more than one neighbor prior to receiving the transaction itself. Upon receiving subsequent INV messages containing the same hash, the node adds subsequent message to a queue, and waits for two minutes before timing out on any outstanding GETDATA messages before sending another a neighbor who had also sent an INV. It is therefore possible to *block* a node from hearing about a transaction for two minutes by sending an INV message and then ignoring the resultant GETDATA.⁶

The InvBlock procedure (Algorithm 2) sends a set of INV messages (corresponding to a set of conflicting transactions when coloring nodes), before sending out the transaction itself. This provides a two minute window over which selected transactions can be sent to specific nodes, without having to win a network latency race. InvBlocks could used to distinctly color each node, but it can also be used to color disjoint sets of nodes, where each node in a set receives the same transaction, but each set receives a different transaction. This latter method allows a more precise control on bandwidth overhead, since the number of INV messages sent to each node is not dependent on the size of the network but on the number of sets. We use this **generalized InvBlock** in the CS and IV algorithms as described next.

5.3.1 Candidate Selection (CS)

The CS algorithm 3 is parameterized with an integer M corresponding to the number of node sets in generalized InvBlock. The set of all known nodes is partitioned into M (roughly) equal sets, with each node having the same probability of being mapped to any set. These sets are colored using generalized InvBlock, i.e., each node in the randomly generated set receives the same transaction, and each set receives a conflicting transaction.

One of these transactions eventually gets included in a block. We identify the set to which this transaction was sent, and increment a “win” score for each node in the set by one.

⁶ Technically, it is possible to delay a node longer by sending multiple INV messages. This bug was found concurrently to our work by Bitcoin developers (see <https://github.com/bitcoin/bitcoin/pull/4547>) but a patch has not yet been deployed.

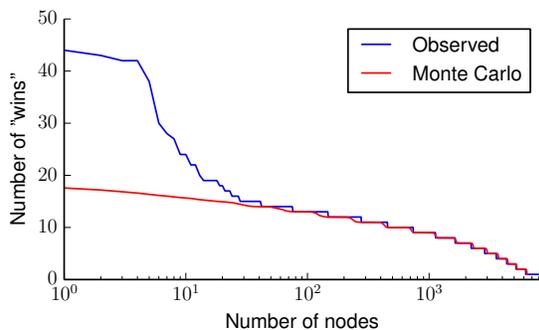


Figure 9: Candidate Selector Results

Over multiple trials with different randomly generated sets, this simple procedure identifies influential nodes as they tend to have disproportionately high scores. Figure 9 shows the observed results of approximately 500 trials on the deployed Bitcoin network with $M = 100$. Each trial was spaced 5 minutes apart over two days. The figure shows the number of wins per node (IP address, port) compared to a distribution obtained through 100 rounds of Monte Carlo simulation in which the winning node is chosen uniformly at random. (We used a MC simulation since the number of nodes in the network changed in each run). The signal from CS is stark: compared to uniform chance, CS cleanly identifies the small number of influential node candidates.

5.3.2 Influence Validation (IV)

We have seen that CS cleanly distinguishes a small set of nodes as influential. The IV algorithm validates this set (or indeed a candidate set generated from any other source) using a similar procedure. IV uses generalized InvBlock with the following structure: each CS identified influential node is a singleton set, and the rest of the network is one other (giant) set. Thus each potential influential node works on a different conflicting transaction, and the rest of the network works on a single (conflicting) transaction. This procedure is detailed in Algorithm 4.

We ran IV 258 times in total—always after running CS—with runs spaced 5 minutes apart. A different conflicting transaction was sent to the influential nodes, and the rest of the network received yet another conflicting transaction. The influential nodes comprised the top 86 from CS and the 14 top relayers of unknown blocks from blockchain.info/pools (together $< 2\%$ of the network in total) and won in 189/258 trials (73%). The candidates identified by CS accounted for 179 out of 189 wins.

6 Bitcoin’s Influential Nodes

In this section, we present an analysis of the influential nodes: how they map to mining pools, and how our procedure can identify pools that were previously unknown.

The IV experiment shows that the candidate nodes found by the CS experiment are indeed influential, accounting for roughly 3/4 of transactions injected during our two day experiment. In this section, we present an analysis of the “winning-most” IV nodes (referred to as the IV set below). Here, we say an IV node “wins” when the transaction sent to it (and it alone) is the unique one (among the set of conflicting transactions) that gets included in a block.

Our analysis infers details about the organization of mining pools (at the time of writing); in some cases we can corroborate these details with supplemental evidence found by public records on the web and DNS records. We present our results in Table 2, which shows aggregate behavior of nodes (columns) and pools (rows) that had multiple “wins”. Each entry shows how often a node, that corresponds to an IP address and port, but we’ve designated with a letter, wins for a pool. The table includes pools our experiment has discovered.

The two largest pools at the time of writing, DiscusFish and GHash.IO, account for 17/35 nodes and 115 wins among the multiple winners in the IV set. Addresses for the DiscusFish nodes (registration, location) or their protocol messages did not distinguish them as influential; we were able to locate these nodes only by the CS experiment. The IP addresses for two of the nodes (L and O) associated with GHash.IO resolve to the DNS name of GHash.IO’s Amsterdam pool server (nl1.ghash.io); the two nodes with the *most* wins for this pool (H and I) are located within the same /24 block.

BitFury is in fact not a mining pool, but rather a commercial entity that manufactures Bitcoin mining equipment and operates large-scale mining centers. All of the “wins” from blocks paying out to BitFury’s publicly-known Bitcoin address correspond to two nodes (S and T) with IP addresses within the same /24 block registered in Iceland (one of the locations where BitFury claims to have a mining operation, see bitfury.com).

Three “unknown” entities, identified only by the addresses they pay out to (1AcAj..., 19vvt..., and 1FeDt...) correspond respectively to nodes located in Germany and Georgia. Nodes U and V are both resolved to by names within the “high.re” DNS hierarchy. The association between node U and 1AcAj... was previously suspected (see below), and was confirmed by CS and IV; the other IP and Bitcoin address associations are new.

Associating Bitcoin addresses with pools (or IP addresses) is an open problem. There is public speculation about an association between the anonymous Bitcoin address 1AcAj... and BitFury [23]. Our techniques independently associated node U’s IP address with the payout address, as it did for the previously unknown nodes V and W with different anonymous Bitcoin addresses. We end by noting that the nodes S, T, U, V, and W are

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
DiscusFish	21	14	13	12	10	3	3																		
GHash.IO								8	6	5	5	3	3	3	3									4	
KnCMiner								1								3	2	2							
BitFury																			13	7					
1AcAj...																					2				
19vvt...																						2			
1FeDt...																							19		
Slush																2								2	2
Total	21	14	13	12	10	3	3	9	6	5	5	3	3	3	3	5	2	2	13	7	2	3	23	2	3

Table 2: Selected results from the Gateway Identifier experiment. Each column (A-Y) represents the IP address of a reachable node, and each row represents a mining pool. Each value indicates the number of times a transaction sent to the corresponding node was included in a block mined by the corresponding pool.

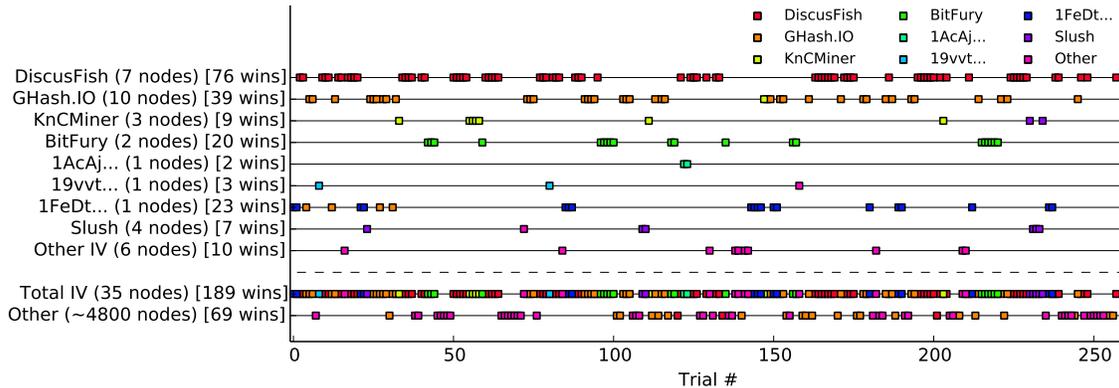


Figure 10: IV set versus the rest of the network: wins over time. The win patterns appear ‘bursty’ because a single block often contains multiple IV transactions.

associated with IP addresses registered in countries that BitFury publicly claims to host infrastructure in, they all run the same version of Bitcoin, host the same web server (with the same version of nginx), and T-W serve the same (insecure) authentication page.

In Figure 10 we show the time-varying behavior during the IV experiment. Horizontal lines across time show wins by different sets of nodes, e.g., all wins for nodes that win for a pool, e.g., A-G for DiscusFish, are aggregated on the highest horizontal line. The penultimate lowest horizontal line shows all wins by the IV set and the lowest horizontal line tabulates the wins by all the other reachable nodes in Bitcoin. Recall that this time-series spans two days in total, with new conflicting transactions, configured as described in the IV experiment, injected every five minutes. The figure provides a visual measure of the pool influence, and also of how influential the IV set is.

7 Conclusion

Bitcoin’s successful, fair operation is predicated on the notion of peers being able to reach a global consensus. A critical component of the Bitcoin protocol is a broadcast substrate that serves as the sole means by which honest peers can learn from and inform other peers. We have

described techniques for efficiently mapping the Bitcoin broadcast network and for identifying nodes who have disproportionate influence on the Bitcoin network.

Our AddressProbe technique is distinguished from prior work in that it is efficient and operates without polluting peers’ local state—we therefore believe that it is more scalable and more likely not to affect Bitcoin peers when applied widely. Indeed, with our CoinScope implementation, we show that we are able to scan the entire network in minutes, and at regular intervals.

Bitcoin network topologies reconstructed using AddressProbe show that the broadcast network is resilient, but does not behave like a traditional random graph.

A major contribution of this paper is the finding that the broadcast topology conceals influential nodes that represent disproportionate amounts of mining power. We introduced novel mechanisms to find these nodes and to measure their impact. Our results show that *roughly 2% of the nodes account for three-quarters of the mining power.*

Our intent is that these techniques can be applied to perform longitudinal analyses of the Bitcoin network. We acknowledge, however, that AddressProbe is somewhat “fragile” in that it depends on undocumented features of the mainline Satoshi client; it would not be in-

feasible for a developer to modify some of the internal data structures in a way that would confuse AddressProbe. Conversely, disabling the decloaking techniques would be far more difficult, as it is based on the three-round exchange on which Bitcoin’s efficient broadcast depends. We hope that the findings in this paper—that understanding topology can identify structural faults to the broadcast—will encourage the Bitcoin community to evolve the protocol to explicitly support efficient topology discovery.

References

- [1] BABAIOFF, M., DOBZINSKI, S., OREN, S., AND ZOHAR, A. On Bitcoin and red balloons. In *Proceedings of the 13th ACM Conference on Electronic Commerce* (2012), ACM, pp. 56–73.
- [2] BACK, A. Hashcash—a denial of service counter-measure. <http://www.hashcash.org/papers/hashcash.pdf>, 2002.
- [3] BAHACK, L. Theoretical Bitcoin attacks with less than half of the computational power (draft). *arXiv preprint arXiv:1312.7013* (2013).
- [4] BAMERT, T., DECKER, C., ELSÉN, L., WATTENHOFER, R., AND WELTEN, S. Have a snack, pay with Bitcoins. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on* (2013), IEEE, pp. 1–5.
- [5] BARBER, S., BOYEN, X., SHI, E., AND UZUN, E. Bitter to better—how to make Bitcoin a better currency. In *Financial Cryptography and Data Security*. Springer, 2012, pp. 399–414.
- [6] BIRYUKOV, A., KHOVRATOVICH, D., AND PUSTOGAROV, I. Deanonymisation of clients in Bitcoin P2P network. *CoRR abs/1405.7418* (2014).
- [7] BLONDEL, V. D., GUILLAUME, J.-L., LAMBIOTTE, R., AND LEFEBVRE, E. Fast unfolding of community hierarchies in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 10 (2008).
- [8] CHRISTIN, N. Traveling the silk road: A measurement analysis of a large anonymous online marketplace. In *Proceedings of the 22nd international conference on World Wide Web* (2013), International World Wide Web Conferences Steering Committee, pp. 213–224.
- [9] DECKER, C., AND WATTENHOFER, R. Information propagation in the Bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on* (2013), IEEE, pp. 1–10.
- [10] EYAL, I., AND SIRER, E. G. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security* (2014).
- [11] GARAY, J. A., KIAYIAS, A., AND LEONARDOS, N. The Bitcoin backbone protocol: Analysis and applications. <https://eprint.iacr.org/2014/765.pdf>, 2014.
- [12] HUANG, D. Y., DHARMDASANI, H., MEIKLEJOHN, S., DAVE, V., GRIER, C., MCCOY, D., SAVAGE, S., WEAVER, N., SNOEREN, A. C., AND LEVCHENKO, K. Botcoin: monetizing stolen cycles. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)* (2014).
- [13] JOHNSON, B., LASZKA, A., GROSSKLAGS, J., VASEK, M., AND MOORE, T. Game-theoretic analysis of DDoS attacks against Bitcoin mining pools. In *Financial Cryptography and Data Security, Lecture Notes in Computer Science*. Springer, 2014, pp. 72–86.
- [14] KARAME, G. O., ANDROULAKI, E., AND CAPKUN, S. Double-spending fast payments in Bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 906–917.
- [15] KOSHY, P., KOSHY, D., AND MCDANIEL, P. An analysis of anonymity in Bitcoin using P2P network traffic. In *Financial Cryptography and Data Security* (2014), International Financial Cryptography Association.

- [16] KROLL, J. A., DAVEY, I. C., AND FELTEN, E. W. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Proceedings of WEIS* (2013), vol. 2013.
- [17] MATONIS, J. The Bitcoin mining arms race: Ghash.io and the 51% issue. <http://www.coindesk.com/bitcoin-mining-detente-ghash-io-51-issue/>, July 2014.
- [18] MEIKLEJOHN, S., POMAROLE, M., JORDAN, G., LEVCHENKO, K., MCCOY, D., VOELKER, G. M., AND SAVAGE, S. A fistful of Bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference* (2013), ACM, pp. 127–140.
- [19] MILLER, A., AND LAVIOLA JR, J. J. Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin. Tech. rep., University of Central Florida, 2014.
- [20] MISLOVE, A., MARCON, M., GUMMADI, K. P., DRUSCHEL, P., AND BHATTACHARJEE, B. Measurement and Analysis of Online Social Networks. In *Proceedings of the 5th ACM/USENIX Internet Measurement Conference (IMC'07)* (San Diego, CA, October 2007).
- [21] MOORE, T., AND CHRISTIN, N. Beware the middleman: Empirical analysis of Bitcoin-exchange risk. In *Financial Cryptography and Data Security*. Springer, 2013, pp. 25–33.
- [22] NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. <http://bitcoin.org/bitcoin.pdf>, 2008.
- [23] NEIGHBOURHOOD POOL WATCH. June 22nd 2014 Weekly Network and Block Solver Statistics. <http://organofcorti.blogspot.com/2014/06/june-22nd-2014-weekly-network-and-block.html?q=bitfury>, 2014.
- [24] OBER, M., KATZENBEISSER, S., AND HAMACHER, K. Structure and anonymity of the Bitcoin transaction graph. *Future internet* 5, 2 (2013), 237–250.
- [25] PALATINUS, M. Stratum mining protocol - ASIC ready. <https://bitcointalk.org/?topic=108533.0>, September 2012.
- [26] PLOHMANN, D., AND GERHARDS-PADILLA, E. Case study of the miner botnet. In *Cyber Conflict (CYCON), 2012 4th International Conference on* (2012), IEEE, pp. 1–16.
- [27] POUWELSE, J., GARBACKI, P., EPEMA, D., AND SIPS, H. The Bittorrent P2P File-Sharing System: Measurements and Analysis. In *4th International Workshop on Peer-To-Peer Systems (IPTPS)* (2005).
- [28] REID, F., AND HARRIGAN, M. An analysis of anonymity in the Bitcoin system. *Security and Privacy in Social Networks* (2012), 197.
- [29] RIPEANU, M., FOSTER, I., AND IAMNITCHI, A. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal* 6 (2002), 2002.
- [30] RON, D., AND SHAMIR, A. Quantitative analysis of the full Bitcoin transaction graph. In *Financial Cryptography and Data Security*. Springer, 2013, pp. 6–24.
- [31] ROSENFELD, M. Analysis of Bitcoin pooled mining reward systems. *arXiv preprint arXiv:1112.4980* (2011).
- [32] SCHOENMAKERS, B. Security aspects of the ecashtm payment system. *State of the Art in Applied Cryptography* (1998).
- [33] TAYLOR, M. B. Bitcoin and the age of bespoke silicon. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems* (2013), IEEE Press, p. 16.
- [34] VASEK, M., THORNTON, M., AND MOORE, T. Empirical analysis of denial-of-service attacks in the Bitcoin ecosystem. In *1st Workshop on Bitcoin Research. Lecture Notes in Computer Science, Springer (March 2014)* (2014).

A Bitcoin Address Propagation

This section lists pseudocode for the behavior followed by the Bitcoin reference client for handling peer addresses. The pseudocode, listed in Algorithm 1 describes the algorithm found in the version 0.9.2 satoshi client, files `addrman.cpp,h`, and `net.cpp`. The data structures used in the algorithm are as follows:

- `addrMan`: A data structure containing every address the Bitcoin node currently knows about.
- `addrKnown`: Bitcoin will not send the same ADDR message to a node twice within 24 hours. This structure maps ADDR sets to the peers they have been sent to.
- `addrBuf`: A buffer that accrues future ADDR messages for a given node.

B Pseudocode for Candidate Selection and Influence Validation Algorithms

In Algorithm 2, we provide detailed pseudocode listings for the `InvBlock` procedure. In Algorithm 3 (resp. Algorithm 4) we provide pseudocode for the Candidate Selection (resp. Influence Validation) routines described in Section 5

```

InvBlock( $n, tx, \tau$ ) :
  for  $\lceil \tau / (2 \text{ minutes}) \rceil$  do
    send INV[ $\mathcal{H}(tx)$ ] to  $n$ 

  on recv GETDATA[ $\mathcal{H}(tx)$ ] from  $n$  do
    Ignore; Do not send  $tx$  to  $n$ .

```

Algorithm 2: The `InvBlock` procedure delays node n from learning about transaction tx for time τ .

```

at node  $x$  store the following data:
┌   addrMan: a mapping from node  $n$  to timestamp  $ts$ 
├   list of connected peers, for each peer
├   ┌    $outbound$  // peer is an outbound connection
├   │   addrKnown // peer knows about address (can set/get)
├   └   addrBuf // set of addresses to send to peer
└

on receive  $(*, y)$  // any message received from peer  $y$  do
┌   if  $y.outbound$  //  $y$  is an outbound connection then
├   ┌   if  $addrMan[y].ts < (now - 20 \text{ minutes})$  then  $addrMan[y].ts \leftarrow now$ 
├   └
└

on receive  $(ADDR[addr\_vector], y)$  //  $ADDR$  message from peer  $y$  with addresses in  $addr\_vector$  do
┌   for each address  $a$  in  $addr\_vector$  do
├   ┌    $y.addrKnown \ll a$ 
├   │   if  $a.ts$  is invalid (very old or 10+ minutes in the future) then  $a.ts \leftarrow (now - 5 \text{ hours})$ 
├   │   if  $a.ts < (now - 10 \text{ minutes})$  then
├   │   ┌   choose 1-2 nodes  $n$  uniformly at random
├   │   │   └    $buffer\text{-}to\text{-}send(n, a)$ 
├   └    $addrMan[a].ts \leftarrow (now - 2 \text{ hours})$  // store in  $addrMan$  with a 2 hour penalty
└

on receive  $(GETADDR, y)$  do
┌    $y.addrBuf \leftarrow \emptyset$  // clear send buffer
├    $q \leftarrow$  up to 2500 addresses chosen uniformly at random from  $addrMan$ 
├    $buffer\text{-}to\text{-}send(y, q)$ 
└

on receive  $(VERSION, y)$  do
┌   send  $GETADDR$  to  $y$ 
├   if  $y.outbound$  then  $buffer\text{-}to\text{-}send(y, x)$ 
└

procedure  $buffer\text{-}to\text{-}send(peer\ y, addr\_vector\ A)$ 
┌   for each address  $a$  in  $A$  do
├   ┌   if  $a \notin y.addrKnown$  then  $y.addrBuf \ll a$ 
├   └
└

every 100 ms do
┌   for one randomly chosen connected peer  $p$  do
├   ┌    $p.addrKnown \ll p.addrBuf$  // upto 1000 ( $addr, ts$ ) in each message
├   └   send  $p.addrBuf$  to  $p$  and clear  $p.addrBuf$  // could be multiple messages
└

every 24 hours do
┌   for every connected node  $p$  do  $buffer\text{-}to\text{-}send(p, x)$ 
└

```

Algorithm 1: Address Propagation Behavior

```

CandidateSelection( $n_1, \dots, n_N$ ) :
  Partition the  $N$  nodes into  $M = 100$  random sets
   $c_1, \dots, c_M$  of size  $C = \lceil \frac{N}{M} \rceil$  where each
   $c_i = (c_{i,1}, \dots, c_{i,C})$ .
  for  $1 \leq i \leq M$  do
    // Each  $\text{tx}_i$  conflicts with the others
     $\text{tx}_i := \text{MakeTx}(\text{tx}_\emptyset[0])$ 
  for  $1 \leq i \leq M$  do
    for  $1 \leq j \leq N$  do
      InvBlock( $n_j, \text{tx}_i, 20\text{m}$ )
  Wait for time  $\Delta_\gamma$  for INV to settle
  for  $1 \leq i \leq C$  do
    for  $c_{i,j} \in c_i$  do
      send TX[ $\text{tx}_i$ ] to  $c_{i,j}$ 
  Wait until a block is found containing some  $\text{tx}_i$ .
  Increment  $\text{wins}_j$  for every  $n_j \in c_i$ .

```

Algorithm 3: Candidate Selection (CS)

```

InfluenceValidation( $w_1, \dots, w_W, n_1, \dots, n_N$ ) :
  /* ( $w_1, \dots, w_W$ ) are the candidates          */
  for  $1 \leq j \leq W + 1$  do
    // Each  $\text{tx}_j$  conflicts with the others
     $\text{tx}_j := \text{MakeTx}(\text{tx}_\emptyset[0])$ 
  for  $1 \leq i \leq W + 1$  do
    for  $1 \leq j \leq N$  do
      InvBlock( $n_j, \text{tx}_i, 20\text{m}$ )
  Wait for  $\Delta_\gamma$  for INV to settle
  for  $1 \leq i \leq W$  do
    send TX[ $\text{tx}_i$ ] to  $w_i$ 
  for  $1 \leq j \leq N$  do
    send TX[ $\text{tx}_{W+1}$ ] to  $n_j$ 
  Wait until a block is found containing some  $\text{tx}_i$ . If
   $i = W + 1$ , then discard. Otherwise, determine which
  mining pool  $P$  is associated with this block and
  increment  $\text{wins}_{i,P}$ .

```

Algorithm 4: Influence Validation (IV)