Exploring the Monero Peer-to-Peer Network

Tong Cao[1], Jiangshan Yu[2,†,‡], Jérémie Decouchant[1,‡], Xiapu Luo[3], and Paulo Verissimo[1]

[1] SnT, University of Luxembourg, Luxembourg
[2] Monash University, Australia
[3] The Hong Kong Polytechnic University, Hong Kong

Abstract. As of September 2019, Monero is the most capitalized privacy-preserving cryptocurrency, and is ranked tenth among all cryptocurrencies. Monero's on-chain data privacy guarantees, i.e., how mixins are selected in each transaction, have been extensively studied. However, despite Monero's prominence, the network of peers running Monero clients has not been analyzed. Such analysis is of prime importance, since potential vulnerabilities in the peer-to-peer network may lead to attacks on the blockchain's safety (e.g., by isolating a set of nodes) and on users' privacy (e.g., tracing transactions flow in the network).
This paper provides the first step study on understanding Monero's peer-to-peer (P2P) network. In particular, we deconstruct Monero's P2P protocol based on its source code, and develop a toolset to explore Monero's network, which allows us to infer its topology, size, node distribution, and node connectivity. During our experiments, we collected 510 GB of raw data, from which we extracted 21,678 IP addresses of Monero nodes distributed in 970 autonomous systems. We show that Monero's network is highly centralized — 13.2% of the nodes collectively maintain 82.86% of the network connections. We have identified approximately 2,758 active nodes per day, which is 68.7% higher than the number reported by the MoneroHash mining pool. We also identified all concurrent outgoing connections maintained by Monero nodes with very high probability (on average 97.98% for nodes with less than 250 outgoing connections, and 93.79% for nodes with more connections).

1 Introduction

As blockchains aim at implementing decentralized and trustworthy systems, they often rely on peer-to-peer (P2P) protocols for membership management and information dissemination. This makes the P2P network a critical element of blockchains, as the security of the underlying consensus protocols and the privacy of transactions are all tightly related to its implementation [1,2,3,4,5,6,7].

Monero is a privacy-centric cryptocurrency, and is currently ranked the first among privacy-preserving cryptocurrencies with a market capitalization of 1.248

† Corresponding author. E-mail: j.yu.research@gmail.com
‡ These authors contributed equally.

Billion USD, and the 10th among all cryptocurrencies[1]. Much research has been done on analysing the privacy of Monero [8,9,10,11,12], with a focus on on-chain data analysis, i.e., how the mixins (a.k.a. decoy inputs) are selected in each transaction and how they provide privacy guarantees. However, little research has been done to investigate Monero's P2P network, even though network level attacks have been studied on the specific networks of Bitcoin and Ethereum [1,2,3,13,14,15].

Analysing the resilience of a blockchain to network level attacks is challenging, as it requires a deep understanding on the underlying network. In this paper, we present a first step of work towards analysing Monero's security and privacy against network level attacks. In particular, we perform an analysis of Monero's network protocol, and identify possible ways to infer the network topology. We develop a toolset to implement our findings. Our tool set includes *NodeScanner* and *NeighborFinder*. *NodeScanner* automatically discovers peers in the Monero network, no matter whether they are currently online or not. We classify the discovered peers in three categories, namely active and reachable nodes, active and unreachable nodes, and inactive nodes. A node is active if it is currently online, and is reachable if *NodeScanner* can successfully connect to it. Compared to previous works [16,17,18,6], *NeighborFinder* is able to identify the unreachable active nodes in the network, which are the active direct neighbors of nodes that could be reached, for the network topology inference.

Our experimental results show that Monero's network is highly centralized — 0.7% of the active nodes maintain more than 250 outgoing connections, and 86.8% of the nodes do not maintain more than 8 outgoing connections. These 86.8% nodes collectively maintain only 17.14% of the overall connections in the network. Our toolset is also very effective in observing the network – after a single week of data collection, our toolset already discovered 68.7% more active peers than Monerohash [19] – a Monero mining pool that is the only known pool providing data on the Monero node distribution. On average, our toolset identified approximately 2,758 active nodes per day, while Monerohash only showed about 1,635 active nodes. Furthermore, we report our analysis of the collected data regarding an estimation of our network coverage, the network connectivity, and the node distribution in the Monero P2P network.

Our contributions are summarized as follows:

– to the best of our knowledge, our work is the first to describe how to infer Monero's peer-to-peer network, which would enable further studies on the network level security analysis of Monero;
– we provide the first toolset to implement our findings on exploring the Monero peer-to-peer network. In particular, *NodeScanner* explores existing and historical nodes in the Monero network, and *NeighborFinder* identifies neighbors of the Monero nodes. We plan to release our toolset as an open source project shortly; and

[1] https://coinmarketcap.com. Data fetched on Sept. 12, 2019.

– we conduct an experiment to evaluate Monero's network, and show the effectiveness of our methods. We provide insights and a security analysis of its network size, distribution, and connectivity.

Disclosure. We have disclosed our research findings to the Monero team, which has been working on patching the peer-to-peer protocol, and publicly acknowledged our findings in their git commit[2].

The remaining of this paper is organized as follows. Section 2 provides a high level overview on the different designs of P2P protocol in different major cryptocurrencies, and highlights the particularities of Monero's P2P membership protocol. Section 3 provides an overview of our analysis pipeline, and the algorithms we used to implement the discovery of the nodes and the inference of their connections. Section 4 details the results obtained after analyzing the data collected during one week, and provides a discussion on the privacy and security issues that can arise after a P2P network exposure. Section 5 reviews related works, and we conclude this paper in Section 6.

2 Monero's P2P membership protocol

Peer-to-peer (P2P) networks have been designed and extensively studied to allow decentralized message exchanges. They have been getting a renewed attention since Satoshi Nakamoto described Bitcoin in 2008. Indeed, inspired by Bitcoin, thousands of cryptocurrencies serving different purposes have been created. However, no standard P2P protocol has been proposed for blockchains. Instead, different P2P protocols have been designed and adapted by different cryptocurrencies [30,31,33].

Monero relies on its peer-to-peer network to disseminate transactions and blocks. Unfortunately, a proper presentation of Monero's peer-to-peer protocol has been missing from the literature. This section describes Monero's peer-to-peer membership protocol based on its source code, which is available from Monero's official working repository[3].

2.1 Initialization

Monero hardcodes a set of hostnames, which can be translated to IP addresses through the DNS service, and IP addresses of seed nodes that new participants can contact to be bootstrapped into the peer-to-peer network. Those seed nodes are operated by the Monero core team.

New joiners can obtain a limited number of active peers' IP addresses from the seed nodes to initialize their peer lists. They can then start initiating connections with peers, exchange membership lists and discover other peers, until they have established their desired number of connections.

[2] https://github.com/monero-project/monero/blob/960c2158010d30a375207310a36a7a942b9285d2/src/p2p/net_peerlist.h.

[3] Commit hash 14a5c2068f53cfe1af3056375fed2587bc07d320, https://github.com/monero-project/monero.

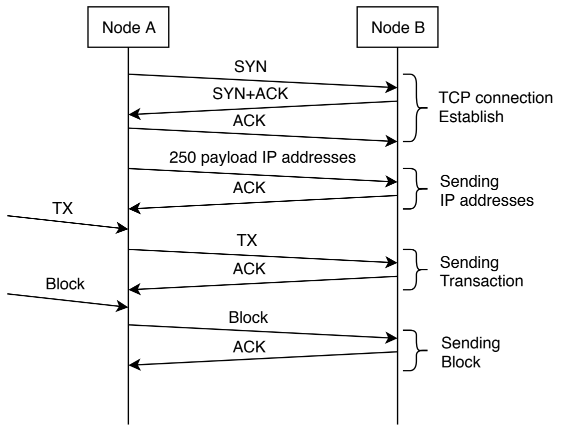


Fig. 1: Message exchange in the Monero network

2.2 Peer list

In Monero, each node maintains a peer list consisting of two parts, i.e., a *white_list*, and a *gray_list*. In the peer list of a peer A, the information of each recorded peer not only contains its identity, its IP address, and the TCP port number it uses, but also a special *last_seen* data field, which is the time at which the peer has interacted with peer A for the last time. All the peers in the lists are ordered chronologically according to their *last_seen* data, i.e., the most recently seen peers are at the top of the list.

Each time a node receives information about a set of peers, this information is inserted into its *gray_list*. Nodes update their *white_list* and *gray_list* through a mechanism called "graylist housekeeping", which periodically pings randomly selected peers in the *gray_list*. If a peer from the *gray_list* is responsive, then its information will be promoted to the *white_list* with an updated *last_seen* field, otherwise it will be removed from the *gray_list*. To handle idle connections, nodes check their connections through the *IDLE_HANDSHAKE* protocol, and update the *last_seen* fields if they successfully connected to the corresponding neighbors, otherwise they drop the associated connection. Nodes also periodically handshake their current connections, and update the *last_seen* field of the associated responsive peers. If a peer does not respond to the handshake request, then the requesting node will disconnect from this neighbor, and connect to a new neighbor chosen from the *white_list*. The disconnected peers will stay in the *white_list*. The maximum sizes of the *white_list*, and of the *gray_list*, are equal to 1,000 and 5,000, respectively. If the number of peers in these lists grow over the maximum allowed size, then the peers with the oldest *last_seen* fields will be removed from the list.

Nodes broadcast messages (e.g., transactions and blocks) to their neighbors through TCP connections. Nodes choose their neighbors from the *white_list*. If not enough peers from the *white_list* are currently online, then a node will choose its neighbors from the *gray_list*. Nodes to which previous connections were established are classified as anchor nodes, and stored in the *white_list*. Monero ensures that every node is connected to at least two anchor nodes to prevent a node from being isolated by an attacker. To discover other participants, nodes exchange membership messages by sending a TCP *SYN* message to their neighbors. Upon receiving a *SYN* message, the neighbors create a message whose payload contains detailed information of its top 250 peers in the *white_list*, and send it back to the requester. The requester inserts the received peer data into its *gray_list*, and runs the graylist housekeeping protocol to update the lists. More details about the TCP connection and data transmission will be presented in Section 2.3.

2.3 Information propagation

By default, each peer maintains 8 outgoing connections and accepts 1 incoming connection. A peer residing behind a firewall or a NAT does not accept incoming connections, and only maintains 8 outgoing connections. Peers are allowed to define their maximum number of outgoing and incoming connections. Monero recommends peers to increase the number of their connections according to their capacity, for an improved network connectivity.

Three types of messages are propagated in Monero, respectively containing peers information, transactions, and blocks. A node establishes connections with others through a TCP handshake (SYN-SYN-ACK) as illustrated in Figure 1, and can subsequently exchange peer information through the established connection.

3 Analysis pipeline overview

In this section, we introduce the different data structures and the processes we implemented, along with the associated network tools they rely on. We also detail our algorithmic approaches to monitor the active Monero nodes and to infer their neighbors. The analysis pipeline is illustrated in Figure 2.

3.1 Construction

We deploy full Monero nodes to collect data in the Monero network. These nodes establish connections with peers in the network, and store packets into their local storage. We adapt two network measurement tools, i.e., *tcpflow*[4] and *nmap*[5], to collect data and analyze the Monero network. As mentioned in Section 2, each received TCP packet contains the most recent 250 IP addresses of the sender's *white_list*. Thus, all received IP addresses are recent out-bound peers of the sender. We then use our first tool, *NodeScanner*, to collect the IP addresses of

[4] https://www.tecmint.com/tcpflow-analyze-debug-network-traffic-in-linux/.
[5] https://nmap.org/.

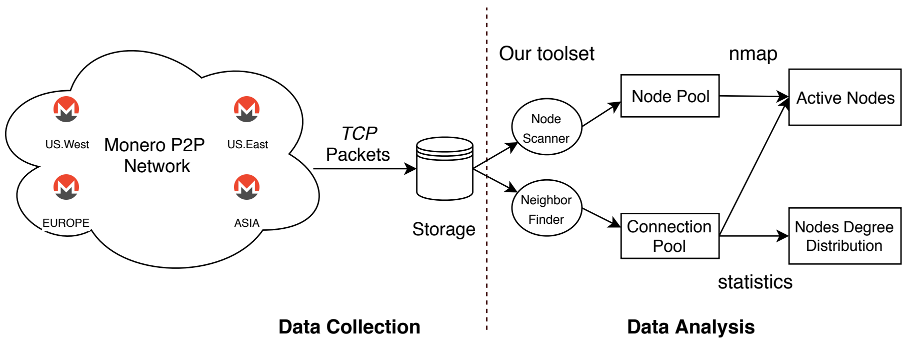


Fig. 2: Analysis pipeline overview

discovered Monero nodes and store them in the *NodePool*. We use our second tool *NeighborFinder* to infer the neighbors of reached nodes that sent the TCP packets to collectors, and store them in the *ConnectionPool*. Each connection consists of a node we reached and of its neighbor, which are both active. We introduce in greater details our developed tools in Section 3.3.

3.2 Neighbor inference based on membership messages

As introduced in Section 2.2, Monero clients execute a gray list housekeeping protocol and an idle connections prevention protocol to evict inactive nodes from their peer list. As a consequence, the outbound neighbors of a node are often associated with the freshest *last_seen* in its peer list, which enables the identification of a node's neighbors from the membership messages it sends.

3.3 Nodes discovery and connections inference

Our deployed nodes accept incoming connections and initiate outgoing connections to receive TCP packets from other nodes. Let $P = \{P_1, P_2, P_3, ..., P_j\}$ be the set of j TCP packets a collector receives from a reached node, such that each packet P_k ($k \in [1, j]$) contains a set $A_k = \{A_{k,1}, A_{k,2}, A_{k,3}, ..., A_{k,250}\}$ of *IP* addresses and a set $T_k = \{T_{k,1}, T_{k,2}, T_{k,3}, ..., T_{k,250}\}$ of *last_seen* timestamps.

NodeScanner. After having received a set P of packets from node $\mathcal{N}$, *NodeScanner* identifies the set $A = \{A_1, A_2, A_3, ..., A_j\}$ of included IP addresses, extracts the set $U = A_1 \cup A_2 \cup A_3 \cup ... \cup A_j$ of unique IP addresses from A, and inserts all unique IP addresses into the *NodePool*.

NeighborFinder. Our second tool aims at identifying a set N_k of neighbors from each P_k ($k \in [1, j]$). Over the various packets P_1 to P_j, it identifies the overall set of neighbors $N = N_1 \cup N_2 \cup N_3 \cup ... \cup N_j$. In the following, we first indicate our neighbors inference approach based on the time difference of the nodes' *last_seen* timestamps in a single packet, and then refine this approach by relying on several received packets.

Neighbors inference based on a single packet. For any received packet P_k from a node $\mathcal{N}$, we assume that it contains $r < 250$ neighbors. Because all neighbors of $\mathcal{N}$ are updated at the same time, the neighbors of $\mathcal{N}$ tend to be the first r adjacent *IP* addresses of A_k, and the difference between any two neighbors' timestamps tends to be small. If we assume that there is a maximum time difference μ[6] between the timestamps of any two neighbors, then we can extract a set $N'_k = \{A_{k,i}, A_{k,i+1}, A_{k,i+2}, ..., A_{k,i+r-1} | \ r \in [1, 250], \ i \in [1, 251 - r], \ \forall x \in [i, i + r - 1], T_{k,x} - T_{k,x+1} \leq \mu\}$ of neighbors from P_k as shown in Algorithm 1.

Algorithm 1: Neighbors inference based on a single received packet

```
  Input  : $P_k$: Packets;
           $\mu$: The maximum time difference between the last_seen timestamps of
  a node's neighbors;
           $A_k$: the IP addresses of $P_k$;
           $T_k$: the last_seen timestamps of $P_k$
  Output: Neighbors set $N'_k$;
1 for $(y = 1, y < 250, y{+}{+})$ do
2     if $T_{k,y}$ - $T_{k,(y+1)} \leq \mu$ then
3         $N'_k \longleftarrow A_{k,y}$
4     end
5     if $T_{k,(y+1)}$ - $T_{k,(y+2)} > \mu$ then
6         $N'_k \longleftarrow A_{k,(y+1)}$; break
7     end
8 end
```

Each node iteratively checks its connections through the *IDLE_HANDSHAKE* procedure, which makes a node send SYN packets to all of its neighbors. Following this procedure, the *last_seen* timestamps of handshaked neighbors are updated with the current time if nodes can be contacted, otherwise connections are dropped. This mechanism prevents idle connections to be maintained. However, the answers to the SYN packet can be received at a different time, which leads to different answer delays. It is therefore necessary to set μ to a value that is large enough to discover all neighbors, but small enough to limit false positives. This problem only exists when we rely on a single packet to infer the neighbors of a target node, and disappears when multiple packets are used.

Improved neighbors inference based on multiple packets. During a connection with a node, it frequently happens that our monitoring nodes successively receive multiple packets from a node. If an IP address appears in successive packets, and its *last_seen* has been updated, then we can conclude that the node corresponding to this IP address is a neighbor of the sender. We use the set $N'' = \{A_{k,y} \mid , A_{k,y} = A_{(k+1),z}, T_{k,y} \neq T_{(k+1),z}, A_{k,y} \in P_k, T_{k,y} \in P_k, A_{(k+1),z} \in P_{(k+1)}, T_{(k+1),z} \in P_{(k+1)}\}$ to denote the IP addresses that have been

[6] We set μ to the value of the *IDLE_HANDSHAKE* interval, i.e., 60 seconds.

Algorithm 2: Neighbors inference based on two received packets

```
Input  : Packets $P_k$ and $P_{(k+1)}$;
         $A_k$, $A_{(k+1)}$: the *IP* addresses in $P_k$, resp. $P_{k+1}$;
         $T_k$, $T_{(k+1)}$: the *last_seen* timestamps in $P_k$, resp. $P_{k+1}$;
Output: Neighbors set $N''_k$;
1 foreach $A_{k,y} = A_{(k+1),z}$ do
2     if $T_{k,y} \neq T_{(k+1),z}$ then
3         $N''_k \longleftarrow A_{k,y}$
4     end
5 end
```

updated between packets P_k and $P_{(k+1)}$. We then extract the neighbors of node $\mathcal{N}$ following Algorithm 2.

4 Experiments

This section describes our experimental settings, validation approach, data analysis methods and results. We also discuss the potential threats of a network topology exposure.

4.1 Settings

We deployed four full nodes in the Monero network: two in the U.S. (California and Virginia), one in Europe (Luxembourg), and one in Asia (Japan). Each node ran on an Ubuntu 16.04 machine with an Intel Xeon Platinum 8000 series processor. We make use of the four nodes not only to collect data, but also to have access to a ground truth and verify our neighbor inference algorithms.

We manually modified the settings on our Monero nodes so that they could establish the largest number of connections with other nodes. First, we set the maximum number of incoming and outgoing connections to 99,999 to force our nodes to actively search for new neighbors. Second, we modified the number of opened files, socket receive buffer, and socket send buffer of used machines to the maximum number (1,048,576, 33,554,432, 33,554,432 respectively) in order to simultaneously maintain a large amount of TCP connections.

We collected 510 GB of raw data containing 12,563,962 peer list messages (as shown in Table 1). We extracted 21,678 IP addresses, which belong to 970 ASs[7]. Out of these collected IP addresses, our nodes established connections with 3,626 peers, and identified 703 peers to which no connection could not be established, but that were active and connected to reached nodes. We say that peers are active and reachable if our nodes can establish connections with them. We say peers are active but unreachable if they are connected to nodes we connected to

[7] We use the whois (https://www.ultratools.com/tools/ipWhoisLookup) database to find the ASN for each IP address.

Table 1: Data collected from Tokyo (T), Luxembourg (L), California (C), and Virginia (V)

#Received Peer List Messages	**Node**		**Connection**	
T: 1,971,514; L: 2,308,968	**IP Addresses**	**ASN**	**Host Level**	**AS Level**
C: 3,892,225; V: 4,391,255	21,678	970	338,023	87,013

and if a connection could not be established with them. We say a peer is inactive if it is neither connected to our nodes, nor connected to responsive peers. If our nodes were not able to connect to a peer, then it either meant that the peer was already fully connected during the data collection, or that it was offline. To reduce the number of possible false negatives, we consider that a peer is offline if the peer is not connected to our nodes or to the neighbors of our nodes, and if their *last_seen* has not been updated during the data collection process.

4.2 Validation

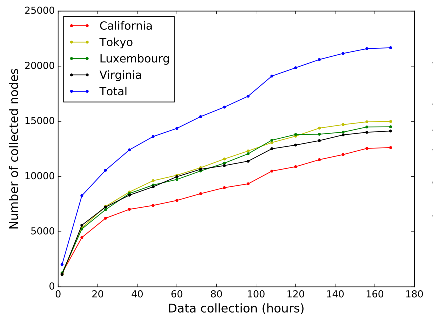
We used the node in Luxembourg to establish three connections with the nodes in California, Virgina, and Tokyo respectively. We compared the identities of the nodes identified by *NeighborFinder* as neighbors with the ground truth of our deployed nodes. Since the payload data of membership messages can contain at most 250 IP addresses, a part of a node's neighbors could not be observed in a single message when it maintained more than 250 outgoing connections. Therefore, we specifically set up a node maintaining more than 250 outgoing neighbors in Tokyo to verify our algorithms. The validation reported a precision of 100% with 97.98% recall (i.e., all inferred neighbors were real neighbors, and 2.12% of the nodes identified as Non-neighbors were false negatives) when the number of neighbors is smaller than 250, and a precision of 100% with 93.79% recall for the node in Tokyo.

4.3 Measuring the network coverage

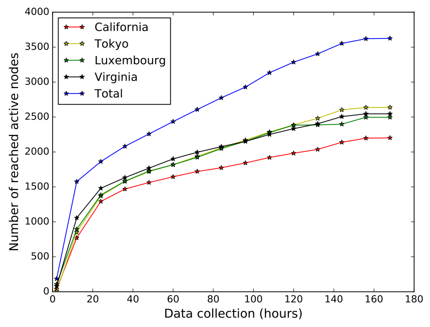
Previous tools [20,21,22,23] relied on the number of reached nodes to estimate their network coverage in Bitcoin and Ethereum. However, unreachable active nodes, which are also a part of the nework, have been overlooked by these tools. In this section, we introduce our method, which takes unreachable nodes into account, to estimate the network coverage. We show the effectiveness of our tools by comparing our results with the data provided by the MoneroHash mining pool [19].

NeighborFinder determined the neighbors of reached nodes even when it was not possible to contact them. This allowed us to:

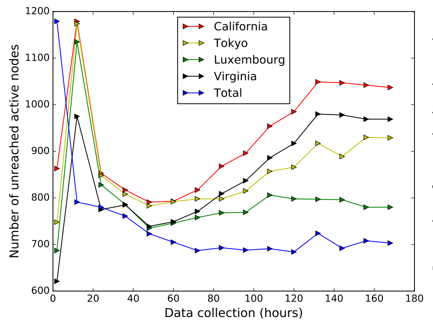
– **identify the fully connected nodes**. When a node has reached its maximum number of incoming connections, it does not accept any new inbound neighbor. In this case, previous approaches cannot identify these fully connected active nodes. However, *NeighborFinder* can discover them through the connections they have established with reached nodes.



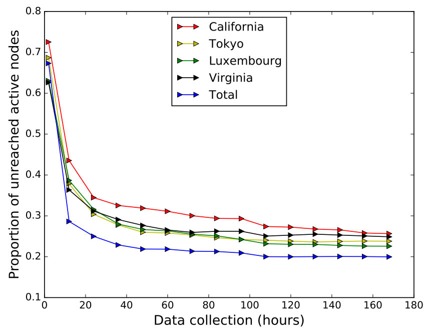
(a) Collected nodes.



(b) Reached active nodes.



(c) Unreached active nodes.



(d) Proportion of unreached active nodes.

Fig. 3: Analysis of the collected IP addresses during the data collection process.

- **estimate the network size by observing the proportion of unreached active nodes**. Unfortunately, there is no ground truth to validate the network size in permissionless blockchains. We use $\frac{num.\ unreached\ active\ nodes}{num.\ collected\ nodes} \in [0, 1]$ as a metric to estimate the proportion of the Monero network that has been reached. In practice, our tools have discovered almost all long-term running nodes in the network when the new reached nodes cannot present information about any new nodes. The overall proportion of unreached active nodes is illustrated in Figure 3(d).

We present the data collection statistics in Figure 3, where we respectively show the data collected by the node in California in red, Virginia in black, Japan in yellow, and Luxembourg in green. The total number of reached nodes is represented in blue. Figure 3(a) shows the number of discovered peers. Figure 3(b) shows the number of active nodes connected to our servers. Figure 3(c) shows the number of active but unreachable peers. Figure 3(d) shows the evolution of proportion of unreached active peers. After the first 80 hours, the proportion of unreached active nodes are stabilizing, which means that our toolset has detected almost all the long-term running active nodes. Thus, it is likely that the Monero network contains around 2,758 active nodes per day as shown in Figure 4. Compared with Monerohash [19], which discovered 1,635 active nodes in average per day, the number of active nodes we discovered is 68.7% higher than the number reported by the MoneroHash mining pool. To the best of our knowledge, Monerohash is the only Monero mining pool providing information related to the number of active nodes in the network. Moreover, the number of daily active nodes in Bitcoin [20] and Ethereum [21] is estimated to be close to 10,000. It is not a surprise to see that Monero has far less daily active nodes than those two more largely used cryptocurrencies.

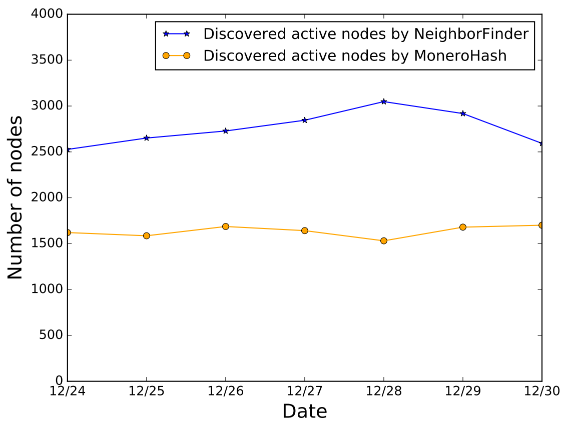


Fig. 4: Active nodes discovered daily by *NeighborFinder* and MoneroHash.

4.4 Node distribution

In a cryptocurrency P2P overlay, different nodes play different roles and exhibit various connectivities in a real world implementation. It is essential to analyze how nodes are connected and located in the network to measure the resilience of the blockchain systems to network level attacks, which are surveyed in Section 5. In this section, we present the experiment results regarding to peer freshness, connectivity, and node distributions alongside with their implications.

Peer freshness. Our approach shows that only about 20% (i.e., 4,329) of the discovered nodes were active, and the remaining nodes were offline during the data collection period. This indicates that a majority of the exchanged IP addresses are inactive in Monero's network, and might decrease the network connectivity.

Connectivity. We say that a node is of degree N if it maintains at most N outgoing connections. We classify active nodes into three categories based on their degree: *light node (degree≤8)*, *medium node (8<degree≤* 250*)*, and *heavy node (degree>250)*. As shown in Table 2, most of the nodes (86.8%) collectively

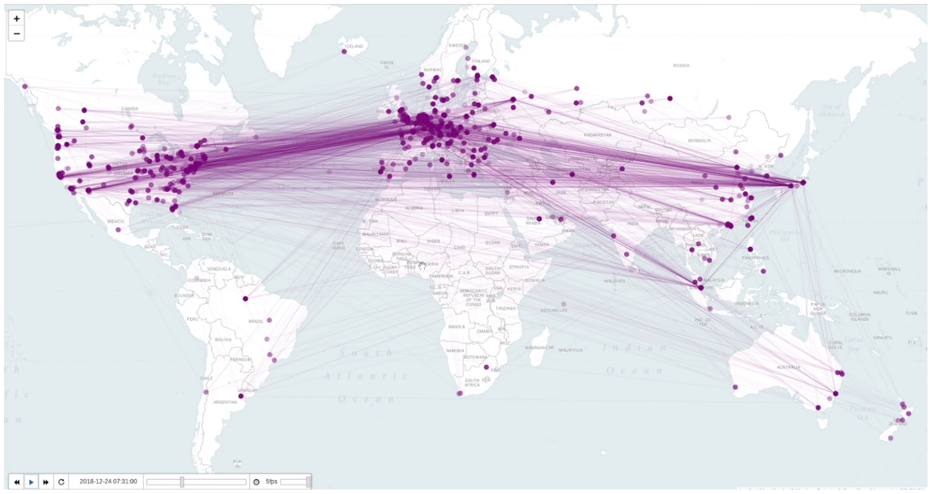


Fig. 5: Snapshot of the Monero network obtained after one hour. Each dot represents a Monero node, whose darkness is proportional to the number of connections it maintains. The lightness of lines denotes their uptimes.

maintain only 17.14% of the connections, while the remaining 13.2% of the nodes maintain 82.86% of the connections. On the other hand, Monero has hardcoded 8 seed nodes in the system, and we initially suspected that all of them would be heavy nodes. Our experiments showed that only 3 of the seed nodes were active, and that two of them were heavy nodes, while another one was a medium node. Later on, we contacted the Monero team for clarification, and they confirmed that 5 seed nodes were not available[8].By comparing the discovered heavy nodes with public Monero mining pools[9] and seed nodes[10], we found that 9 heavy nodes are maintained by mining pools, and that 2 heavy nodes are Monero seed nodes. Due to the lack of public information, we could not identify the other 17 heavy nodes. However, we assume that the remaining unidentified heavy nodes are likely to be the front-end nodes of private mining pools.

Table 2: Number of active nodes in the ConnectionPool.

	Light nodes	**Medium nodes**	**Heavy nodes**	**Total**
Reached	3146 (86.8%)	452 (12.5%)	28 (0.7%)	3626
Unreached	-	-	-	703
Total	3146	452	28	4329

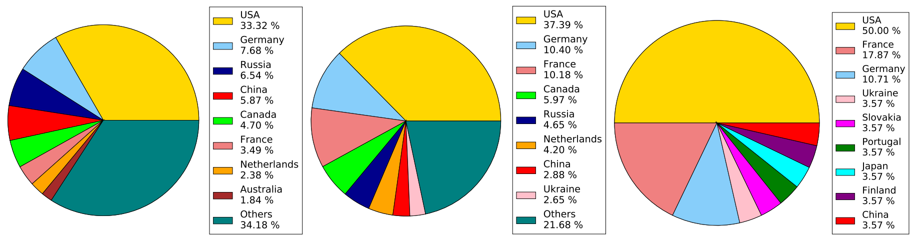
[8] https://github.com/monero-project/monero/issues/5314.

[9] http://moneropools.com/.

[10] https://github.com/monero-project/monero/blob/577a8f5c8431d385bf9d11c30b5e3e8855c16cca/src/p2p/net_node.inl.

Snapshot of the Monero network topology. We collected snapshots of the network topology thanks to the *ConnectionPool*, which continuously records the connections' updates. Those snapshots provide useful information concerning the network structure. We represent a one hour snapshot of the Monero network topology observed on 12/24/2018 in Figure 5. It is obvious that an user's IP address is exposed along with its connections. This leaves a chance for the adversary to identify different roles (miner or client) in the network depending on their connectivity. On the other hand, we hypothesize that the vast inequality of node connectivity (In our experiment, the heaviest node could maintain more than 1000 connections, the lightest node just maintain 8 connections) might lead to network vulnerabilities [24], where the high degree node could significantly affect low degree node to select neighbors.

Geographic distribution. We present in Figure 6 the location of the Monero nodes depending on their classification. Approximately 50% of the heavy nodes, which are likely the mining pools, are located in the US, while the light nodes, which are likely clients, are more evenly distributed around the world.



(a) Light nodes. (b) Medium nodes. (c) Heavy nodes.

Fig. 6: Nodes location distribution

Degree distribution. Monero's peer-to-peer network is unstructured, permissionless and very dynamic. In particular, a node is allowed to change its neighbors as we analyzed in Section 2. To further analyze how nodes are connected over time, we counted the numbers of neighbors of active nodes during one week, and plot their distribution in Figure 7. The blue dots represent the distribution of outgoing neighbors of the nodes. The results indicate that a small fraction of the nodes have more than 1000 outgoing neighbors, while a large fraction of nodes have less than 100 outgoing neighbors. The red dots represent the distributions of both incoming and outgoing neighbors. Comparing with the blue dots, one can see that the node with a large number of outgoing neighbors are likely to maintain a large number of incoming neighbors as well. More importantly, the small jumps in both blue and red dots indicate that a number of nodes have not kept the number of connections fixed by default in order to gain a better connectivity. We point out that this is an unique feature of Monero, which implies a high network dynamism.

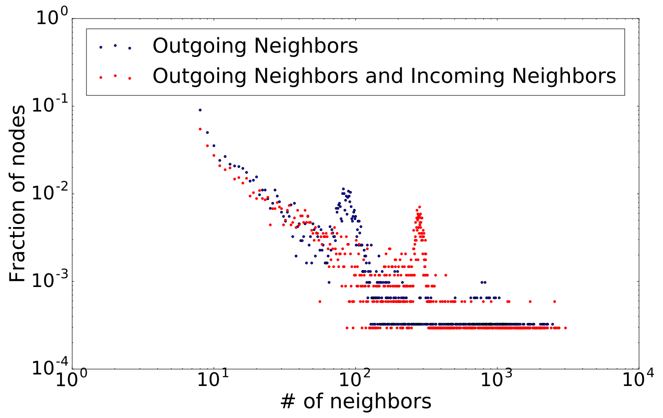


Fig. 7: Number of outgoing neighbors of heavy, medium, and light nodes.

4.5 Potential threats

Using our tools, one can identify Monero's network topology and the connectivity of nodes. An example is shown in Figure 8, which illustrates the neighbors of a *light node (5.X.X.X)*[11] during the 9-hour monitoring process. Each color represents a neighbor of the node. It shows that neighbor 1-6 stayed connected with the node for the entire 9 hours, whereas the connection with neighbor 7 is dropped around the 8th hour, and a connection with neighbor 11 was established to replace neighbor 7. Similarly, a connection with neighbor 9 was established to replace neighbor 8 after 3 hours.

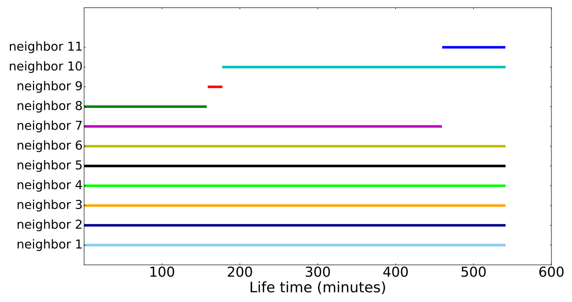


Fig. 8: Dynamic neighbor tracking of a light node in 9 hours.

With such knowledge, an attacker can potentially launch different types of attacks. For example, an attacker could launch a targeted attack by monopolizing all connections of a victim node [1], selectively partition the network [3], or

[11] Hidden IP address to protect the privacy of this light node.

even deanonymize transactions by identifying the first node relaying a transaction [5,6].

5 Related work

Previous works studied the network information of leading cryptocurrencies, e.g., Bitcoin and Ethereum. Decker and Wattenhofer [16,25] measured the rate of information propagation between reached nodes in Bitcoin. Relying on the received messages from reached nodes, interconnections of reached nodes were inferred in Bitcoin [26,18] to evaluate network properties. To infer whether two reached nodes are connected in Bitcoin, Grundmann et al. [27] suggested to use double spent transactions as probing messages, and S. D. Segura et al. [28] suggested to use orphan transactions. Kim et al. [23] deployed 30 nodes on one machine to collect network messages to measure the Ethereum network. However, the node interconnections are difficult to infer in Ethereum, and unreachable nodes cannot be observed.

Network level attacks have been studied in Bitcoin and Ethereum. Routing attacks [3,15] are facilitated by the fact that Bitcoin's protocol makes nodes exchange messages in plain text during the peer-to-peer communications. This allows an adversary to partition the network, and delay the dissemination of messages among nodes. Eclipse attacks, in Bitcoin [1,13] and in Ethereum [2], pointed out that unsolicited incoming connections can be leveraged by an adversary to continuously send large amount of fake packets to a given node, and fill the table of its stored IP addresses, forcing it to restart. These attacks demonstrated that an attacker can monopolize all connections of a targeted node with high probability. Deanonymization attacks [29,5,6] have been introduced to track transactions and discover the generator's IP address. These attacks aim at linking the IP address of a node with the transactions it created, with the requirement of monitoring interconnections. Such attacks require, or are facilitated, by an understanding of the peer-to-peer overlay and topology.

6 Conclusion

In this work, we presented methods we developed to observe Monero's peer-to-peer network, and infer its topology. We described how one can deploy Monero nodes to discover all the nodes participating in the protocol, and their interconnections, using the *last_seen* timestamps in the peer lists that nodes exchange. For accuracy, we compared our methods' results with the ground truth of our deployed nodes. Our experiments show that even though Monero is a privacy-preserving cryptocurrency, it is still possible to accurately discover the nodes in the network and their interconnections. Our analysis provides insights about Monero's degree of centralization, and about the privacy and security issues potentially caused by a network topology exposure. As future work, we will conduct a deeper network-based security and privacy analysis of Monero, based on the tools provided in this paper.

Acknowledgments. This work is partially supported by the Fonds National de la Recherche Luxembourg (FNR) through PEARL grant FNR/P14/8149128.

References

1. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: USENIX Security Symposium. (2015)
2. Marcus, Y., Heilman, E., Goldberg, S.: Low-resource eclipse attacks on ethereum's peer-to-peer network. IACR Cryptology ePrint Archive **2018** (2018) 236
3. Apostolaki, M., Zohar, A., Vanbever, L.: Hijacking bitcoin: Routing attacks on cryptocurrencies. In: IEEE Security and Privacy (SP). (2017)
4. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: 2015 IEEE Symposium on Security and Privacy (SP). (May 2015)
5. Biryukov, A., Khovratovich, D., Pustogarov, I.: Deanonymisation of clients in bitcoin P2P network. In: ACM CCS. (2014)
6. Biryukov, A., Tikhomirov, S.: Deanonymization and linkability of cryptocurrency transactions based on network analysis. In: EuroS&P. (2019)
7. Natoli, C., Yu, J., Gramoli, V., Veríssimo, P.J.E.: Deconstructing blockchains: A comprehensive survey on consensus, membership and structure. CoRR (2019)
8. Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., Hogan, K., Hennessey, J., Miller, A., Narayanan, A., Christin, N.: An empirical analysis of traceability in the monero blockchain. PoPETs **2018**(3) (2018) 143–163
9. Kumar, A., Fischer, C., Tople, S., Saxena, P.: A traceability analysis of monero's blockchain. In: ESORICS. (2017) 153–173
10. Yu, Z., Au, M.H., Yu, J., Yang, R., Xu, Q., , Lau, W.F.: New empirical traceability analysis of cryptonote-style blockchains. In: FC. (2019)
11. Yu, J., Au, M.H.A., Veríssimo, P.: Re-thinking untraceability in the cryptonote-style blockchain. In: IEEE Computer Security Foundations Symposium (CSF). (2019)
12. Wijaya, D.A., Liu, J., Steinfeld, R., Liu, D., Yu, J.: On the unforkability of monero. In: ACM Asia Conference on Information, Computer and Communications Security (ASIACCS). (2019)
13. Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. (2016)
14. Natoli, C., Gramoli, V.: The balance attack or why forkable blockchains are ill-suited for consortium. In: DSN. (2017)
15. Ekparinya, P., Gramoli, V., Jourjon, G.: Impact of man-in-the-middle attacks on ethereum. In: SRDS. (2018)
16. Decker, C., Wattenhofer, R.: Information propagation in the bitcoin network. In: IEEE P2P. (2013)
17. Koshy, D.: An Analysis of Anonymity in Bitcoin Using P2P Network Traffic. Pennsylvania State University (2013)
18. Neudecker, T., Andelfinger, P., Hartenstein, H.: Timing analysis for inferring the topology of the bitcoin peer-to-peer network. In: IEEE UIC. (2016)
19. MoneroHash: Monerohash - monero mining pool. https://monerohash.com/nodes-distribution.html (2018) Accessed: from 2018-12-23 to 2018-12-30.
20. Bitnodes: Bitnodes. https://bitnodes.earn.com/nodes/ (2018) Accessed: from 2018-12-23 to 2018-12-30.

21. Ethernodes: Ethernodes. https://www.ethernodes.org/network/1 (2018) Accessed: from 2018-12-23 to 2018-12-30.
22. Neudecker, T.: Characterization of the bitcoin peer-to-peer network (2015-2018). http://dsn.tm.kit.edu/bitcoin/publications/bitcoin_network_characterization.pdf (2019)
23. Kim, S.K., Ma, Z., Murali, S., Mason, J., Miller, A., Bailey, M.: Measuring ethereum network peers. In: ACM IMC. (2018)
24. Singh, A., Castro, M., Druschel, P., Rowstron, A.: Defending against eclipse attacks on overlay networks. In: Proceedings of the 11th Workshop on ACM SIGOPS European Workshop. EW 11 (2004)
25. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A.E., Miller, A., Saxena, P., Shi, E., Sirer, E.G., Song, D., Wattenhofer, R.: On scaling decentralized blockchains - (A position paper). In: Financial Cryptography and Data Security - FC. (2016)
26. Miller, A., Litton, J., Pachulski, A., Gupta, N.S., Levin, D., Spring, N., Bhattacharjee, B.: Discovering bitcoin's public topology and influential nodes. (2015)
27. Grundmann, M., Neudecker, T., Hartenstein, H.: Exploiting transaction accumulation and double spends for topology inference in bitcoin. In: FC. (2018) 113–126
28. Delgado-Segura, S., Bakshi, S., Pérez-Solà, C., Litton, J., Pachulski, A., Miller, A., Bhattacharjee, B.: Txprobe: Discovering bitcoin's network topology using orphan transactions. CoRR (2018)
29. Fanti, G., Viswanath, P.: Deanonymization in the bitcoin p2p network. In: NIPS. (2017)
30. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. (2008)
31. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151** (2014) 1–32
32. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: IPTPS. Volume 2429 of Lecture Notes in Computer Science., Springer (2002) 53–65
33. Van Saberhagen, N.: Cryptonote v 2.0 (2013)