Exploring the Monero Peer-to-Peer Network

Tong Cao*, Jiangshan Yu†, Jérémie Decouchant*, Xiapu Luo‡ and Paulo Verissimo*
*SnT - University of Luxembourg, †Monash University, ‡The Hong Kong Polytechnic University
Email: *firstname.lastname@uni.lu, †jiangshan.yu@monash.edu, ‡csxluo@comp.polyu.edu.hk

***Abstract*—As of 12th January 2019, Monero is ranked as the first privacy-preserving cryptocurrency by market capitalization, and the 14th among all cryptocurrencies.**

This paper aims at improving the understanding of the Monero peer-to-peer network. We develop a tool set to explore the Monero peer-to-peer network, including its network size, distribution, and connectivity. In addition, we set up four Monero nodes that run our tools: two in the U.S., one in Asia, and one in Europe. We show that through a short time (one week) data collection, our tool is already able to discover 68.7% more daily active peers (in average) compared to a Monero mining pool, called Monerohash, that also provides data on the Monero node distribution.

Overall, we discovered 21,678 peer IP addresses in the Monero network. Our results show that only 4,329 (about 20%) peers are active. We also show that the nodes in the Monero network follow the power law distribution concerning the number of connections they maintain. In particular, only 0.7% of the active maintain more than 250 outgoing connections simultaneously, and a large proportion (86.8%) of the nodes maintain less than 8 outgoing connections. These 86.8% nodes only maintain 17.14% of the overall connections in the network, whereas the remaining 13.2% nodes maintain 82.86% of the overall connections. We also show that our tool is able to dynamically infer the complete outgoing connections of 99.3% nodes in the network, and infer 250 outgoing connections of the remaining 0.7% nodes.

I. Introduction

Monero is one of the most popular privacy-preserving cryptocurrencies, with a market capitalization of 761 Million USD. It is ranked the first among privacy-preserving cryptocurrencies [1], [2] by market capitalization, and the 14th among all cryptocurrencies[1]. Monero proposes to protect the sensitive transactions from being traced or linked. While many studies [3]–[5] have been conducted to understand the privacy guarantees of Monero through analyzing the data recorded in the blockchain, little has been done to study the network level security and privacy. Like Bitcoin, Monero relies on a peer-to-peer membership communication system. Understanding Monero's network therefore a first step towards network level security analysis.

This work provides the first tool set, with systematic investigation, to help understand Monero's network. More precisely, we develop a tool set that allows us to explore the Monero peer-to-peer network, including its network size, distribution, and connectivity.

The tool set includes *NodeScanner* and *NeighborFinder*. The former automatically discovers peers in the Monero network, no matter whether they are online or not. The discovered peers are categorised in three types, 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 the host running *NodeScanner* can establish a successful connection with the node. We use *NodeScanner* to find IP addresses from reached nodes, and use *NeighborFinder* to infer links between different nodes. This helps us understanding the connectivity of active nodes in the system. To deploy our tool set, we set up four nodes in the Monero network: two in the US, one in Asia, and one in Europe. During one week of data collection, we collected 21,678 IP addresses participating in the Monero network, where only 4,329 peers (about 20%) are active and the rest are inactive. For the active peers, we are able to connect to 3,626 peers (about 83.8%), and the rest could not be connected to during the week as all of their outgoing connections were occupied. Even though our data collection process only takes a single week, we discovered 68.7% more active peers than Monerohash [6] – a Monero mining pool that is the only known pool providing data on the Monero node distribution. In average, our tool shows that there are around 2,758 active nodes per day, while Monerohash only shows about 1,635 active nodes. This shows the effectiveness of our approach.

Based on the collected data, we show that the Monero network connections follow the power law. We found out that only 0.7% active nodes maintain more than 250 outgoing connections simultaneously, and a large proportion (86.8%) of nodes maintain less than 8 outgoing connections. Also, these 86.8% nodes collectively maintain only 17.14% of the overall connections in the network, whereas the remaining 13.2% nodes maintain the remaining 82.86% connections.

We also present a snapshot of the Monero network topology. We show that our tool is able to dynamically infer the complete outgoing connections of 99.3% nodes in the network, and infer 250 outgoing connections of the remaining 0.7% nodes. To verify the accuracy of connectivity results of Monero nodes, we use one of our nodes to run *NeighborFinder*, and use it to infer the connections of our other three nodes. Our results indicate that *NeighborFinder* is able to discover all the neighbors of the nodes accurately. Inferring the connections of Monero nodes may allow an attacker to launch network level attacks, such as the eclipse attack [7], [8]. However, a detailed network-level security analysis of Monero is future work.

Our contributions can be summarized as follows:

- To the best of our knowledge, our work is the first to describe how to infer the Monero peer-to-peer network, which would enable further studies on the network level security analysis of Monero.

[1]https://coinmarketcap.com. Data fetched on 11.Jan.2019.

- We provide the first tool set to explore the Monero peer-to-peer network, including its network size, distribution, and connectivity. In particular, our *NodeScanner* explores existing and historical nodes in the Monero network; and our *NeighborFinder* identifies neighbors of the Monero nodes. We plan to release our toolset as an open source project shortly.
- We conduct an experiment to evaluate Monero network, and show the effectiveness of our methods. We also identified highly influence nodes in the system, and analyzed their distributions.

The remaining of this paper is structured as follows. We show how the Monero peer-to-peer membership protocol operates in Section II. Section III provides an overview of our analysis pipeline. Section IV details the algorithms we used to implement the discovery of the nodes and the inference of their connections. Section V details the results obtained after analyzing the data collected during one week. We discuss the limitations of our approach in Section VI. Finally, we review related works in Section VII, and conclude the paper in Section VIII.

II. Monero peer-to-peer membership protocol

Monero relies on its peer-to-peer network to disseminate messages such as transactions and blocks. Unfortunately, a proper presentation on how exactly the peer-to-peer network operates is missing in the literature. This section describes Monero's peer-to-peer membership protocol based on the source code, available from its official working repository[2], with commit hash 14a5c2068f53cfe1af3056375fed2587bc07d320.

A. Initialisation

Monero hardcodes a set of hostnames, which can be translated to IP addresses through DNS, 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 request IP addresses of active peers from the seed nodes to initialize their peer lists. They can then start initiating connections with peers, thereby discovering new peers thanks to membership list exchanges, until they have established their desired number of connections.

B. Peer List

In Monero, each node maintains a peer list consisting of three parts, i.e., an $anchor\_list$, a $white\_list$, and a $gray\_list$. In the peer list, the information of each recorded peer contains not only the peer's identity, its ip address, and TCP port number, but also a special $last\_seen$ data, which is the unix time at which the peer has been seen 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.

[2]https://github.com/monero-project/monero

The $anchor\_list$ is a special list, in the sense that it records seed nodes the local node connected to during its initialization, and a subset of the peers later connected to. Monero requires that each $anchor\_list$ must contain at least two seed nodes. These seed nodes can later be used during the entire life time of the node to establish trustworthy connections. Each time a node receives information about a set of peers, this information is merged into the $gray\_list$. 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 online, then this peer will be promoted to the $white\_list$ with an updated $last\_seen$ field, otherwise it will be removed from the list. To handle idle connections, nodes check their connections through "IDLE_HANDSHAKE", and update the $last\_seen$ field if they managed to successfully connect, otherwise they drop the associated connection. Also, nodes periodically handshake their current connections, and update the $last\_seen$ field of the associated peers if they respond. If a peer does not respond to a handshake message, then the node will disconnect with the neighbor peer, and seek new neighbor 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 respectively 1,000, and 5,000. 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 $anchor\_list$. If there are not enough currently online peers from the $anchor\_list$, then the neighbors will be chosen from the $white\_list$ and then the $gray\_list$. Nodes previously connected to are classified as anchor nodes. In this way, Monero ensures that every node is connected to at least two anchor nodes to prevent a node from being isolated by an attacker. To further establish new connections, nodes randomly select a number of peers from their $white\_list$, and send a TCP *SYN* message to the selected peers. Upon receiving a *SYN* message, the peer creates a message where the payload contains the information of its top 250 peers in the $white\_list$, and sends it back to the requester. The requester then combines the received peer data into its $gray\_list$, and run the graylist housekeeping protocol to update the lists. More details about the TCP connection and data transmission will be presented in Section II-D.

C. Number of Connections

With the default settings, each peer maintains 8 outgoing connections and accepts 1 incoming connection. If a peer sits behind a firewall or a NAT, then it does not accept incoming connections, and only maintains 8 outgoing connections. Peers are allowed to update their numbers of outgoing and incoming connections. In fact, Monero recommends peers to increase the number of connections according to their capacity, to have a better connectivity.

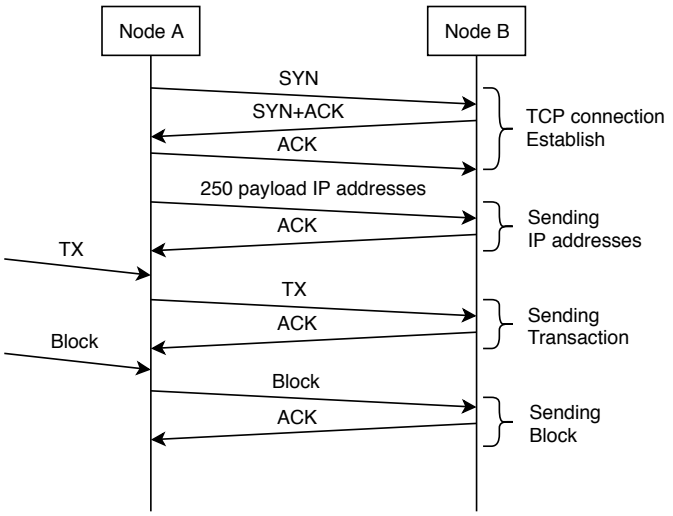


Fig. 1: Information propagation in the Monero network.

D. Information Propagation

Three types of messages are propagated in Monero, namely containing peer information, transactions, and blocks.

A node establishes connections with others through a TCP three ways handshake (SYN-SYN-ACK), and sends peer information through the established connection. This process is illustrated in Figure 1, where a node A establishes a connection with a node B. Node A initializes the communication by sending a synchronize message SYN to B, and expecting from B an acknowledge message SYN+ACK where the SYN and the ACK bits are both turned on (set to 1) in the TCP header. Upon receiving SYN+ACK, A completes the handshake by sending an acknowledge message ACK to B. SYN messages and ACK messages are indicated by the turned on SYN bit and the ACK bit inside the TCP header, respectively. After establishing a connection, Node A prepares a TCP packet where the payload contains information of the top 250 peers in A's $white\_list$, and sends this packet to B.

Each node inserts the most recent 250 IP addresses of its $white\_list$ into a TCP packet, and sends it to its neighbors as payload data. In this way, nodes disseminate and update their peer lists. Similarly, to disseminate received and verified transactions and blocks, nodes establish TCP connections with their neighbors as above, then transfer the verified transactions and blocks to their neighbors through the established TCP connections.

III. Analysis Pipeline Overview

In this section, we introduce the different data structures and processes we implemented, along with the associated network tools we used and algorithmic approaches we developed to count the number of active Monero nodes, and to infer the neighbors of a target node.

A. Construction

As illustrated on the left of Figure 2, our method sets up four nodes in the Monero network to collect data. These

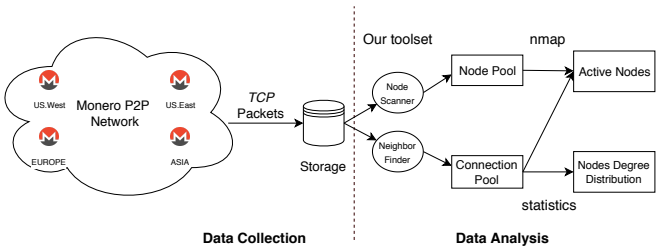


Fig. 2: Analysis pipeline overview.

nodes establish connections with peers in the network, and store packets into their local storage. We rely on two network measurement tools, i.e., *tcpflow*[3] and *nmap*[4], to collect data and analyze the Monero network. First, when establishing TCP connections with a node, we use *tcpflow* to collect the IP addresses of peers communicated by this node. The collected TCP packets of connections with different nodes are stored in different files. In other words, we store all data collected during connections with a single node into a single file. This allows us to efficiency manage and analyze each of the connected nodes.

As mentioned in Section II, each received TCP packet contains the most recent 250 IP addresses of the sender's $white\_list$. Thus, all received IP addresses are recent outbound peers of the sender. We then use our first tool, *NodeScanner*, to collect the IP addresses of Monero nodes and store the results in the *NodePool*. To test whether the nodes associated to the received IP addresses are still active, we use *nmap* to test each IP address in the *NodePool*. In Monero, the port 18080 is used to receive incoming connections. Hence, if a node is responsive and its TCP port 18080 is open, then the node is deemed active. In this case, we will lose the active node if it does not respond the *nmap* request. Nevertheless, our second tool, *NeighborFinder*, infers the neighbors of the Monero nodes we received packets from, and store them in the *ConnectionPool*. Both reached nodes and their neighbors from *ConnectionPool* are active. This provides a better way to discover active nodes without *nmap*. We introduce in greater details our developed tools in the next section.

B. Neighbor Inference Based on Membership Messages

We used two complementary approaches to understand how to identify a node's neighbors. First, we checked the Monero source code, and found that the function *peerlist_manager::set_peer_just_seen* updates the IP addresses in the peer list by updating the $last\_seen$, using the instruction *ple.last_seen = time(NULL)*. This function is triggered when an IP address is promoted from the $gray\_list$ to the $white\_list$, or when a node is connecting with the host. This updating scheme leads to outbound neighbors of a node to be associated with the freshest $last\_seen$ in its peer list. Therefore, we conclude that the IP addresses of the outbound

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

neighbors of the packet sender are included in the TCP packet that are sent to its peers.

Second, we setup our own nodes in Monero network to prove that the neighbors' IP addresses are binded into the TCP packet for being sent, as detailed in Section V. We further analyzed the peer lists and connections of our controlled nodes, and the payload IP addresses our nodes sent to their neighbors. We observed that the payload IP addresses our nodes sent out include all their outgoing connections when their maximum number of outgoing connections is less than 250. Therefore, by analyzing the payload IP addresses in a packet, we are able to infer the outgoing connections of its sender. We discuss the case of nodes that maintain more than 250 outgoing connections (heavy node) and the influence in section V.

IV. Nodes discovery and connections inference

As previously mentioned, by implementing our controlled full nodes in the Monero network, we can accept incoming connections and initiate outgoing connections to receive TCP packets from other nodes.

In the following, let us assume that a node has received j TCP packets from a Monero reached node. We use $P = \{P_1, P_2, P_3, ..., P_j\}$ to denote the set of packets received from that node. 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 $\mathcal{N}$, we identify the set $A = \{A_1, A_2, A_3, ..., A_j\}$ of included IP addresses. The NodeScanner first collects the unique IP addresses. We denote U the set of unique IP addresses that are obtained from $\mathcal{N}$, i.e., $U = A_1 \cup A_2 \cup A_3 \cup ... \cup A_j$. We then insert 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 firstly indicate our neighbors inference approach based on the time difference of the nodes' $last\_seen$ timestamps in a single packet. We then refine this approach by relying on several received packets.

1) Neighbors inference based on a single packet: For any received packet P_k from a node $\mathcal{N}$, we assume that it contains r $(r < 250)$ 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 between the timestamps of any two neighbors, then we can extract a set $N'_k = \{A_{k,i+1}, A_{k,(i+2)}, A_{k,(i+3)}, ..., A_{k,(i+r)} |\ r \in [1, 250],\ i \in [1, (250 - r)],\ \forall x \in [i + 1, i + (r - 1)], T_{k,x} - T_{k,(x+1)} \leq \mu\}$ of neighbors from P_k as shown in Algorithm 1.

Each node checks its connections iteratively by using *IDLE_HANDSHAKE*, which sends SYN packets to all of its neighbors, the $last\_seen$ timestamp of handshaked neighbors are updated with the local time if nodes can be contacted, otherwise connections are dropped. This mechanism prevents idle connections to be maintained. However, each connection may answer the SYN packet 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 the value should not be too large as it may cause 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.

Algorithm 1: Neighbors inference by using single packet

Input : P_k: Packets;
μ: 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;

```
for y = 1, y < 250, y++ do
    if T_{k,y} - T_{k,(y+1)} ≤ μ then
        N'_k ⟵ A_{k,y}
    end
    if T_{k,(y+1)} - T_{k,(y+2)} > μ then
        N'_k ⟵ A_{k,(y+1)}; break
    end
end
```

2) Improved neighbors inference based on multiple packets: During a connection with a node, it frequently happens that we receive multiple packets. 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 $N'' = \{A_{k,z} \mid \forall$ z,w, $A_{k,z} = A_{(k+1),w}$, $T_{k,z} \neq T_{(k+1),w}$, $A_{k,z} \in P_k$, $T_{k,z} \in P_k$, $A_{(k+1),w} \in P_{(k+1)}$, $T_{(k+1),w} \in P_{(k+1)}\}$ to denote the IP addresses that have been updated between packets P_k and $P_{(k+1)}$. We then extract the neighbors of node $\mathcal{N}$ following Algorithm 2.

Algorithm 2: Neighbors inference by using two packets

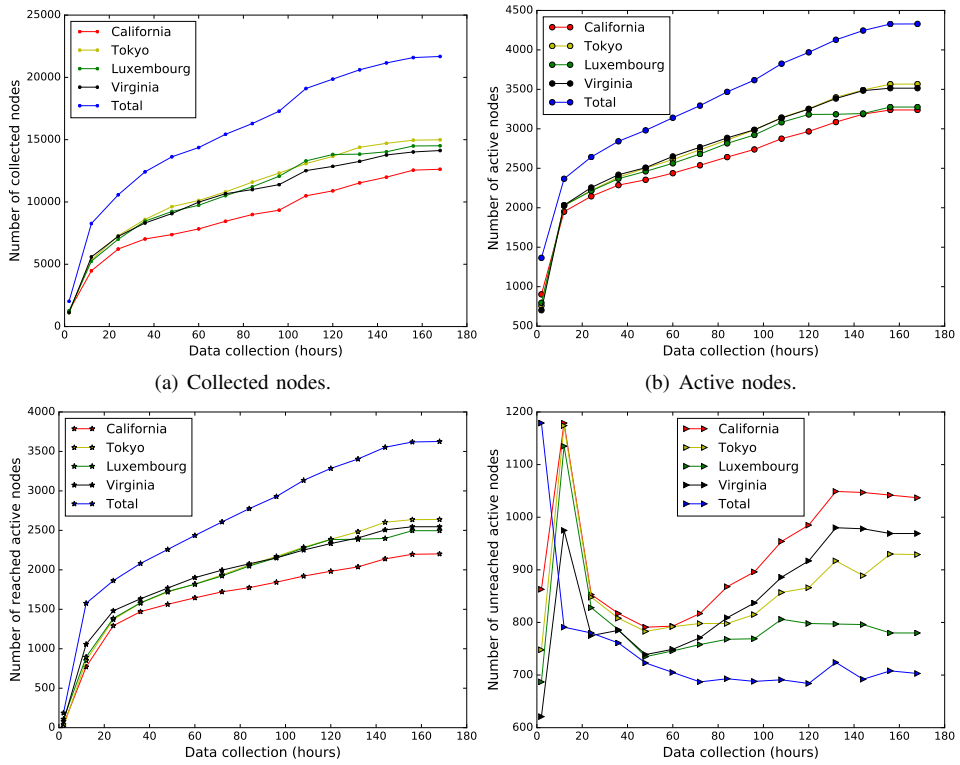
Input : Packets P_k and $P_{(k+1)}$;
A_k, $A_{(k+1)}$: the *IP* addresses in P_k, resp. P_{k+1};
T_k, $T_{(k+1)}$: the $last\_seen$ timestamps in P_k, resp. P_{k+1};
Output: Neighbors set N''_k;

```
foreach A_{k,y} = A_{(k+1),z} do
    if T_{k,y} ≠ T_{(k+1),z} then
        N''_k ⟵ A_{k,y}
    end
end
```

V. Experiments

A. Data Collection Settings

We set up four full nodes in the Monero network two in the U.S. (California and Virginia), one in Europe (Luxembourg),



(a) Collected nodes.

(b) Active nodes.

(c) Reached active nodes.

(d) Unreached active nodes.

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

and one in Asia (Japan). In particular, the node in Luxembourg was set up on an Ubuntu 16.04 machine with Intel Xeon CPU E5-2695 V3 2.30 GHz processor, and each of the remaining nodes was established on an independent Ubuntu 16.04 machine with an Intel Xeon Platinum 8000 series processor. The four nodes were connected to each other in order to gain the ground truth for verifying our neighbor inference approach.

We collected data by participating in the Monero protocol starting from 2018.12.23 during a week. Our goal was to establish connections with other nodes as much as possible. We manually modified the settings on our controlled nodes so that they could establish adequate connections with other nodes. First, we modified the maximum number of incoming and outgoing connections to 99,999, which makes our nodes actively look for new nodes, in order to reach this maximum number of neighbors. Second, we increased the maximum number of TCP connections our nodes could maintain with other nodes simultaneously.

Even though we collected data during a single week, our results indicate that our approach discovered more than 40% new active nodes in comparison to the Monerohash[5]'s results. To the best of our knowledge, Monerohash is the only Monero mining pool providing information on the number of active nodes in the network.

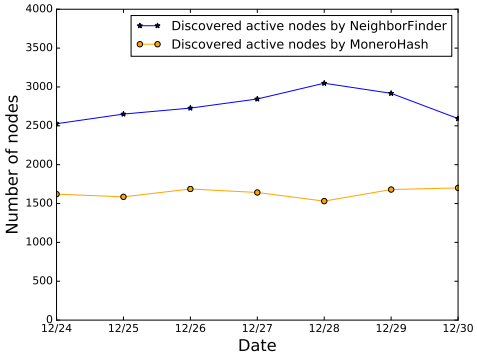


Fig. 4: Number of active nodes per day.

B. Measuring the Network Coverage

Through our *NodeScanner*, we have collected 21,678 peer IP addresses. Out of these collected IP addresses, through *NeighborFinder* we have established connections with 3,626 peers, and identified 703 peers that we could not connect to, 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, but we could not directly connect to them. We say a peer is inactive if it is neither connected to our nodes, nor connected

[5]https://monerohash.com/nodes-distribution.html

to our connected peers. If our nodes could not connect to a peer, then it either means that the peer was already fully connected during the data collection, or that it was offline. To reduce the potential false positives, 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. Our approach shows that only about 20% (i.e., 4,329) of the discovered nodes are active, and that the remaining nodes were offline during the data collection period. We assume that the majority of transmitted IP addresses are inactive in Monero network.

By using *NeighborFinder*, we built a *ConnectionPool* that includes all the active nodes. To further guarantee that these nodes are indeed active, we use *nmap* to check if their IP addresses can be pinged and determine whether their TCP port 18080 is open. We indicate the number of active nodes per day in Figure 4. There were approximately 2,758 active nodes per day. Compared with Monerohash, which discovers 1,635 active nodes in average per day, we identified 68.7% more nodes. Moreover, the number of daily active nodes in Bitcoin [9], [10] and Ethereum [11], [12] is around 10,000, it is therefore not a surprise to see that Monero has far less daily active nodes.

Figure 3 presents the data collection statistics, 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. Figure 3(c) shows the evolution of the number of active nodes connected to our servers. Figure 3(d) shows the number of active but unreachable peers.

Figures 3(a) and 3(b) show that the number of discovered peers, and active nodes including both reachable active nodes and unreachable active nodes, increased dramatically in the first 18 hours of the data collection. However, while the number of unreached nodes in each of the data collection server increases rapidly, the total number of unreached nodes goes the other way. This means that the sets of nodes that our data collection servers connected to are fairly non-overlapping until the 18th hour. Around the 23rd hour of the data collection process, the Virginia, Luxembourg, Tokyo, and California server respectively connected to 1481, 1385, 1371, and 1292 nodes. Each server connected to a number of nodes that other servers did not connect to. Overall, we reached 1863 nodes. In addition, the number of discovered peers stabilized after 160 hours of data collection, as Figure 3(c) shows.

Figure 3(d) indicates that we have detected most of the long-term running active nodes within 12 hours, when the total number of unreached active nodes dramatically decreased. Then, this number keeps decreasing showing that we connected to most of them. In particular, we have connected to most of the nodes between 2 to 12 hours. Comparing total number of unreached active nodes (blue line) with the number of unreached active nodes detected in different locations between 2 to 12 hours, we can find that each deployed server instantly discovered more unreached active nodes, which had been discovered by other deployed servers. One interesting finding is that we detected almost all long-term running active nodes after the first 80 hours. This indicates that new participants mostly connected to the nodes that we had discovered. Sometimes, new participants connected to some nodes that we had not discovered (like the jump at 132 hour in the blue line of Figure 3(d)). Thus, it is likely that the Monero network contains around 2758 active nodes per day (according to our experiment) that maintain all connections in the system.

C. Node Distribution

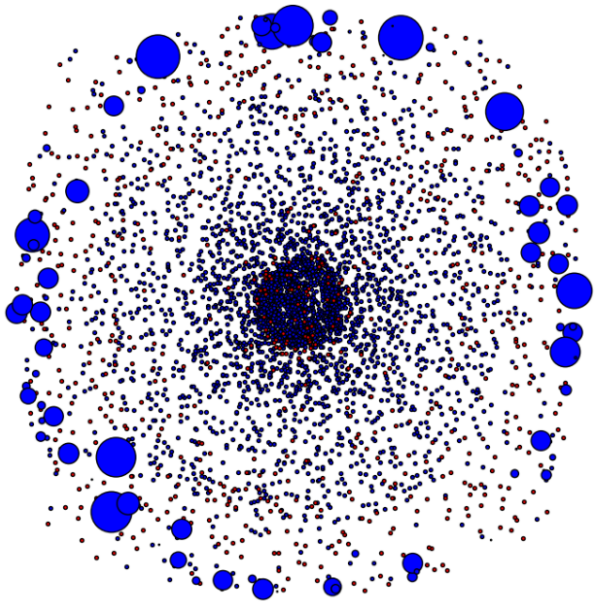
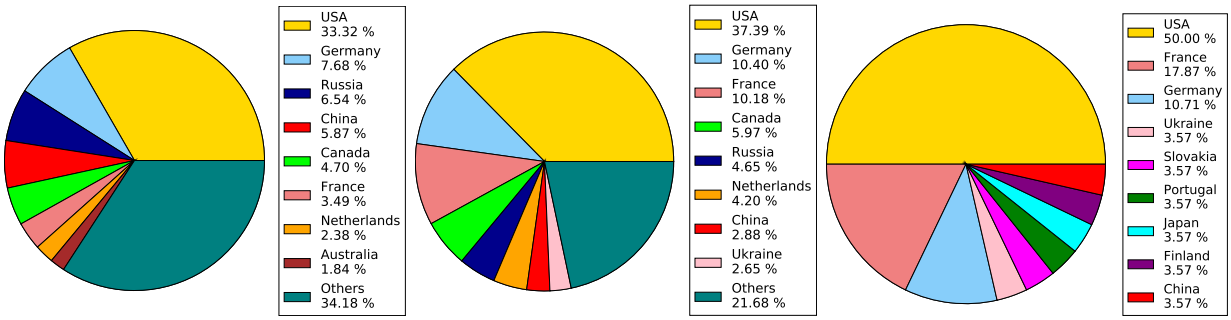


Fig. 5: Representation of the Monero nodes. Each dot represents a Monero node. The size of the dots represents the outgoing degree of the node. Red represents unreached nodes and blue represents reached nodes.

Snapshot of the Monero network topology. Based on the *ConnectionPool*, we draw a snapshot of the Monero network topology as shown in Figure 5. In this Figure, we use red dots to denote unreached nodes, and blue dots to denote reached nodes. The size of a dot represents the node's degree. This figure shows a potential re-centralization problems in the Monero network. We discovered that some nodes operated by mining pools are always maintaining many more connections than normal client users. In this case, the actual maximum number of connections depends on the network capacities of the host. Mining pools are then implementing a Monero node on a well equipped and connected host to reach as many other nodes as possible, so that they receive the the broadcast messages faster to gain an advantage to win coins. Generally, smaller client users use their node to transfer coins or check their balance, and are not willing to reach more other nodes. Therefore, those clients are likely to use the system



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

Fig. 6: Nodes location distribution.

rather than participate in it.We believe that such difference in nodes' behaviors cause the inequality of connectivity in the Monero peer-to-peer network, and might lead to network vulnerabilities.

Connectivity. We say that a node is of degree N, if it maintains at most N outgoing connections. We classify active nodes into three types based on their degree: *light node (degree$\leq$8)*, *medium node (8$<$ degree$\leq$ 250)*, and *heavy node (degree$>$250)*.

As shown in Table I, out of the 3,626 active and reachable nodes, 3,146 nodes (86.8%) are light nodes, i.e., the number of their outgoing connections is limited to 8, and most of the light nodes accept only 1 incoming connection; 452 nodes (12.5%) are medium nodes maintaining from 8 to 250 connections, and only 28 nodes (0.7%) are heavy nodes maintaining more than 250 connections simultaneously. In addition, the majority of the nodes (86.8%) collectively maintain only 17.14% connections, while the remaining 13.2% of the nodes collectively maintain 82.86% of the network connections.

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

TABLE I: Number of active nodes in the ConnectionPool.

We present in Figure 6 the location of the Monero nodes depending on their classification. This Figure shows that the light nodes, which are likely to be clients, are more distributed around the world than the heavy nodes. In particular, over 78% of the heavy nodes are located in only three countries, namely US, France, and Germany, and half of the heavy nodes in Monero are located in US. In fact, US has the largest number of nodes in all three types. France, which comes to the second place and the third place considering the total number of hosted heavy nodes and medium nodes. However, only a small proportion (3.59%) of light nodes are located in the France. Germany is more balanced in terms of the percentage of different types of nodes — 10.71% heavy nodes (3rd place in the chart), 10.4% medium nodes (2nd place), and 7.68% light nodes (2nd place) are located in Germany. Moreover, we present in Figure 7 the geographic distribution of active nodes.

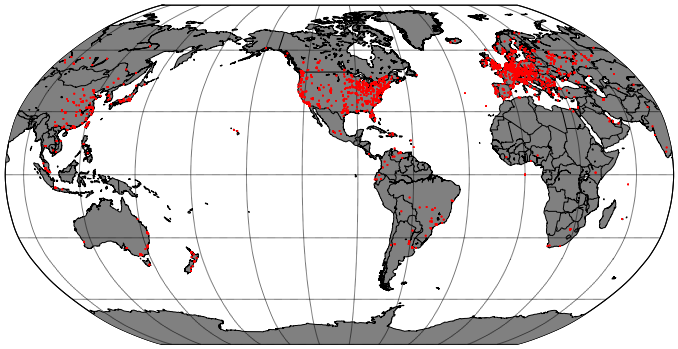


Fig. 7: Geographic distribution of active nodes.

Monero has hardcoded 8 seed nodes in the system, and we initially suspected that all of them would be heavy nodes. Interestingly, our experiment shows that only 5 seed nodes can be pinged, and that only 2 seed nodes are heavy nodes. By comparing the discovered heavy nodes with public Monero mining pools [6] and seed nodes [7], we found that 9 heavy nodes are maintained by mining pools, 2 heavy nodes are Monero seed nodes. Due to the lack of public information, we cannot identify the other 17 heavy nodes. However, we assume that the remaining heavy nodes are likely to be private mining pools.

Figure 8 presents the number of inferred neighbors of a heavy node, medium node, and light node. For instance, a light node represented in green maintains 8 outgoing connections, a medium node represented in orange maintains 98 outgoing connections. However, the maximum number of outgoing connections of heavy nodes is difficult to confirmed. First, heavy nodes might have an unlimited number of connections in order to maintain the highest connectivity they can. Second, the actual number of outgoing connections might never reach the maximum number for some heavy node. This is due to the fact that even though a heavy node may have more than 250

[6]http://moneropools.com/

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

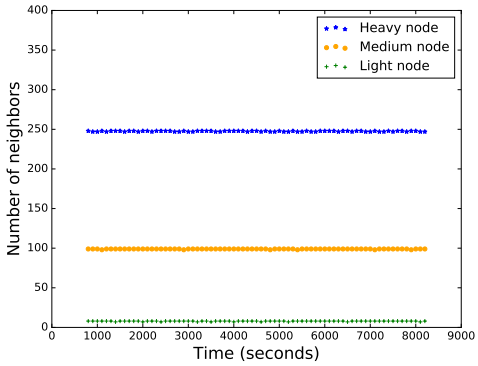


Fig. 8: Number of outgoing neighbors of heavy node, medium node, and light node.

neighbors, it can only select and broadcast 250 of its neighbors in its messages.

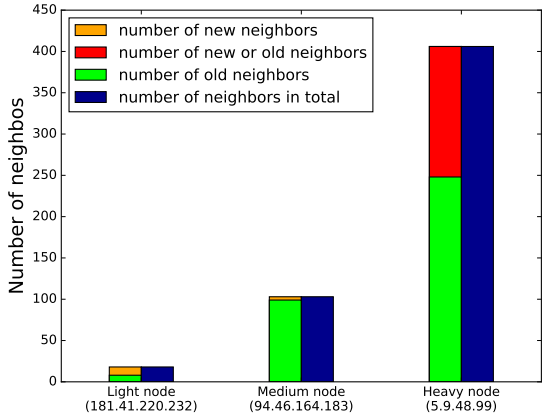


Fig. 9: Neighbor updates in 9 hours.

Figure 9 details the number of new neighbors per category of node. In this Figure, we use orange to denote the number of new neighbors during 9 hours. The green color denotes the number of old neighbors, i.e., the neighbors that were connected at the beginning. The dark blue denotes the total number of neighbors during 9 hours. The red color denotes the number of neighbors for which we could not confirm whether they were old neighbors or new neighbors. Figure 9 shows that we are able to infer all the neighbors of light and medium nodes with their neighbors updating history. However, for heavy nodes, it is difficult to infer the maximum number of their outgoing connections. Nevertheless, monitoring heavy nodes for a longer period of time would allow us to get a more accurate maximum number of outgoing connections of heavy nodes.

By using our neighbor inference approach, we are able to find all of their neighbors during the data collection. As shown in Figure 10, we show all neighbors of a *light node* during the 9-hour monitoring. Each color represents a neighbor of the node. It shows that neighbor 1-6 stayed in connection with the node in the entire 9 hours, whereas the connection with neighbor 7 is dropped around the 8th hour, and neighbor 11

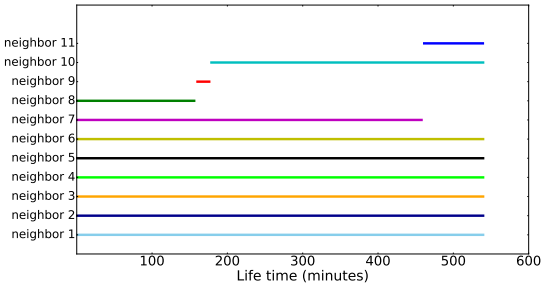


Fig. 10: Neighbor updating of a light node in 9 hours.

is connected to replace neighbor 7. Similarly, the connection to neighbor 8 is dropped within 3 hours, and the node is reconnected to neighbor 9 for a very short time period, before establishing a more stable connection with neighbor 10. This indicates that the *NeighborFinder* not only discovers the number and information of neighbors at a single time point, but also the number and information of neighbors during a period of time. This allows us to monitor the dynamic connectivities of existing active Monero nodes.

Power Law Distribution. Previous works [13], [14] pointed out that the cryptocurrency networks follow the power law distribution, where the majority of the nodes maintains few connections, and a small number of nodes maintains most of the connections. Our findings, as presented in the Figure 11, confirm this statement in the Monero network. This indicates that successfully attacking a small number of heavy nodes might cause large communication delays, or even a network partition. Such a network attack might open a window for attacker to launch a network level attacks. We leave for future work the detailed analysis on the feasibility of such attacks.

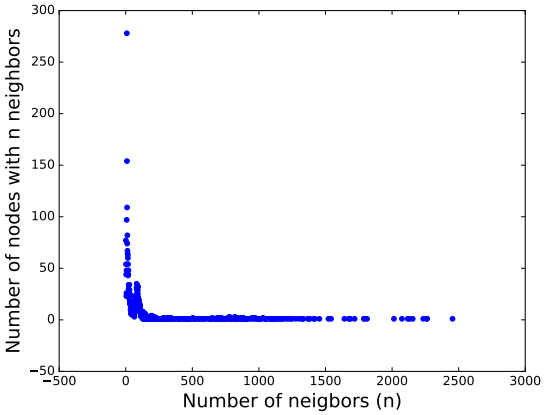


Fig. 11: Degree distribution.

VI. Discussion

A. Unreached Active Nodes

During the data collection process, we measured that our nodes could not directly connect to 16.2% of the active nodes that are connected to our neighbors. We indirectly observed their presence in the network from the payload TCP packets

that we received from the nodes we reached. The following two reasons might explain why they could not be contacted directly by our nodes. First, the nodes might be fully connected and could not accept new connections. Second, if a host receives several connection tentatives simultaneously, it will prefer to establish the connection with the nodes that were previously connected to. The second reason comes from our analysis of the Monero peer-to-peer network. In particular, as detailed in Section II, to establish connections nodes will first choose peers from its $anchor\_list$, where all recorded nodes are previously connected.

B. Potential Attacks

We briefly discuss potential network level attacks on Monero, however, it is very important to conduct a formal study on the attacks as a part of the future work. As we showed previously, the links between the nodes in Monero can be inferred due to the $last\_seen$ associated to the IP addresses. This might help an adversary to launch BGP hijacking attacks [15], where the adversary would isolate a node from the network. On the other hand, if the adversary can connect to the node that does not accept incoming connections, then the adversary could dominate the victim's peer list by continuously sending its IP addresses with fresh $last\_seen$, and then monopolize all the connections of the victim to launch attacks [7].

We also show the possibility to detect the destination of outbound connections of the heavy nodes. In practice, miners would try to broadcast their created blocks to the network as fast as they can, to compete against other miners who might have mined a block at the same time. Thus, the front-end node of a mining pool is likely to maintain a large amount of outgoing connections to speed up the message propagation. An adversary can connect to the front-end node of the mining pool, and infer its outgoing connections. The adversary can then launch DOS attacks to harm the mining pool's connectivity. By comparing with Bitcoin and Ethereum, the network size of Monero is far smaller. This might make similar attacks easier and cheaper to launch on Monero.

VII. Related work

Previous works studied the topology of leading cryptocurrencies, e.g., Bitcoin and Ethereum, that are not privacy preserving. Decker and Wattenhofer [16] measured the rate of information propagation in the network, Miller et al. proposed CoinScope [14] to discover a node's degree and the network topology in Bitcoin. Neudecker et al. [13] proposed a comprehensive timing analysis to discover Bitcoin's network topology. Their approach is based on an estimation of the sending time of a transaction/block. This estimation is obtained by subtracting the measured time needed to transfer a message between the two nodes to the receiving time of the transaction at the receiver side. As a result, they can link a transaction with the IP address that generated it with high probability. Later on, the Bitcoin developers noticed this timing leakage, and added a random delay to the dissemination of a message at a sender's side. Kim et al. [17] evaluated the network size and evaluated several properties of the Ethereum network. They used the number of daily online nodes to evaluate the network size in Ethereum. However, links between different nodes are difficult to discover in Ethereum.

Network level attacks, such as routing attacks [7], [15], [18] or man-in-the-middle attacks [19], have been explored on Bitcoin and Ethereum. Such attacks are facilitated by the fact that Bitcoin's protocol makes nodes exchange messages in plaintext during the peer-to-peer communications. An adversary is therefore able to explore the network for the purpose of harming the blockchain system. Deanonymization attacks [18] have been described by Fanti and Viswanath. These attacks aim at linking the IP address of a node with the transactions it created, even if the node is located behind a firewall or a NAT. Such attacks are based on monitoring the messages exchanged between nodes of the network. Apostolaki et al. described BGP hijacking attacks in Bitcoin [15], where malicious messages are injected to broadcast wrong IP prefixes in Bitcoin, which causes packets to be delivered to incorrect locations. By launching a hijacking attack, an adversary can isolate a target node in the network, and delay the dissemination of messages among nodes in order to spend one coin twice, or in order to win the mining competition. Eclipse attacks, in Bitcoin [7] and in Ethereum [8], pointed out that unsolicited incoming connections can be applied by the 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.

VIII. Conclusion

In this work, we presented a method to observe the Monero peer-to-peer network, and infer its topology. More particularly, we described how to rely on Monero nodes to discover all the nodes participating in the protocol, and how to infer their connections, using the $last\_seen$ timestamps in the peer lists that nodes exchange. 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 connections with each other. Our analysis provides insights about how centralized the Monero network is. As future work, we will conduct a deeper network-based security and privacy analysis of Monero, based on the tools provided in this paper.

Acknowledgement

This work is partially supported by the University of Luxembourg - SnT and by the Fonds National de la Recherche Luxembourg (FNR) through PEARL grant FNR/P14/8149128.

References

[1] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *IEEE Security and Privacy (SP)*, 2013.

[2] N. Van Saberhagen, "Cryptonote v 2.0," 2013.

[3] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin, "An empirical analysis of traceability in the monero blockchain," *PoPETs*, vol. 2018, no. 3, pp. 143–163, 2018.
[4] A. Kumar, C. Fischer, S. Tople, and P. Saxena, "A traceability analysis of monero's blockchain," in *ESORICS*, 2017, pp. 153–173.
[5] Z. Yu, M. H. Au, J. Yu, R. Yang, Q. Xu, , and W. F. Lau, "New empirical traceability analysis of cryptonote-style blockchains," in *FC*, 2019.
[6] "Monerohash - monero mining pool," https://monerohash.com/nodes-distribution.html, accessed: 2019-01-12.
[7] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network." in *USENIX Security Symposium*, 2015.
[8] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network," *IACR Cryptology ePrint Archive*, vol. 2018, p. 236, 2018.
[9] "Bitnodes," https://bitnodes.earn.com/nodes/, accessed: 2019-01-12.
[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[11] "Ethernodes," https://www.ethernodes.org/network/1, accessed: 2019-01-12.
[12] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.
[13] T. Neudecker, P. Andelfinger, and H. Hartenstein, "Timing analysis for inferring the topology of the bitcoin peer-to-peer network," in *IEEE UIC*, 2016.
[14] A. Miller, J. Litton, A. Pachulski, N. S. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering bitcoin's public topology and influential nodes," 2015.
[15] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *IEEE Security and Privacy (SP)*, 2017.
[16] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P*, 2013.
[17] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring ethereum network peers," in *ACM IMC*, 2018.
[18] G. Fanti and P. Viswanath, "Deanonymization in the bitcoin p2p network," in *NIPS*, 2017.
[19] P. Ekparinya, V. Gramoli, and G. Jourjon, "Impact of man-in-the-middle attacks on ethereum," in *IEEE SRDS*, 2018.