DLT Peer-to-Peer Network mapping

1st Yu Gao
Informatics
Zürich University

***Abstract*—Blockchain systems rely on peer to peer networks to support decentralized and privacy preserving communication. Due to protocol level privacy mechanisms, real peer connections are not directly observable, which complicates the analysis of network structure.**

In this study, a clustering based method is applied to infer real peer connections in Monero using peer list data extracted from TCP flow level observations. The method is designed for Monero's latest protocol version, where timestamp information is no longer available and peer sharing follows protocol specific selection rules.

We validate the inference approach and reconstruct the network structure, showing that inference accuracy improves with longer observation periods. In contrast, applying similar techniques to networks that rely solely on textitlast_seen information, such as Dogecoin, remains challenging. This report provides insights into peer to peer connectivity and the limits of network inference imposed by peering protocols.

***Index Terms*—Monero P2P Protocol, Monero P2P Network Mapping, Robustness in Monero P2P network**

I. Introduction

The peer-to-peer (P2P) network constitutes the foundational communication layer of decentralized ledger technology (DLT) systems. It enables decentralization by disseminating transactions and blocks, supports consensus among distributed nodes, and contributes to the overall security and robustness of the system [1]–[3]. Unlike traditional P2P networks, DLT systems assign additional responsibilities to peers, including consensus participation and ledger synchronization, which significantly increase the complexity of network operation and analysis.

Although different DLT systems adopt distinct protocol designs, they all rely on a P2P overlay network as a backbone to facilitate trustless interactions among independent computing devices, referred to as peers or nodes [4], [5]. In this overlay, peers play symmetric roles in message routing and resource sharing [6]. The design and dynamics of this P2P network directly influence critical system properties, including decentralization, resilience, and security.

A key feature of DLT P2P networks is their self-organizing connectivity, whereby nodes autonomously discover and maintain connections without centralized control [7], [8]. While this property enhances robustness and scalability, it also complicates network measurement and structural analysis. In the absence of a global view or central authority maintaining a complete network map, understanding the network topology requires empirical measurement techniques based on partial observations, such as traceroute-style probing and traffic analysis.

Monero is an open-source blockchain platform that employs advanced cryptographic techniques, including ring signatures, stealth addresses, and ring confidential transactions, to create a permissionless P2P network with a focus on transaction anonymity and untraceability[1]. Like Bitcoin, it uses proof of work (PoW) as its consensus mechanism. However, Monero differentiates itself from most other Distributed Ledger Technologies (DLTs) through its unique feature: transactions conducted on Monero are untraceable. The study of DLTs spans several key areas, including smart contracts [9], [10], consensus mechanisms [11], malicious attack [12]–[14], decentralized finance [15], and transactions [16], [17]. While substantial research has focused on analyzing transaction networks, assessing the performance of consensus mechanisms, and evaluating the levels of decentralization they achieve, P2P networks remain relatively underexplored. This is particularly noteworthy, as P2P networks play a critical role in ensuring blockchain security and warrant further research attention.

For most other DLTs, there is significant research investigating the structure and characteristics of their P2P networks [13], [18]–[21]. However, for Monero, research on this topic is limited. The only study focusing on Monero's P2P network is Cao et al. [22] paper, Monero's network protocol was updated to hide the *time_stamp* field in the peer list. In other words, nodes no longer include *last_seen* information when sharing their peer lists. Existing studies on Monero primarily concentrate on transaction traceability [23]–[25], making its P2P network an area in need of more research attention. For most other DLT systems, substantial research has investigated the structure and characteristics of their P2P networks [13], [18]–[21]. In contrast, studies focusing on Monero's P2P network remain limited. To date, the only work explicitly examining Monero's P2P topology is that of Cao et al. [22]. Since then, Monero's network protocol has been updated to remove the *timestamp* field from shared peer lists, meaning that nodes no longer disclose *last_seen* information when exchanging addresses. Existing research on Monero has therefore primarily focused on transaction traceability rather than network structure [23]–[25], leaving its P2P connectivity underexplored.

By contrast, Dogecoin largely follows the legacy Bitcoin style peer discovery protocol [13], [18], where peer lists include *last_seen* information but do not apply preferential peer selection when sharing addresses. While the availability of *last_seen* timestamps enables certain temporal analyses, the

[1]https://www.getmonero.org/

absence of protocol-level sharing preferences makes reliable inference of direct peer connections more challenging. As a result, methods that exploit protocol-induced biases in peer dissemination are not directly transferable from Monero to Dogecoin.

In this paper, we collected TCP data exchanged between our Monero node and its peers, extracting peer lists shared through the TCP flows. Given that the updated protocol no longer includes handshake timestamps in peer lists, we propose an algorithm to infer peer connections (neighbors) using timestamp-free TCP packet analysis. We validate the algorithm by comparing the inferred connections with the actual neighbors of our nodes, demonstrating high accuracy despite the absence of timestamp references. Finally, we visualize the network architecture and analyze the Monero P2P network topology. This analysis enhances our understanding of the network structure, identifies potential vulnerabilities, and provides a basis for simulating various attack scenarios. These findings offer valuable insights into the risks inherent in the current protocol design, highlighting areas for potential improvement.

II. Monero Peer-to-Peer Protocol

Similar to general P2P networks, each peer in Monero maintains both incoming and outgoing connections. Incoming connections are established by other peers requesting to connect, allowing data to flow from those peers to the host peer. Outgoing connections, on the other hand, are initiated by the host peer to other nodes, enabling data sharing and communication within the network. By default, if no specific configuration is set for a Monero node, a peer establishes 8 outgoing connections, and a node must have at least one outgoing connection to join the network, while the number of incoming connections can be 0. When a new node joins the network for the first time and lacks knowledge of existing nodes, it connects to public seed nodes that facilitate its integration. These seed nodes help the new node discover other active nodes in the network. The new node then establishes connections with these discovered nodes through a handshake protocol, which ensures the connection's validity, and exchanges network also blockchain information. It synchronizes data such as blocks and peer lists. Once connected, the node continually maintains and updates its list of peers, handles incoming connection requests, and manages connection and disconnection processes to ensure overall network stability.

Unlike many blockchain platforms that are largely informed by Bitcoin's P2P protocol, Monero is built on its own uniquely designed P2P network protocol. Each peer in the Monero network has two lists as its peer database, **white_list** and **gray_list**. There are 1000 and 5000 slots for **white_list** and **gray_list**, respectively. The **white_list** is used to save the most recent handshake peer addresses, and the grey is for the peer who didn't answer or the IP addresses sorted lower by timestamp will be moved to the **gray_list**. Under the new Monero P2P protocol, in the TCP flow process, there are 120% peers returned by a peer, which is randomly selected from its top(according to the *last_seen* timestamp sequence) 300 IP Addresses from the white list[2].

III. Peer list collection and neighbour inference

In the context of P2P networks in distributed ledger technologies (DLT), it is crucial to distinguish between "active neighbours" and "potential peers".

A. Definition of 'Neighbour' and 'Interaction'

To formalize the distinction, we define the concepts of "connection" and "interaction" as follows:

Neighbour/Connection: A connection refers to a persistent and active communication channel between peers, enabling the exchange of substantive data. Active neighbours are defined as peers with which such direct and ongoing connections are established. These connections are critical for maintaining the network's integrity and performance, even as peers dynamically join and leave the network over time.

Interaction: An interaction encompasses any superficial or temporary relationship between peers. This includes peers that:

- Establish brief connections with a node in a short duration, such as during initial handshakes, without forming a persistent communication channel.
- Appear in a node's peer list as potential peers, such as addresses obtained through peer discovery protocols or shared lists, without any established communication or validation.

This distinction allows for a precise representation and management of the network topology, ensuring that only meaningful peer interactions are classified as active neighbours within the P2P network.

To infer P2P connections between Monero nodes, we first setup a peer list data collection pipeline similar to the one described in [22], and then proceed to devise an algorithm for the inference of Monero P2P connections. The Monero peer list sharing protocol changed in the last few years so that the *last_seen* timestamp is not shared anymore, so we could not use the previously published P2P inference algorithm [22] but had to proposal a different one which works without this *last_seen* timestamp information.

B. Peer list data collection

Our data collection pipeline (Figure 1) consists of three machines deployed in three different regions - the United States, Europe, and Singapore. Geographical dispersal of data collection nodes is important to capture as much of the Monero P2P network as possible, and at the same time different nodes which we ourselves control allow us to collect validation data for the Monero P2P inference. Peer list data is collected by running a Monero node on a machine and monitoring TCP traffic with the *tcpflow*[3] program which is setup to listen on port 18080 through which communication with other Monero nodes is established. Peer lists are then extracted from

[2]Monero GitHub Repository

[3]https://github.com/simsong/tcpflow

the binary dump of the collected traffic data, as this data is otherwise not exposed through the Monero node's RCP interface. The connection data from each of our three nodes then serves to validate the P2P network inference using our devised method. The data used in this paper was collected between 21.12.2024 and 10.1.2025, covering three weeks.

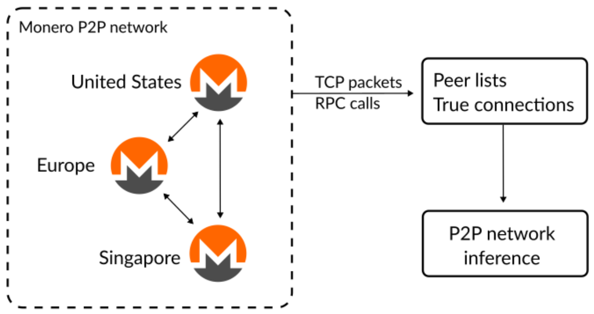

Fig. 1: Monero Data collection pipeline.

C. *Algorithm of peer connection Inference*

After collecting and analyzing peer-list data, we need to filter and identify which peers qualify as "real neighbours." According to the P2P protocol, each peer shares up to 250 peers from the top 300 entries in its whitelist. However, if a node's whitelist contains fewer than 250 peer addresses, it will share all the available peer addresses from its current whitelist.

An important observation is that, by default, real neighbours in a P2P network are likely to appear at the top of a peer's whitelist. This is because real neighbours perform regular handshakes to maintain their connections. Consequently, real neighbours are expected to appear more frequently in shared *peer_lists* compared to peers that are not directly connected.

According to the interactions in the data set, we can broadly categorize nodes into two cases:

- **Nodes with numerous incoming or outgoing connections:** These nodes often have a large number of connections due to their higher activity or role in the network, for instance, the public seed nodes, which have a lot of incoming connections.
- **Nodes with general connections:** For example, without special configurations in the node's setup file, a general node will have 8 outgoing neighbours by default.

The challenge lies in determining the frequency threshold for identifying real neighbours based on how often the interactions appear in the TCP flow packets by shared *peer_lists*. Since real neighbours are expected to appear more frequently than non-neighbours after multiple times *peer_list* return to the same handshake nodes, we can leverage this property for filtering. One thing to note is that, since there is no information about interaction direction in the packages our nodes received, each interaction in the data is regarded as bidirectional (undirected).

The underlying assumption is that a node's neighbours will appear most frequently in the peer list returned. This assumption holds because the peer list consists of 250 peer addresses from the top 300 nodes in the white list of a node. Due to the `IDLE_HANDSHAKE` protocol, the node verifies the connection with its neighbours every minute [22], which results in the frequent update of their timestamps in the **white_list**, ensuring their presence in the top 300. Additionally, *grey_list* nodes are randomly promoted to the **white_list**, and once this happens, they move to the top of the **whitelist** due to timestamp updates.

Let $P^{(i)} = \{P_1, \ldots, P_{n_i}\}$ be the set of n_i TCP packets the observing node receives from node i in the time window t; t is the period of the experiment. We may want to discard $P^{(i)}$ from the dataset when n_i is too small. We define an experimental sample S, a sample of nodes we detected, then $\forall i \in S$ we can find the relative presence p_i of observed addresses a as:

$$p_i(a) = \frac{\sum_{j=1}^{n_i} \chi(a \in P_j)}{n_i} \tag{1}$$

where χ is an indicator function with 1 if $a \in P_j$ is true and 0 otherwise. We count the number of times an address appears in the peer lists returned by node i.

We simulate a simple experiment to illustrate the peer selection process in the Monero P2P protocol. Suppose a node has 8 neighbors. In 100 handshakes, the node receives a list of 250 peer IP addresses each time. According to the Monero protocol, all 8 neighbours consistently appear within the top 300 candidates in the list. Peer IP addresses are randomly selected from this top 300. Consequently, each neighbour has a uniform probability of being selected during a handshake, calculated as: $P_{\text{neighbour}} = \frac{250}{300} \approx 0.833$.

For non-neighbor nodes, which are not guaranteed to appear in the top 300 candidates, the selection involves two stages:

- A random node enters the top 300 candidate pool with probability: $P_{\text{enter}} = \frac{300}{992} \approx 0.302$, where 992 is the total number of non-neighbour nodes, calculated as $1000 - 8$.
- Given that a random node is in the top 300, it has the same selection probability as any other node in the pool: $P_{\text{selected}} = \frac{250}{300} \approx 0.833$.

Combining these probabilities, the overall probability for a random node to be selected is: $P_{\text{random}} = P_{\text{enter}} \cdot P_{\text{selected}} = 0.302 \cdot 0.833 \approx 0.252$.

This creates a notable frequency disparity between neighbours and random nodes. For example, the mean cumulative frequency of neighbour nodes 0.833 is more than three times that of random nodes 0.252. This discrepancy reflects the structural bias of the protocol, as neighbour nodes are always included in the candidate pool, while random nodes must first pass a filtering stage. Consequently, neighbour nodes exhibit consistently higher selection frequencies, resulting in a relatively distinct frequency distribution.

D. *Neighbor identification by k-means*

The above trivial method and synthetic calculating intuitively identify high-frequency peer interactions compared to low-frequency peer interactions. This observation led us to apply k-means clustering to separate high-frequency links from low-frequency ones.

To identify real neighbors from the collected data, we first gather the frequency of each peer's connections appearing in shared *peer_lists* or TCP flows. Then, we apply the k-means clustering algorithm to differentiate between high-frequency and low-frequency links as follows:

Algorithm 1 Data Filtering and Neighbor Inference Process

```
Require: Raw dataset $D$ with columns ip1, ip2, count.
 1: $C_{\text{min}} \leftarrow 2$, threshold of minimum connection count.
 2: $N_{\text{min}} \leftarrow 8$, threshold of minimum number of out-going neighbors.
 3: $D_{\text{filtered}} = \{row \in D : row[\texttt{count}] \geq C_{\text{min}}\}$
 4: for each unique source IP $s$ in $D_{\text{filtered}}$ do
 5:     $G_s = \{row \in D_{\text{filtered}} : row[\texttt{ip1}] = s\}$
 6:     if $|G_s| \geq N_{\text{min}}$ then
 7:         Perform $k$-means clustering on the unique count values in $G_s$ with $k = 2$. Greater count values' labels are set as 1 and as 0 otherwise.
 8:     end if
 9: end for
10: $D' = \{row \in D_{\text{filtered}} : row[\texttt{label}] = 1\}$
11: return $D'$
```

The dataset consists of triplets $(\texttt{ip1}, \texttt{ip2}, \texttt{count})$, where `ip1` and `ip2` represent IP addresses, and `count` indicates the frequency of interactions between them. We also removed all the rows where `ip1` is either Singapore, US, or EU, since our nodes are performing high-frequency handshakes constantly to receive the TCP packets. To identify relevant connections and infer real neighbours, we applied a series of filtering and clustering techniques as follows:

First, we remove rows where $\texttt{count} \leq 1$, as interactions that only appeared once are likely to represent noise or insignificant connections. The remaining dataset, denoted as D_{filtered}, contains pairs with meaningful interaction frequencies.

Secondly, to analyze the connections for each source IP (`ip1`), we grouped the data by `ip1`, yielding a set of groups G_s, where each group corresponds to all connections originating from a specific source IP address `ip1`. For each group G_s:

1) **Minimum Connection Threshold**: Groups with fewer than two times interaction ($|G_s| < 2$) were discarded, as these interactions only appear once in the data, and do not need to be considered for clustering filter.
2) **k-means Clustering**: For groups with $|G_s| \geq 2$, we performed k-means clustering on the unique `count` values with $k = 2$. A peer appearing in the peer list of another peer is either its true or non-true connection, which explains our setting $k = 2$. The reason why we perform k-Means method on unique $(count)$ values is to exclude the effect of most low-frequency interaction pairs. The clustering method will divide the connections into two clusters based on frequency. The `ip2`s with higher interaction frequency pairs are labeled as 1 and as 0 otherwise. A value of 1 indicates that the corresponding peer is considered a true neighbor by our method.
3) **Filtering Connections**: All connections labeled as 1 were retained which is denoted as D'. This dataset includes only the interactions based on frequencies and the clustering process, which represent the inferred relevant connections/neighbours for each source IP. The final resulting dataset D' was written to an output file for further analysis. Each row in D' corresponds to a high-confidence connection between two IPs, where the relevance of the connection was determined using frequency-based clustering.

By applying k-means clustering on frequency data, we can identify clusters of interactions with comparatively different patterns, and peers with higher frequency are inferred as the most likely real neighbours of given peers. The k-means method's computational efficiency is very high and has the ability to highlight meaningful patterns, especially in large-scale interaction datasets just like our dataset.

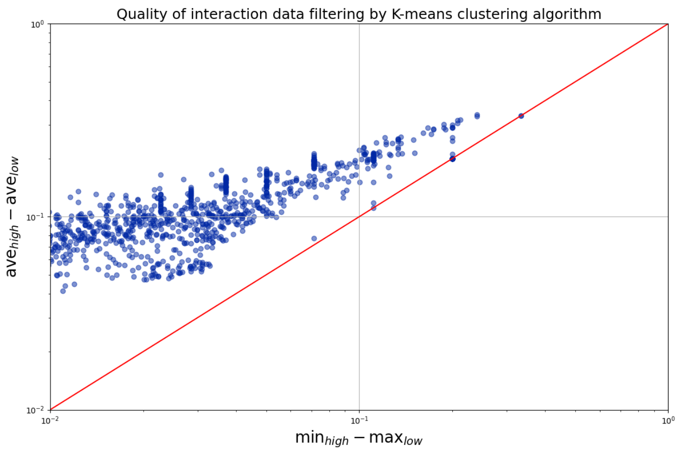

Fig. 2: The difference between the minimum frequency in the high-frequency cluster and the maximum frequency in the low-frequency cluster ($\text{min}_{high} - \text{max}_{low}$), versus the difference in average connection frequency between the high- and low-frequency clusters ($\text{ave}_{high} - \text{ave}_{low}$). Each spot represents a peer (node); the farther a spot is from the slope, the better the clustering result.

To evaluate the reliability of k-means in clustering by frequency data, we calculate "the difference between the minimum frequency in the high-frequency cluster and the maximum frequency in the low-frequency cluster ($\text{min}_{high} - \text{max}_{low}$)" and "the difference in average connection frequency between the high- and low-frequency clusters ($\text{ave}_{high} - \text{ave}_{low}$)". The result of k-means on our dataset is shown in Fig.3. The relatively large distance between the blue nodes and the red slope suggests that k-means achieves excellent performance in clustering true neighbours based on frequency.

E. Neighbor inference validation

We compare the inferred neighbors of our Singapore, EU, and US nodes with the real connection lists obtained using Monero's monero-daemon-rpc. These real connection lists serve as a benchmark for validating the neighbor inference.

Since peers dynamically connect and disconnect by time, even the real connection list represents a snapshot of transient connections. To validate our method, we first identify the neighbors of our nodes in the inferred network and then determine how many of these inferred neighbors appear in the real connection lists during our data collection time.

We calculate the accuracy rate as the ratio of "Number of our node's neighbors in the inferred network" to "Neighbors identified from real connections over one week of data." The following tables present the validation results.

TABLE I: One week data neighbor inference accuracy

Monero P2P network Benchmark comparison for one week data			
Location	***Singapore***	***US***	***EU***
Number of our node's neighbors in the inferred network	489	534	534
Number of neighbors appears in our node's connection list	388	367	371
Neighbor identification Accuracy rate for one week data	79.35%	68.73%	69.48%

TABLE II: Three weeks data neighbor inference accuracy

Monero P2P network Benchmark comparison for three weeks data (the data we use in this paper)			
Location	***Singapore***	***US***	***EU***
Number of our node's neighbors in the inferred network	394	437	713
Number of neighbors appears in our node's connection list	327	319	572
Neighbor identification Accuracy rate for three weeks data	82.99%	73.00%	80.22%

Without incorporating the *last_seen* timestamps, the prediction accuracy of our inference method is already relatively high. This result in Table I is based on only one week of listening data, excluding the *last_seen* timestamps in the peer lists under the current protocol. When the data collection period was extended, the accuracy improved by around 10% (check Table II), highlighting the significant impact of a larger dataset collected over a longer duration on prediction accuracy. This improvement aligns with the theoretical expectation that longer TCP listening times enhance accuracy.

IV. Dogecoin Peer protocol

In Dogecoin, peer discovery follows a Bitcoin series peer-to-peer protocol in which each node maintains a local address database (`addrman`) composed of two sets: a *tried* table containing peers with which successful connections have previously been established, and a *new* table containing addresses learned through peer announcements but not yet contacted. Addresses are added to or updated in this database upon receipt of `addr` messages from other peers.

When processing an incoming `addr` message, address entries are incorporated according to predefined update rules. Timestamps associated with advertised addresses are subject to a fixed time penalty ($2h$) and are only updated if the newly received timestamp exceeds the previously stored value by a minimum update interval, which depends on whether the address is considered recently active. Addresses that have been successfully connected to and placed in the *tried* table are no longer updated through subsequent `addr` announcements.

Addresses that remain in the *new* table may be referenced multiple times by different peers. However, insertion into multiple storage buckets and subsequent updates are governed by probabilistic rules, and the assignment of addresses to buckets depends on both the advertised address and the announcing peer.

In response to `getaddr` requests, a node does not return its current neighbors, but instead, a node returns a randomly sampled subset of entries from its local address database, excluding addresses considered invalid or of low quality. The number of returned addresses is bounded by a fixed fraction of the database size, typically up to approximately 23%, and by an absolute upper limit of 1000 entries per response.

A. Motivation

In contrast to Bitcoin, which has received continuous attention and frequent protocol updates, the peerlist shearing is restricted to no-update caching inside $24hs$ to any connection, Dogecoin largely retains an older Bitcoin-style peer discovery mechanism and has not incorporated many of these recent changes[4]. This motivates our exploration of whether meaningful features can still be observed from Dogecoin's peer protocol and whether the network topology can be inferred under these constraints.

B. Dogecoin data collection

Accordingly, we collect TCP flow-level data from the Dogecoin network using *getaddr*-based peer discovery, following measurement approaches similar to those applied in Bitcoin, while accounting for protocol-level differences in address dissemination.

C. data description

we use two connectors,which mentioned above, collected 10 days of the tcp flow data and decode the tcp flow information into readable peerlist file.

- In the tcp flow raw data, there are 559 unique IPs.
- In our sigaple groundtruth node, there are 919 unique IPs.

a) TCP-derived peer address observation fields.: Peer discovery traffic was extracted from TCP flow–level measurements by parsing peer address announcements exchanged over established peer-to-peer connections. Each observation corresponds to the appearance of a peer address x in an

[4]Dogecoin Core addrman implementation

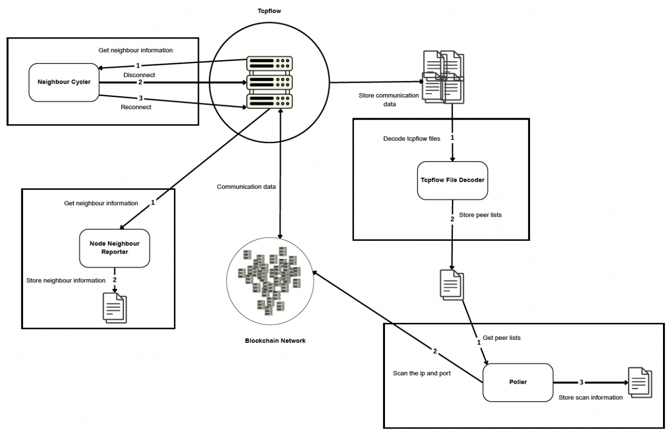

Fig. 3: Dogecoin peerlist data collection

`addr` message received by the measurement node. For each observation, we record the following fields:

- **Source IP and port** $(s_{\text{ip}}, s_{\text{port}})$: the network address of the peer that transmitted the `addr` message containing the advertised address.
- **Destination IP and port** $(x_{\text{ip}}, x_{\text{port}})$: the advertised peer address contained in the `address ip` message.
- **Timestamp of observation** (t_{obs}): the local time at which the `addr` message was received by the measurement node.
- **Advertised last-seen time** $(t_{\text{last\_seen}})$: the timestamp carried within the `addr` entry, representing the sender's locally stored time associated with address x.
- **Time difference** (Δt): the difference between the observation time and the advertised timestamp, defined as $\Delta t = t_{\text{obs}} - t_{\text{last\_seen}}$.

In subsequent analysis, address observations may be filtered or aggregated according to additional criteria derived from these fields, such as bounds on Δt or uniqueness constraints on advertised timestamps across different sources.

D. neighbor inference based on the dogecoin peerlist data

TABLE III: Inference of directed edges between nodes A and B based on the uniqueness and time difference of their last_seen timestamps.

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 $A \nrightarrow B$
Unique $ts <$ 2hr	$\exists$ edge $A \rightarrow B$	$\exists$ edge $A \leftrightarrow B$	$\exists$ edge $A \rightarrow B$
Not unique $ts <$ 2hr	Unclear $B \nrightarrow A$	$\exists$ edge $B \rightarrow A$	Unclear

Table III summarizes the idealized rules for inferring directed peer connections based on the relative freshness and uniqueness of exchanged *last_seen* timestamps. Under the assumption of stable connectivity, negligible relay delays, and the absence of protocol-level obfuscation, reciprocal and timely observations provide evidence of direct connections, while stale or non-unique timestamps indicate indirect propagation. These rules represent a conceptual baseline for neighbor inference and do not account for protocol countermeasures or gossip-induced distortions observed in real networks.

E. Problem of dogecoin peerlist under current protocol

The inference in the above table doesn't work anymore under current protocol. since the outbound connection rules chaned into: no update if the outbound neighbor is still connected, only update the timestamp when the connection is broken[5]. so what we expected was, even the protocol updated, the real neighbor still may have a higher frequency.

a) Link frequency of True and False inference: According to the result Figure 4, shows the distribution of advertised last-seen timestamps for addresses x observed in SI-initiated `getaddr` responses in the direction SI $\rightarrow$ x, restricted to Δt $<$ 2.5 hours. Addresses are labeled according to ground-truth neighbor information. Both true neighbors and non-neighbors exhibit a pronounced peak in the range of approximately 120–150 minutes, and their timestamp distributions largely overlap. Non-neighbor addresses substantially outnumber true neighbors across the entire range, including in the region of most recent timestamps. This indicates that, under the freshness constraint, advertised last-seen timestamps alone do not provide sufficient discriminatory power to reliably distinguish true neighbors from non-neighbors.

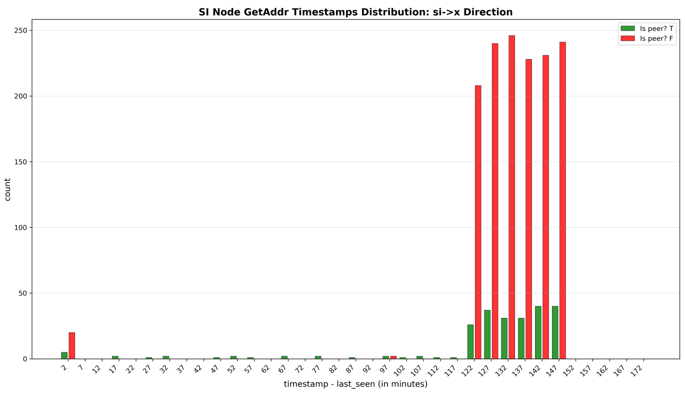

Fig. 4: Dogecoin peerlist true and false neighbor frequency analysis

b) Temporal increments of advertised timestamps.: Figures 5 and 6 analyze the temporal dynamics of advertised peer timestamps by considering the increments of successive last-seen values observed for each address x in SI-initiated `getaddr` responses. For a given address, we define the *last-seen increment* Δt_s as the difference between two consecutive advertised timestamps associated with that address. Intuitively, Δt_s captures how frequently the stored last-seen time of an

[5]Dogecoin Core addrman outbound timestamp rules update

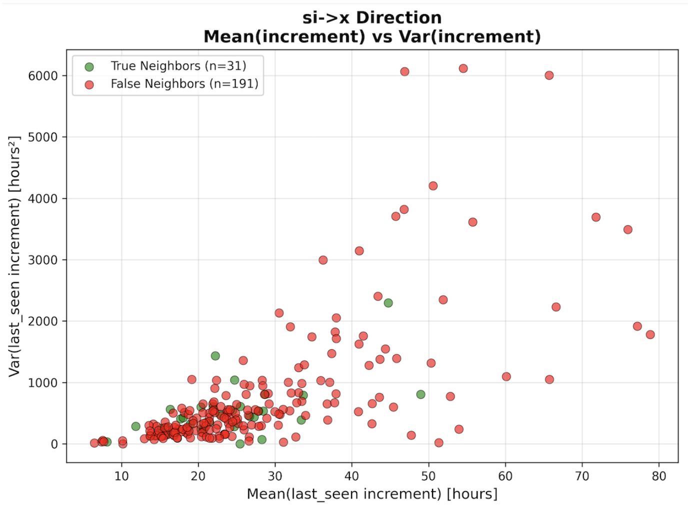

Fig. 5: Dogecoin peerlist true and false neighbor delta_ts analysis

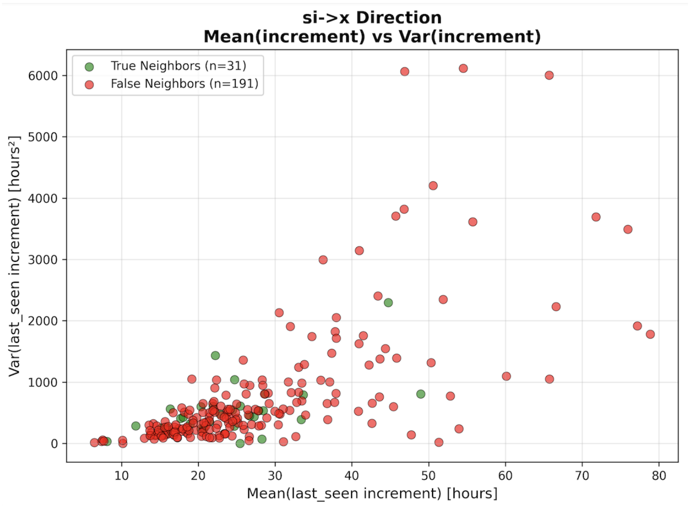

Fig. 6: Dogecoin peerlist true and false neighbor delta_ts analysis

address is updated in the peer address database, rather than the absolute freshness of the timestamp itself.

For each address, we compute the mean and variance of Δt_s over the observation period and compare these statistics between ground-truth neighbors and non-neighbors. In both figures, true neighbors and false neighbors occupy largely overlapping regions in the mean–variance space. Most addresses cluster at moderate mean increments with relatively low variance, while a smaller number of addresses exhibit large mean increments and high variability. Notably, false neighbors dominate across the full range of observed values, including regions with low mean and low variance, where true neighbors are also present.

These results indicate that the temporal increment statistics of advertised last-seen timestamps do not provide a clear separation between true and false neighbors. Even when considering higher-order temporal features beyond absolute timestamp freshness, the update behavior of advertised timestamps remains strongly influenced by protocol-level constraints, such as time penalties, minimum update intervals, and probabilistic acceptance of updates. As a consequence, mean and variance of last-seen increments alone are insufficient for reliable neighbor inference in Dogecoin peer address data.

V. Conclusion

In conclusion, K-means clustering enables each node to obtain its own classification of true and false neighbors, provide a novel method for mapping the previously uncharted terrain of the Monero peer-to-peer network by detecting real neighbors between nodes. Our inference approach demonstrates high accuracy in uncovering the network's structure, offering an intuitive visualization that enhances understanding of its topology. The findings provide the first comprehensive insights into the connectivity patterns within the Monero P2P network under the new peer protocol, shedding light on its complex landscape.

In contrast, applying classification methodology to Dogecoin reveals fundamental limitations imposed by the peer discovery protocol. Under the current Dogecoin design, true neighbors and non-neighbors exhibit highly overlapping temporal and statistical characteristics in `getaddr` data. As a result, features identified at a single measurement node do not generalize to reliable classification of peer lists across the network.

Together, these results highlight a protocol-driven (result in data-driven) divergence in inferability. While Monero exposes sufficient structure to support neighbor identification and network mapping, the Dogecoin peer protocol deliberately obscures distinctions between neighbors and non-neighbors. This comparison underscores how protocol design choices directly determine the feasibility of network inference from passive peer discovery data.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of lipschitz-hankel type involving products of bessel functions," *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 247, no. 935, pp. 529–551, 1955.

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[3] R. Böhme, N. Christin, B. Edelman, and T. Moore, "Bitcoin: Economics, technology, and governance," *Journal of economic Perspectives*, vol. 29, no. 2, pp. 213–238, 2015.

[4] A. M. Antonopoulos and D. A. Harding, *Mastering bitcoin*. " O'Reilly Media, Inc.", 2023.

[5] N. El Ioini and C. Pahl, "A review of distributed ledger technologies," in *On the Move to Meaningful Internet Systems. OTM 2018 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part II.* Springer, 2018, pp. 277–288.

[6] J. Buford, H. Yu, and E. K. Lua, *P2P networking and applications*. Morgan Kaufmann, 2009.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Satoshi Nakamoto*, 2008.

[8] V. Buterin, "A next-generation smart contract and decentralized application platform," *Ethereum White Paper*, 2014.

[9] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE transactions on software engineering*, vol. 47, no. 10, pp. 2084–2106, 2019.

[10] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[11] L. M. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO)*. Ieee, 2018, pp. 1545–1550.

[12] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on {Bitcoin's}{peer-to-peer} network," in *24th USENIX security symposium (USENIX security 15)*, 2015, pp. 129–144.

[13] T. Neudecker, P. Andelfinger, and H. Hartenstein, "A simulation model for analysis of attacks on the bitcoin peer-to-peer network," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 1327–1332.

[14] L. Bahack, "Theoretical bitcoin attacks with less than half of the computational power (draft)," *arXiv preprint arXiv:1312.7013*, 2013.

[15] S. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. Knottenbelt, "Sok: Decentralized finance (defi)," in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, 2022, pp. 30–46.

[16] M. J. M. Chowdhury, M. S. Ferdous, K. Biswas, N. Chowdhury, A. Kayes, M. Alazab, and P. Watters, "A comparative analysis of distributed ledger technology platforms," *IEEE Access*, vol. 7, pp. 167 930–167 943, 2019.

[17] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in bitcoin," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 906–917.

[18] T. Neudecker, P. Andelfinger, and H. Hartenstein, "Timing analysis for inferring the topology of the bitcoin peer-to-peer network," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. IEEE, 2016, pp. 358–367.

[19] V. Deshpande, H. Badis, and L. George, "Btcmap: Mapping bitcoin peer-to-peer network topology," in *2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*. IEEE, 2018, pp. 1–6.

[20] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring ethereum network peers," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 91–104.

[21] K. Wüst and A. Gervais, "Ethereum eclipse attacks," ETH Zurich, Tech. Rep., 2016.

[22] T. Cao, J. Yu, J. Decouchant, X. Luo, and P. Verissimo, "Exploring the monero peer-to-peer network," in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 2020, pp. 578–594.

[23] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan *et al.*, "An empirical analysis of traceability in the monero blockchain," *arXiv preprint arXiv:1704.04299*, 2017.

[24] Y. Li, G. Yang, W. Susilo, Y. Yu, M. H. Au, and D. Liu, "Traceable monero: Anonymous cryptocurrency with enhanced accountability," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 2, pp. 679–691, 2019.

[25] A. Miller, M. Möser, K. Lee, and A. Narayanan, "An empirical analysis of linkability in the monero blockchain," *arXiv preprint arXiv:1704.04299*, 2017.