TU WIEN Informatics

TECHNISCHE UNIVERSITÄT DARMSTADT

Bitcoin Network Topology Discovery Using Timing Analysis

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Timo Klonowski
Matrikelnummer 12019245

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Mag. Dr. techn. Edgar Weippl
Mitwirkung: Dipl.-Ing. Dr.techn. Johanna Ullrich
Prof. Dr. phil. nat. Marc Fischlin

Wien, 27. Jänner 2023

Timo Klonowski   Edgar Weippl

Informatics

TECHNISCHE UNIVERSITÄT DARMSTADT

Bitcoin Network Topology Discovery Using Timing Analysis

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Timo Klonowski
Registration Number 12019245

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Dipl.-Ing. Mag. Dr. techn. Edgar Weippl
Assistance: Dipl.-Ing. Dr.techn. Johanna Ullrich
Prof. Dr. phil. nat. Marc Fischlin

Vienna, 27th January, 2023

Timo Klonowski	Edgar Weippl

Erklärung zur Verfassung der Arbeit

Timo Klonowski

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 27. Jänner 2023

Timo Klonowski

Bitcoin Network Topology Discovery Using Timing Analysis

Masterarbeit von Timo Klonowski (Matrikelnummer: 2311797)
Tag der Einreichung: 27.01.2023

1. Gutachten: Univ.-Prof. Dipl.-Ing. Mag. Dr. techn. Edgar Weippl
2. Gutachten: Dipl.-Ing. Dr.techn. Johanna Ullrich
3. Gutachten: Prof. Dr. phil. nat. Marc Fischlin
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

TECHNISCHE UNIVERSITÄT WIEN

Fachbereich Informatik
Cryptoplexity

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Timo Klonowski, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Wien, 27.01.2023

T. Klonowski

Danksagung

Ich möchte mich bei meinen Betreuern und Co-Betreuern Edgar Weippl, Marc Fischlin und Johanna Ullrich für ihre Geduld und außerordentliche Unterstützung bedanken, die diese Arbeit erst möglich gemacht haben. Die spannenden Diskussionen mit euch und eure wertvollen Ratschläge haben diese Arbeit maßgeblich geprägt und die immer wieder eingebrachten neuen Ideen und Hinweise haben mich sehr unterstützt. Ein ganz besonderer Dank geht auch an Jakob Rosenblattl, der mit seiner Arbeit den Grundstein für mich gelegt hat. Die Einführung in die Thematik durch dich und deine Unterstützung bei Code- und Verständnisfragen waren mir eine große Hilfe.

Von ganzem Herzen möchte ich allen meinen Freunden innerhalb und außerhalb der Universität danken. Danke für die schöne Zeit, die Lacher und die Ablenkung vom Stress und dafür, dass ihr immer Verständnis hattet, wenn ich mal viel für diese Arbeit zu tun und weniger Zeit hatte. Ihr habt mich stets motiviert und unterstützt. Ganz besonders möchte ich mich bei denen bedanken, die diese Arbeit korrekturgelesen und durch viele hilfreiche Anmerkungen verbessert haben. Florian, für sprachlich-inhaltliche Rückmeldungen. Lukas, für Hilfe bei mathematischen Problemen und außerordentliches Gespür für Schreibstil.

Nicht zuletzt möchte ich mich bei meiner Familie, meinen Geschwistern und vor allem meinen Eltern bedanken. Ihr habt mich während meines gesamten Studiums immer unterstützt und mir den Weg gewiesen. Ohne euch wäre mein Studium niemals möglich gewesen.

Kurzfassung

Die Kryptowährung Bitcoin besteht aus einer Menge von Minern, die zu einem Peer-to-Peer-Netzwerk verbunden sind. Innerhalb dieses Netzwerkes werden Nachrichten mithilfe eines Gossip-Protokolls effizient zwischen Clients weitergeleitet. Die genaue Topologie dieses Netzwerks soll dabei geheim gehalten werden; jeder Miner kennt nur die Verbindungen zu seinen Nachbarn sowie eine Auswahl an möglichen Adressen potenzieller Verbindungspartner. Eine Kenntnis der Topologie würde es Angreifern ermöglichen, bestimmte Arten von Angriffen durchzuführen, und die Anonymität der Nutzer gefährden. Allerdings erlaubt die Messung der Topologie Forschern auch, die Eigenschaften des Netzes zu analysieren.

Wir schlagen eine Methode vor, die die Weiterleitung von Adressnachrichten im Peer-to-Peer-Netzwerk ausnutzt, um dessen Topologie zu ermitteln. Zu diesem Zweck kombinieren wir eine Schätzung des Knotengrades mit einer Inferenz von potenziellen Verbindungen. Insbesondere zeigen wir, dass Eigenheiten bei der Weitergabe von ADDR-Nachrichten im Rahmen des Gossip-Protokolls diese Messmethoden ermöglichen. Dies gilt, obwohl die Grundidee der beiden Angriffe bereits bekannt ist und die Gegenmaßnahmen Trickling und Ratenbegrenzung bereits in der Referenzimplementierung umgesetzt sind. Wir zeigen, dass diese Schutzmechanismen die Messungen nicht verhindern und, dass beide Arten von Messungen gemeinsam ausgeführt werden können, um eine umfassende Messung der Topologie mit erheblicher Genauigkeit durchzuführen.

Dabei vergleichen wir auch Schätzungsmethoden, welche auf idiosynkratischem Bitcoin-spezifischem Verhalten basieren, mit einer statistischen Timing-Analyse von Flooding-inhärenten Zeitverzögerungen. Wir validieren unsere Methode an einem Testbed-Knoten und stellen fest, dass vor allem der zeitbasierte Schätzer die Topologie trotz künstlich eingeführter Weiterleitungsverzögerungen mit signifikanter Präzision schätzen kann. Der relative Fehler der Gradabschätzung ist kleiner als 10 % und die Genauigkeit und Trefferquote der Verbindungsinferenz liegen jeweils bei etwa 40 %.

Des Weiteren führten wir eine aktive Messung im offenen Bitcoin-Mainnet durch und analysierten die graphentheoretischen Eigenschaften der gefundenen Topologie. Unser Angriff kann mit wenigen, leistungsschwachen Systemen sowie in kurzer Zeit durchgeführt werden. Die Analyse zeigt, dass das Netzwerk nicht-zufällige Eigenschaften in seiner Struktur aufweist, vor allem in der Verteilung der Knotengrade und der lokalen Clusterkoeffizienten.

Abstract

The cryptocurrency Bitcoin consists of a set of miners that are connected to a peer-to-peer network. Within said network, messages are efficiently forwarded between clients using a gossip protocol. The exact topology of this network is supposed to be kept secret; each miner only knows the connections to their neighbours, as well as a set of potential peers' addresses. Knowledge of the topology would enable attackers to launch certain attacks and compromise the anonymity of clients. In contrast, measuring the topology also allows researchers to analyse the properties of the network.

We propose a method to exploit the relaying of address messages in the network to measure the topology of the network. To do this, we combine a node degree estimation with an inference of potential connections. In particular, we show that peculiarities in the propagation of ADDR-messages in the context of the gossip protocol enable these attacks. This holds true despite the basic idea behind both attacks being known already and the countermeasures trickling and rate-limiting having been implemented by the reference client. We show that said countermeasures do not prevent these measurement methods and that both methods can be executed together to perform a comprehensive topology discovery with considerable accuracy.

In doing so, we also compare estimation methods based on idiosyncratic Bitcoin-specific behaviour with a statistical timing analysis of flooding inherent time delays. We validate our method on a testbed node and find that, in particular, the time-based estimator can estimate the topology with significant accuracy despite artificially introduced forwarding delays. The relative error of the degree estimation is less than 10 %, and the connection inference's precision and recall are approximately 40 % each.

We performed an active measurement on the open Bitcoin mainnet and analysed the graph-theoretic properties of the topology we found. Our attack can be performed with low hardware expenses as well as in a short time. The analysis shows that the network exhibits non-random properties in its structure, especially in the distribution of node degrees and local clustering coefficients.

Contents

CHAPTER 1

Introduction

Bitcoin is a prime example of a cryptocurrency with a strong focus on pseudonymity and decentrality, two merits which have been its focus ever since its introduction in 2008 [Nak08]. It is based on a peer-to-peer network specifically designed to be robust so that even in the presence of attackers, the transaction data essential for the currency can still be exchanged between peers. In this sense, the network ought to be resistant to disruptions, while being able to provide the service it offers with high throughput [XCW05]. The network should also function with as little central authority as possible in order to avoid single points of failure and remain true to the ideal of a decentralised currency with no controlling authorities [Nak08]. In particular, regarding the exchange of network participants' addresses, this means that individual nodes should not be used to inform nodes about the presence of other nodes, as it is the case in a conventional hierarchical network; if possible, individual DNS-like services should not have to be relied on to maintain the network. For this purpose, Bitcoin uses a so-called *gossip protocol* [NAH16].

The aim of this procedure is to exchange addresses between network participants in a coordinated manner such that, in the end, everyone knows everyone else without a central authority keeping record. Analogous to gossip between people, in a gossip protocol between computers, information is passed on between hierarchically equal neighbours until the desired information has permeated the network and reached every participant [Wei19]. A number of different protocol message types are subject to the gossip protocol in Bitcoin, one of them being the so-called *ADDR*-messages. These contain information about IP addresses through which other Bitcoin clients can be reached. Such messages are *flooded* throughout the entire network, which means that everyone shares them with their direct neighbours according to certain rules. When these neighbours receive the messages, they pass them on to their own neighbours. This then leads to a series of forwardings such that the original message — for example, about the presence of a new network participant — reaches every other node in the connected network [Prob]. For this procedure, it is not necessary for a participant to know the *topology* of the network at any time, i.e., which peer is connected to whom; to flood the messages, it is sufficient for each node to know its own neighbours, with the gossip protocol providing the message distribution necessary for network operation without central servers.

In recent years, Bitcoin has grown in size, and with it has the need for security analyses of the currency. After all, a well-founded and comprehensive security analysis is essential, especially in the field of cryptocurrencies, to strengthen trust in the currency. Naturally, in the case of cryptocurrencies, the focus often lies on the analysis of weaknesses in the cryptographic part of the blockchain layer of the currency. Less attention has been paid to the network part of the protocol in previous works [HKZG15]. However, a securely functioning network layer is equally important for the orderly operation of Bitcoin. This is due to the fact that attacks on vulnerabilities of network aspects can cause a wide range of undesirable, disruptive effects. Attacks such as an eclipse attack, for example, can specifically exclude participants from currency operations and thus cause serious financial damage to users or groups of users. Such attacks are partly prevented by the fact that Bitcoin's peer-to-peer network protocol attempts to hide the topology of the network in order to impede the execution of these kinds of attacks [DSBPS+19].

Therefore, one way of looking at Bitcoin's network security is through the lens of topology discovery. This is an attempt to measure the structure of the network as accurately as possible, in spite of the mechanisms Bitcoin employs to prevent exactly that. If, even in theory, knowledge of the topology enables attacks, then it is of utmost interest to the Bitcoin community to close any loopholes so as to keep the currency secure. A demonstration that such a measurement is possible would, therefore, be of relevance for the community [DSBPS+19].

In addition to the Bitcoin community's interest in keeping the currency secure by hardening the network protocol, there is also academic interest in analysing the network. The fact that the network tries to make itself non-transparent also means that it is difficult for researchers to make statements about its structure. However, it would be interesting to monitor whether the network's characteristics are suitable for a peer-to-peer network designed for robustness [DBG18, EPJ20]. Metrics that can be calculated for a network graph can provide information about whether Bitcoin exhibits appropriate properties in this regard. These metrics can only be calculated if the topology is known [CLA16]. In this respect, the deliberate lack of transparency in the network prevents further analysis of the network. However, topological security assessments are a desirable way of assessing whether the structure of the network is fundamentally appropriate to fulfil defined security goals, or if further adjustments to the protocol are necessary. Therefore, there is a vested academic interest in utilising this chance to demonstrate the possibility of a topology measurement as a means of analysing the network structure. From a scientific perspective, the existence of a topology discovery vulnerability would thus be an occasion to study the network in terms of these properties [DSBPS+19].

A promising approach to topology discovery that was presented in related works in the past is the so-called *timing analysis*. This involves sending specially crafted messages from researchers to participants in the Bitcoin peer-to-peer network. The recurrence of these messages elsewhere in the network can then be observed, and the time it took to do so is measured. Afterwards, the measured time delay is used to make statements about topological properties of the network along the message's path, such as the approximate length of the path. If this process is repeated often enough and evaluated statistically, it can be used for finding connections within the network. This provides a way to measure the graph based solely on time delay characteristics inherent to any computer network: messages are always time-delayed to some extent when they are passed between communication partners [NAH16].

This is contrasted by a topology discovery based on behavioural properties that are not inherent to the network, but rather due to peculiarities in the implementation of the protocol. Such an approach takes advantage of the fact that the behaviour of network participants in the Bitcoin protocol is predictable. After all, the source code for the most wide-spread client software, the Bitcoin Core client, is open source. This means, in particular, that the Bitcoin network protocol specification is public. However, the prerequisite for this type of topology discovery is that the manner in which a network node handles a packet during flooding is sufficiently characteristic. Therefore, the observable behaviour of the client must be clear and idiosyncratic enough to allow for conclusions to be drawn about connection properties [Wei19]. In past work, examples of such idiosyncratic behaviour were for example the way clients internally assigned timestamps to connections [MLP+15], or how some data packets are not forwarded via certain connections under specific conditions [GNH18]. After identifying sufficiently characteristic protocol behaviour, the observation of packets can generally be used to infer connections between clients.

In this thesis, we address whether topology discovery is possible in Bitcoin in one of the following two ways: by statistical timing analysis or by using predictable idiosyncratic behaviour alone. More specifically, we focus on the way ADDR-messages are forwarded in Bitcoin's peer-to-peer network protocol for both approaches. As described above, these messages are flooded through the network following a gossip procedure. This is done for each node according to well-defined rules, which provide reasonably predictable behaviour as we show in this thesis. Thus, ADDR-message flooding exhibits both of the properties we wish to analyse. We compare the two types of analyses in regard to the degree to which topology discovery in Bitcoin through ADDR-message flooding analysis is possible through either method.

Here, the topology discovery attack we perform takes place in two steps. First, a degree estimation is performed on nodes. In this process, the node degree of all network participants is estimated based on the measured behaviour. Then, in a second step, connection inference is performed. This involves determining the probability of a given connection's existence in the measured network for all possible node links. We show that it is possible to perform both of these procedures — degree estimation and connection inference — on the basis of the same measured behavioural data. This allows both partial estimates to be conducted in the same experiment run while using the same setup.

A behaviour-based method for degree estimation was presented by Biryukov et al. in

2014 [BKP14] and carried out network-wide by Grundmann et al. in 2021 [GBH21]. The researchers assume that a rate-limiting for address messages introduced since then now prevents this kind of attack [GBH21]. We make use of the same basic idea for degree estimation in this thesis. This allows us to test the extent to which this countermeasure does indeed work.

We also examine the effectiveness of countermeasures already implemented explicitly for this purpose by performing connection inference. To do this, we use a method presented by Neudecker et al. in 2016. The researchers performed a timing analysis of messages forwarded in Bitcoins gossip protocol to detect connections. For this, they focused on block announcement message (*INV*) relay. Furthermore, they presented the possibility of artificially added time delays, so-called *trickling*, to prevent attacks of this type [NAH16]. This trickling has since been implemented for the Bitcoin protocol [Prob]. We therefore apply connection inference following this idea to verify the effectiveness of this measure. To better link connection inference with degree estimation, we use the observation of the Bitcoin message type ADDR instead of INV for this purpose. We find that both message types are flooded in a similar manner and thus exhibit comparable vulnerability to timing analysis.

The goal of this thesis is to implement a framework for topology discovery. This framework implements the Bitcoin network protocol, which allows the framework to connect to clients on the network and send and receive messages. Using this framework, we performed an active network measurement of a subnet of the reachable Bitcoin network. This was done by injecting ADDR-messages and measuring the time delay until their eventual reoccurrence. We show how this time difference data can be passed to an estimator to make classification decisions about node degrees and the existence of connections between nodes. In this thesis, the classifier is operated in two modes: First, the estimation is performed only on the basis of the forwarding frequency of the ADDR-messages. As described above, a sufficiently characteristic property in the relay algorithm is used to estimate node degrees and connections. In the second mode, time delay information is added to obtain additional information about possible connections according to a timing analysis. We subsequently use the connection data estimated in these ways to form a network graph. Using this model, we examine and discuss graph-theoretic properties employing network metrics relevant to peer-to-peer networks. In short, we apply topology discovery based on ADDR-message relay to assess the network design in terms of robustness and throughput.

To validate the results, we determine the performance of the framework in a qualitative evaluation. For this purpose, a testbed is set up in which a local victim node under our control is measured. The framework's estimate of the node's neighbours is then compared with the actual known neighbours to calculate the precision and recall of the estimator. In particular, the performance of the two estimation modes is benchmarked and compared to conclude whether the addition of timings in the estimation provides an advantage over estimation based on reoccurrence rates alone. Subsequently, we compare the accuracy metrics of the estimators and the network metrics to values from similar experimental setups from related works.

Fundamentally, this master thesis is dedicated to whether topology discovery using time delay or behavioural analysis is possible in the gossip protocol of Bitcoin's address transmission. The research questions are therefore:

- **RQ1:** *Does the implementation of rate-limiting for ADDR-message relay in Bitcoin prevent the conduction of degree estimation?*
- **RQ2:** *Does the implementation of trickling for ADDR-message forwarding in Bitcoin prevent the conduction of timing analysis?*
- **RQ3:** *Are the topological properties of the Bitcoin network appropriate for a network designed for resilience and information distribution?*

For reasons that are discussed further in the "Discussion" section, the framework only covers nodes that can be reached via IP and explicitly does not cover Onion-nodes. Thus, this thesis is to be understood as an analysis of a subnetwork of the entire Bitcoin network. Furthermore, this thesis was written at the same time as the thesis of Jakob Rosenblattl [Ros] about the theoretical aspects of the topology discovery attack using timing analysis. Rosenblattl's thesis also provides the time-based estimator implementation. The development of this estimation mode is therefore not part of this thesis. Additionally, a code basis from which to build the measurement framework was also kindly provided by Rosenblattl. For reference, an overview of his work can be found at `https://gitlab.sba-research.org/jrosenblattl/diplomarbeit`.

The remainder of this thesis is structured as follows: Chapter 2 summarises the required basics of mathematics, network technology, and the Bitcoin protocol. These are necessary to understand the main body of this thesis. In Chapter 3, the general idea and methodology of the measurement are described in detail and justified on the basis of additional small-scale measurements on the Bitcoin network. In Chapter 4, the results of the experiment are analysed in detail, discussed, and their relevance validated using a testbed evaluation. Finally, we give a summary and an outlook on possible further work in Chapter 5.

CHAPTER 2
Background

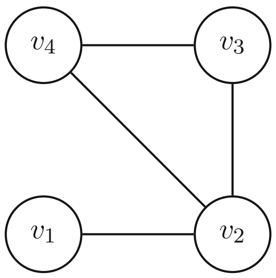
In this chapter, we cover the basics of the topics that are referenced in the following chapters. This includes mathematical fundamentals such as graph theory and classification problems, network theory and metrics, and an introduction to the Bitcoin peer-to-peer network protocol with its most important message types and associated behaviour. An understanding of this is presumed from Chapter 3 onward.

Particularly, Section 2.1 gives an introduction to graph theory and random graphs. This is necessary for the discussion of the Bitcoin network and topology discovery. Section 2.2 provides an overview of classification problems, as topology discovery itself is one, and related concepts from statistics. In Section 2.3, a detailed summary of Bitcoin nodes' behaviour is given. This includes protocol messages and, in particular, ADDR-message relay behaviour, which is the basis of the attack we present in Section 3.1. Section 2.4 presents general concepts of peer-to-peer networks. This includes, in particular, metrics for robustness and performance, as we explore later in this thesis for the Bitcoin network. A subsequent description of attacks on such networks highlights security considerations and the influence of topology discoveries.

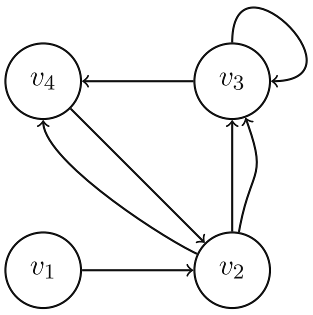
Readers who are already familiar with these topics are invited to skip to Section 2.5, where we give an overview of related work and discuss how we expand on it in this thesis.

2.1 Graph Theory

Graphs mathematically formalize networks like computer networks and allow to use a clear mathematical language with well-defined terms to describe networks. We therefore use this notation to describe the Bitcoin network. To this end, we first give an introduction to the most important concepts and definitions from the mathematical field of graph theory. The definitions are based on standard works by West [W$^+$01], Wilson [Wil79], and Diestel [Die00]. Afterwards, we briefly expand on the concepts of random graphs.



(a) Simple, undirected graph



(b) Complex, directed graph with loops and multiple edges

Figure 2.1: Examples for a simple and a complex graph

2.1.1 Graphs

A *graph* is a pair $G = (V, E)$ where V is a set of *vertices* and $E \subseteq \{\{x, y\}|x, y \in V\}$ is a set of *edges* [BW10]. Graphs are commonly drawn by placing a circle for every vertex and lines from one vertex to another along the edges, see Figure 2.1a and Figure 2.1b.

The following terms are used for graphs [BW10, JD16, Die00]:

- For an edge $e = \{x, y\}$, the two vertices it connects are called *endpoints*.
- An edge $e = \{x, x\}$ is called a *loop* if its endpoints are the exact same vertex.
- When two vertices are endpoints of an edge, they are *adjacent* and *neighbours*.
- The *degree* $\deg(v)$ of a vertex v is the number of edges in G that have v as an endpoint, i.e. the number of neighbours of v.
- *Degree distribution* is the function giving the probability that an arbitrary node has exactly k neighbours [JD16].
- A *walk* is a sequence of edges that lead from one vertex to another.
- If a walk does not contain a vertex multiple times, it is called a *path*.
- A walk that leads from a vertex v to the same vertex v is called a *circle* or *cycle*.
- If a graph contains no cycles, it is *acyclic*.
- For some graphs it is possible to contain so-called *multiple edges*, i.e. separate edges $e_1, e_2 = \{v_1, v_2\}$ connecting the same vertices.

- A graph for which neither multiple edges nor loops exist is called a *simple graph*, see Figure 2.1a.
- *Connected graphs* are graphs where for all pairs of vertices there is a path connecting the two.
- A graph consisting of multiple connected pieces is called *disconnected graph*.
- If for the edges of a graph G the edges are defined as ordered pairs instead of an unordered set, G is called a *directed graph* or *digraph* instead of a *undirected graph*. In this case, the edges are then drawn with arrows to indicate the direction, see Figure 2.1b.
- For edges $e = (v_1, v_2)$ of a directed graph, v_1 is called the *tail* and v_2 is the *head*. Furthermore, v_1 is then the *predecessor* of v_2, and v_2 is the *successor* of $v1$.
- A *subgraph* of a graph G is a graph H for that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We say that G *contains* H.
- The *distance* $distance(v_1, v_2)$ between two nodes v_1 and v_2 is defined as the number of edges on the shortest path between the two nodes.
- *Network diameter* is the maximum distance between any two nodes [JD16].
- For a node v, the *local clustering coefficient* is given by dividing the number of actual links between the nodes in the neighbourhood of v by the maximum number of links that could exist there if the nodes of this neighbourhood formed a fully connected graph [HLY04].
- The *global clustering coefficient* is defined as the arithmetic average local clustering coefficient of all nodes. It measures the degree of compactness for a graph [HLY04].
- If for two subsets A and B of the vertices V all $A - B$ paths contain edge or vertex x from a set X, we say that X *separates* A and B in G, or X separates G.
- A vertex separating two vertices is called a *cutvertex*, and an edge separating two vertices is called a *bridge*.
- The graph G is *k-connected* if no two vertices in G are separated by fewer than k other vertices.
- The *connectivity* of a graph G is defined as the greatest integer k such that G is k-connected.
- An acyclic graph is called a *forest* and a connected forest is a *tree*.
- The *bisection width* is defined as the minimum number of edges between any two partitions of the graph of equal size [LKRG03].
- For a node, the *betweenness centrality* denotes how many shortest paths between all other pairs of nodes traverse that node [ZFDS22].

2.1.2 Erdös-Rényi Random-Graphs

For graphs of unknown structure, we can either assume that they follow a certain probability distribution or are the result of a random process. The latter are called random graphs. The theory of random graphs combines graph theory with probability theory [ER⁺60], and is useful to both 1) model complex graphs and 2) to derive and discuss their properties [DSBPS⁺19, CLA16].

A common formalization of random graphs is that of Erdös and Rényi [ER⁺60]. Here, a random graph is defined as a random undirected graph $G = (V, E)$ with fixed number of vertices $n = |V|$ and fixed number of edges $N = |E|$. It has no multiple edges and no loops. The total number of possible graphs is thus $\binom{\binom{n}{2}}{N}$ and a random graph can be formed in one of the following two ways:

1. One element of the set of all possible graphs is chosen randomly, so that each element has the same probability $\frac{1}{\binom{\binom{n}{2}}{N}}$.
2. The forming of the random graph is considered as a stochastic process: iteratively, one of the possible edges connecting the vertices $V_1, \ldots, V_N$ is chosen in turns, one at a time, and added to the graph. Since each edge can be chosen only once, at time $t = k + 1$ there are exactly $\binom{n}{2} - k$ edges left to choose from, all of which can be chosen with the same probability $\frac{1}{\binom{n}{2}-k}$.

From a mathematical point of view, these two ways of defining a random graph are identical [ER⁺60].

2.2 Classification Problems

The goal of this thesis is to decide whether an actual edge is present or not based on observations of the Bitcoin network. From a mathematical point of view, this is a *classification problem*: there are two possible distinct values for an edge (present or absent) and the task is to define a classifier that can decide, based on observed behaviour, which one of these two values a given edge should be correctly assigned. Even though classification of multiple labels is possible simultaneously, for the sake of simplicity, only *binary classification* in the single label setting is considered in the following. This means that each entity has only one label at a time, i.e. only the membership or non-membership to one set is predicted.

Given is *training data* of the form $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ where each x_i is a *feature vector* of dimension d, and each y_i is a correct binary *label*. A probabilistic classifier then outputs a *model* that gives, for a feature vector, the conditional probability that its label belongs to a particular class. In the single-label setting, the model thus outputs a column vector C with probabilities [LEN14].

The following definitions apply [LEN14]:

- A *decision rule* $D(C) : \mathbb{R}^n \rightarrow \{0,1\}^n$ transforms a vector C of probabilities into binary decisions P.
- *Gold standard* $G \in \mathbb{R}^n$ is the vector containing all true labels of the data.
- A *performance metric* $M(P|G) : \{0,1\}^n \times \{0,1\}^n \rightarrow \mathbb{R} \in [0,1]$ assigns a score to a prediction given the gold standard.
- *True positives* tp denotes the number of all elements, which were correctly assigned to a set.
- *False positives* fp gives the number of all elements that were incorrectly assigned to a set.
- *True negatives* tn counts the number of all elements that were correctly not assigned to a set.
- *False negatives* fn is defined as the number of all elements that were incorrectly not assigned to a set.
- *Precision* $p = \frac{tp}{(tp+fp)}$ measures the fraction of correct assignments within all assignments of the model.
- *Recall* $r = \frac{tp}{(tp+fn)}$ is the fraction of correctly made assignments out of all possible assignments that would be correct for the estimator [1].
- The *F1 score* is defined as $F1 = \frac{2}{\left(\frac{1}{r}+\frac{1}{p}\right)} = \frac{2tp}{2tp+fp+fn}$ and combines precision and recall into a single scalar value

Three similar and mutually related concepts in classification problems are those of *Binary Classification*, *Bipartite Ranking*, and *Binary Class Probability Estimation*.

1. For *Binary Classification*, the goal of the classification model is to predict a label by assigning one of the binary values *true* or *false*
2. For *Bipartite Ranking*, the goal is to learn a ranking model that can rank new objects so that those of one class are ranked strictly higher than those of another class.
3. In the case of *binary class probability estimation (CPE)* a probability shall be given that an object belongs to a class.

[1] In literature, recall is sometimes referred to as *sensitivity* or *true positive rate* [Wei19]. However, in papers dealing specifically with topology discovery in Bitcoin, we encounter the term recall most often. Therefore, we will use this term in this thesis.

These three concepts and their relationships between each other are discussed in detail by Narasimhan and Agarwal [NA13]. The researchers show that a Binary CPE model can be transformed into both a Bipartite Ranking and a Binary Classification Model. Furthermore, there is a weak transform relationship between Bipartite Ranking models and both Binary CPE and Binary Classification models. In the context of this thesis, a Binary CPE is used for edge estimation. This means that a model is used to predict the probability that an edge is present between two given nodes of the Bitcoin Network. The probability is then turned into a Binary Classification by applying a threshold value, as we discuss in Section 3.2.

The application of classifiers often involves the trade-off of different error types. In some application scenarios, a false positive is much worse than a false negative, while in other cases it may be the other way around. We argue that for our application false positives are acceptable up to a certain point: If an attacker wants to find a connected subset of nodes in the network for an attack, a false positive in the edge set means that the set of targets and thus the cost of the attack would increase. On the other hand, a false negative means that an edge is not detected and not the whole desired subset is found, which could then jeopardize the attack. This conclusion is also hinted at by Neudecker et al. [NAH16], as discussed by Weinstock [Wei19].

2.2.1 Precision-Recall Graph

Non-binary classifiers output a score for classifications. This can be converted into binary classifications by applying a threshold in a decision rule. For example, a binary decision can be made by labeling each possible classification with a score below the threshold as *false* and each with a score above it as *true*. Depending on how this threshold is chosen, the resulting classification has different precision and recall. The values for precision and recall are inherently inversely related to each other as the threshold rises: as precision increases, recall generally decreases, and vice versa. The exact ratio of these two values per threshold can then be displayed in a *Precision-Recall (PR) graph*. To build such a graph, the performance of the estimator is determined for each possible threshold and entered into the PR graph as a step function. Thus, it visualizes the tradeoff between precision and recall for an estimator [BG94]. An example of how a PR graph may look like can be seen in Figure 2.2.

2.2.2 Receiver Operating Characteristic

Similar to the PR graph, the *Receiver Operating Characteristic (ROC) graph* also visually represents the performance of an estimator. The ROC graph plots the True Positive rate on the y-axis against the False Positive rate on the x-axis. While a discrete binary classifier again only produces one point in the ROC space, a curve can be formed for non-binary classifiers over all threshold values, just as with the PR graph. An example of how such a ROC graph may look like is shown in Figure 2.3.

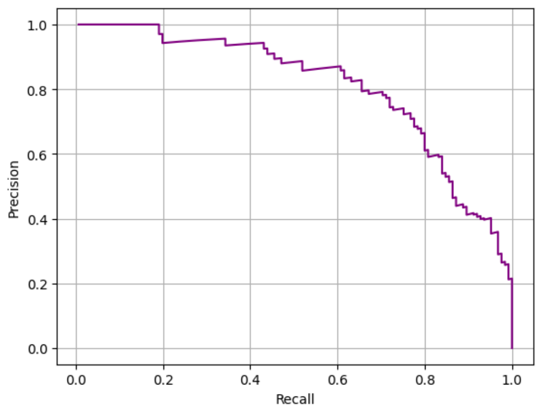


Figure 2.2: Example of a PR graph. The x-axis shows the recall, and the y-axis the precision of a non-binary classifier for various thresholds

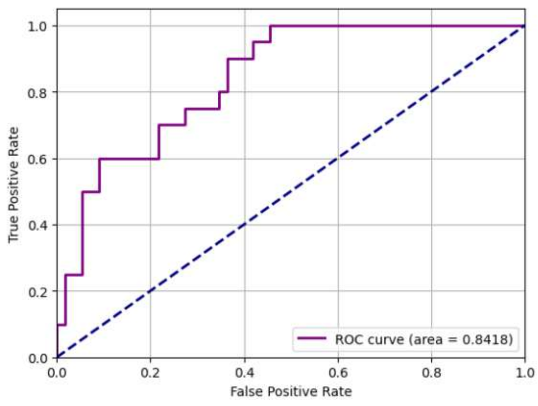


Figure 2.3: Example ROC graph. It shows the true positive rate against the false positive rate of a non-binary classifier for various thresholds

Here, points on the diagonal represent a random guess. Each classifier below the diagonal performs worse than a random guess, but can be turned into a better classifier above the diagonal by inverting it. The upper left corner represents a perfect classifier and entries to the upper left of others tend to be better. Entries on the left side of the graph are considered conservative in the sense that they only classify with high confidence. Classifiers with entries in the upper right corner, on the other hand, tend to be liberal and make many positive decisions at low certainty.

An inherent property of ROC graphs is that they are not susceptible to changes in class distribution. This means that the ROC curve does not change when the distribution of positive to negative entries in the test set changes. PR curves can differ significantly in such a scenario. Thus, depending on the expected class distribution deviation in the data set, ROC graphs are better suited to represent properties of the classifier in a comparable way.

To compare classifiers, it may also be desirable to reduce ROC performance to a single scalar value. This can be done by calculating the area under the ROC graph (AUC). The resulting area is a value between 0 and 1, where 0.5 is the worst realistic value and 1 is the best. Also, this AUC-value is equivalent to the probability that a given classifier scores a random given positive instance higher than a random given negative instance [Faw04].

The concept of the AUC is closely related to the so-called *Gini coefficient*. In just the same way, this coefficient can also be used as a scalar value quality measure for classifiers. It is defined as twice the area between the ROC curve and the diagonal. Thus, $Gini + 1 = 2 \times AUC$ [HT01].

2.3 Bitcoin

Bitcoin is the most well-known cryptocurrency. Its goal is to create a decentralised means of payment. To this end, the currency is based on a decentralised blockchain that acts as a publicly disclosed and distributed transaction ledger. This blockchain is operated by individual network nodes that are joined into a peer-to-peer network to organize and document transactions according to the rules of a consensus procedure [ECP21, Nak08].

In this section, we describe and explain the most important characteristics and the behaviour of the peer-to-peer network on which the blockchain is based. Aspects of Bitcoin that are not directly related to the operation of the peer-to-peer network are not considered. In particular, this includes transactions, mining and verification of blocks, the blockchain, consensus procedures and block acceptance, and all those network messages that are related to requesting or responding to data related to transactions and blocks. Instead, the remainder of this section focuses on the network aspects of Bitcoin and the associated message types, particularly those used to control network connections between two nodes or to exchange information about other nodes in the network. An understanding of these features is necessary to understand the topology discovery method presented in this thesis.

The source of the following information is the community documentation [Wika], network reference [Proc] and the developer guide [Prod] of the Bitcoin developer site. Since these are not official documentations, we have compared and verified the information with the source code [Proa] of the Bitcoin Core client. Thus, the information mainly refers to the behaviour of nodes using the *Bitcoin Core* standard client. While it is possible that nodes run client software that does not in every detail conform to the protocol described in this section, we assume that alternative clients behave similarly and exhibit the same basic behaviour on the network.

2.3.1 Network Structure and Node Behaviour

The Bitcoin network is operated as an unencrypted *overlay network* based on TCP/IP. Individual independent *nodes* use the protocol to exchange information and establish Bitcoin connections. Since the nodes are on the same hierarchical level and act in a fundamentally similar way, the network is a peer-to-peer network.

There are two different types of nodes: full nodes and lightweight nodes. A *full node* downloads all blocks and transactions and verifies that they meet the consensus rules. Whereas *lightweight nodes* do not perform this verification completely but only on the basis of the block headers and therefore have to trust the full nodes in the network. Within the network protocol, full nodes can be requested to filter and distribute historical block and transaction data to other nodes for this purpose [Prod]. For the remainder of this thesis, it is sufficient to group these two types of nodes under the common term "node".

From a node's point of view, a peer is an *outbound peer* if the node itself has established the connection to this peer. Similarly, a peer is called an *inbound peer* if it initiated the connection to the node. Outbound and inbound peers are treated the same except for minor differences. It is possible that a network node only allows outbound peers due to its configuration. The default value for the number of outbound peers to which a node attempts to connect is eight. The default maximum number of inbound connections is 117 [BKP14].

Another type of connection is the so-called *block-relay-only connection*. As the name suggests, these connections are only used to forward blocks. In particular, no addresses are forwarded over this type of network link. The purpose of these connections is to protect the node against topology discovery attacks. The fact that no addresses are relayed over the connections makes them more difficult to detect. In the presence of an attack, these links can thus continue to be used to exchange blocks in an undetected way. By default, a node establishes two such connections [Cmm].

In the context of Bitcoin, the term *address* is used for two different things. On the one hand, address is a string of characters that represents the destination for transactions on the blockchain. This type of address can be used to make payments on the blockchain layer of the currency. On the other hand, address also refers to the network addresses of computers on the network layer of the currency. This type of address can be used to establish connections between Bitcoin nodes and to reach the computers that operate the clients. Since Bitcoin's peer-to-peer network is an overlay network on TCP, these addresses generally correspond to the IP addresses of the nodes. In this way, an address can take the form of one of three address types: IPv4 addresses, IPv6 addresses and Onion-addresses, which belong to the TOR network. The latter can be further subdivided into V3 Onion-addresses and the deprecated V2 Onion-addresses. In this thesis, we use the term address exclusively to refer to network addresses in the sense of one of the three address types.

A node connects to the network by establishing outgoing connections to known peers. If a node wants other peers to connect to it, it advertises its own network address on the network. To do so, it simply sends unsolicited address messages containing its own address to its neighbours. These nodes then forward this address in the network. By default, a client does so once every 24 hours [Proc].

The distribution of information in the Bitcoin peer-to-peer network follows a so-called *gossip protocol*. Analogously to gossip in groups of people, the gossip protocol passes information between network participants in order to distribute it in the network. This means that a Bitcoin node forwards network messages to its neighbours according to certain rules. This ensures that the knowledge spreads throughout the network. This mechanism is used to distribute transaction data, but also address data in the Bitcoin network [ECP21, NAH16].

The first time a node wants to connect to the network it does not yet know addresses of other nodes to connect to. This is because it has not yet had a chance to receive such

Length	Field
4 Bytes	start string magic bytes
12 Bytes	command name + null padding
4 Bytes	payload size
4 Bytes	checksum

Table 2.1: Bitcoin message header fields and structure

addresses, as it would have to be connected to the network for that. For this reason, it is necessary to resort to the use of a ground truth. In Bitcoin, this problem is solved by the fact that a node has a list of hardcoded *DNS seeds*. These DNS seeds are services maintained by community members that scan the network for addresses and maintain a list of available nodes. A node that wants to learn about addresses sends a DNS request. The DNS service makes a random selection from their internal list and sends the addresses to the node. The node then has everything it needs to establish an initial connection to the network [Prod].

Once a node has established a connection to other nodes, it asks them for known addresses or waits for unsolicited address messages via the gossip protocol. This allows a node to accumulate knowledge about addresses of other participants in the network. This *node discovery* for the network is therefore possible both actively and passively. For address storage, nodes operate an internal list called address manager or *addrman*. In subsequent attempts to connect to the network, the node falls back on previously accumulated addresses and doesn't have to rely on DNS-seed services [Prod].

In addition to the main Bitcoin network *mainnet*, there are other alternative networks. For example, the so-called *testnet*. This testnet represents an alternative blockchain, where it was agreed that the currency on it should have no monetary value. The goal of the testnet is to provide developers and researchers with an environment in which aspects of Bitcoin can be tested without fear of having a distorting influence on the real blockchain [Wikb].

2.3.2 Message Structure

All peer-to-peer communication takes place via TCP. Messages of the Bitcoin protocol are therefore located in the payload area of a TCP packet. This Bitcoin message part of a data packet consists of a mandatory header and an optional payload depending on the type of Bitcoin message. The Bitcoin header thereby is made up of multiple fields and is structured as shown in Table 2.1.

Here the magic bytes at the beginning of the header differ based on whether the message refers to the testnet or mainnet. The maximum payload size is 32 MiB, and the checksum is obtained by using the first 4 bytes of the SHA256 hash of the SHA256 hash of the payload [Proc].

While the Bitcoin protocol knows a variety of different message types, the most important control messages for the operation of the network are VERSION, VERACK, PING, PONG, GETADDR and ADDR.

It is noteworthy that none of these messages are authenticated while being handled. As a result, these messages may contain false, harmful or maliciously falsified information [Proc].

VERSION: To establish a connection to a peer, a node sends a VERSION message to a peer. It contains useful information for communication, such as the version number of the protocol supported by the sender, as well as the system time, offered services, and the current state of the block information. The remote peer responds to a VERSION message by sending a VERSION message with information about itself. The handshake is completed by both communication partners each sending a VERACK message without payload. This establishes the connection, and sending of other message types. A connection is closed if no message is received from the peer for more than 90 minutes [Proc, Prod].

PING: For latency probing and keeping alive the connection, a node sends PING messages at a standard interval of 2 minutes. These contain an 8 byte random value called nonce [Prob]. If a TCP error occurs during the delivery of the PING message or if no PING message is answered correctly for 20 minutes, it is assumed that the peer is no longer connected. The communication partner properly responds to a PING message with a PONG message. This contains the same nonce value, which confirms that the peer has received the corresponding PING [Proc].

GETADDR: With a GETADDR-message, a node asks a communication partner for addresses of other network participants. This allows filling its address memory without having to passively wait for incoming addresses. Such a GETADDR-message has no payload. The proper response to a GETADDR-message is either an ADDR or ADDRV2 message [Proc]. Notably, a Bitcoin Core client is generally unwilling to return more than 23 % of addresses stored in its address manager in response to a GETADDR request, presumably to avoid information leaks.

ADDR: An ADDR-message is used to transport information about addresses of network participants. Such a message contains between one and a thousand such address entries at once. For this purpose, the payload of an ADDR-message consists of a flexible number of fields: first a counter of the number of address entries to follow, and then the address entries one after the other. Each of these address entries consists of:

- A 4-Byte timestamp indicating when the node behind this address was last seen.
- 8 Bytes of information about what services this node provides

- 16 Bytes network address
- 2 Bytes port number

Since the 32 byte addresses of Bitcoin nodes behind Tor v3 hidden services do not fit into the 16 byte network address field of ADDR-messages, ADDRV2-messages were introduced as an advanced version of the ADDR-message type. The only difference to ADDR-messages is that they have fields for the address entries that specify the length of the address and the type of network. A node can inform its communication partners that it supports ADDRV2-messages by sending them a SENDADDRV2-message. It has no payload and requests a node to send only ADDRV2-messages instead of ADDR-messages. This request has to happen before the connection is fully established with a VERACK message [Proc, Prob].

2.3.3 Address Relay Theory

A gossip protocol is used to distribute information about network participants throughout the network. This ensures the spreading of ADDR and ADDRV2 messages to all network participants. Individual nodes send these messages to their neighbours according to certain rules. Since this behaviour does not differ for ADDR and ADDRV2, these two message types are combined under the term "ADDR" for the remainder of this thesis.

First, all ADDR-messages from block-relay-only nodes and those containing more than 1000 address entries are ignored. Then, the individual address entries within the ADDR-message are processed in random order, and for each of them, the following procedure is applied:

1. Rate-limiting is applied to ensure that on average only one address entry is processed every 10 seconds. Any further addresses are skipped. This form of rate-limiting cannot be circumvented by sending more address entries in fewer ADDR-messages. However, due to the usage of so-called *rate limiting buckets*, address entries can be sent faster than average for a short period of time after time spans with equivalently lower than average throughput.
2. Addresses that offer neither a useful address directory nor all desired services are ignored. To fulfill this condition, the service flags of the advertised node must either point to a full node, a node that claims to be able to provide data for at least the past 2 days of the blockchain, or all of the service flags that the forwarding node has defined as interesting for itself. Any other address is skipped. Therefore, address entries should contain at least the service flag *NODE_NETWORK* or *NODE_NETWORK_LIMITED*. In particular, this means that only addresses to full-nodes are stored.
3. Timestamps that are either more than 10 minutes in the future or unrealistically far in the past are adjusted so that their timestamp equals the current time minus 5 days.

4. The node remembers that the sending peer knows this address. This ensures that the same peer does not receive the addresses it already knows.
5. Addresses banned under the reputation system are not handled beyond this point, except that the node remembers that they were received.
6. The address is forwarded if it was not part of an ADDR-message with more than 10 entries, was not a response to a GETADDR request, its associated lastseen timestamp is less than 10 minutes old, and the address is generally routable.
7. The address is stored in the AddressManager if it is in principle a network address that can be reached by the node.

The forwarding in step 6) works as follows:

- The address is not forwarded to the node from which the ADDR-message comes.
- Among all other peers to which the receiving node is connected, the number of forwarding destinations to be selected is either 2 if the address is reachable for the node, or otherwise equally distributed randomly either 1 or 2. The reachability is not checked by probing, but decided on the basis of which network the address is part of.
- The random selection is based on hash values and is therefore deterministic. Thus, the choice falls on the same selection of peers within a 24-hour timeframe, if the list of peers connected to the node does not change.
- While it does not influence the selection, the actual sending off the address is skipped for a peer if we know that this peer already knows the address.

Next, the message is sent to the selected peers. However, to be precise, the message is not relayed immediately. To make timing analysis more difficult, a random time delay is built into the sending of ADDR-messages. For this purpose, the addresses to be sent are first appended to a queue for the corresponding peer. A node has one such queue for each of its communication partners. The queue for a connection to a peer is then emptied at irregular intervals by sending all pending addresses in it at that point in time in a single ADDR-message. The delay between each ADDR-message transmissions for a peer is randomly sampled from an exponential distribution with an average of 30 seconds. To be precise, the function that is used to get the random delay calculates each delay amount as

$$-\ln(1-x) \times 30s + 0.5\mu s$$

where x is a uniform random value of $0 \leq x < 1$. Due to this, the probability of the message queue being emptied after time t is

$$1 - e^{-\left(\frac{t}{30s}\right)}$$

[Prob]. This artificial time delay is called *trickling* and is deliberately introduced to hinder timing analyses [NAH16].

In order to avoid redundant address forwarding, a client keeps a list for each connected peer, in which a record is kept of which addresses this peer already knows. This list is cleared every 24 hours on average. Addresses are added to the local internal list when they are to be sent to or received from the peer. Before sending, the system checks whether a node already knows this address and if this is the case, sending is skipped. This has the effect of preventing unnecessary bandwidth wastage during relay, which would otherwise occur without immediate benefit [Prob].

We argue that this means that two mechanisms are built into the system that ensure that flooding ends over time. Since the time stamp for forwarding may not lie more than 10 minutes in the future or past, a given address entry is normally forwarded for 10 minutes, but not more than 20 minutes. In this sense, the timestamp for the address flooding defines a kind of time to live of the message, and after this time, the forwarding of this address ends abruptly. During this time period, multiplying the number of addresses by 2 in each forwarding step increases the total volume of addresses in circulation. The effect is slowed down by the fact that, as described above, the address is not forwarded to peers who know that they already know the address. This slows down the forwarding as soon as a high saturation is reached, in the sense that the address has already been sent to many network participants. In particular, as mentioned above, the random selection of peers for forwarding always selects the same neighbours within each 24-hour time frame if the peer list remains constant. Because a node remembers which addresses its neighbours knows and then may not forward an address to them, a node generally only handles the relay of a given address once. This is another reason why the chain of forwarding an address in the network ends in the context of flooding.

In regard to the properties of every handled address, there is a mechanism in place to prevent attackers from misusing the Bitcoin network to overload services on the Internet with unwanted network traffic. To do this, nodes running the Bitcoin Core client try to avoid connecting to addresses that do not use the default port 8333 [Proc]. However, this port preference does not affect the forwarding behaviour of the addresses. For the address relay procedure, the port specified in the address is therefore irrelevant.

2.3.4 Reputation System and Misbehaving Peers

To protect network participants from misbehaving peers, Bitcoin relies on a reputation system. A node assigns an internal behaviour score to each peer connected to it. This score is increased if the node detects misbehaviour on the part of this communication partner. If a certain threshold value, which is set to 100 by default, is exceeded, this communication partner is considered harmful. The connection is then terminated, and future connection attempts are aborted. This does not apply to explicitly whitelisted peers, manually established connections, and local peers [Prob].

Behaviour	Score
providing invalid block data	100
outbound peer being on an invalid chain	100
previous block in the chain is invalid	100
previous block in the chain is missing	10
providing invalid transaction data	100
sending a getblocktxn with out-of-bounds transaction indices	100
sending 10 headers that don't connect in a row	20
sending an invalid header	10
sending a non-continuous header sequence	20
sending more than 1000 addresses in a single ADDR-message	20
sending an INV message with more than 50000 entries	20
sending a GETDATA message with more than 50000 entries	20
invalid filter constraints in FILTERLOAD message	100
send a data item of more than 520 bytes in a FILTERADD message	100

Table 2.2: Behaviour punished in the reputation system in Bitcoin and the corresponding punishment scores

The behaviours that are penalized by the Bitcoin client and their assigned penalty score are listed in Table 2.2. We note that almost all of these penalized forms of misbehaviour relate to block and transaction verification. The only restriction regarding Bitcoin connections is that ADDR-messages must contain at most 1000 addresses. It is also worth noting that penalty scores are only stored internally; they are never sent through the network to warn other nodes about misbehaving nodes [Prob].

2.4 Networks

2.4.1 Peer-to-Peer Networks

Conventional computer networks often follow a *client/server* structure. This means that network participants have different roles: A single *server* is a central unit and provides content or a service. Several *clients* are connected to the server and access the resources or services without sharing their own resources. The server is often a higher performance system than the clients. In this sense, there is a hierarchical imbalance between the network participants [Sch01].

Outside of client/server networks, in which participants always assume exactly one of these two roles, a third role is also possible: the *Servent*. This participant simultaneously fulfills the roles of server and client in the sense that it can provide resources and make requests at the same time. A network of peers that share resources without hierarchical differences to jointly provide the service or content of the network is called a *peer-to-peer* network [Sch01].

A further distinction is made between *hybrid peer-to-peer networks* and *pure peer-to-peer networks*. A pure peer-to-peer network is characterized by the fact that it is a peer-to-peer network that may only consist of servents. This means that there is no central entity in the network, and it is possible to remove individual, randomly selected network participants without any loss of network service. On the other hand, hybrid peer-to-peer networks are those that also allow central entities to provide parts of the service or resource offered by the network. These differ from client/server networks in that the other network participants also still share resources [Sch01].

2.4.2 Network Metrics

Certain definitions given in Section 2.1.1 can be used as metrics to judge properties of the network from a graph theoretical point of view. Some metrics provide insights into the properties of individual nodes, while others are applicable to the entire network:

- *Node degree* reflects the importance of a node within the network.
- *Average shortest path length* expresses the degree of separation between nodes in the network. A low average path length implies a low latency for resource lookup [HLY04]. Loguinov et al. argue that this average path length is a better representation of what the user can expect from the actual performance of the network than the diameter, as the latter only gives an upper bound and not a balanced metric. Additionally, they refer to the unintuitive fact that it is possible to increase the average routing distance while reducing the diameter of the graph [LKRG03].
- High values of *Global Clustering Coefficient* imply robustness in object lookup even under heavy load [HLY04], while simultaneously hinting towards weaker networks because of its implications on edge expansion [LKRG03].
- *Edge Expansion* measures how robust the graph is in the presence of edge failures. It is defined as the minimal fraction of all edges of a graph that connect any subset of nodes with the corresponding rest of the graph. As a higher clustering index leads to fewer connections between the neighbourhood and the rest of the graph, the edge expansion is contrasting the clustering coefficient [LKRG03].
- *Node Expansion* measures how robust a graph is against node failures. It is defined as the minimal fraction of all vertices of a graph that connect any subset of vertices with the corresponding rest of the graph. Loguinov et al. argue that node failure is more present in peer-to-peer networks than edge failure is, making node expansion of a graph a better indicator for resilience [LKRG03].
- *Bisection Width* offers a metric on how difficult it is to split the graph into two large components by splitting individual edges. While this is a metric for the resilience of the graph on the one hand, a low value can also indicate congestion problems

under the assumption that both halves produce the same amount of traffic. This is because that traffic would then need to pass through a possible throughput bottleneck [LKRG03].

- *Betweenness Centrality* shows for a node how high its control over the graph is. This is because it can be assumed that for high values of betweenness centrality, proportionally more traffic passes through the node [ZFDS22].

We argue that it is to be expected that a truly random graph according to the Erdös-Rényi-Model would have both a low average path length and a low global clustering coefficient, which would in turn speak for a high node- and edge expansion.

Zabka et al. [ZFDS22] apply the *Gini coefficient*, which is used in economics to express unequal distribution as a scalar value, to network metrics. To this end, they calculate the coefficient for distributions of node metrics to determine how equally they are distributed within the network. Specifically, they calculate the Gini coefficient for betweenness centrality to determine an uneven distribution of high centrality values among a few nodes. First, they form the Lorenz curve, which indicates what proportion of nodes has what proportion of the sum of the metric across all nodes. A perfect uniform distribution would thus form a diagonal line. The Gini coefficient is then calculated as the area between this diagonal and the Lorenz curve, divided by the total area under the diagonal. Following this calculation, a Gini coefficient of 0 corresponds to a perfect uniform distribution, and a value of 1 corresponds to a maximally unequal distribution. This principle can thus be applied to express the distribution of node properties within the network as a scalar value. We use the Gini coefficient to derive a network metric from a node metric. This approach has been used in previous work related to graph theoretic metrics in computer networks [CLA16, DSBPS^{+}19].

2.4.3 Security and Attacks

The security relevance of individual network aspects also depends on the purpose of the network. For example, unstructured peer-to-peer networks such as those underlying permissionless blockchain systems sometimes have different requirements than anonymity-providing networks such as TOR. In the case of permissionless blockchain systems, Bitcoin is a widely considered prototype. However, the security requirements imposed on Bitcoin are in principle transferable to many other such networks. Functional requirements for this kind of peer-to-peer network are: 1) Openness of the network, i.e. the possibility for any peers to connect to it. This requirement presupposes a sufficiently large proportion of accessible peers. 2) Dissemination of information, i.e. the distribution of relevant information to all network participants, is ensured by flooding or broadcast/gossip protocols [NH18].

Non-functional requirements that can also be placed on this kind of network include: 1) low cost of participation, i.e., there should not be too great a hurdle to setting up a node.

This includes the necessary memory, bandwidth and computation. 2) performance, i.e. the speed of dissemination of information [NH18, BMS01].

Mechanisms that are supposed to ensure these requirements are often the focus of attacks on this type of network. Attacks frequently discussed in the context of security considerations for such peer-to-peer networks include sybil attacks, eclipse attacks and denial of service attacks.

Sybil Attack: A sybil attack is an identity attack in which a single entity impersonates multiple simultaneous identities. This kind of attack is a fundamental problem for systems that assume that each participant controls only one identity [LSM06]. In the case of the Bitcoin peer-to-peer network, such an attack occurs when a single attacker connects a number of malicious nodes to the network. Since, according to the requirements listed above, the cost of participating in the network should be low, an attacker can easily create and connect many such nodes. The number of such sybil nodes may well exceed the number of honest nodes in a network [NH18].

Eclipse Attack: In an eclipse attack, an attacker is positioned within the network to control all connections to a victim node. This gives the attacker full control over all information received by the victim node, and thus over the victim node's view of the information shared on the network. This disrupts the assumption of perfect information exchange in the network, which is necessary for many peer-to-peer networks. In particular, an attacker can force a target node to waste computing power on false chain-state information in case of Bitcoin or have them use said power in favor of the attacker. By deliberately withholding blocks from the rest of the network from the attacked node, the node's participation in the service is fundamentally disrupted. This can lead to scenarios in which the attacker illegally transfers funds multiple times, once to the victim node and secondly to the rest of the network (so-called *double spending*). If a substantial number of nodes are simultaneously targeted in a coordinated eclipse attack on a larger scale, this can favor mining attacks such as the *51 %-attack* [HKZG15]

DoS: Denial-of-service (DOS) attacks generally refer to attacks that result in a given system no longer being able to provide the services it would offer under regular circumstances. In the context of computer networks, this can mean that individual network participants can no longer take part in the distribution of information. In practice, such an attack can take different forms. On the one hand, the attacker could, for example, make specific faulty requests with the aim of causing the target system to crash. On the other hand, one possibility is to make valid requests to a system, but so many at once that the victim either crashes under the load or no longer has the capacity to handle legitimate requests made by the other network participants as part of the operation [Rou11].

Topology Discovery

A special position among attacks on peer-to-peer networks is held by the so-called topology discovery attack. Although there are various invasive methods in practice, the primary objective of this attack is not to disrupt operations. Instead, the attacker aims to gain information. Specifically, the goal of this attack is to find out the topology of the network, that is, the graph $G = (V, E)$ that maps the network [Wei19].

An obvious problem here is the privacy of the users. An attacker who can trace the topology may be able to de-anonymize users [DSBPS+19, BKP14, NAH16]. For privacy-oriented network services such as Bitcoin, this may be in direct conflict with the defined security objectives of the service.

Topology discovery also enables analysis of the network topology. From the perspective of benevolent network participants and researchers, this can thus be a good means of verifying, for example, compliance with decentralization goals and understanding the actual interactions between clients. As a result, this process is a key to detecting vulnerabilities or attacks and disruptions already in progress in the network [DSBPS+19, BKP14, NAH16]. However, malicious attackers can also use this knowledge to target attacks on the network's topology. For example, knowledge of the network structure can be used to find a minimal cut of the graph, which can then be carried out in practice via targeted attacks on individual connections to split the network in half [NAH16, BKP14].

A more subtle approach by which an attacker can take advantage of topology discovery is to deliberately and strategically choose a position within the network. For example, well-chosen connection to selected peers can be used to optimize the speed at which information reaches a node. In particular, advantages in mining can be drawn from advantages in positioning within the topology [MLP+15].

All in all, we argue that topology discovery gains strategic value for attackers, especially in combination with other attacks. For example, detailed knowledge of network construction can facilitate eclipse attacks, as an attacker may be able to gain control over nodes more easily when knowing their legitimate neighbours. In this sense, a basic knowledge of the topology is also useful for targeting DoS attacks. All in all, we therefore assume that topology discovery can provide a useful foundation for more advanced attacks with higher damage potential.

2.5 Related Work

Biryukov et al. [BKP14] proposed the first paper on general de-anonymization through topology discovery in Bitcoin in 2014. The researchers show that the pseudonyms of Bitcoin users can be mapped to IP addresses with a success rate of up to 60 percent if the set of entry nodes for a non-listening peer is known. They suggest a method to find the number of neighbours of a node by exploiting address propagation. The basic idea is that a node v, which connects freshly to the network, sends its own address via ADDR-messages to all connection partners P. Then, these ADDR-messages are forwarded to random neighbours of these peers P. If an attacker has established enough connections to the nodes P, they receive with some probability the ADDR-message containing the address of v. The attacker has then identified a subset of the network that is connected to the victim node. The advantage of this approach is that it can also be used to determine the connection partners of those nodes that are behind NATs or firewalls and do not allow any incoming connections from the attacker. In addition, the researchers present an alternative method for degree estimation and connection inference. For the former, they inject a series of addresses into a node and observe how many of them are directly forwarded to listening connections of monitors. The node degree can be estimated by the ratio of observed relays to the originally injected number. The researchers validated the average relative error rate of this experiment to be between 3 % and 10 %. In this thesis, we reproduce the degree estimation following the same idea. Additionally, the researchers also present a connection inference method. For this, the addresses are first injected into a node. Afterwards, GETADDR queries are used to check, which other nodes in the network have these addresses in their address memory. Using this method, connections to network nodes could be determined with 100 % precision and recall. However, since the way in which GETADDR queries are answered by Bitcoin nodes has been changed afterwards, this procedure is no longer feasible [BKP14].

Directly building on this work is the 2015 paper of Miller et al. [MLP$^+$15]. Their *AddressProbe* framework uses the way in which nodes use timestamps in their internal address memory to determine the recency of addresses. This timing information is then used to determine connections between nodes. More specifically, the researchers exploit the fact that nodes keep the "lastseen" timestamp up to date for outgoing connections, but not for incoming connections. By querying the address memory of nodes through GETADDR-messages, connections between pairs of nodes are detected. Furthermore, artificially added "ageings" of the timestamp by exactly two hours in the ADDR-message relay allow reconstruction of the relay path through the network by detecting discrete steps of exactly 2h in these timestamps. Thus, this topology discovery is based on the address manager's handling of timestamps at the ADDR-message relay and not on actually measured time delay. Unlike Biryukov et al., this method only works for nodes that allow incoming connections. The vulnerability enabling this attack was patched shortly thereafter by the Bitcoin developer community. The researchers conclude their paper by examining network properties such as node degree and randomness. They find that a small number of nodes have an exceptionally high node degree, and that the network graph, even without the high node degree outliers, falls statistically significantly below the expected level of randomness [MLP$^+$15].

The first work to explore the topology not based on client implementation peculiarities but by exploiting packet flooding properties was proposed by Neudecker et al. [NAH16] in 2016. The researchers present a model that allows measuring time delays in the forwarding of data packets, and then to statistically estimate a path length. This model is generally applicable for all peer-to-peer networks that use flooding, as well as for all floodable packet types. As a proof of work, the researchers then apply this model to the

Bitcoin network. For this purpose, time delays in the forwarding of block announcement messages (*INV*) are measured and analysed using the model. In practice, the framework achieves precision and recall values of about 40 %. The main difference between their work and this thesis is that they measure a different type of message with INV messages instead of ADDR messages. Apart from this, the time-based estimator we use in this thesis is directly based on the statistical model of Neudecker et al. As the researchers also discuss the now implemented artificial forwarding time delays (trickling) as a countermeasure, our work further investigates the effectiveness of this countermeasure in practice [NAH16].

In 2018, Grundmann et al. [GNH18] proposed two additional ways to measure Bitcoin topology. The first exploits the way pending transaction messages are artificially held in queues before being sent as part of the trickling process. This attack is theoretically possible but of little practical relevance, since its execution has high monetary costs and even then it only has 10 % recall. The second attack uses conflicting transactions. Here, contradictory transactions are sent to nodes. Target nodes that receive these transactions drop all but one of the conflicting transactions and forward it. By then matching which transactions were injected and received back from where, they determined who is connected to whom using a set of observations. In this way, they were able to determine connections with a recall of 87 % and a precision of 71 %, costing a total of 99 transaction fees to perform [GNH18].

A similar topology discovery attack was presented in 2019 by Delgado-Segura et al. [DSBPS+19]. Here, peculiarities of Bitcoin clients in the way they handle mis-ordered incoming transaction data are exploited. However, since this attack is exceedingly invasive and could paralyse large parts of the network, the researchers tested their framework only on the testnet. They were able to determine precision and recall values of over 90 %. The researchers use this approach to identify the connections of the Bitcoin testnet. They then apply graph-theoretic metrics to compare the resulting topology with random graphs [DSBPS+19].

Finally, there is the 2021 paper by Grundmann et al. [GBH21]. The researchers describe how numerous injections of ADDR-messages with spoofed addresses by an unknown actor were observed in the network. They hypothesize that these could potentially be used to perform a degree estimation attack. Therefore, the paper presents a method of degree estimation by address injection. The idea of the presented method is the same as that of Biryukov et al. However, they do not inject the ADDR-messages themselves. Instead, they observe an unknown actor injecting a known number of addresses into all nodes on the network. This way, they were able to perform the degree estimation with a relative error of 4.1 %. The researchers also mention that since the incident, a rate-limiting system has been implemented to prevent such attacks [GBH21].

The diploma thesis of Jakob Rosenblattl [Ros] is particularly worth mentioning at this point. His thesis is being written at the same time as this thesis, in close collaboration. The thesis deals in detail with the theoretical foundations of topology discovery using ADDR-relay and provides the time-based estimator. In this respect, it was developed to build on the work of Neudecker et al. in terms of timing-analysis. In particular, Rosenblattl's work describes the methodology by which the time delay data is collected. It also derives the formulas by which degree estimation and connection inference can be performed using the statistical methods of timing analysis. The project furthermore provides a basic code base for performing the estimation on individual accounts [Ros]. This thesis builds on Rosenblattl's work by validating the procedure and comparing it to an alternative behaviour-based estimation procedure. We further adapt the method for scaling it up to the entire network. For this purpose, the code-base is heavily expanded. In this sense, the product of this thesis is the practical implementation and verification of Rosenblattl's work.

CHAPTER 3

Methodology

In this chapter, the topology measurement experiment is presented in detail. Section 3.1 first describes the idea for the procedure, justifies model assumptions, and defines terms. Both the technical approach for collecting the data and the general way in which information can be deduced based on this data are presented. Section 3.2 will explain in detail how the time-based estimation and the relay-based estimation are conducted, respectively. In Section 3.3, a series of smaller complementary measurements are presented. These were performed before the main experiment to determine details of the behaviour found in Bitcoin client nodes and the network as a whole. In essence, these additional experiments test the boundary conditions of the main experiment by determining the availability and stability of nodes and their connections to each other and to the probing system. Afterwards, the outline and workflow of the main measurement is presented in Section 3.4. This also includes considerations of experiment scalability, and the selection of the peers to be measured.

3.1 Idea

As described in Section 2.3, address messages are flooded through the Bitcoin network using a gossip protocol. A properly implemented Bitcoin client forwards an ADDR-message at random to two of its currently connected neighbours, which in turn forward the message further. The goal is address spread so that eventually every participant in the network has knowledge of a sufficiently large number of client addresses.

The way in which this propagation takes place allows two types of information to be gained: Node Degree Estimation and Connection Inference.

3.1.1 Definition of Terms

To ease the description of the measurement framework, we define a series of terms related to individual components and actors of the setup and measurement procedure:

- *Injection*: The act of sending specially crafted ADDR-messages to a Bitcoin client on the network.
- *Monitor*: An endpoint of the framework that implements part of the Bitcoin network protocol to perform the measurement. In a way, it is a fake sybil peer for communication with other clients in the network. Monitor nodes connect to regular nodes, without suspicion, to inject ADDR-messages as well as receive forwarded Bitcoin messages from the client. Monitors that inject messages are called *injectors*, and those that listen are called *receptors*. The role that a monitor takes might change over the course of the experiment. In fact, any monitor that is technologically capable of receiving and sending messages can fulfill both roles simultaneously.
- *Victim*: A node in the network that is under active measurement, i.e., connected to a Monitor that injects messages to determine node degree and neighbours.
- *Client*: A node on the network, which is connected to one or more monitors, but into which no messages are injected. Clients are primarily used to retrieve forwarded messages in the connection inference. In the same way that a monitor node can act as an injector and a receptor at the same time, a network node can also have different roles depending on the perspective. For example, if several measurements are performed simultaneously on different nodes, a victim of one measurement can simultaneously assume the role of a client in another measurement.
- *Conductor*: The framework component that coordinates the experiment workflow. The Conductor connects the monitors to victims and clients and orchestrates the injection of messages.
- *Bullet*: ADDR-messages created and used specifically for injection into victims. In a way, they are "weaponized" for the attack and are "fired" at the victims.
- *Estimator*: Standalone module into which the measured time delay data is fed. It provides estimates of victim node degrees as well as victim neighbours.

An overview of the general layout and interaction between these components is depicted in Figure 3.1.

3.1.2 Node Degree Estimation

Upon receipt of an ADDR-message, the node forwards it to two randomly selected neighbours. Since it makes no sense to relay an ADDR-message back to its origin, the

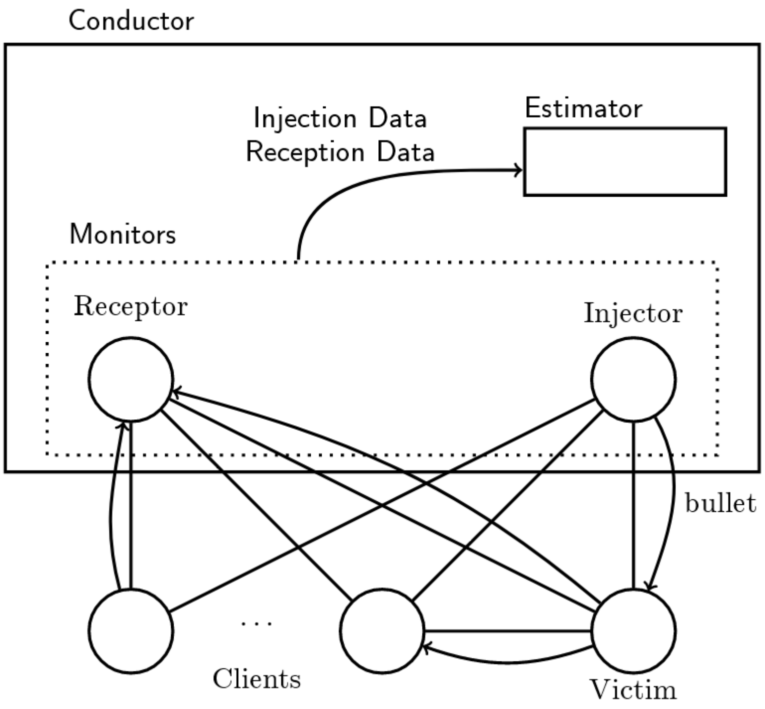


Figure 3.1: Overview of the general layout of the experiment components and their interactions between each other

sender is excluded from the forwarding. Apart from that, the number of neighbours to which it is forwarded is fixed and does not change with the node degree. Intuitively, this means that the probability of receiving a forwarded ADDR-message from a node v decreases with node degree $\deg(v)$; if a node v has many connected neighbours, the probability that a given neighbour v' receives the message is lower than with fewer neighbours. This change in probability is exploited to estimate the node degree.

For this purpose, nodes from a node set $M = \{m_1, m_2, \ldots, m_n\}$ of size $|M| \geq 2$ connect to the victim node v, taking the role of the attacker. One of the attacker nodes, for example m_1, sends a sequence of ADDR-messages to node v. The remaining attacker nodes, m_2 through m_n, pay attention to which addresses they receive from v. In normal operation, the eavesdropping attacker nodes are expected to receive some of the previously injected addresses as well as addresses that were randomly passed to them and are unrelated to the experiment. Since the injected addresses are known, it is easy to filter them. All other addresses are neither relevant for the degree estimation nor for the connection inference. The proportion of addresses recovered after injection then gives an indication of the ratio of attacker nodes M and non-attacker nodes $C = \{c_1, \ldots, c_n\}$ connected to v. Since the number of connected attacker nodes is known, this can be used to estimate the node degree $\deg(v)$.

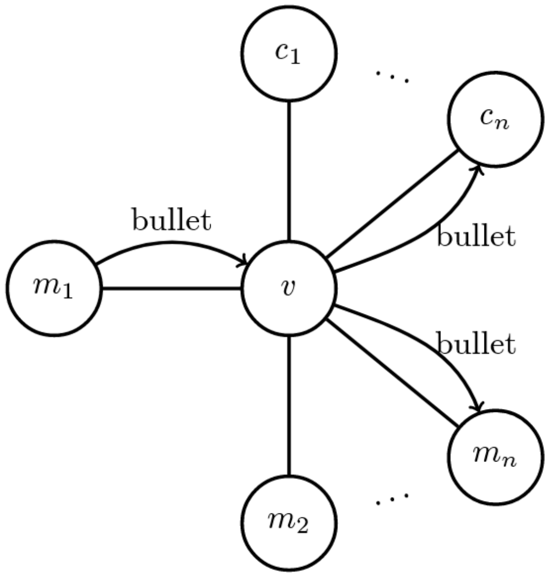


Figure 3.2: Sketch of the parties involved in degree estimation

A figure showcasing the parties involved in the process of degree estimation, and their connections is shown in Figure 3.2.

In practice, the node sizes for Bitcoin clients can become rather large; 125 to several hundreds are possible [GBH21]. Since a given receiver node's share of relayed addresses decreases substantially for such nodes, it may make sense for practical reasons to keep the number of connected attacker nodes as large as possible to increase the total number of relays observed by the conductor.

The idea of being able to actively measure node degrees using this experiment setup was already presented in 2014 by Biryukov et al. [BKP14]. For this thesis, we use their setup to conduct our experiment. However, as opposed to their paper, we acknowledge that the minimum number of connected nodes for this measurement is two: one injector node to send ADDR-messages to the victim and one receiver node to listen out for any ADDR-messages being passed on by the victim. As the injector node never receives its own address messages back by the victim, a single node cannot fulfill both of these roles at once.

3.1.3 Connection Inference

After a node forwards a received ADDR-message to its peers, these peers also start forwarding this message according to the same forwarding rules. If a monitor node of the attacker receives a previously injected address from a client, it means that there is a path between the injecting monitor and receiving monitor. This path necessarily includes the victim node and an unknown number of client nodes and corresponds to

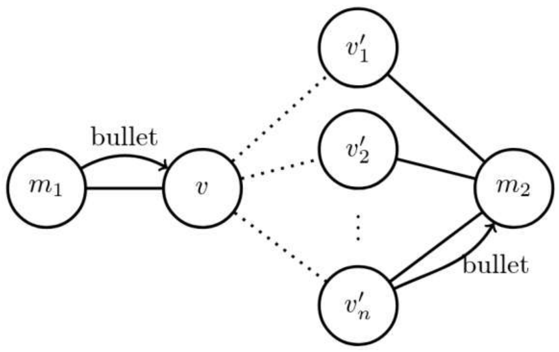


Figure 3.3: Sketch of the parties involved in connection inference

the path through which the address message was forwarded. The exact number of client nodes through which the injected address was forwarded after the victim and before the receiving monitor is unknown and may be difficult to determine. However, a large quantity of observed address recurrences can be used to determine with a certain degree of certainty whether two nodes are adjacent.

In this method of connection inference, an ADDR-message is therefore inserted at one point in the network, and the time delay δ until its reappearance at another point is measured. For this purpose, an attacker node m_1 connects to the victim node v that is to be measured. Simultaneously, another attacker node m_2 connects to as many nodes $V' = \{v'_1, \dots, v'_n\}$ from the network as possible. A large number of uniquely identifiable ADDR-messages are then passed from m_1 to v, and the timestamp of each of these injections is noted. If the messages are then forwarded from one of the nodes $v' \in V'$ to the receiving node m_2, the time of reception of the message is noted. This allows the attacker to obtain a total delay $\delta_{(v,v')}$ of the respective message for the transit between v and v'.

As seen in Figure 3.3, it is not known to the attacker, to which of the nodes of V' v is connected. Whether a connection (v, v') exists is then estimated on the basis of the measured time delays and message reoccurrences. The idea is to find nodes that frequently forward messages injected into victim v to monitors with as little delay as possible. These are then most likely to be direct neighbours of v.

Similar to degree estimation, a large number of monitors connected to the clients is helpful to increase the probability of injected addresses being rediscovered. However, the technical minimum number of monitors is 1 per observed node. This is because, unlike in the degree estimation, registering immediate returns from the victim to a second monitor is not necessary for the connection inference.

3.1.4 Possibility of Combined Conduction

We propose a procedure that allows degree estimation and connection inference to be performed simultaneously. This exploits an inherent synergy between the methods in several aspects.

With respect to the active measurement, we note that the setups of the two measurements overlap when they are performed for multiple nodes simultaneously. For example, let there be a situation where address set B_1 is injected by monitors M_1 into victim node v_1. At the same time, address set B_2 is injected into another node v_2 by monitors M_2. In this sense, two separate degree estimation experiments take place at nodes v_1 and v_2. But because addresses from B_2 can also travel through the network to monitors M_1, connection inference takes place simultaneously. Thus, over the same injections that were intended for degree estimations, exactly the same observations are made that would also be made in a connection inference of the connection (v_1, v_2). When performing degree estimation measurements for all nodes of a node set V, this means that a connection inference of all connections between nodes of V is also performed. In practice, this means that only one measurement needs to be performed to apply both methods to the network. A separate measurement is therefore not necessary.

However, the simultaneous execution of both procedures for a node is only possible if more than two monitor connections can be established to it. If only one monitor can connect, connection inference alone is still possible. Alternatively, injections can be omitted in this case, and an injectionless deduction of the connections from information of the neighbouring nodes conducted: since in Bitcoin connections are always bidirectional, a node always has connections to the nodes that have a connection to it. For example, a network-wide connection inference can determine that a set N of nodes each has a connection to victim v. A connection inference of v itself would then no longer be necessary to be sure that the set of neighbours of v is exactly N.

The simultaneous execution is furthermore not associated with an increased number of monitor nodes. Since a monitor node can connect to all victim nodes that are to be measured at the same time, a total number of two monitors is sufficient for the complete measurement.

A second point of synergy between the two methods is the estimation that is performed after the measurement. This is because a node with a high node degree is inherently more likely to be connected to a random node than a node with a low node degree would be. In this sense, the node degree can be incorporated into the connection inference in the form of an a priori probability [Ros]. Therefore, it can be useful to pass the results of the degree estimation to the estimation of the connection inference.

Finally, the result of the degree estimation can also be used to translate the non-binary probability score results of the connection inference estimation into a binary classification. As we saw in Section 3.2, the determined node degree is the basis for applying a threshold. This exploits the property of nodes that the number of connections is equal to the estimated node degree.

All in all, both methods can be carried out in measurement at the same time using the same setup, and furthermore support each other in estimation. According to this approach, we also perform degree estimation and connection inference simultaneously in this thesis. In the context of this thesis, the joint execution of both of these methods constitutes a topology discovery attack.

3.1.5 Model Assumptions

In order to be able to carry out the measurement in the manner presented, four basic model assumptions are made about the Bitcoin network

1. *The ADDR-messages are flooded through the network as part of a gossip protocol.*
2. *The ADDR-messages used as bullets are uniquely identifiable.*
3. *The network is open, i.e. we can connect to any node to send and receive messages.*
4. *All network connections are fixed during the duration of the experiment.*

We note that these assumptions are consistent with those made by Neudecker et al. [NAH16]. Assumption 1) is satisfied by Bitcoin's network protocol as described in Section 2.3. Assumption 2) is ensured by the choice of bullet addresses as discussed in Section 3.4.2. The extent to which assumptions 3) and 4) hold is determined in complementary measurements conducted in Section 3.3.

As for the attacker model, the assumptions are that the attacker can:

1. create at least two sybil nodes
2. send messages from sybil nodes to network participants
3. monitor traffic between network participants and sybil nodes

In particular, it is explicitly not a requirement that the attacker can observe or modify message traffic between network participants (man-in-the-middle). The fulfillment of all three of these assumptions is a direct consequence of the openness of the network, as well as the open source nature of the network protocol. Therefore, no further vulnerabilities are needed to provide the attacker with the necessary capabilities to carry out this attack. We argue that this attacker model is not particularly strong, which means that any sufficiently motivated adversary should be able to conduct the attack without significant operational difficulties.

3.1.6 Vulnerable Design Decisions

Our attack relies on the way messages are relayed in the Bitcoin protocol ADDR as part of the gossip protocol. As described in Section 2.3.3, a node always forwards a received address to two neighbours. This design decision enables degree estimation using Equation 3.2.2 we presented in Section 3.2.2. Likewise, connection inference in the relay-based mode also builds on the gossip mechanism. Through this, as shown in Section 3.2.2, inferences about a node's neighbours might be possible by means of observable return probabilities. The behaviour described in Section 2.3.3 thus provides the sole core of this attack.

This is especially true for the idiosyncratic way messages relay is handled for the relay-based estimator. Although time-based estimation by its nature is not based on behavioural characteristics [NAH16, Wei19], we nevertheless locate an enabling circumstance in design decisions of the relay, i.e., in the implementation of the trickling.

Furthermore, we argue that the choice of the value for the rate-limiting of ADDR-messages, 10 seconds per address, also enables the attack. We assume that this parameter is low enough to allow the attack to be performed with a few sybil nodes in a reasonable amount of time.

We would like to point out that none of these points represent a vulnerability or a bug in the conventional sense. Rather, this attack is enabled by deliberate design decisions of the developer community. Countermeasures against the topology discovery presented in this thesis can therefore not be bug fixes, but must be done by rethinking the trickling mechanism and rate-limiting parameters.

3.2 Estimations

In the context of the experiment, the estimation is separate from the measurement. The injections and receptions of ADDR-messages with their corresponding timestamps are first measured and saved, before they are passed as input to a standalone estimator.

In this thesis, we have the estimator produce estimations based on two different approaches: time-based estimations and relay-based estimations.

3.2.1 Time-Based Estimation

For the time-based estimation approach, it is necessary to measure time delays of the injected addresses. For this purpose, the two procedures are executed as described in Section 3.1. When an injection is made into a victim v, an internal record is kept of the time at which the injection occurred. If any node v' forwards the same address to a monitor some time later, the total transit time of the address $\delta_{(v,v')}$ between the two nodes is recorded. Across all injections, this procedure results in a set Δ of time delays measured across all nodes of the network. Based on this set of measured time delays, a timing analysis is performed.

The estimation with the help of these timings is carried out in this thesis by applying statistical methods according to the procedure presented by Rosenblattl in his work [Ros]. To enable a comparison with the alternative estimation method, we would like to give a brief, superficial overview of the underlying idea as presented in his thesis, but without going into too much detail.

Degree Estimation

In the degree estimation only the number of direct relays is important. That is, the number of delays where the input node and the output node are identical so $K := |\{\delta_{(v,v')} \in \Delta \mid v = v'\}|$. Also, let k be the number of addresses recovered after a single injection. By the fact that node v selects exactly two peers without duplications for relaying, we know that $0 \leq k \leq \min(2, \deg_{mon}(v) - 1)$ where $\deg_{mon}(v)$ is the number of monitor nodes simultaneously connected to v. We also know that k follows the hypergeometric distribution of $H(\deg(v), \deg_{mon}(v) - 1, 2)$, where $\deg(v)$ is the node degree including monitor nodes. With the associated probability mass function h, it then follows for the likelihood function L

$$L(\deg(v) = d \mid K) = \prod_{k\in K} P(k \mid \deg(v) = d) = \prod_{k\in K} h(k \mid d-1, \deg_{mon}(v) - 1, n),$$

The estimated node degree can then be derived from this using maximum likelihood estimation, as $d_{MLE} = \operatorname{argmax}_d L(\deg(v) = d \mid K)$. Alternatively, an expected node degree distribution can also be included in the estimation as an a priori probability. This expected distribution could be the result of another research work. Then a maximum a posteriori estimation can be performed with $d_{MAP} = \operatorname{argmax}_d L(\deg(v) = d \mid K)P(\deg(v) = d)$ [Ros].

For this thesis, the time-based estimator produces degree estimation results using the maximum a posteriori estimation. The node degree distribution for the a priori probabilities is taken from Grundmann et al. [GBH21].

Connection Inference

When an ADDR-message is forwarded from one node to the next, there is a time delay in each step through the network. This is due to processing overhead and network delay, but in the case of Bitcoin mainly due to intentional delays (so-called trickling, see Section 2.3). This means that for individual messages the delay over several steps is the sum of the individual delays. The expected amount of time delay can at best be estimated, but in general it can be assumed that the delay increases with the number of steps in the network.

The general idea of connection inference through timing analysis is now to measure the times needed to transfer data from one peer in the network to another. Based on these measurements, the topology of the network can be identified by drawing conclusions about the distances and connections between the peers. Basically, Timing Analysis exploits the fact that the transmission speed of data in a network depends on the distance between peers. The farther apart the peers are, the longer it takes to transmit the data. By measuring the transmission times required, one can draw conclusions about the topology of the network. For example, if one finds that it takes longer to transfer data from peer A to peer C than from peer A to peer B, one can conclude that peer C is farther away from peer A than peer B.

In this way, we can estimate the connections and distances between peers in the network based on the delay set Δ. The delays used in this procedure are exactly those where the input node and the output node are not identical, i.e. $K := |\{\delta_{(v,v')} \in \Delta \mid v \neq v'\}|$. For the purpose of this estimation, we add expiries to the set of observed deltas, i.e. $\delta = \infty$ for all bullets that were injected into a victim but not returned by another client. The following a priori probabilities are then included in the probability estimate of the existence of a direct connection $(v, v') \in E$:

- the probability $p(r \mid c)$ that a delta between a directly connected pair of nodes is a recurrence
- the probability $p(e \mid c)$ that a delta between a directly connected pair of nodes is an expiry
- the probability $p(r \mid \neg c)$ that a delta between a pair of nodes not directly connected is a recurrence
- the probability $p(r \mid \neg c)$ that a delta between a pair of nodes not directly connected is an expiry

These probabilities are initially unknown for the Bitcoin network. Rosenblattl, however, presents a method of approximating them for the network. To do this, the injection and forwarding of address messages, as would take place in the context of the experiment, is simulated. A large number of messages is used in a simulated network, which in terms of size, node degree distribution and connectivity can be taken to be as similar as possible to the real network to be measured. The a priori probabilities given above can then be observed in the simulation and used to estimate the real measurement data. This is therefore done under the assumption that the network to be measured behaves similarly to the simulated network in this respect.

In addition, the node degrees of both nodes are included in the estimation of a direct connection between v and v' as another a priori probability $p((v, v') \in E)$. This is because in a random given network, nodes with large node degrees inherently have a larger probability of being connected, since a larger proportion of all edges lead to the nodes, respectively. For the time-based estimator, node degrees previously estimated in the degree estimation can be used here.

The connection inference is a special case of a path length estimation for the case that the length of the path is $l = 2$, i.e. only the partial delays of the edges (v, v') and (v', m') are included in the total delay. Since trickling is only added to outgoing messages and injecting monitor m refrains from doing so, we assume that the delay of (m, v) is negligible here. Moreover, based on the Bitcoin Core reference client delay equation presented in Section 2.3.3, we know that the individual partial delays follow an exponential distribution with average $\lambda = \frac{1}{30s}$. Therefore, the sum of the partial delays across $l = 2$ trickling processes follows an Erlang distribution $Erl(\lambda, 2)$. Let f be the associated probability density function, the probability of a given time delay δ assuming direct connection is therefore $P(\delta \mid v \rightarrow v') = f(\delta \mid \lambda)$ Using the observed recurrences $\Delta^r_{(v,v')}$ and expiries $\Delta^e_{(v,v')}$ relevant for node pair (v, v'), we can summarise that the posterior probability $P((v, v') \in E \mid \Delta_{(v,v')})$ of a connection (v, v') can be estimated using likelihood $L((v, v') \in E \mid \Delta^r_{(v,v')}, \Delta^e_{(v,v')})$ of the same [Ros].

It is worth noting that connection inference based on timings is not based on characteristic behaviour of Bitcoin clients. Instead, the observed time delays are inherent to all networks that distribute information through flooding. The timing analysis used in this thesis is based on the method presented by Neudecker et al., who conducted a comparable timing analysis based on another flooded address type [NAH16]. For this thesis, the time-based estimator is taken from the work of Jakob Rosenblattl, who developed it based on the work of Neudecker et al. Their estimator works by training a model based on data generated by a simulated network, and then statistically comparing the measured experiment data to this trained model. To change the results, the model can be tweaked by providing different training data generated by simulated networks generated with different network parameters. In the end, the time-based estimator produces a degree estimation based on *maximum a posteriori estimation (MAP)* together with the corresponding posterior probability. The connection inference of this estimation method results in a posterior probability for every possible pair of two peers. A decision whether a connection should be assumed between two peers can then be made based on this posterior value.

Intuitively, one can simply apply a threshold value t for the decision rule $D(x)$:

$$D(x) = \begin{cases} \text{True}, & \text{if } x \geq t \\ \text{False}, & \text{otherwise} \end{cases}$$

for all possible connections between two nodes for which a posterior probability above this threshold is estimated, it is then be assumed that a connection exists. As a threshold value, one can use a value that is known to produce good results. However, we propose an alternative method with a flexible threshold for the estimation. Instead of the decision depending only on the threshold, the node degree previously estimated in the degree estimation is also taken into account. If the estimator has estimated a node degree of n for a node, then exactly n links with the highest assigned posterior value are assumed as connections to possible neighbouring nodes for this node. This method has the advantage that the degree estimated by the degree estimation is better reflected in a network graph created by connection inference. Alternatively, the flexible threshold can be based on ground truth node degree knowledge if the attacker wants to apply the connection inference independently of the degree estimation.

3.2.2 Relay-Based Estimation

The relay-based estimator produces the estimation based on the number of messages that reappear after injection. In contrast to the time-based estimator, the measurement of exact timing data is therefore not strictly necessary. We note that the timing data required for the timing analysis can simply be reduced to occurrence counts for this estimation method.

Degree Estimation

For an estimation of the node degree, the attacker exploits the fact that the mechanism for ADDR-message flooding is to some extent predictable.

As described in Section 2.3, we know that for every injected message $m \in M$, the victim node v randomly selects exactly two of its peers for message forwarding. As no peer is selected twice, and the injecting monitor is not selected [Prob], every neighbour of v that was not the originator of the message, has a chance of

$$p = \frac{2}{\deg(v) - 1} \tag{3.1}$$

of being passed the message [1].

We can derive this equation as follows: Node v selects exactly two of its neighbours. Since the injector is not considered, the set of all considered nodes has the size $k := \deg(v) - 1$. Because the order of nodes in the selection does not matter, there are $\binom{k}{2}$ possible choices of pairs of nodes to forward to. At the same time, there are exactly $k - 1$ pairs of nodes for a given neighbour node of v in which it is contained, since the order of the pair is also indifferent here. Thus, the chance to choose one of these relevant pairs from the set of all possible pairs is exactly as follows:

$$\frac{k-1}{\binom{k}{2}} = \frac{k-1}{\frac{k!}{2!(k-2)!}} = \frac{2!(k-1)(k-2)!}{k!} = \frac{2!(k-1)!}{k!} = \frac{2!}{k} = \frac{2}{k} = \frac{2}{\deg(v)-1}.$$

q.e.d.

More specifically, this equation can be rearranged to get the victim degree given the proportion of injected messages that were observed to be passed to monitor nodes. Since this fraction is well observable experimentally, we can rearrange the equation to obtain the node degree. Because the node degree is briefly increased during the measurement by the monitor nodes, but in the end only the node degree without monitor nodes is of interest, it makes sense to distinguish between these two types of connection in the formula. For this, let n be the victim degree without monitor nodes, n_{mon} the number

[1] Of course, this is only true under the assumption that the node degree is not unrealistically small, i.e. $\deg(n) > 2$. For the edge case of $\deg(v) = 2$ we have $p = 1$ and for $\deg(v) = 1$ there are no peers that were not originators

of connected monitor nodes including the injector and all receptors, and $p_{observed}$ the observed recurrence rate as the total number of observed immediate message recurrences across all monitor nodes divided by the total number of injected messages. For the set of observed immediate message recurrences, only those that come directly from the victim node are to be counted. Bullets that are passed on from another peer back to the monitor through a longer route are not relevant for this experiment.

We then know that each of the $n_{mon} - 1$ receiving monitor nodes had an equal chance of

$$p = \frac{2}{n + n_{mon} - 1}$$

for receiving any of the messages. Solving this equation for n yields

$$p = \frac{2}{n + n_{mon} - 1} \tag{3.2}$$

$$p \times (n + n_{mon} - 1) = 2 \tag{3.3}$$

$$n + n_{mon} - 1 = \frac{2}{p} \tag{3.4}$$

$$n = \frac{-n_{mon} \times p + p + 2}{p} \tag{3.5}$$

By entering the observed probability $p_{observed}$ for p and the known number of monitors into the equation, we get the estimated degree of the node. However, when using more than the minimum of two monitor nodes for the measurement, we must also take into account that the final result must be adjusted to the number of monitor nodes. This is done by simply dividing by the number of simultaneously connected receiving monitors (i.e. $n_{mon} - 1$). In this way, it is compensated that the total number of observed returns was previously summed over all monitors. Combining the sets of receptions and injections each over all monitors does not influence the degree estimation in any other way.

All in all, we get

$$n = \frac{-n_{mon} \times p_{observed} + p_{observed} + 2}{p_{observed} \times (n_{mon} - 1)} \tag{3.6}$$

This equation can then be used to estimate the node degree of any Bitcoin client based on the message relay count alone, without any timings. Should p be particularly small, as might be the case for nodes with a large degree, then many injections are required to acquire a small number of rediscoveries. This means that a small variation in absolute terms in the number of messages recovered can have a large impact on the estimated node degree, as can be seen from Equation 3.2.2. This fact leads to the assumption that there is an intrinsic benefit in conducting as many injections as possible with as large a number of connected monitor nodes as possible to obtain accurate results.

We also point out that injected bullets forwarded from the victim to a monitor do not necessarily have to be part of these immediate message recurrences. Theoretically, they could also have, with a certain probability, taken a path of the form $monitor \rightarrow victim \rightarrow peer_1 \rightarrow peer_2 \rightarrow victim \rightarrow monitor$. For degree estimation these bullets would then not be relevant, but difficult to detect. Possible ways to filter out these bullets would be: 1) set a maximum value for the time delay to filter out all unusually long paths. 2) ensure that the receiving monitor does not receive any other ADDR-messages between injection and receipt. In the case of direct forwarding, these should have included the bullet in question. 3) ensure that the receiving monitor is not equal to the injecting monitor. This could happen with such false message recurrences, but never with a true immediate relay.

This idea to use observed address return rates to estimate the degree was first presented by Biryukov et al. [BKP14]. In their paper, they presented Equation 3.1 and described how they used this information to deduct node degrees. We use this idea to build the degree estimation part of our relay-based estimator for this thesis.

Connection Inference

For a path between the injecting and receiving monitor across a number of Bitcoin nodes, the probability that an address message is forwarded along this path decreases proportionally with the length. The final probability is calculated as the product of the individual probabilities $p(c)$ for all nodes c along the path. For a path $monitor \rightarrow victim \rightarrow client \rightarrow monitor$, for example, this means that each address message used follows this path with probability

$$\begin{aligned} p(total) &= p(c_1) \times p(c_2) && (3.7) \\ &= \frac{2}{\deg(c_1) - 1} \times \frac{2}{\deg(c_2) - 1} && (3.8) \end{aligned}$$

as can be derived from individual probabilities according to Equation 3.1. Because the degree of a given node can be expected to be much greater than three, the probability for a given step is correspondingly much smaller than 1. Even if the node degree is only slightly larger than 3, it is easy to see from the equation that the probability of an address occurring decreases with distance, since $p(c) < 1$ can be assumed for all nodes c. With realistic node degrees, which in practice can be many times larger, an amplification of this effect is to be expected.

However, we can only assume this effect for short path lengths. For long path lengths, we expect the opposite effect: Since the address is forwarded twice in each step on the path, the number of multiples of each address in circulation increases exponentially. In this way, the flooding protocol ensures that the address reaches all network participants via a sufficiently large number of forwardings. We expect that through this effect, network nodes far away from the victim node also return the bullet addresses in large quantities. To filter out these returnees, we can filter returning bullets according to how long they were in transit ahead of the relay-based connection inference. In this way, the influence of this interference factor decreases, and we can assume more returning bullets for less distant nodes. We argue that this filtering by return times below a fixed threshold does not constitute a timing analysis as with the time-based estimation. This is because the application of statistical methods, as done in Neudecker et al. and Rosenblattl's work, is missing. In addition, the estimation here is still based on idiosyncratic behaviour-based observations, and not on inherent flooding delays as is usual in timing analysis. Nevertheless, the consideration of timings in this naïve estimation approach can be seen as a kind of informal, naïve timing analysis. Through this, the influence of this time limit highlights the general influence of timings on the estimation, and thus reflects the relationship between relay-based and time-based estimation.

We suspect that clients that forward many messages previously injected into v to monitors may then have a higher probability of being adjacent to v. Should this theory hold true, the relay-based estimation can be performed by sorting all remaining clients c for each victim v according to the forwarding rate of relevant messages. Again, only messages that were previously injected into v are relevant here. The certainty score assigned to the possible connection (v, c) by the estimator is then the number of addresses injected in v that came back to the monitor through c divided by the sum of all addresses that came back through any client $c' \neq v$ after being injected into v.

For measurements outside artificially optimizable testbed environments, a practical problem is the variable number of connections from monitor nodes to individual peers. To account for the fact that some nodes have more connections to monitors than others, the average number of connected monitors per total connection time can also be calculated for the respective node. Then either the certainty score or the number of bullet returns counted is divided by this number of monitors. This compensates for the fact that a peer that is connected to many monitors is more likely to be the source of a bullet return.

After that, similar to the flexible threshold of the connection inference in the time-based estimator, it can be useful to use a previously determined node degree from the degree estimation. The estimator then again estimates a connection between the victim and the given number of nodes ranked best by this method.

3.3 Complementary Measurements

Before the main experiment, we conducted a series of smaller measurements. The goal is to investigate general aspects of the Bitcoin network and the node addresses. Above all, the experiments include the gathering and validation of node address datasets, but also general node reachability and connectivity.

The technical execution of these experiments was done with the help of functionality implemented for the framework of the main experiment. Fundamentally, there is an inherent similarity of these experiments to the main measurement in the sense that Bitcoin connections are established and the response to certain message types is observed. This made it possible to use large parts of the framework for this purpose with only minor adaptations. Therefore, it was possible to carry out these experiments with only little additional effort after the framework had been programmed.

The results of these experiments are also partially incorporated into the design decisions and parameter choices of the main experiment. Hence, these complementary measurements are an important first step for the realization of this thesis' main experiment.

The experiments we conducted in the process are:

1. GETADDR and address gathering: We collect lists of active nodes by GETADDR requests and by Bitnodes.io to have a basis for the further experiments. We receive over $200,000$ addresses of which about $15,000$ can be considered currently active.

2. Address type statistics: Examining the distribution of the different address types in the two collected datasets, as well as the respective service flags and age of the address entries, we get a first basic overview of the expected structure of the bitcoin network. We find a considerable percentage of Onion-addresses. Almost all nodes indicate in their service flags that they support the network services required for the experiment.

3. Node fluctuation rate: We examine the churn rate that the network addresses exhibit over time. This has an influence on how long the topology measured by our attack remains valid. The churn we observe is 30 % in 24h, differing greatly from values reported in related work.

4. Monitor connection duration: The average time a monitor node is able to maintain connections to network nodes is an indicator of how long the measurement can last without being distorted. We find that our monitor nodes can maintain connections to the network for an average of more than seven hours, longer than the expected measurement duration.

5. Node neighbourhood stability: How long the topology identified by our attack remains valid is reflected by the average time a network node maintains connections to its neighbours. At the testbed node, we observe that more than 80 % of the connections remain stable for more than an hour. This allows us to assume that the network connections between nodes are stable enough to be measured.

These experiments are presented in detail below.

3.3.1 Testing Environment

To test the basic functionality of the developed framework and to perform the tests from this section, as well as Section 4.1.2, a test setup was built. This primarily includes a filter proxy to prevent ADDR-messages from leaving the local network of the test device. The reason for this being that, for moral reasons, we do not want to disturb the Bitcoin network more than necessary. In order not to disrupt Bitcoin's operations, ADDR-messages should only reach the network as part of the main measurement.

However, a simple firewall cannot be used to filter the packets: the packet type distinction is within the TCP payload and not in the TCP header. A regular firewall usually only looks at the TCP header and would therefore not necessarily recognize the Bitcoin packets correctly. To make matters worse, we do not want to filter all Bitcoin packets, as the victim node should be able to connect to nodes outside the local network. Consequently, one requirement of the filter is that the exact type of Bitcoin packet should be detected, and "harmless" packets such as PING or VERSION should be forwarded. For this reason, we decided to use a filter proxy: this can perform deep packet inspection, i.e. look inside the packet payload and thus detect the packet type. We verified the functionality of the filter-proxy in console outputs and packet captures.

3.3.2 GETADDR and Address Gathering

For all experiments, it is necessary to connect to a number of different active nodes. One prerequisite is therefore to know the addresses of as many currently operational nodes as possible. A possible way to gather a list of active nodes is to connect to all the nodes that are already known and send GETADDR-requests to ask for more addresses. This short payloadless Bitcoin-message serves exactly this purpose in the protocol: to enable nodes to inform themselves about other nodes without having to wait passively for incoming ADDR-messages [Proc].

To this extent, we used GETADDR-requests to gather the addresses needed for our experiments. We tried to connect a single monitor with a total of 7000 nodes known initially from previous test runs. Roughly 3700 of these connection attempts were successful. The monitor then sent GETADDR-messages to each of the connected peers.

In response to this inquiry, a total of 3, 202, 410 addresses were returned across all peers. After taking into account double entries, the returned dataset consists of 212, 660 individual addresses.

Popular Nodes: It is clear that individual addresses can occur multiple times across all *GETADDR* responses of all nodes. In fact, taking into account duplications, the number of addresses received decreases from 3, 202, 410 to 212, 660 by 93.36 %. This reveals a large overlap between the addresses provided by the individual nodes. Within this overlap, certain addresses may, on average, be mentioned more frequently by different nodes than others. If this difference becomes significant, there may be a single node, or a group of nodes, that are particularly popular in the network. We suspect that this level of popularity could have several reasons: 1) the node advertises itself aggressively 2) the node connects to many nodes at the same time or in close succession 3) the address of the node does not change for a long time because it has a static IP or Onion-address 4) the node is a reliable connection partner and is therefore advertised more often.

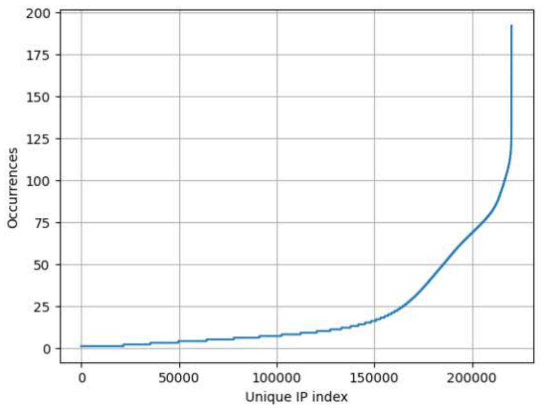


Figure 3.4: Number of occurrences of a node's IP address in the *GETADDR*-responses gathered

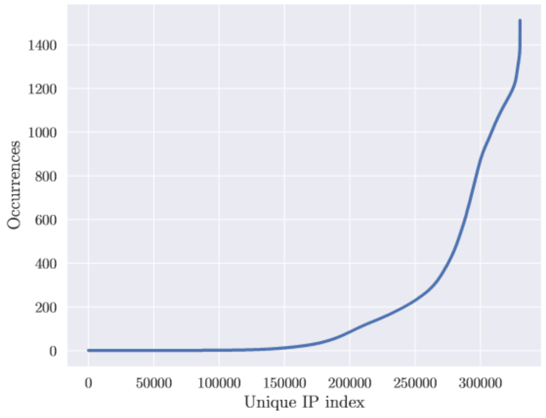


Figure 3.5: Number of occurrences of a node's IP address in the *GETADDR*-responses gathered, as presented by Eisenbarth et al. [ECP21]

Knowledge of particularly popular nodes is also of academic interest. This is because it may enable certain attacks, as these nodes could be a target of choice for attackers [ECP21]. To provide an understanding of potential popular nodes, we therefore also examine the results of our measurement for such nodes.

A histogram of the mention rates can be found in Figure 3.4. We see that this curve follows the same general shape as the one measured by Eisenbarth et al. [ECP21]. However, we note that in our graph the relative distribution of popular, average and unpopular nodes is shifted towards nodes being generally less frequently mentioned. This means that the fraction of identified popular nodes was smaller in our dataset compared to Eisenbarth et al. In our measurement, the total number of occurrences was significantly lower for the given addresses, weakening the conclusiveness of the results. Still, we argue that our measurement confirms the finding that there is a group of popular nodes.

Bitnodes.io: A public service whose goal is also to map as many current addresses of the Bitcoin network as possible is *Bitnodes.io*. This service provides a public API that we can also use to obtain current addresses as an alternative to our own measurement. The address gathering procedure of Bitnodes is similar to ours: according to information given on their website, Bitnodes has one or more crawlers set up, which try to get a picture of the current nodes by means of *GETADDR* queries. In contrast to us, the crawler here works recursively: by further trying to connect to any node found to be advertised by GETADDR responses, it is supposed to be ensured that only reachable addresses are collected. A successful connection then serves as both a proof of the validity of the returned address, and a new target to send further GETADDR to [Bitb].

In fact, it is likely that nodes respond to a GETADDR request with addresses that are no longer available or outdated. This is because the default Bitcoin client, when prompted with a *GETADDR*-request, takes addresses out of its address memory that are up to 30 days old since the time they were last seen online [Prob]. If some of these addresses went down or changed their IP address in this time, these addresses would then be irrelevant for our purposes.

Our previously presented workflow lacks a procedure for filtering out such unreachable nodes. It is therefore to be expected that Bitnodes reports significantly fewer addresses. In fact, a query of the API via the URL „`https://bitnodes.io/api/v1/snapshots/latest/`“ returns only about 15, 000 addresses. We find that our dataset is missing only 100 addresses that are present in the list returned from Bitnodes, while, on the other hand, having more than 200, 000 addresses more. It is therefore roughly a superset of the Bitnodes dataset.

While we certainly have many outdated addresses, we argue that Bitnodes may in turn not list all accessible addresses; if a node rejects only some requests it might be considered unreachable for Bitnodes, while it could be reachable again for our attempts. This would then constitute a false negative in the measurement of Bitnodes. Nevertheless, for the following experiments and the main experiment, we use the list of Bitnodes to limit the set of addresses in a meaningful way.

3.3.3 Address Type Statistics

After receiving a list of accessible node addresses from Bitnodes, another potentially interesting statistic is the one about the address type and possible fluctuations over time. There are three different address types for Bitcoin nodes: 1) IPv4 2) IPv6 3) Onion, where Onion-nodes are only accessible as a hidden service on the TOR network and are represented by an ".onion"-address instead of an IP address, similar to other hidden services on the TOR network. Onion-addresses can be further subdivided into the old V2 and the new V3 ones, whereby the old ones are 16, and the new ones are 56 characters long [TB]. Since these Onion-nodes cannot be reached directly by IP addresses, in practice a connection must usually be established over a proxy that feeds incoming traffic into the TOR network.

We suspect that such a connection through proxies and Onion-routing could prove problematic in practice for our experiment. This is because in the main experiment we carry out an analysis of time measurements with as much precision as possible. We suspect that these could become inaccurate over several forwardings. This would then in turn influence the timing analysis. Because of this, we have decided to carry out the measurement only for IPv4 and IPv6 nodes for the course of this thesis. The measurement therefore explicitly only concerns a subnetwork of the entire Bitcoin network. Performing the experiment for Onion-nodes and possibly solving the timing problem is a subject for future work. It is therefore all the more important to find out how the address types are distributed in practice and, in particular, what proportion of addresses are allocated to the TOR network.

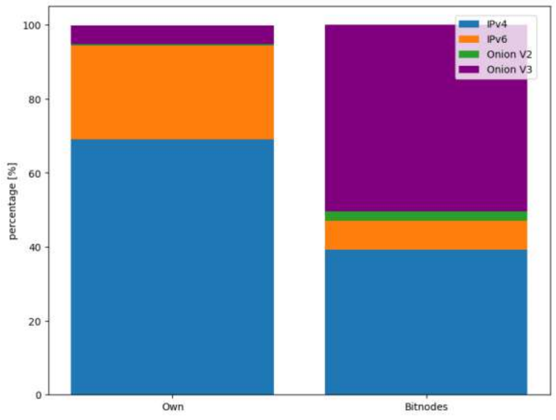


Figure 3.6: Comparison of address type percentages between our dataset and the one returned by Bitnodes.io

	IPv4	IPv6	Onion V2	Onion V3
Own Dataset	69.01 %	25.39 %	0.31 %	5.16 %
Bitnodes.io dataset	39.29 %	7.72 %	2.49 %	50.51 %

Table 3.1: Address type percentages between our dataset and the one returned by Bitnodes.io

Figure 3.6 and Table 3.1 show the rounded percentage distribution of addresses obtained from Bitnodes in the individual address types. This distribution is consistent with the statistics provided by Bitnodes on their node statistics webpage [Bita]. It is compared with the percentage distribution within the data set we determined in the previous side experiment.

We note that the two distributions significantly differ in the number of Onion-addresses present. While our dataset consists of only 5.47 % Onion-addresses, the dataset from Bitnodes is made up of 53.00 % Onion-addresses. Our decision to use the Bitnodes dataset for the main experiment and exclude Onion-addresses as previously discussed therefore results in the effective size of network measured being halved.

It is also striking that this address type distribution differs significantly from that found by Deshpande et al. The researchers had been able to find on average 83.83 % IPv4 nodes, 13.49 % IPv6 nodes and 2.69 % Onion-nodes across 6 months in 2018 [DBG18]. Thus, the difference from our datasets suggests a shift toward IPv6 and Onion-addresses in recent years.

Service Flags: As described in Section 2.3.2, each address entry is assigned a series of service flags. This is a single integer number whose equivalent binary notation indicates which services the respective Bitcoin client offers. The 64 individual bits are interpreted

	our	bitnodes.io
all flags	5.0965 %	0.0000 %
no flags	0.0009 %	0.0207 %
NODE_NONE	18.4557 %	10.3236 %
NODE_NETWORK	81.5443 %	89.6764 %
NODE_BLOOM	31.4144 %	32.3994 %
NODE_WITNESS	98.2778 %	99.5100 %
NODE_COMPACT_FILTERS	10.6198 %	5.3481 %
NODE_NETWORK_LIMITED	97.7804 %	98.3162 %
at least one experimental flag	11.6166 %	2.0910 %
no experimental flag	88.3834 %	97.9090 %
conflicting flag information	10.5450 %	N/A

Table 3.2: Percentage of addresses in the datasets, where specific service flags were set

as flags, each of which stands for a possible service. Five of these bits are defined as services in the Bitcoin Core source code. These represent the basic functionality of a Bitcoin client. All other bits can be used for experimental services that may be offered by alternative clients. Only the flag *NODE_NONE* is a special case, as it is considered active exactly when the *NODE_NETWORK* flag is not set. It should be noted that each client is free to choose the service flags it wants to advertise, without having to actually provide the services. An overview of the services advertised by addresses in the Bitnodes dataset as well as in the dataset we collected can be seen in Table 3.2.

Address Age: Similar to service flags, timestamps are also assigned to the received addresses. These represent the time when the respective peer was last observed as active. These service flags can thus give us an indication of how up-to-date the data in the two datasets is.

As expected, both datasets show an accumulation of entries younger than one week. For the Bitnodes dataset, this part is 84.67325 %, in the collected with GETADDR 37.3225 %. After that, the occurrence of ages in the getaddr-dataset is evenly distributed up to an age of just over 30 days. The Bitnodes dataset has few entries in this age range, indicating that the way the dataset was collected does indeed reflect current nodes. Both datasets have only a small proportion of addresses older than that. All in all, the timestamps support our assumption that the Bitnodes dataset generally has more up-to-date addresses.

3.3.4 Node Fluctuation Rate

Another interesting data point is how the addresses fluctuate over time. This means that while some addresses are stable within the datasets and still accessible after a certain amount of time, other nodes may leave the network or join it. A reason for this can be that the physical node is switched off, i.e. has actually left the network. Particularly with dynamic IP addresses, another reason is that the address of the node has simply changed, and the same node can now be reached at a different address. A low fluctuation rate suggests a stable network, which is possibly also reflected in the topology measured in the main experiment.

The determination of the fluctuation rate only makes sense if the data set is sufficiently up-to-date. We know that the dataset collected by us using GETADDR is not filtered by its up-to-dateness and therefore contains many outdated addresses. However, the dataset provided by Bitnodes is well filtered for actuality. Thus, we argue that the fluctuation rate measured for this dataset may better reflect the actual rate of the network.

Therefore, we determine the fluctuation of addresses only in the dataset obtained from Bitnodes. This is done by directly comparing which addresses were added or dropped after a set amount of time. We calculate the fluctuation rate as $1 - \frac{(n_{stable})}{n_{total}}$ where n_{stable} is the number of addresses that are in both datasets and n_{total} the number of addresses that are present in any of the two sets. This metric is sometimes also referred to as *churn rate* [ECP21, ISTY19].

We find that within 24h, the network had a total churn of 30.65287 % with 4620 out of 15072 addresses changing over time. This value is composed of 15.80474 % of the old nodes leaving and 20.27460 % of the new nodes having recently joined. Meanwhile, the total number of reported active addresses changes by only 5.30892 %. We also note that the churn rate differs between the different address types: while IPv4 and IPv6 addresses have a fluctuation rate of only 11.71185 % and 18.90145 %, respectively, Onion-addresses have a rate of 49.19366 %.

It is worth noting that our results therefore differ greatly from those of Eisenbarth et al., who found the rate of both leaving and joining nodes across 24 hours to be stable between 5 % and 6 % [ECP21]. These results can also generally represent a factor that limits the validity period of the main experiment's results. This is because a topology that has been found is no longer valid for new nodes that have joined the network or for those that have left it. The node fluctuation rate thus plays an important role in justifying the results of the main experiment, in addition to the average connection duration between persistent nodes.

3.3.5 Monitor Connection Duration

To tune the duration of the experiment for the best possible success, it is good to know how long connections between monitors and nodes of the network last. This then makes it possible to select the parameters of the injections so that the total runtime remains as far as possible below a value adapted to the average connection duration. This prevents too many nodes from losing their connection during the experiment and thus disturbing the measurement.

For this purpose, we performed the following measurement: simultaneously connecting to all IPv4 and IPv6 nodes and maintaining the connection for up to 24 hours. The time

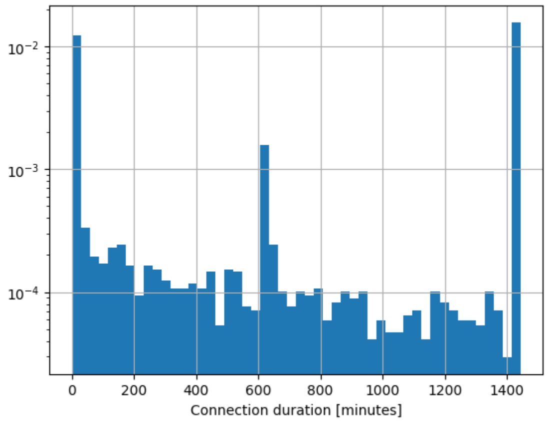


Figure 3.7: Time after which the connection of the monitor to a node is broken. Limited to 24h. Normalized and on a logarithmic scale.

of successful connection establishment is measured, as well as the time of connection termination. Nodes to which no connection could be established are not included.

The results of this measurement are shown in a histogram in Figure 3.7. It can be seen that a large number of connections break off within the first few minutes. Connections that last longer are rather stable. 45.26 % of all connections last until the end, and 34.63 % of connections that last longer than the first 60 seconds break off prematurely. The average connection duration for nodes that last longer than 60 seconds but not until the end is 7.8 hours. The median connection duration for the same set of nodes is 8.27 hours.

In principle, in addition to network problems, the reason for a disconnection can also be that the remote node wants to drop the connection, possibly in order to connect to other nodes.

For our main experiment, a disconnection to a node would have the following impact, depending on the role of the peer in the measurement: If the connection to a client is lost, the connection between victim nodes and this client can only be inferred with a low certainty. In case of a connection loss to a victim, the accuracy of both connection inference and degree estimation of this node decreases. In either case, if the connection is lost, the estimator can at least use the data that was collected up to that point.

Since we have observed in this experiment that the connections of the monitors are rather stable, we can assume that the measurement can be conducted without major disruptive factors. We can set the experiment parameters to a relatively long measurement period of more than one hour without having to worry that the proportion of nodes that can be measured well continues to drop rapidly compared to shorter measurement periods. The results of this measurement indicate that connections breaking off during the running experiment should not be a major disturbance factor for an experimental time of less

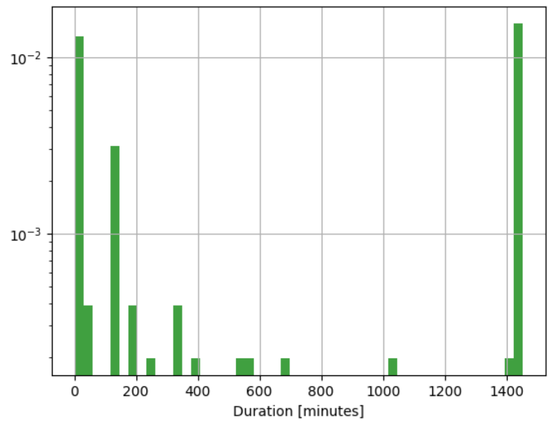


Figure 3.8: Time after which connections of a node to its peers are broken. Limited to 24 h. Normalized and on a logarithmic scale.

than a few hours.

3.3.6 Node Neighbour Stability

Similar to the node connection duration experiment, it is also interesting to know how long a client of the Bitcoin network keeps the connections to its peers. This refers to the average duration of the connection between the victim and a client. This information is good to know to be able to estimate how large the accuracy of the connection inference can be. A long connection duration here would indicate that the results of the main experiment are valid for longer. The average connection duration thus also represents a further limiting factor for the measurement duration of the main experiment: Since the estimator assumes that the connections in the network are stable, a suitable measurement can only be made if the measurement duration is set to a suitably low value in relation to the average connection duration.

For this reason, we prepared the following experiment: 1) A victim node under our control connects to the network as widely as possible. 2) Using RPC calls, the list of all peers connected to the victim node is queried at regular intervals. 3) The duration of each connection is calculated.

As a victim node, we run a Bitcoin Core version 22.0 client without any modifications. We can assume that this node behaves in a way that is typical for the Bitcoin network, especially when it comes to actively disconnecting and reconnecting to other nodes.

The measurement is performed for 24 hours, and the sampling interval is 30 seconds. Initially, the node is connected to 109 peers, at the end of the experiment to 81. During the experiment, 94 connections were broken and 66 were established. 7 connections are resumed later after their termination, after an average of 10.6 h with a standard deviation of 7.2 h.

A histogram of connection durations is given in Figure 3.8. As expected, we see that the histogram follows the general shape of the node connection duration histogram in Figure 3.7: Many connections last only a short time and just as many last a long time. Interestingly, this histogram also shows a spike of connections being dropped after roughly 10 minutes. The average connection duration in this experiment is 11.7 h, the median is 3.9 h with a standard deviation of 11.7 h.

We argue that the large number of short-lived links is of little importance for our main experiment: What is important for us is not so much the average connection time, but the fraction of stable connections at a given moment. In the data we collected in this experiment, we see that at each sample time, between 83.9 % and 100 % of the connections persist for more than one more hour before breaking. The average percentage of such connections at any one time is 98.8 % with a standard deviation of 3.0 %. For long-lived connections of more than 20h remaining time, this average value is 83.0 % with a standard deviation of 7.8 %.

In summary, the connections between nodes in the Bitcoin network appear to be stable. We expect that this means for the main experiment that the measurement can be carried out over a longer period of time without significantly distorting the results.

3.4 Main Experiment

In this section, we describe how the main experiment is performed after the preliminary measurements were carried out. To this end, we first describe decisions regarding the main parameters of the measurement, such as the selection of the network participants to be measured and the bullet addresses used for injection. We then discuss and justify the scaling mechanisms of the experiment. Finally, the technical experimental setup is described.

The results of the experiment are interpreted, evaluated, and discussed in Chapter 4.

3.4.1 Peer Selection

Deciding which participants of the Bitcoin peer-to-peer network will be measured, we have defined requirements for the addresses:

1. Only current addresses. Outdated addresses are more likely to be unreachable.
2. Only reachable peers. This is technically necessary to establish connections from monitors to peers. In particular, peers that do not accept incoming connections, for example because they are behind a NAT, are not measured.
3. IPv4 and IPv6 only. Onion-addresses are not measured to simplify the experiment.

As described in Section 3.3.3, we can use the address dataset provided by bitnodes.io to obtain addresses with little effort. Due to the pre-filtering by the provider, 1) and 2) are already ensured. The addresses for the test are obtained from Bitnodes immediately before the test is carried out. About 14000 addresses were acquired.

A pre-filtering of the addresses to a ".onion" ending is then also easily possible. This left about 7000 addresses for the measurement. In this respect, the experiment in this thesis represents a measurement of only a subnetwork by this selection of peers. In this setup, this subnetwork thus comprises about 50 % of the entire reachable network.

3.4.2 Bullet Choice

It is particularly noteworthy that before forwarding an ADDR-message for flooding, it is not verified by the nodes if there is a Bitcoin service reachable at this address. This means that random addresses with no real node behind them are forwarded with the same probability as those that register an actual client. The reachability of the bullet address is not checked by the relaying node, but only decided on the basis of the network the address is part of. This makes it much easier to choose addresses for injection, as there does not need to be a service responding to requests behind bullet addresses. The exception is addresses in ADDR-messages that the victim node knows it cannot route to, like IPv6 addresses to a node that only supports the IPv4 protocol or IP addresses that are associated with local networks. These are randomly forwarded to either one or two neighbours, which can distort the estimation. This must therefore be taken into account when choosing which addresses to inject: they must be chosen in such a way that no node considers them to be non-routable. In particular, this means not using Onion-addresses as it can be assumed that a large portion of nodes cannot connect to Tor hidden services. Furthermore, it is important to note that this restriction does not only apply to the immediate target of an injection: one could assume that it is safe to send IPv6 addresses as bullets to those nodes that are themselves operating under an IPv6 address. However, in this case, it cannot be ensured that this bullet address is only forwarded by the immediate victim to those nodes that also operate the IPv6 protocol. We argue that this circumstance can mean that forwardings from possible IPv4 neighbours of the victim to the monitor nodes only happen with reduced probability, which could possibly falsify the results of the connection inference. IPv4 addresses are a safe choice here, as it can be assumed that every node regards them as routable.

Another helpful fact is that the port portion of a Bitcoin address does not affect forwarding. An address with the default Bitcoin port 8333 is forwarded just like an address with any other port. This circumstance allows us to use each IP address as a bullet many times, simply by using different ports. Together with the fact that there are over 65000 usable port numbers per IP address, this means that the number of possible bullets per address range increases dramatically.

Since the main goal of the measurement is to track individual bullets on their way through the Bitcoin network, it is important that the bullets remain individual and uniquely identifiable. This means in particular that the same bullet cannot be used for different victims. Otherwise, it would not be possible to determine exactly which victim the bullet was injected into if it reappears at a certain point in the network.

It is also important to ensure that addresses used as bullets are not sent through the network independently of the experiment. From this we derive the restriction that the addresses must not belong to real and advertised nodes. To this extent, addresses must also not have been used as bullets recently. This holds true even though it is possible to identify these previous injections of the same bullet or bullet addresses sent independently of the experiment by the timestamp used in the corresponding ADDR-messages address entry. As described in Section 2.3.3, this is because nodes store already known addresses and are less likely to forward them to their neighbours if they know that they already know the address.

All in all, the requirements for the bullet addresses are as follows:

1. must be IPv4 addresses
2. must be unique within the set of all used bullets, no bullet reuse
3. must not be an address previously known to any Bitcoin node

Since only a tiny fraction of all IPv4 addresses are running an actual Bitcoin node, it is possible to use random IP addresses as bullets to meet these restrictions. The probability that a given bullet address is then already known as a real address within the network would then be negligible. However, distributing random addresses makes it more likely that uninvolved IP endpoints are exposed to connection attempts by Bitcoin clients after the experiment. Because of this, we decide to use addresses under our control as bullets for this thesis instead.

We use an IPv4 range provided by the university faculty as it fulfils these requirements. The range encompasses 64 IP addresses. For each individual of these addresses we use the port range 8333 to 65535, which results in $64 \times 57203 = 3,660,992$ possible IP-port-pairs to be used as bullets for our measurement.

The number of injections possible per victim node is therefore limited by this maximum available total number of bullets. In this regard, measuring more victims simultaneously reduces the number of bullets per victim. Also, reusing addresses within a 24 hour timeframe may distort how they are relayed by peers under certain circumstances, as described in Section 2.3. Therefore, IP addresses should not be used as bullets more than once a day, even across separate experiment runs. Since we expect initial connections up to 5000 victim nodes, some of which may break off prematurely, we can assume about 800 available bullets per victim in the experiment.

For injection, the service flags must be added to each address. Since these are not relevant for the experiment, we can specify any values here. For the forwarding of addresses, as described in Section 2.3.3, a minimum requirement is the presence of the flags *NODE_NETWORK* or *NODE_NETWORK_LIMITED*. For moral reasons, we make a minimal selection of flags here. We hope that this then leads to nodes remembering the addresses we advertise as less interesting, keeping nodes from trying to connect to them. Since there is no real Bitcoin node behind the addresses we use, this keeps the number of failing connection attempts that occur as a direct consequence of our experiments as small as possible.

In addition, a timestamp must be added to each address. As described in Section 2.3, deliberately setting this field to a spoofed value can reduce the time for which the address is forwarded through the network as part of the gossip protocol. From a technical point of view, this allows the flooding time for individual bullets to be easily limited. To reduce the load on the network, we therefore set this timestamp for each bullet to a value that is a fixed number of seconds in the past compared to the respective bullet injection time. We choose this timestamp such that addresses are forwarded for 200 seconds. This idea of controlling the desired relay time by utilising this time stamp has also been addressed in related work. Biryukov et al. kept the time as small as possible to reduce the influence of false positives [BKP14]. in contrast, Grundmann et al. describe that the attacker they observed increased this time up to 19 minutes [GBH21].

We also note that service flags and the timestamp do not provide a way to create more individual bullets for fewer IP addresses by variation. So it is not practically possible to use the same $(host, port)$-tuple of an IP address in multiple bullets by changing the other fields. This limitation exists, although it would be possible for the estimation to distinguish these bullets from each other. The reason for this is that, as described in Section 2.3, a victim node keeps track of which of its neighbours it has already forwarded IP addresses to. Reusing an IP address in multiple bullets would therefore prevent the bullet from being forwarded to monitor nodes multiple times.

While we refrain from doing so, we would like to point out that it is possible to use real addresses used in the network as bullets. These bullets can become uniquely identifiable to the attacker by adjusting the time and service field. The forwarding behaviour would then possibly be distorted to some extent by the fact that the measured nodes may have already seen the addresses and forwarded them before. We expect that this would lead to reduced accuracy of the results. Arguably, an attacker can decide to take advantage of this opportunity to act less conspicuously by distributing plausible and real addresses. A reduced accuracy of the results could be accepted for this purpose.

3.4.3 Scaling-Up and Workflow Planning

The larger the number of nodes to be measured, the more important the planning of the workflow and its scaling; More connections have to be established and maintained simultaneously, the computational overhead increases with the number of bullets and victims, and the total experiment runtime increases with the number of total injections. In this subsection, we discuss design decisions that we make with respect to the scaling of the experiment. However, we limit ourselves to the conceptual design and not to technical optimizations of the code.

Workflow: We propose that scaling the experiment workflow to the full network can be done according to two different approaches; we call them *broad* and *narrow*.

In the narrow approach, the conductor connects to a small selection of nodes and injects messages only into individual victims. This measurement is resource efficient, but only those nodes can be identified as neighbours of victim nodes that lie within the set of nodes selected by the conductor. In iterative further steps, the set of connected nodes can be further adjusted after initial estimates to further search for likely neighbours of victims. As a result, the experiment setup would become more and more narrowed down to probable neighbors while multiple rounds of injections are conducted.

In the broad approach, the conductor connects to the entire connectable Bitcoin network at once. Thus, the actual neighbours of the victim nodes are certainly within the selected client set. Due to this circumstance, a single round of injections would be sufficient to find all neighbors. The experiment time is further reduced in comparison to the narrow approach in that it is possible to measure all connectable nodes at the same time. A prerequisite for this approach, however, is measurement hardware sufficiently powerful to hold all necessary connections to all nodes simultaneously.

For our main measurement we choose the broad approach, i.e. we measure all relevant Bitcoin nodes at the same time.

Multiple Injecting Monitors: A constraining factor is the throttling of accepted incoming addresses to an average of 0.1 per second, as explained in Section 2.3. For the experiment, this means that measurements with as many bullet addresses as possible can take a considerable amount of time. However, it is desirable to have as short an experimental runtime as feasible. This keeps the time frame for a possible network disturbance and detectability small. It also increases the credibility and usefulness of the results, since fewer changing connections can be expected in the network in a shorter time.

A practical way to increase the injection rate is to inject with multiple connected monitor nodes at the same time. Since the throttling of the injection rate applies per connection, the actual total injection rate can be increased many times over by using multiple simultaneous connections. This means that the experiment runtime can be significantly reduced in this way, while having no impact on the estimation. It should be noted that the number of connected monitors is limited by the network connection of the conductor and the associated number of practically sustainable TCP connections. At the same time, the number of simultaneous incoming connections accepted by the victim and client nodes is, of course, also a limiting factor.

While it is possible to configure the conductor for the largest possible number of monitors and tolerate a high number of disconnections, this approach has major disadvantages.

Initial large connection numbers demand a lot from the network connection of conductor, victim and client. In addition, the conspicuousness of the attempt increases, and the attacker risks no longer operating undetected. We therefore set the number of monitor nodes to the value of 5. In previous attempts, we empirically determined this to be a good compromise. Of these five monitor nodes, all take the role of injector and receiver simultaneously during the experiment to speed up the experiment as much as possible. This means that all monitors inject one after another, instead of only one sending messages and the others exclusively waiting for the receipt of these messages. This approach allows for a greater number of injections by the total number of monitors while maintaining the same experiment runtime. In addition, the increased total number of listening monitors also increases the number of bullet addresses received back. Thus, the accuracy of both connection inference and degree estimation can be increased, as we show in Section 4.1. To combine the data from the monitors after the experiment, it is only necessary to combine the sets of receptions and injections each across all monitors. Further adjustments to the estimation are not necessary.

However, increasing the number of connected monitors per node is likely to also increase the rate of disconnections per victim node. This means that the number of connected monitors for a node decreases over the duration of the experiment. To prevent this variation from being a disturbing influence on the estimation, the connection times for all monitors are measured. A duration-weighted average of the number of monitors can then be taken as the value for the estimator.

However, when increasing the injection speed to as large a value as possible, care must be taken not to overload the network. We learned in Section 2.3 that during the artificially added trickling wait time, incoming addresses are added to a queue before being sent in bulk. If this queue reaches a size of more than 10, the messages are still sent but not forwarded by the next node. Since these messages can then no longer be used for connection inference, this circumstance should be avoided as far as possible. Furthermore, the average bulk size should even be smaller than 3, as this value would suggest that the total message volume is at the throttling speed[2]. We validate the extent to which our experiment affects these congestion metrics in Section 4.1.2. In any case, we limit the flooding of bullet addresses to limit overloading by decreasing the timestamp by 400 seconds, i.e., to 200 seconds of flooding.

Bulk Injections: It is possible to inject multiple addresses with a single ADDR-message. There can be a bulk of up to 10 addresses per message without changing the way the individual addresses are forwarded. However, from the point of view of the throttling taking place through the victim node, it is necessary to increase the waiting times between the ADDR-messages to a corresponding extent. This is because the rate-limiting takes into account the number of addresses rather than the number of

[2]Sending a bulk message containing 3 addresses would, with the average trickling delay of 30 seconds, correspond to one address every 10 seconds. This is the maximum average relay speed before throttling takes place.

individual ADDR-messages these addresses come in. Due to this, no time gain can be achieved for the test execution. However, this increase in bulk size can be useful from a technical point of view to reduce computational overhead. This is because less frequent switching between the active injectors is then required. For the experiment, we therefore always inject 10 messages in bundles. In doing so, we keep the experiment similar to Biryukov et al. and Grundmann et al., who reported also using a bulk size of 10 for their respective experiments [BKP14, GBH21]. Additionally, we increase the waiting times between injections per victim node tenfold accordingly.

Final Workflow

All in all, the overall workflow looks like the following:

1. The conductor creates the monitors
2. All monitors connect to all selected victim nodes that are to be measured.
3. To let rate-limiting buckets build up, the monitors wait for a time depending on the block size before starting to inject.
4. Taking into account the minimum waiting times due to rate-limiting, the full addresses are injected into each victim node to which at least two monitors are connected. A maximum injection rate of 10 seconds per address per monitor is maintained. For each victim node, all monitors connected to this node are used for injection. Addresses are sent in blocks of the designated block size and their timestamp is artificially decreased relative to the time of injection.
5. To give the network time to relay addresses back to the monitors, the monitors wait for a while after the last injection.
6. The connections are terminated and the measured data is collected.
7. The measured data is passed to the estimator, which makes an estimation using two estimation modes: time-based and relay-based.
8. Graph-theoretic metrics are calculated for the estimated network graph

3.4.4 Experiment Conduction

As an active probe of the open network, we carry out the final measurement on the Bitcoin mainnet using the prototype framework that we developed. The experiment is conducted using a single a *Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-96-generic x86_64)* system. It is hosted on a dedicated measurement server with 32 AMD EPYC 7313P 16-Core CPUs @ 3GHz, 255GiB RAM and a 10Gb/25Gb RDMA Ethernet Controller. At the time of measurement, the measured actual network speed is about 2483.77 Mbit/s in download and 899.25 Mbit/s in upload.

Instead of focusing on a single test node, this time the measurement is directed against all IPv4 and IPv6 nodes of the Bitcoin network. The filter proxy that was used in the additional experiments to limit them locally is therefore also not used.

A total of 6795 IP addresses associated with Bitcoin nodes were obtained from Bitnodes.io immediately before the experiment. This set forms the set of victims. Up to 5 monitor nodes connect to each of these victims, if possible. All of them perform injections simultaneously. Up to 800 bullet addresses are injected into each victim. Each injection consists of an ADDR-message containing 10 of the bullet addresses at the same time.

For reference, the entire codebase of the framework used for the measurement, and the subsequent estimations can be found at `https://gitlab.sba-research.org/johanna/coninf_klonowski`.

CHAPTER 4

Evaluation

In this chapter, we validate the estimator performance, present the results of the experiment and discuss our findings.

The validation in Section 4.1 measures the precision and recall of the estimator. The two estimation modes are compared to each other, and the influence of experimental parameters, such as the number of monitor nodes and injections, is measured and discussed. In particular, the credibility of the measurement results is also justified through this validation. A brief overview of the raw results data is given in Section 4.2. The topology of the Bitcoin network determined by the measurement is presented in Section 4.3. Graph-theoretical metrics are also applied here, and the implications of their results are discussed. The discussion in Section 4.4 summarises the most important results of this thesis and places it in the context of this field of science.

4.1 Validation

To validate the results, it is necessary to make a comparison with known values. We therefore perform quantitative validation by testing the framework in a testbed. Specifically, this means that we measure a Bitcoin node under our control and use the estimator to estimate its degree and connections. Since the ground-truth connections are known for this node, it is thus possible to evaluate the accuracy of the estimation. This allows us to specify precision and recall of the framework, and compare them for different experiment parameters. In this way, we measure and compare the performance of the estimator in terms of degree estimation and connection inference for relay-based estimation and time-based estimation. It also allows us to justify the training data of the estimator by comparing which generated training data gives the best results. The estimator performance evaluation is furthermore conducted for different node degrees to determine how well the method covers all possibilities of nodes in the network. Finally, the performance of the method presented in this thesis is compared with methods from related work.

4.1.1 Setup

Hardware: The tests are run on a virtual machine with 2 virtual Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz, 4GiB RAM and a 10Gbit/s Ethernet interface. At the time of measurement, the measured actual network speed is about 1430Mbit/s in download and 1080Mbit/s in upload.

Software: The tests are performed on a *Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-109-generic x86_64)* system. The measured testbed node runs the Bitcoin client *Bitcoin Core version 22.0*.

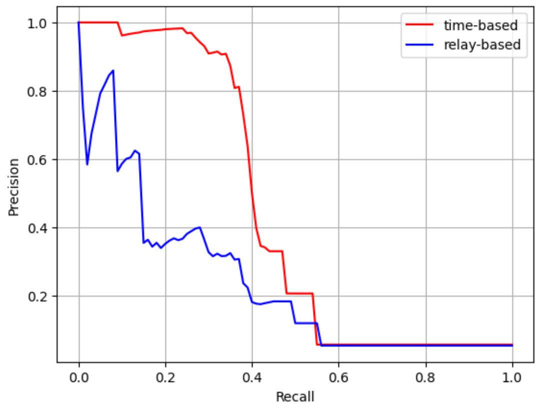
Parameters: For the sake of comparability, we measure the test node with parameters similar to those that were used for the main experiment: 1000 bullet addresses are injected into the victim node with a delay of 10 seconds each. The victim node has a stable node degree of 120, and 5 monitors are connected to the victim and 2000 clients each. All monitors were used for injection. Thus, the total run time of the measurements is about 33 minutes. The training set for the estimator is generated based on the following parameters: A network is built with 14,000 nodes in total, out of which 4000 are connected to 5 monitors each. 1000 injections are simulated with a 10-seconds delay between each other into 5 victims with node degrees 0, 20, 40, 60 and 80.

These parameters are applied to all the validation runs if not stated otherwise. For validations that determine the results as a dependency of one of these parameters, only this parameter under consideration is changed and all others are the same as above.

Datasets: Usually, it is desirable for validation to collect as much data as possible in order to be able to robustly average the results against variation. However, in this thesis, collecting data for validation means interfering with the operation of the Bitcoin network. This is because the test of connection inference requires injected addresses to reach the main network beyond our test node. For this reason, we decided to perform the validation with as few data sets as possible. As a result, the exact results may differ from the true average. However, we see that trends can nevertheless be identified in the evaluation. We therefore argue that the experiments presented below can still be used to validate the choice of parameters.

4.1.2 Validation Results

For validation, we determined the influence of the following experimental parameters on estimation performance: monitor count, victim node degree, training dataset, setup connectivity, injection amount, and address timestamp. We also measure the overall performance of the estimator in both modes studied as PR and ROC graphs, and validate the influence of the main measurement in terms of possible network overload. Validated are both the degree estimation and the connection inference. The results of this are presented in this section and briefly interpreted where appropriate.



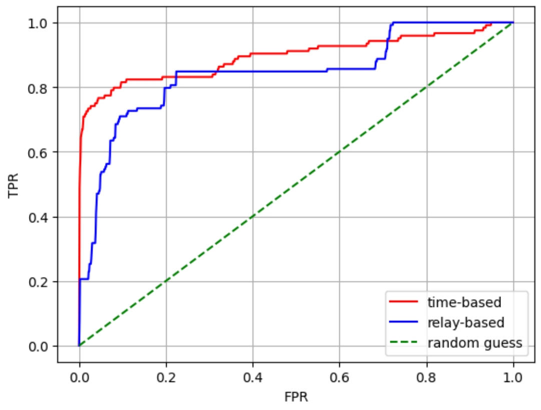


Figure 4.1: Precision-recall graph for the time-based estimator and the relay-based estimator

Figure 4.2: Receiver Operating Characteristic graph for the time-based estimator and the relay-based estimator

Connection Inference Precision and Recall per Threshold

As described in Section 3.2, the estimator produces probability values for all possible connections during the connection inference. These are then turned into a binary classification based on estimated probabilities, to which a threshold value is applied. By setting this threshold value, a tradeoff between good values for precision or recall can be achieved. In this section, we evaluate the performance of the estimator by building the precision-recall graph and the receiver operating characteristic graph as presented in Section 2.2. The curve of the PR graph for both estimation modes is shown in Figure 4.1. We see that the time-based mode consistently outperforms the relay-based mode. The precision value of the relay-based estimator decreases even for low recall values from up to 20 % to about 40 %. Thereafter, it decreases less rapidly to below 10 % at 60 % recall. The time-based estimator, on the other hand, shows more than 90 % precision at up to 35 % recall. Only then does its precision drop steeply to the level of the relay-based estimator of below 10 % at 60 % recall.

The ROC graph also shows how the time-based estimator generally outperforms the relay-based one. In doing so, the time-based estimator shows a steeper increase than the relay-based estimator. While the former reaches a true positive rate of 80 % at a false positive rate of 10 %, the latter has this value only at 20 %. For higher FPR values, the TPR of the time-based estimation mode moves steadily towards 100 %. The TPR of the relay-based estimator, on the other hand, has a clear jump at 70 %.

For the PR graph and the ROC, both the AUC value and the Gini coefficient for both estimation modes can be obtained from Table 4.1.

We can also compare the PR graph of our estimator with that of Neudecker et al. [NAH16]. For the latter, it was reported that it maintains a precision of 40 % for a recall of 40 %. Thus, we find that the researchers' timing analysis shows better performance than the

	PR		ROC	
Estimator Mode	AUC	GINI	AUC	GINI
time-based	0.44147	-0.11706	0.89423	0.78847
relay-based	0.23448	-0.53104	0.84198	0.68396

Table 4.1: Estimator metrics for the two estimation modes

		Average relative error		
Real connections	Injections	time-based	relay-based	Biryukov et al.
10	500	12.88889	10.88889	3.2
30	1000	-4.62811	-5.98443	10
70	1000	3.49365	3.56328	3.4
100	2000	-6.88543	-7.83781	3.0

Table 4.2: Relative degree estimation error depending on node degrees, compared with the estimator by Biryukov et al. [BKP14]. A negative error value means that the node degree was estimated lower than it actually is.

relay-based estimator of this thesis. However, the performance of our timing analysis for this sensitivity is equal with also a precision of about 40 % at 40 % recall. For the range of recall values from 20 % to 40 % the precision of our time-based estimator drops off slightly less steeply.

Degree Estimation Relative Error

To measure the performance of the degree estimation, we set up and measured the testbed node with different node degrees. For better comparability, we chose the same parameters as Biryukov et al. [BKP14] in their validation. That is, we measured the node with degrees 10, 30, 70 and 100. Between 500 and 2000 injections were used in accordance to the reference evaluation the researchers describe. Subsequently, the node degree was estimated on the basis of these measurement results in both estimation modes.

The results of this can be seen in Table 4.2. As expected, the relative error of the relay-based estimator is similar to that from the reference paper. For both, the relative error ranges between 3 % and 10 %. This is also in line with the validation results of Grundmann et al., who report an average relative error of 4.1 % for a comparable degree estimation experiment. We assume that the similarity is due to the fact that the same equation was used for the estimations across all papers.

We note, however, that the researchers' estimator consistently overestimates the node degree, while in our estimate the sign of the error fluctuates. In addition, we see a large deviation in our estimate for node degree 10, possibly caused by stabilization problems

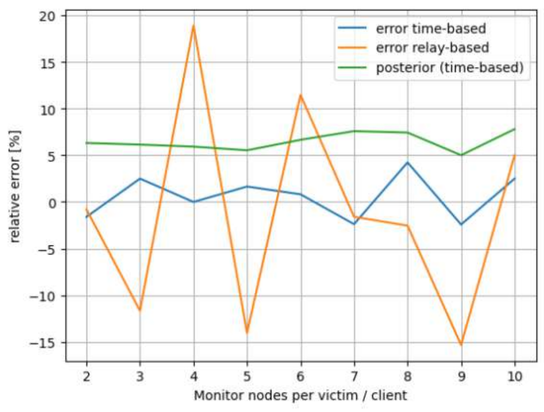


Figure 4.3: Degree Estimation performance for different numbers of monitors

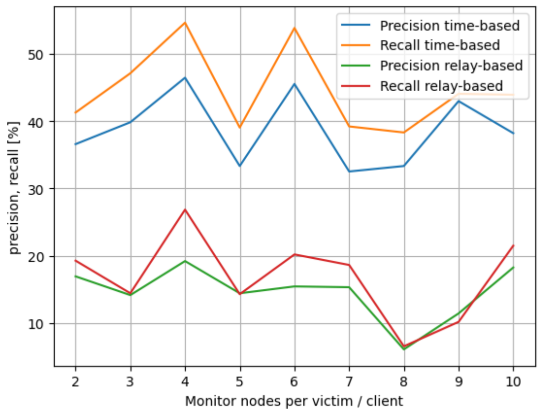


Figure 4.4: Connection Inference performance for different numbers of monitors

of the actual node degree in the testbed. For all node degrees except 10, the absolute value of the relative error in the time-based estimation is lower than in the relay-based estimation, but only to a small extent.

Overall, we can verify that the estimator developed in this thesis produces estimates with similar accuracy as the reference estimator from related literature.

Monitor Amount

In the context of the experiment, there are two factors in favor of connecting as many monitor nodes as possible to the victims and clients: 1) more injectors at the same time allow for parallel injection and thus speed up the experiment 2) more receivers at the same time increase the chance that an address is forwarded back to the attacker. Especially 2) hints towards an improvement of the results by increasing the total relay observation count, especially in case of low numbers of injections. We measured the effect of changing the number of monitors beyond these two points on the quality of the estimation using the testbed node. For this purpose, 1000 bullets were injected into the node with different numbers of monitors per victim and client. The results of this can be seen in Figure 4.3 and Figure 4.4.

We find that the posterior value returned by the time-based estimator for the degree estimation remains constant. For the error of the degree estimation, no clear trend can be observed over the monitor number. Both the time-based and the relay-based estimation modes produce fluctuating results that sometimes overestimate and sometimes underestimate the node degree. The fluctuation of the error rate is significantly smaller with the time-based estimator.

The results are similar for connection inference. Here, both the time-based and the relay-based estimation produce varying results across different monitor amounts. Again,

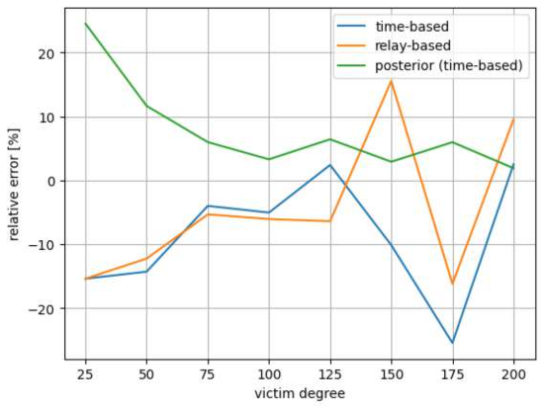


Figure 4.5: Degree Estimation performance for different victim degrees

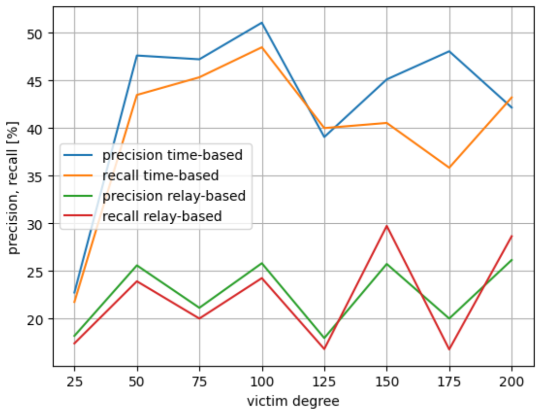


Figure 4.6: Connection Inference performance for different victim degrees

the deviation does not show a clear trend in relation to the number of monitors. We conclude that the number of monitor nodes has no clear influence on the results of the experiment. One possible explanation for this could be that the other experimental parameters can guarantee stable results even for low numbers of monitors. Specifically, we argue that this would mean that 1000 injections into a victim with node degree of 120 can lead to enough returning bullets for estimation, even when using a smaller number of receiving monitors.

Victim Degree

As shown by Grundmann et al. [GBH21], the nodes in the Bitcoin network can assume a wide range of node degrees. We therefore validate the performance of the estimator for nodes with different node degrees. This allows us to determine the extent to which the estimator can produce valid results across the entire heterogeneous network. The results of the measurement series are shown in Figure 4.5 and Figure 4.6. We see that the posterior value for the degree estimation is slightly higher at lower node degrees, and then becomes constant above approximately 100. The relative error of the degree estimation develops similarly for the relay-based estimation as for the time-based estimation. For low node degrees, both overestimate the degree. The error decreases until around 125. From then on, the relative error increases strongly for increasing node degrees, but varies by sign. Both estimation modes are affected. We suspect that this may be because variations of single recurring bullets carry more weight at large node degrees due to the lower absolute expected number of returns. In this way, small statistical deviations have a greater influence on the degree estimation at large node degrees.

In terms of connection inference, there is no clear tendency for precision or recall in either mode. Only the time-based estimation mode shows an unusually low performance at node degree 25. Otherwise, all values remain rather constant with small fluctuations.

Training Set Parameters

To run the estimator in time-based mode, a model must first be created. This can then be used to interpret the measured data and perform the estimation. In this sense, training is a necessary first phase for this estimation mode before the actual estimation. This is not necessary for the relay-based estimator.

As described in Rosenblattl's study, a training data set is required for this training phase. This data set contains forwarding data in a format similar to the measurement data format. More specifically, the training data sets are generated in the context of this thesis by a simulator provided by Rosenblattl's work. This first creates a random graph as similar as possible to Bitcoin and then simulates the forwarding of ADDR-messages through this graph using the forwarding rules described in Section 2.3.3. The forwarding delays observed during this simulation, together with the known ground truth regarding the connections in the simulated network, then form the training basis for the statistical estimator [Ros].

While the random node degree distribution of the simulated graph is designed to follow that determined by Grundmann et al. [GBH21], the remaining parameters of the simulation can be freely chosen to fit the resulting training data set for better estimation results. The choice of parameters here include:

- total number of clients
- number of clients connected to monitor nodes
- number of monitor nodes
- number and degrees of the measured victims
- number of injections per victim

We choose different values for these parameters to create a series of training data sets. We then use these to estimate the same set of measured reference data. In this way, we determine the influence of the different aspects of the training data set on the final estimation. The results of this series of experiments are listed in Table 4.3.

We find that all parameters have only a negligible impact on the quality of the estimation results. In particular, the performance with a large number of victims with a wide variation of degrees is not different from the performance of small sets with only one victim. Under the training dataset with the number of monitors increased to 20, connection inference is slightly better, but degree estimation is significantly worse compared to the other datasets. When the number of monitors is increased even further to 1000, this effect is reversed. Noticeably, even the number of simulated injections for the model to train on barely changed the estimation outcome. However, increasing the total size of the simulated network had a noticeable effect on the quality of connection inference, while maintaining good degree estimation results.

Dataset Name	A	B	C	D	E	F
clients	1400	1400	1400	1400	1400	1400
connected clients	1200	1200	1200	1200	1200	1200
monitor nodes	5	5	5	5	5	5
victim degrees	120	120	10	[120 for 10 times]	[10, 20,. . . , 200]	[120 for 20 times]
injections	1000	100	1000	100	100	100
relative error degree estimation	0.00787	0.00787	0.00787	0.00787	0.00787	0.00787
degree estimation posterior	0.07233	0.07516	0.07506	0.07375	0.07655	0.07225
precision (flexible threshold)	0.46094	0.47656	0.42188	0.46875	0.46094	0.48438
recall (flexible threshold)	0.50427	0.52137	0.46154	0.51282	0.50427	0.52991

Dataset Name	G	H	I	J	K	L	M
clients	1400	1400	1400	1400	14000	500	14000
connected clients	1200	1200	1200	1200	12000	250	12000
monitor nodes	5	2	20	1000	5	5	5
victim degrees	120	120	120	120	120	120	[10, 20,. . . , 200]
injections	5000	1000	1000	1000	1000	1000	100
relative error degree estimation	0.00787	-0.01575	0.12598	0.01575	0.00787	-0.00787	0.00787
degree estimation posterior	0.07091	0.07033	0.05957	0.14733	0.07215	0.08472	0.06903
precision (flexible threshold)	0.47656	0.47200	0.47552	0.18605	0.51562	0.46032	0.52344
recall (flexible threshold)	0.52137	0.50427	0.58120	0.20513	0.56410	0.49573	0.57265

Table 4.3: Estimator results with different training sets

Dataset M combines these aspects in an attempt to add up their respective benefits. However, we notice that the estimation performance under this training set is slightly, but not considerably higher than using any other training set. We therefore conclude that the choice of training data has little effect on the quality of the estimation results.

Connectivity Rate

As described in Section 3.1, the estimation of both node degrees and connections works by retrieving bullets from Bitcoin nodes. Especially in connection inference, it is important to maintain a large number of connections simultaneously. This includes the links from monitors to the respective victim node, but also all connections between the monitors and all neighbours of the victim node. If individual such connections do not exist, the bullets cannot be routed from the respective peer to the monitors. As a direct consequence of this, the peer cannot be recognized as a neighbour of the victim node, since no bullets are recognized as recurring through this client.

To define a measure of the extent to which this condition is met, we define the *connectivity rate*. This denotes the proportion of all actual connections between monitors and neighbours of the victim node out of all possible such connections. Since it is expected that not all such connections can be maintained in practice for the full duration of the experiment, we expect the value of this parameter to be less than 1 during the measurements.

For validation experiments, it is possible to achieve a high connectivity rate by suitable selection of the connections of the monitors and by controlling the desired neighbours of a controlled victim node. However, this is not feasible for the main experiment, as it has been shown in practice that not all desired connections of monitors are always holdable. Thus, there is not always a desirable degree of connectivity to the neighbours of all victim nodes.

In this validation, we therefore investigate how different connectivity rates affect the estimation results. To this end, we perform several measurements in a control environment with a high connectivity rate. Subsequently, we artificially simulate a worse such rate by trimming the measurement data, and average the precision and recall values for both the time-based estimator and the relay-based estimation mode.

The results of this series of measurements can be seen in Figure 4.7. We notice that the performance of both estimation modes increases strictly with increasing connectivity. The growth is strongest up to the point of a connectivity of about 30 %. After that, the influence of the connectivity rate on the estimator performance decreases.

We can conclude that the connectedness of the monitors to the network should in principle be as high as possible. At the same time, however, we argue that our measurement setup can still produce credible results even at connectivity of as low as 30 %.

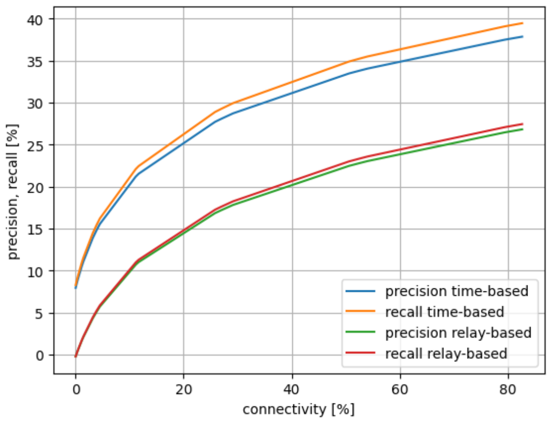


Figure 4.7: Connection inference performance per connectivity rate

Injection Amount

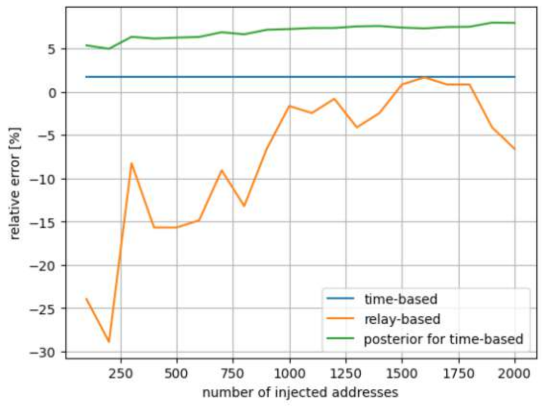
The stochastic nature of the message forwarding process in the ADDR-message relay leads to an inherent imprecision of the estimation. Due to random decisions in the selection of forwarding partners and the corresponding time delay, it is possible that a node behaves differently for individual messages than an observer would expect: With some probability, for example, monitor nodes may be underrepresented in the choice of forwarding partners during the experiment, or long forwarding chains with unusually low total delay may be observed.

This circumstance suggests that a large number of injections might lead to better estimation results. The intuition is that some of these misleading random events then lose weight in the total number of events measured. A larger data set could therefore lead to better estimations.

We investigate whether this assumption is true by validating the number of injections. For this purpose, a measurement with a high number of injections is conducted first. Then, a dataset with an arbitrarily small number of injections is simulated by subsequent trimming of the data. The performance measures of the estimator are calculated on those trimmed datasets.

The results of this evaluation are shown in Figure 4.8 and Figure 4.9. Regarding the degree estimation, the time-based estimation mode produces constant results independent of the number of injections. The relay-based mode, on the other hand, produces constantly improving results with increasing number of injections up to a point of around 1500 injections.

For connection inference, both precision and recall decrease slightly throughout the relay-based mode up until around 1000 injections. After this, both metrics start to rise steeply. In the time-based mode, performance increases sharply with increasing number



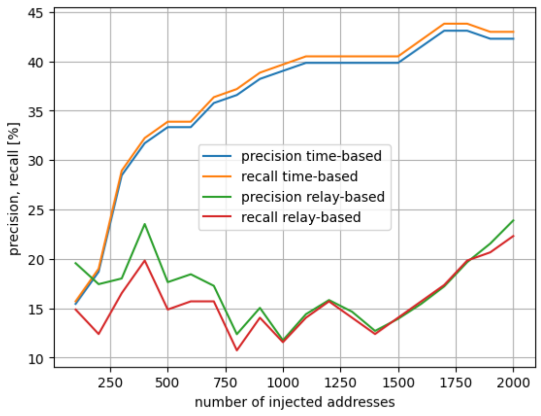


Figure 4.8: Degree estimation performance per number of injected addresses

Figure 4.9: Connection inference performance per number of injected addresses

of injections up to the point of about 1000 injections, after which it continues to grow only marginally.

From this we can conclude that the total number of injections should be as high as possible, but at least 800 for decent estimation results. Accordingly, we can assume that the number of injections we used in the main experiment, although not optimal, is sufficient to provide valid results. However, a larger number of injections, as it would be desirable according to the results of this validation, was not feasible due to availability limitations as discussed in Section 3.4.2.

Trickling and Time to Live

In Section 2.3 we describe that addresses are forwarded together with an associated timestamp. This can be between 10 minutes in the past and up to 10 minutes in the future. The purpose of the timestamp is to abort the flooding of addresses after a predefined time. To the attacker, this mechanism allows controlling how long messages are forwarded through the network, or their *time to live (TTL)*. An address that is given the current timestamp at the time of injection is forwarded for 10 minutes. If the injection timestamp is artificially reduced, the flooding for this message ends sooner. Alternatively, the time can also be increased, since time stamps that lie slightly in the future are also tolerated. In this case, the message is forwarded through the network for a longer duration.

Since there is an average time delay of 30 seconds with each forwarding of the message as part of the trickling process, messages that return a long time after their injection have most likely taken a large number of steps between the injector and the receiver. For connection inference, however, only messages with a small number of relays are relevant, since it is these that come from the neighbours of the victim node. To reduce the number of irrelevant messages that the monitors receive from the network, an attacker

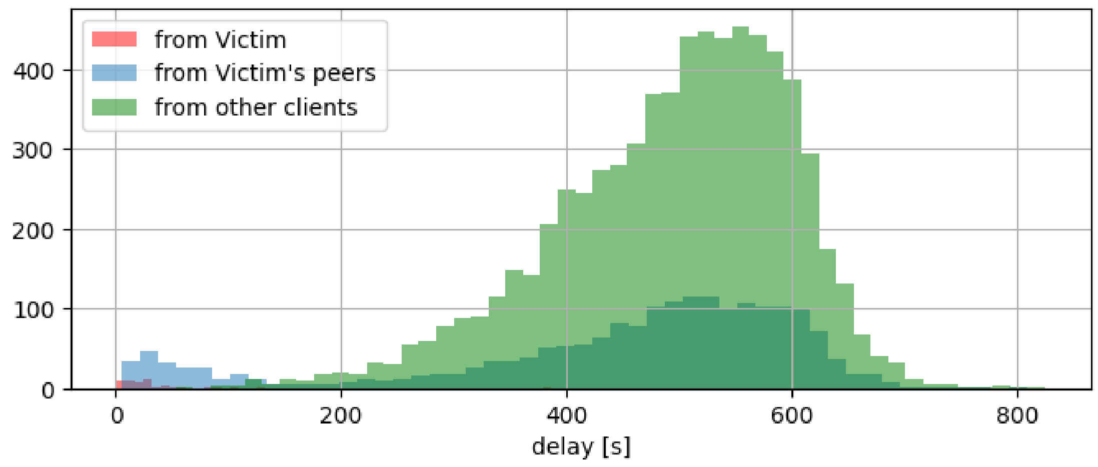


Figure 4.10: Histogram of the delay times of bullet receptions from different sources

can therefore choose the timestamp of the addresses such that they are forwarded only briefly. Alternatively, it is of course also possible to filter bullets that come back with a long delay after the fact in preparation for estimation.

In addition, the number of multiplications of an injected address increases quadratically network-wide: Once received, an address is forwarded twice. This squaring takes place up to a saturation point. After all, addresses are only forwarded less often when many network subscribers already know them. This increase in message volume suggests that reducing the lifetime of an address for flooding may reduce the network load.

We can observe both of these effects in a histogram of the measured delay times. Such a histogram is given in Figure 4.10. We notice that the set of messages that were forwarded often is significantly larger due to the multiplicative factor. These messages can be recognized well by the high measured delay. This results in a large accumulation of messages in the right part of the histogram. Addresses that come from a client that is not adjacent to a victim node are almost exclusively found in this right part of the histogram.

We note, however, that there is also a large proportion of packets among the addresses with a large delay that come from neighbours of the victim node. This contradicts the intuition that addresses coming back through neighbours of the victim should generally have a low delay. We assume, however, that this proportion of packets may also be packets that do not return directly through the victims' neighbour, but have reached said neighbour over a longer path. In other words, addresses arriving through neighbours of the victim can have the delay associated with minimal path length expected of a direct path, but can also show a delay time of an indirect path. As a result, the histogram appears roughly divided into two parts: A cluster on the left side represents predominantly packets coming back directly from victim nodes and their neighbours. The much larger cluster

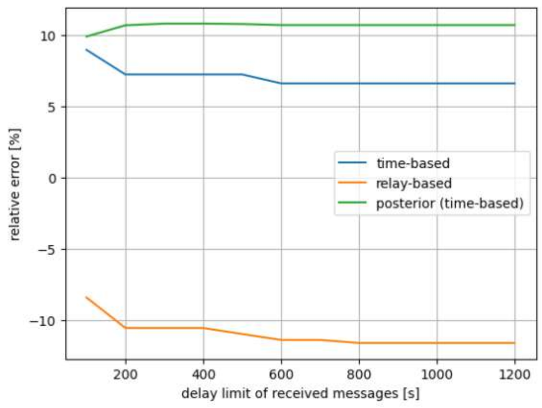


Figure 4.11: Degree estimation metrics for receptions limited to being lower than specific delay times

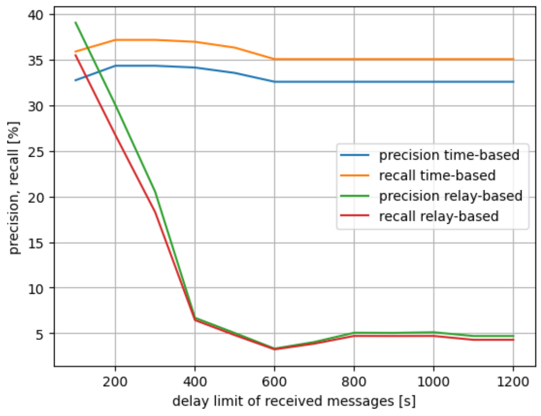


Figure 4.12: Connection inference metrics for receptions limited to being lower than specific delay times

on the right side shows addresses returning over paths of long length. The separation between these two areas appears to be around 200 seconds.

We can assume that the total delay of individual bullets across n forwarders follows an Erlang distribution. This is because the total delay is the sum of independently exponentially distributed partial delays. We see in the histogram that the curve roughly follows this expected distribution. On a site note, we can find that the observed maximum is at about 550 seconds delay. Since it should be $\frac{n-1}{\lambda}$ for an Erlang function $Erl(\lambda, n)$, we calculate the average number of bullets relays within the network to be $n = 550 \times \lambda + 1 = 550 \times \frac{1}{30} + 1 = 19.34$. This suggests that ADDR-messages are forwarded on average about 19 times on their way through the network.

The bullets used in Figure 4.10 all have an unadjusted timestamp. Thus, the expected flooding time is 10 minutes or 600 seconds. We see that the number of bullets received after this time drops sharply, but not immediately to 0. This is because the consideration of the timestamp takes place when the bullet is received, and thus before the trickling duration is applied. In this way, even after more than 10 minutes after injection, bullets can still be returned by network nodes.

A strategy suggested by the observed bi-partition of the histogram can therefore be to limit time delays in such a way that only messages with small delays arrive or are considered further. The goal of this strategy would be to limit the interfering influence of messages from the right part of the histogram by limiting the estimation to messages from the left part. In a sense, we filter for exactly those messages that we can assume are most likely to come from the victim and its neighbours.

Intuitively, it is to be expected that a certain influence on the degree estimation by this procedure is to be noticed. On the one hand, the bullets forwarded from the victim to the monitors without further hops generally show a low overall delay. On the other hand, we expect that in the set of all bullets returning through the victim node there are also some that have followed a longer path. These would then not have been forwarded to monitors again directly after the injection into the victim, but would have taken a detour over multiple other peers before returning to the victim and finally to the monitors. In a sense, they would have taken a detour through the network before reaching the monitors through the victim. As we discussed in Section 3.2.2, these bullets can be effectively filtered out by introducing a maximum delay. This would correct the degree estimation error downwards.

The actual impact of this approach on the performance of degree estimation can be seen in Figure 4.11. We find that the true impact on the degree estimation is negligible. This is the case for both the relay-based and the time-based estimation. The reason for this may be that in practice hardly any bullets return to the monitors via long detours through the victim. In terms degree estimation performance, therefore, no filtering of these bullets is necessary. This observation is consistent with Biryukov et al., who point out that the TTL of the message should be kept artificially small for the degree estimation [BKP14].

For connection inference, the effect of the time to live of an address can be seen in Figure 4.12. We observe, that for the time-based estimation mode, both precision and recall remain constant across all possible values for the time to live of bullets. The relay-based estimation mode however is heavily influenced by the delay time: For low values, both the precision and recall show good results of up to 40 % at 100 s. With an increasing time to live of the bullets, both of these metrics drop significantly to less than 10 % at 400 s, where they remain constant for all higher values.

This fact is illustrated by the comparison of the corresponding Gini coefficients. The coefficient of the time-based estimator is constant at $0.6 - 0.7$ for all time-to-live values greater than 400 seconds. Between 100 and 400 seconds the value increases slightly to a maximum value of $0.75 - 0.80$. The relay-based estimator, on the other hand, has a constant value of about 0 for TTL values above 400 seconds. This circumstance shows that the relay-based estimator does not perform better than a random guess for unfiltered high TTL values. With lower maximum TTL, however, the Gini coefficient rises steeply to values of up to 0.75. Thus, the time-based estimator is an upper bound for the performance of the relay-based estimator depending on TTL.

In summary, the effect of the maximum forwarding time can therefore be regarded as significant for only the relay-based estimation mode. This estimation method benefits greatly from a low maximum time delay. Generally, this parameter does not significantly influence the time-based estimation method. Filtering the receptions for a maximum delay of 100 s represents a good parameter choice that greatly benefits the relay-based estimation without being at the expense of the time-based estimation.

Network Overload

It is easy to see that it is important for our experiment not to overload the network. As described in Section 2.3, all messages that a node is holding back to send over a connection are combined into a single ADDR-message during the trickling delay period. As the volume of ADDR-messages within the network increases, the number of messages a node accumulates in this way generally increases, and so does the average number of addresses per message. We therefore argue that, in addition to the total number of addresses per connection duration, the average bulk size of messages is a good measure of network utilisation. This metric is particularly important because packets with a bulk size of more than 10 are not forwarded and therefore distort the results of the estimation. That means that if messages are injected so fast that they accumulate to a bulk of more than ten addresses in the outbound queue, these bullets are ignored by all non-monitor network participants.

It is therefore in the interest of the attacker to inject slowly enough so that the average bulk size of incoming data packets from the network does not become too high. Also, the number of messages for which a bulk size more than 10 is detected at the monitors should be kept as low as possible. The average address rate should be below 0.1 per second per connection, as this value would suggest a maximum network load. With an average trickling delay of 30 seconds, this corresponds to a maximum average bulk size of 3.

In order to be able to determine the additional load on the network during the experiment, we compare the message volume before and during the execution of the experiment. Figure 4.13 shows the speed at which addresses are sent through the network via ADDR-messages per connection in a 24-hour time frame without our involvement. It can be seen that a large proportion of connections have a message rate between 0.002 and 0.015 while only few connections send more than 0.02 messages per second. The average is 0.0054 per second, which would amount to one message roughly every 185 seconds on any given connection. We observed the average bulk size to be 1.26147 over the course of this timespan. Only 0.00655 % of all received ADDR-messages had a bulk size of more than 10. This indicates that the total number of address forwardings during normal network operation is far from the technical limit.

By comparing this amount of traffic to the volume of messages sent during the experiment execution, we can then derive a consideration of how invasive the active measurement of the network in our main experiment is to be assessed. Here we compare the data obtained not from the smaller validation measurements, but from the full measurement of the entire network according to the parameters described in Section 3.4.4. In this case, the average bulk size increased to 2.22230. The number of ADDR-messages containing more than 10 addresses increased several fold to 1.65897 %. The average number of incoming address entries per second also increased to 0.01895. This corresponds to one address every 53 seconds on average.

We can conclude from this that the load on the network increased by a multiple during the experiment. Nevertheless, the load on the Bitcoin network remained below the technical limit at which throttling would occur. We can therefore assume that the results of the measurement were not distorted to any significant extent by network overload.

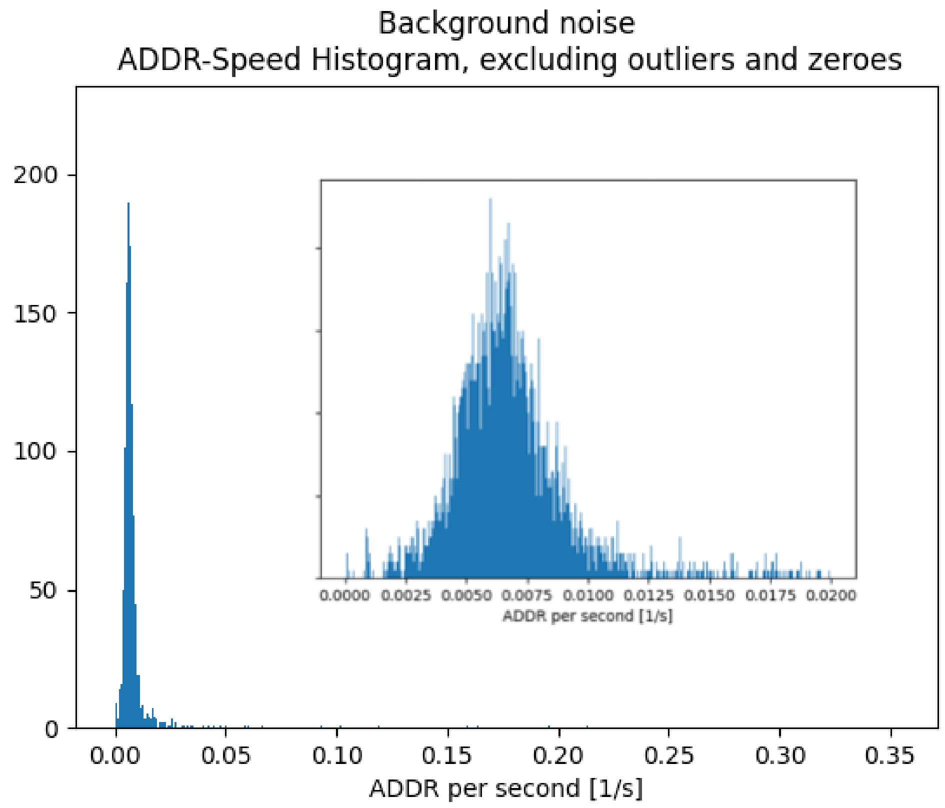


Figure 4.13: Histogram of the rates of incoming ADDR-messages per connection

The network load in terms of bandwidth, on the other hand, is difficult to measure. This is because the Bitcoin protocol is an overlay protocol on TCP. Therefore, multiple ADDR-messages can be combined into single TCP packets, reducing the impact of the TCP header. However, we can provide an upper bound estimate of the additional bandwidth required. For this, we assume that each address is sent individually in an ADDRv2-message (bulk size 1) in a separate TCP packet. We have measured that the complete data frame in such a case has a size of 92 *Byte*, although the payload part of the Bitcoin message is only 14 *B*. Thus, the amount of data traffic would increase from $0.4968\,\frac{B}{s}$ to $1.7434\,\frac{B}{s}$ per connection. A node with an average of 120 connections would therefore have an additional data traffic of only $149.592\,\frac{B}{s}$ per upload and download.

4.2 Raw Results

The measurement was performed on October 31, 2022. It lasted for a total of 90 minutes, from 16:42:13 to 18:11:44 UTC.

Before starting the measurement, 15537 of 33975 (= 45.73068 %) of the monitor connections were successfully established. Reached were 4615 of the 6795 individual nodes of the network that were to be measured. Out of these, 2985 nodes (43.92936 %) had connections to at least two monitors, and thus the minimum number to be actively measured. 2553 nodes had all five monitor connections, 89 only four, 100 were reached by three monitors, 243 by two and 1630 nodes had a connection to only a single monitor node. This connectivity percentage is roughly in accordance to the expected value. Grundmann et al. estimate, that around 53 % of accessible nodes have either no open connection slots left or are close to their respective maximum slot amount [GBH21].

At the end of the injections, the number of remaining active connections had decreased to 13608 (= 40.05298 %) to 4357 individual nodes. As a result, as many as 2547 of the original 2985 nodes still had the required monitor connection count for active measurement. The number of nodes that still had all five monitor connections upheld had dropped to 2015. Also, the proportion of nodes that still had only between four and two monitor connections had shifted to 277, 105, and 150 respectively. Nevertheless, 1810 nodes still held exactly one persisting connection to a monitor, which meant that receiving bullets through these nodes was still possible.

The total number of addresses received from the monitors during the experiment is $1,578,809$. Of these, $582,355$ (= 36.88572 %) are bullets that were previously injected.

Due to some connections breaking prematurely, not all planned injections were performable. Hence, only to 2288 of the victims 100 % of the bullets intended for them could be injected. For 2591 of the victims, more than 75 % of the injections were performed, and for 2609, at least half were performed. The number of nodes for which at least 10 % of the injections could take place is 2684.

Estimation: The data collected during the measurement process was subsequently passed on to the estimator to obtain an estimate of the measured Bitcoin subnetwork. After a single initial training phase of 50.5 seconds on simulated network data, the time-based estimator performed the estimation for a total of around $24.5\,h$ [1].

4.3 Graph-Theoretical Results

Due to the better performance, the time-based estimation mode was chosen as a basis for the graph analysis. To transform the resulting estimates into a graph, they were first converted into a binary classification. For this purpose, as discussed in Section 3.2, a flexible threshold was applied to the posterior values. Thus, for each node, exactly as many connections were positively estimated as given by the estimated node degree.

Subsequently, we have implemented the estimated connections in a *directed* graph. In this aspect, the model differs from the real Bitcoin graph, which would be undirected. The reason for this is that the labels for the mutually opposite edges (u, v) and (v, u)

[1] We suspect the significant time cost to be rooted in implementation issues in the time-based degree estimation code. For reference, the corresponding relay-based estimation would take mere seconds.

between two nodes u and v in our binary classifier can differ from each other. Unifying this circumstance into an error-minimized graph that still has the desired node degrees poses a significant problem. We leave solving this issue as a subject for future work. In this thesis, we work with a simplified model of the resulting graph. For this reason, the results obtained by us may differ from the values of the real network.

With the help of this graph reconstruction derived from the estimation, we determine graph-theoretic metrics. By doing so, we hope to gain insights into the nature of the peer-to-peer network that Bitcoin is based on. The results of the created metrics can be seen in Table 4.4.

On the one hand, we use metrics that produce a single number representing the entire graph. On the other hand, we also compute metrics that produce values for all individual nodes or node pairs. For the latter, we calculate the standard deviation, minimum and maximum as well as the Gini coefficient in addition to the arithmetic mean of all these values for a better breakdown of the value distribution. All metrics are applied only to those nodes that were part of the active measurement. Client nodes that were determined as the destination of a connection, but that do not have any outgoing connection data themselves, are thus explicitly excluded.

In addition, we compare the results obtained for the estimated graph with the results of a random graph. This comparison allows for a better understanding of what values might be expected or desired for the graph. The random graph is an Erdös-Rényi graph. It is created using the same parameters and therefore contains the same number of nodes and edges as the main graph.

Metrics: The formed graph consists of 3328 nodes, out of which 2614 are actively measured victims and 714 are clients. A total of $255,827$ connections are estimated between the nodes. Thereby, the victim nodes have an average node degree of 97.86802. A histogram of the node degree distribution is shown in Figure 4.14. For reference, the degree distribution of the relay-based estimation is also included in the histogram.

We can see that the node degree results of the two estimation modes overlap significantly. Both see a large proportion of the node degrees distributed over low values below one hundred. Furthermore, both estimators find an outstanding accumulation of nodes with degrees between 125 and 150. Overall, the relay-based estimator tends to show a smoother distribution of node degrees, while the time-based estimator shows a more pronounced accumulations in small value ranges. This is reflected in the spikes of the time-based histogram.

For clarity, the histogram cuts outliers in the degree estimation with estimated node degrees above 400. We found that there are 8 of these in our relay-based result data, with degrees of up to 1407, but none in the time-based estimation.

For comparison, we include the degree distribution as presented by Grundmann et al. in Figure 4.15. Comparing our degree distribution results with the result provided by

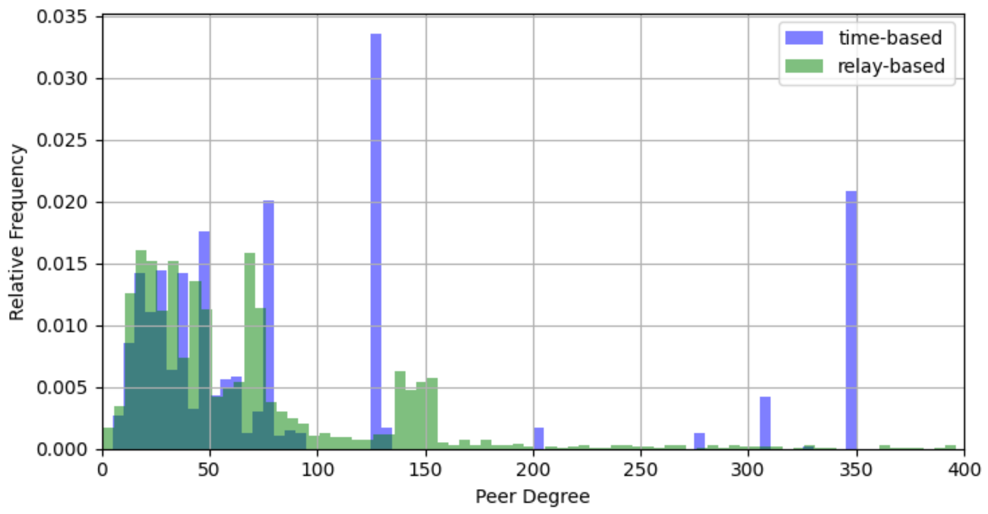


Figure 4.14: Normalized histogram of the estimated nodes degrees in the network in both relay-based and time-based estimation. Outliers of degree > 400 combined to the single maximum value for the relay estimation

their work, we see that both results agree in the detection of the large number of nodes with low degrees. The accumulation around node degree 125 is also confirmed by our findings. This makes sense, as 125 is the default maximum number of nodes connected to the standard client. However, the results differ in that the spike around 125 found by our time-based estimator is less pronounced. The accumulation that our estimator finds for high node degrees is completely absent from the comparative histogram. Also, the accumulation at about degree 75 that we find in both estimation modes is not confirmed by the researchers' results [GBH21]. It is unclear whether these differences are due to topological changes in the network or to differences in the experimental setups.

Naturally, the average degree of the nodes in the estimated graph is exactly equal to that of the random graph. The deviation of the node degrees from each other is much larger on the other hand. Thus, the distribution of the node degrees is exceptionally unequal, which can also be seen in the high Gini coefficient. In particular, this can mean that a large number of nodes with large degrees might play an important role in the network.

This is also reflected in the betweenness centrality. Here, the distribution of the values for the nodes is similarly highly dispersed, and the standard deviation is much higher than with the random graph. For Bitcoin, this means that there are a lot of nodes that occupy a particularly central position in the routing of the network. Interestingly, however, the Gini coefficient is equally low for both graphs. We can therefore conclude that there are differently central nodes in both graphs, but that the unequal distribution of centrality does not exceed the expected level.

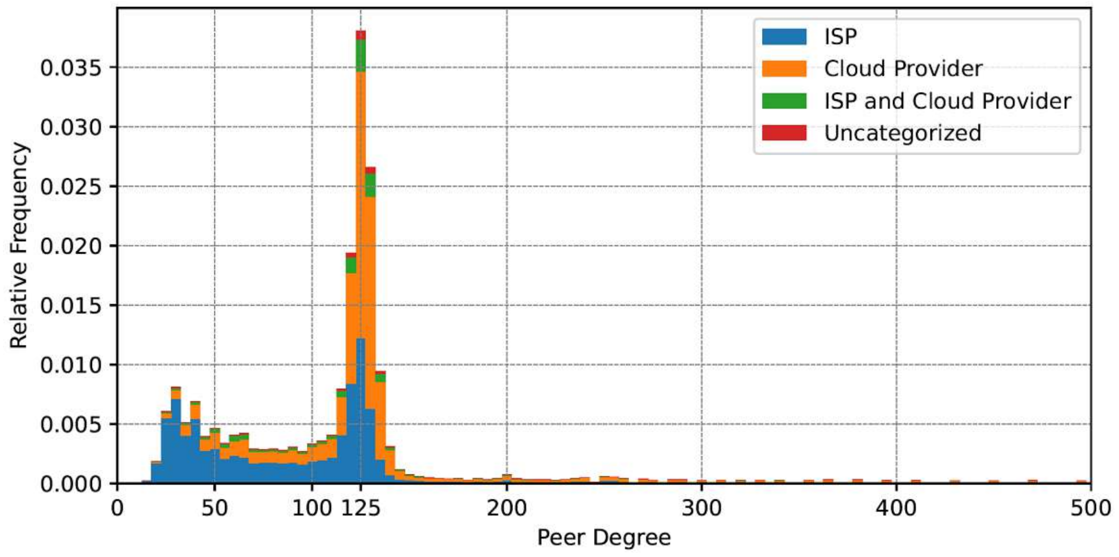


Figure 4.15: Normalized histogram of the estimated nodes degrees in the network as presented by Grundmann et al. [GBH21]

	Metric	Degree	Clustering	Betweenness	Shortest Path Length
estimated	min	5	0	0	1
	max	346	0.24863	30165	5
	avg	97.86802	0.06050	3101.82708	2.19854
	std	101.61283	0.02340	3241.19010	0.48393
	gini	0.50233	0.19179	0.09376	0.10030
random	min	67	0.02116	1233	1
	max	133	0.02999	5665	3
	avg	97.86802	0.02590	2792.31714	2.07134
	std	9.78032	0.00117	551.44565	0.35378
	gini	0.05627	0.02539	0.11015	0.05991

Table 4.4: Results of graph-theoretic metrics for the estimated Bitcoin network, and a random Erdös-Rényi graph with the same number of nodes and edges for comparison. The degree, clustering coefficient, and betweenness centrality metrics were applied to each node, and the shortest path length was applied to each pair of nodes in the network. For each of the metrics, the minimum, maximum, arithmetic mean, standard deviation, and Gini coefficient are given.

As expected, the clustering coefficient is consistently low in the random graph, with a negligible standard deviation. However, this is not the case for the estimated graph. Here we find a strongly unequal distribution of the clustering. The values range to a maximum that exceeds that of the random graph by a factor of more than eight. It is therefore evident that clusters of nodes are present in Bitcoin's peer-to-peer network.

In terms of the shortest path length, however, the two graphs hardly differ. Both have low values altogether, which indicates a fast distribution of information in the network. Here, the Gini coefficients of the shortest path length between all pairs of nodes are small for both networks. This means that the fast spread of information is roughly equally true for all nodes. The diameter of the networks is 5 and 3, respectively.

In order to evaluate the resilience of the estimated graph, we also calculated the bisection width of our graph and the reference graph. For this, we used the Kernighan-Lin bisection algorithm. This represents a heuristic for estimating the bisection width, making it runtime efficient enough to be performed for the large number of nodes in the graph [KL70]. The resulting value for the estimated graph is $70,218$, for the random one it is $113,310$. So we find that the bisection width is about 38.0302 % lower than it would be for a random graph. The reason for this reduced value may be the increased clustering. In practice, this would mean that the difficulty of certain attacks on the network is reduced. For example, an attacker who wants to separate the network into subnetworks would have to attack fewer connections.

Overall, our results are consistent with those of Delgado et al. who calculated network metrics for Bitcoin's testnet following their topology discovery attack. The researchers determined a diameter of 5, which corresponds exactly to our value. They also find an average clustering coefficient that is slightly higher than that of random graphs at 0.052 % [DSBPS+19]. Thus, although these values refer to the testnet instead of the mainnet and, with a total of only 733 nodes and 6090 edges, the size of the data set is much smaller, the evaluation results are about the same in this respect.

In direct comparison with the metrics of the random graph, we find that the Bitcoin peer-to-peer network does not appear to be a random graph. In particular, the uneven distribution of clustering coefficients and node degrees represents a clear difference between the graphs.

Overall, it can be seen that there appear to be relative clusters in the Bitcoin peer-to-peer network. The node degrees are also unevenly distributed, wich might lead to the assumption that individual nodes hold increased importance in the graph. In these aspects, the characteristics of the network graph differ greatly from those of a true random graph. The betweenness centrality is broadly scattered but not less uniformly distributed than it would be expected from a truly random graph. Accordingly, we find no evidence that the routing influence of individual nodes is strongly increased beyond the expected level in the network, despite the dode degree distribution hinting towards this fact. The low average shortest path length also indicates that the information exchange in the network is fast for all nodes.

We can conclude that the Bitcoin network is not a random graph. Nevertheless, the topology exhibits characteristics that we assume could be indicative of routing fairness and fast information exchange. We argue that it can be assumed that the Bitcoin network could therefore fulfill the typical requirements that can be placed on peer-to-peer networks in this respect. In terms of resilience, however, the network graph shows significantly reduced values.

4.4 Discussion

In this thesis, we present a method for degree estimation and connection inference of the underlying peer-to-peer network of Bitcoin. This is done by exploiting message forwarding of the address relay process. We show that both of these methods can be conducted together and are then suitable to form a comprehensive topology discovery of the network.

We perform validation experiments in a lab setting against a testbed node under our control. For the degree estimation, both relay-based and time-based estimation exhibit similar performance. Here, the relay-based estimation has relative error rates of up to 10 %, while the timing based estimation showed an error of up to 12 % for nodes with low degrees. This is in line with the results of Biryukov et al. reporting 3 – 10 % [BKP14] and Grundmann measuring 4.1 % [GBH21] error in similar experiment setups. In terms of the connection inference, the relay-based estimator had a precision of 20 % for a recall value of 40 %. The time-based estimator proved to have a higher performance here, with a precision value of 40 % at the same recall. This superiority of the timing analysis over the estimation based on idiosyncratic behaviours is also visible through the PR-Graph and the ROC-Graph. Here, the time-based estimation mode shows a slightly larger AUC for the ROC-Graph with a value of 0.89423, compared to the 0.84198 of the purely relay-based one. In the PR-Graph, the difference is considerably bigger with 0.44147 as opposed to 0.23448. Both estimation modes perform significantly better than a randomly guessing classifier, which indicates that the topology discovery is generally possible using the methods presented in this thesis.

We also validate the parameter choices of the experiment against our testbed node. Validating node degree and monitor count, we see no clear trend. When trimming the receptions by transit times, we note that the time-based estimator is nearly unaffected. The relay-based estimator on the other hand changes performance drastically depending on the maximum time to live of bullets: For any value above 400s, it performs the connection inference no better than random guessing. The lower the value below 400s, the more the performance rises steeply, all while still having the time-based estimator as upper bounds. The parameters, with which the training set for the time-based estimation was generated, seem to have no noticeable effect on the estimation results.

Furthermore, we conducted the topology discovery attack against the peer-to-peer network that underlies Bitcoin. Based on the estimated connections, we then apply metrics to the topological model that we found. To this end, we also build a random Erdös–Rényi-Graph based on the same parameters, to compare the metric results against. In doing so, we see that the network seems to have clusters, and a degree distribution that distinguishes it from a random graph. The betweenness centrality is evenly distributed, suggesting a fair routing. The shortest path length is generally low, which may indicate a fast information distribution within the network. Lastly, we find that the bisection bandwidth is reduced significantly compared to the random graph.

Because we find that the estimator performance is decent even with rather low connectivity values and a reasonably low number of injected bullets, we are confident these results hold true. In the end, these are the parameters that the practical experiment conduction was most restricted by. Furthermore, we use a series of complementary measurements to show that the overall node churn rate in the network is approximately 30 %. The link stability in the network is high, with more than 80 % of a given nodes peers being still connected to it after one hour. This increases the credibility of the results further, as it indicates that the estimation results would be valid for a long period of time.

We observed that the network load during the experiment noticeably increased. This suggests that it would be possible for observing nodes to detect such large network wide measurements in the future. As it was the overall load that was increased, a detection can be done even when the injections are not made directly into nodes of the observer. In fact, it was this increase of observed addresses in circulation that led to the belief that an attack similar to our procedure has already been performed by unknown actors. Grundmann et al. report that in July and August 2021, an unknown actor connected nodes in the network to send them spoofed addresses within a large number of ADDR-messages [GBH21]. This observation precedes all experiments we conducted in the context of this thesis. However, we note that this observed behaviour is identical to the one that would be observed from the perspective of network participants that are victims to the experiments described in this thesis. In their paper, they restrict their analysis of the event on the idea that the observed influx in addresses could relate to a degree estimation experiment. However, they do not note the possibility of a connection inference and thus a full topology discovery having been conducted with the same event. In this thesis, we have shown that the topology discovery attack they observed in 2021 could have gone further than they have previously considered. The unknown actor that sent out the initial suspicious ADDR-messages could have obtained a detailed topological map of the Bitcoin network with considerable accuracy, in much the same way that we did.

With respect to the observed spam wave, we also note that, according to Grundmann et al., the performer adjusted the timestamp of the ADDR-messages, thereby artificially increasing their transit times [GBH21]. This would not be necessary for a mere degree estimation, as Biryukov et al. note [BKP14] and we verify in Section 4.1.2. Also, for simple connection inference using the naïve relay-based estimation approach, this would not be beneficial either, and it has no impact on timing analysis as we have also shown in Section 4.1.2.

All in all, the experiment we have conducted in this thesis has mainly shown that past countermeasures implemented by Bitcoin are not sufficient to adequately prevent attacks of this type. Grundmann et al. claim that the rate-limiting implemented in September 2021 prevents the degree estimation they performed. A detailed justification for this claim was not given [GBH21]. However, we have seen that degree estimation is still feasible despite rate-limiting. On the one hand, this is due to a suitable scaling of the injections with several sybil nodes. Thus, an injection of sufficient messages for the degree estimation is still possible in reasonable time. On the other hand, it is also because even small amounts of injected messages are enough to accurately determine the node degree, as we validated in Section 4.1.2.

Similarly, Neudecker et al. introduced trickling as a countermeasure to the timing analysis they performed [NAH16]. However, we have seen that a timing analysis is still possible in ADDR-message relay despite the trickling that has been implemented in the meantime. We have validated that the performance of our timing analysis is at least as high as that of the researchers without trickling. Furthermore, the performance of the naïve relay-based estimation increases with cutting the messages according to transit times, as we were able to validate in Section 4.1.2. The time-based estimator remains the upper bound of the relay-based estimator in precision and recall. We therefore conclude that the influence of the timing data is still essential for the connection inference. Evidently, timing analysis has not been distorted, as it would be the goal of the trickling.

It is particularly noteworthy that the costs of the attack in terms of hardware are exceptionally low. The attacker only needs to operate a few Sybil nodes. These, in turn, are mostly passive listeners, so they have little computational overhead. For this thesis, we decided to use the university's dedicated measurement server. However, we found that the framework we developed can also be run on average mid-range hardware. Equipment of this scale could well be owned by a potential attacker for private use. This means that there is only a low hurdle to the implementation of the presented attack. This underlines the possibility that attacks of this kind could continue to be detected in the wild in the future.

Limitations: The measurement and subsequent application of network metrics was limited mainly by the fact that stable connections could not be established to the entire network at the same time. There are three main reasons for this: 1) The measurement works by design only for nodes that allow incoming connections, which corresponds to about 19 % according to estimations [GBH21]. 2) We have chosen to reduce the scope of this thesis by not measuring Onion-addresses. These would account for about 53 % of the openly reachable nodes. 3) Not all open nodes accept all incoming connections. This is possibly because nodes do not have enough open connection slots left [GBH21]. Due to this, the percentage of nodes that held enough initial connections to be measured was 43.93 % of all designated victim peers. In total, the proportion of nodes that could be actively measured simultaneously was thus about 22.79 % of all reachable clients and 4.33 % of all total Bitcoin clients. In this sense, only a subnetwork of the Bitcoin network was measured. It can therefore be assumed that the determined graph-theoretical metrics deviate to a certain degree.

For the nodes where a measurement could take place, we were able to validate that the estimation produces solid results that go far beyond random guessing in their precision and recall. Nevertheless, an accuracy of 100 % could not be achieved. Accordingly, the presented topology discovery attack cannot be used to lead to a situation where an attacker knows all neighbours of a victim node.

Furthermore, topology discovery based on ADDR-message relay can only identify non-block-relay connections by design. Attacks can therefore not build on the detected topology with the aim of preventing block relay.

Finally, the number of injections per node was also limited by the availability of addresses. An attacker with access to a larger usable IP address space, or one that uses random addresses, is able to perform more injections. This could then further lead to an increased performance of the estimation.

4.4.1 Moral Considerations

Where possible, we have tried to affect the Bitcoin network as little as possible for the experiment and preparatory tests. It is not the goal of the presented attack to have a disruptive impact on the operation of the peer-to-peer network. We argue that a malicious attacker also shares this goal, since a non-disruptive execution of topology discovery means that the attack is more likely to go undetected. To achieve this goal we have taken the following measures:

- A filter proxy was used as described in Section 3.3 to prevent our addresses from being forwarded to the network during testing unless necessary.
- Only IP addresses from an address range under our control were used to prevent uninvolved foreign addresses from being advertised as Bitcoin node operators and consequently being exposed to connection attempts.
- By lowering the associated timestamp, the forwarding time of each address and thus the overall traffic was reduced.
- A minimal choice of specified services per address ensures that bullets, even if stored in nodes in the address manager of victim peers, are seen as uninteresting for connections. This reduces the number of failed connection attempts as a result of our experiment.
- By choosing the number of monitors to be 5, the total number of sybil nodes in the network is negligible. Admittedly, these sybil nodes did not productively participate in network operations, for example by delivering blocks or transactions. However, sybil nodes represented only a small proportion of all connections of a given network participant. We can see this from the fact that in the results of the estimation, the average node degree is much greater than the number of the given

> sybil node connections. Therefore, it is not reasonable to assume that any node of the network was disrupted in its regular operation by the connection of these sybil nodes.

The extent to which overall ADDR-message traffic on the network increased during the experiment was determined in Section 4.1.2. We argue that this increase is measurable network-wide but is too small to be perceived as a disruptive factor. Overall, we consider the value for the estimated upper bound of the additional data consumption of $1.7434\,\frac{B}{s}$ per connection per node to be negligible.

The network can also be disrupted if the bullet addresses that are not accessible as Bitcoin clients are stored in the address memory of victim nodes during the experiment. However, in our reference node, which was also exposed to the measurement, we found only 12 bullet addresses in the address memory after the measurement. This means that the number of addresses added to the address managers of the nodes in the network as a direct result of the experiment is negligible. Therefore, it cannot be assumed that the experiment had a significant negative impact on the shared address knowledge in the network.

All in all, we therefore consider the influence of the measurement on the network to be sufficiently low to justify the actual execution of the attack on the basis of academic interest.

CHAPTER 5

Conclusion

In this thesis, we explored a method to discover the topology of the Bitcoin peer-to-peer network. For this purpose, we introduced methods for degree estimation and connection inference. We also have presented an approach for performing these methods simultaneously in a single active measurement. The underlying idea is to observe the forwarding of ADDR-messages sent by the attacker. We used two methods for estimation, one based on characteristic forwarding behaviour, and another one based on flooding inherent timings. The goal was to evaluate the performance of the timing analysis by using the naïve estimation method as a baseline for comparison.

The validation of our approach shows that connection inference can be performed with considerable precision and recall of about 40 % each. The relative error of the degree estimation also remains below 10 %. For both estimations, timing-based analysis proved to be superior to the relay-based method. This shows that trickling as a countermeasure against timing analysis is unable to reduce the accuracy of such an attack in practice. Even with the implementation of rate-limiting in the Bitcoin reference client as a countermeasure in place, one can still achieve sufficiently high injection speeds such that the analysis yields useful results.

We implemented a framework to carry out this approach. We used this framework to actively measure a portion of the Bitcoin mainnet, demonstrating that such measurements can be performed with minimal time and hardware. The topological model resulting from the estimation was then analysed using graph-theoretic metrics. We showed that the network exhibits non-random structural properties, especially in the distribution of node degrees and local clustering coefficients. In addition, we showed that the network possesses significantly lower bisection width than a comparable random graph.

5.1 Future Work

We have shown that this attack is possible in the current way the protocol works. Whether it can be prevented by changing the relay algorithm remains to be seen. Thus, future work could investigate possible adjustments of the trickling and rate-limiting parameters. For example, without structurally changing the flooding protocol per se, the rate-limiting could be tightened to make it unprofitable to perform the measurement we have presented. Alternatively, the utilisation of entirely different flooding algorithms could be examined. Another approach might be, for example, avoiding relay-based degree estimation by setting address forwarding probabilities per neighbour to a fixed value instead of being dependent on the node degree.

It would also be worth examining whether the identified network metrics change when Onion-nodes are included in the measurement. We measured only about half of the open mainnet without connection to Onion-services. With an adjustment to the framework, the remaining nodes could also be measured to see whether this changes the resulting network metrics.

Furthermore, it remains to be seen to what extent the methods presented in this thesis can be applied to other peer-to-peer networks, such as Ethereum or Dogecoin. Networks that implement information flooding in their protocol similar to Bitcoin's gossip protocol may be affected in the same way. In addition, networks that have implemented trickling may test whether it provides the desired level of protection against timing analysis by adapting and applying our work.

Finally, we suggest that future work should discuss the trickling procedure once more. We have shown that trickling in its current form does not prevent timing analysis. Accordingly, a discussion of possible changes is of utmost necessity.

List of Figures

List of Tables

Bibliography

[BG94] Michael Buckland and Fredric Gey. The relationship between recall and precision. *Journal of the American society for information science*, 45(1):12–19, 1994.

[Bita] Bitnodes.io. Bitnodes dashboard. `https://bitnodes.io/dashboard/`, visited 21.04.2022.

[Bitb] Bitnodes.io. Reachable bitcoin nodes. `https://bitnodes.io/`, visited 21.04.2022.

[BKP14] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in bitcoin p2p network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 15–29, 2014.

[BMS01] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *International Workshop on Information Hiding*, pages 245–257. Springer, 2001.

[BW10] Edward A Bender and S Gill Williamson. *Lists, decisions and graphs*. S. Gill Williamson, 2010.

[CLA16] Fabio Caccioli, Giacomo Livan, and Tomaso Aste. Scalability and egalitarianism in peer-to-peer networks. In *Banking beyond banks and money*, pages 197–212. Springer, 2016.

[Cmm] Bitcoin Cmmunity. P2p design philosophy. `https://github.com/bitcoin-core/bitcoin-devwiki/wiki/P2P-Design-Philosophy`, visited 21.07.2022.

[DBG18] Varun Deshpande, Hakim Badis, and Laurent 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)*, pages 1–6. IEEE, 2018.

[Die00] Reinhard Diestel. *Graph Theory*. 2000.

[DSBPS$^+$19] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. Txprobe: Discovering bitcoin's network topology using orphan transactions. In *International Conference on Financial Cryptography and Data Security*, pages 550–566. Springer, 2019.

[ECP21] Jean-Philippe Eisenbarth, Thibault Cholez, and Olivier Perrin. A comprehensive study of the bitcoin p2p network. In *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 105–112. IEEE, 2021.

[EPJ20] Meryam Essaid, Sejin Park, and Hong-Taek Ju. Bitcoin's dynamic peer-to-peer topology. *International Journal of Network Management*, 30(5):e2106, 2020.

[ER$^+$60] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[Faw04] Tom Fawcett. Roc graphs: Notes and practical considerations for researchers. *Machine learning*, 31(1):1–38, 2004.

[GBH21] Matthias Grundmann, Max Baumstark, and Hannes Hartenstein. Estimating the peer degree of reachable peers in the bitcoin p2p network, 2021.

[GNH18] Matthias Grundmann, Till Neudecker, and Hannes Hartenstein. Exploiting transaction accumulation and double spends for topology inference in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 113–126. Springer, 2018.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on {Bitcoin's}{peer-to-peer} network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[HLY04] Ken YK Hui, John CS Lui, and David KY Yau. Small world overlay p2p networks. *In other words*, 345(6):2, 2004.

[HT01] David J Hand and Robert J Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine learning*, 45(2):171–186, 2001.

[ISTY19] Muhammad Anas Imtiaz, David Starobinski, Ari Trachtenberg, and Nabeel Younis. Churn in the bitcoin network: Characterization and impact. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 431–439, 2019.

[JD16] Feng Jian and Shi Dandan. Complex network theory and its application research on p2p networks. *Applied Mathematics and Nonlinear Sciences*, 1(1):45–52, 2016.

[KL70] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[LEN14] Zachary Chase Lipton, Charles Elkan, and Balakrishnan Narayanaswamy. Thresholding classifiers to maximize f1 score. *arXiv preprint arXiv:1402.1892*, 2014.

[LKRG03] Dmitri Loguinov, Anuj Kumar, Vivek Rai, and Sai Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 395–406, 2003.

[LSM06] Brian Neil Levine, Clay Shields, and N Boris Margolin. A survey of solutions to the sybil attack. *University of Massachusetts Amherst, Amherst, MA*, 7:224, 2006.

[MLP$^+$15] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Discovering bitcoin's public topology and influential nodes. *et al*, 2015.

[NA13] Harikrishna Narasimhan and Shivani Agarwal. On the relationship between binary classification, bipartite ranking, and binary class probability estimation. *Advances in neural information processing systems*, 26, 2013.

[NAH16] Till Neudecker, Philipp Andelfinger, and Hannes 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)*, pages 358–367. IEEE, 2016.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[NH18] Till Neudecker and Hannes Hartenstein. Network layer aspects of permissionless blockchains. *IEEE Communications Surveys & Tutorials*, 21(1):838–857, 2018.

[Proa] Bitcoin Project. Bitcoin core. `https://github.com/bitcoin/bitcoin`, visited 21.07.2022.

[Prob] Bitcoin Project. Net procesing. https://github.com/bitcoin/bitcoin/blob/master/src/net_processing.cpp, visited 22.07.2022.

[Proc] Bitcoin Project. P2p network. https://developer.bitcoin.org/reference/p2p_networking.html, visited 20.04.2022.

[Prod] Bitcoin Project. P2p network. https://developer.bitcoin.org/devguide/p2p_network.html, visited 21.07.2022.

[Ros] Jakob Rosenblattl. Topology discovery within the bitcoin network.

[Rou11] Derrick Rountree. 3 - network security. In Derrick Rountree, editor, *Security for Microsoft Windows System Administrators*, pages 71–107. Syngress, Boston, 2011.

[Sch01] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings First International Conference on Peer-to-Peer Computing*, pages 101–102. IEEE, 2001.

[TB] Tor-Blog. V3 onion services usage. https://blog.torproject.org/v3-onion-services-usage/, visited 21.04.2022.

[W$^+$01] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[Wei19] Itsi Weinstock. The bitcoin p2p network topology: a review of inference methods and machine learning approach. 2019.

[Wika] Bitcoin Wiki. Protocol documentation. https://en.bitcoin.it/wiki/Protocol_documentation, visited 21.07.2022.

[Wikb] Bitcoin Wiki. Testnet. https://en.bitcoin.it/wiki/Testnet, visited 21.07.2022.

[Wil79] Robin J Wilson. *Introduction to graph theory*. Pearson Education India, 1979.

[XCW05] Dong Xuan, Sriram Chellappan, and Xun Wang. Resilience of structured peer to peer systems: Analysis and enhancement. *Handbook On Theoretical And Algorithmic Aspects Of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, Auerbach Publications*, pages 767–786, 2005.

[ZFDS22] Philipp Zabka, Klaus-Tycho Foerster, Christian Decker, and Stefan Schmid. Short paper: A centrality analysis of the lightning network. 2022.