TU WIEN Informatics

Topology Discovery within the Bitcoin Network

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Jakob Rosenblattl, BSc
Registration Number 01527437

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Assistance: Univ.Lektorin Dipl.-Ing. Dr.techn. Johanna Ullrich, BSc

Vienna, 24th December, 2023

Jakob Rosenblattl	Edgar Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Jakob Rosenblattl, BSc

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, 24. Dezember 2023

Jakob Rosenblattl

Danksagung

Ich möchte mich herzlich bei all jenen bedanken, die mich bei der Fertigstellung dieser Arbeit unterstützt haben.

Zunächst gilt Johanna Ullrich großer Dank für ihre wertvollen Ideen und Ratschläge. Ich danke für ihre scheinbar unerschöpfliche Geduld sowie für ihr Vertrauen in meine Kompetenz und Selbstständigkeit. Dank geht auch an meinen Betreuer Edgar Weippl.

Weiters bedanke ich mich bei meinem Kollegen Markus Maier, der mir Zugang zur virtuellen Maschine für die Durchführung meiner Simulationen verschafft und mir bei technischen Problemen weitergeholfen hat. Durch die erhöhte Rechenleistung konnte ich die Qualität meiner Arbeit bedeutend steigern.

Ganz besonders möchte ich mich bei meinen Eltern bedanken. Ohne eure Unterstützung wäre mein Studium nicht möglich gewesen und diese Arbeit wäre nicht zustande gekommen. Ich habe mich immer schon auf euch verlassen können. Danke, dass ihr mir mein ganzes Leben lang nicht nur finanziell, sondern auch seelisch und tatkräftig beiseitegestanden seid.

Abstract

Bitcoin is the first decentralised digital currency, which to this day has a significant impact on the body of scientific literature, the global economy and to a certain extent even climate change. The topology of its underlying network is largely hidden; however, knowledge of it could on the one hand provide valuable insights into the state of the system and on the other hand aid in attacks against the network. Previous research on topology inference is either no longer applicable due to countermeasures implemented in the reference client or is otherwise impractical.

In this thesis we present a method for finding the degree of a target Bitcoin node as well as the network addresses of a subset of its peers. Our technique exploits the gossip protocol around `ADDR` messages and consists of (1) connecting monitor nodes to as much of the Bitcoin network as possible, (2) sending `ADDR` messages containing unique marker addresses to target nodes, (3) recording the recurrence of each marker address at any monitor node, and (4) analysing the recorded samples. We estimate node degrees from the fraction of recurred marker addresses and calculate the probability of connection between two nodes from the delays between sending markers to one and receiving the same markers from the other node.

We validated our techniques against simulated traffic data. Our degree estimation method was nearly unbiased and produced errors showing a standard deviation of 1.5 in an idealised model and errors showing a standard deviation of 3.6 in a more realistic one, although in both models the standard deviation of relative errors amounted to only 5 %. Using conservative classifier sensitivities, our connection inference method performed with a precision of 40 % and a recall of 99.8 % in the idealised model and both precision and recall of 56 % in the more realistic one. Using optimal sensitivities at the risk of overfitting, precision and recall come out to be 94 % and 98 % in the former model and 83 % and 47 % in the latter model.

These findings suggest that cheap and moderately successful topology inference may be possible in the real Bitcoin network. More research is required to assess the effectiveness of our approach in greater detail. Nonetheless, Bitcoin developers are advised to consider adapting current countermeasures or implementing new ones if they want to ensure that the Bitcoin network topology remains obscured.

Contents

CHAPTER 1

Introduction

Bitcoin is a decentralised digital currency. Its white paper titled *Bitcoin: A Peer-to-Peer Electronic Cash System* [Nak08] was published by an unknown party under the pseudonym Satoshi Nakamoto in 2008, making it the first of its kind. It allows for direct transactions between Bitcoin users without the need for a central authority or trusted mediator, thus eliminating reliance on financial institutions or national governments. As a consequence, Bitcoin can offer high degrees of independence, security, freedom and—if used carefully—pseudonymity and even anonymity.

Since its inception Bitcoin has accrued considerable academic attention, media coverage and controversies. Over the years, Bitcoin's popularity for investment has increased drastically, reaching a market capitalisation of $1 trillion for the first time [coi]. In 2021 El Salvador became the first country to adopt Bitcoin as legal tender [Del21]. As both its demand and value increased, more and more effort was exerted into a process called *mining*, which refers to a computationally intensive task that allows for the creation of new bitcoins. In recent years bitcoin mining reached a scale large enough to motivate scientific investigations on its potential environmental impact. In 2022 it was estimated to be responsible for 65.4 megatonnes of CO_2 per year, representing 0.19 % of global emissions [DVGKS22].

In order to make use of Bitcoin, one has to run a Bitcoin client implementation, the most popular being *Bitcoin Core*, and thereby take part in the Bitcoin network. Every transaction issued by a client is broadcast to the entire network. Over time transaction records are collected and compiled into *blocks*. These blocks are logically chained together, forming a data structure called the *block chain*. Each Bitcoin client maintains its own copy of the block chain. Every time a network node creates a new block, it is broadcast to the entire network, allowing for the constant synchronisation of the block chain within the network. This way the block chain can serve as a public ledger through which every network node can confirm the balance of any other network node.

An important technical aspect of Bitcoin is the quality of its underlying network. As the title of the white paper gives away already, Bitcoin builds on a peer-to-peer (P2P) network of Bitcoin clients. The defining characteristic of a P2P network is that all nodes have the same role in the sense that they are equally privileged and capable, i.e. there is no hierarchical or role-based structure as would be the case in client-server network application designs, for example. In a P2P network, connected network nodes are said to be *peers* of each other. The *topology* of the network refers to the arrangement of connections between the nodes. Knowing this arrangement would help researchers and developers to assess the health of the Bitcoin network with regards to fairness, robustness and performance [DSBPS+19]. However, Bitcoin Core is designed to leak as little information as possible that would allow for discovering a node's peers or the number thereof. Bitcoin tries to hide its topology due to the risk of abuse. Knowledge on the topology of the network could aid attackers in performing certain kinds of attacks on individual nodes or larger parts of the network. One such attack is called *eclipse attack*, where an adversary tries to monopolise all connections to and from a victim node [HKZG15].

An early landmark paper in the field of Bitcoin network topology discovery was published by Biryukov et al. [BKP14] in 2014. Their main goal was not primarily topology discovery but rather deanonymisation of Bitcoin users (i.e. linking public keys to network addresses), for which topology inference was a necessary step. They achieved the latter by exploiting mechanisms related to `ADDR` and `GETADDR` messages implemented in Bitcoin Core. In the protocols of the Bitcoin network `ADDR` messages are used to exchange network addresses (e.g. IPv4, IPv6 or onion addresses) of network nodes so that Bitcoin clients can establish connections to each other in order to participate in the network and thus the Bitcoin system as a whole. When a Bitcoin client receives an `ADDR` message from one of its peers under the right conditions, it relays the contained network addresses by packing them into new `ADDR` messages and sending them to some of its other peers, which in turn will repeat the process. This type of protocol is called a *gossip protocol* and together with `ADDR` messages they would form a recurring theme in research following this publication. However, in response to this paper, countermeasures have been implemented into the reference client, rendering their methods ineffective.

In 2015 Miller et al. [MLP+15] published a different topology inference method that also exploited `ADDR` messages. Again, Bitcoin Core was patched soon after to disable this kind of attack. In 2016 Neudecker et al. [NAH16] presented a timing-based traffic analysis technique for topology inference. The idea was to observe the propagation of `INV` messages[1] and to calculate the probability of connection between two clients given the set of related message timings. As will be discussed in detail later, this is the type of inference technique the legacy of which we try to continue in this thesis. The authors validated their methods in the real Bitcoin network, where their classifier achieved both precision and recall over 40 %. As a countermeasure for this attack, Bitcoin Core was patched to introduce random delays between relays of `INV` and `ADDR` messages, as the

[1] Used for notifying peers about new data objects. Short for *inventory*.

latter could also be used for this technique. The researchers found this countermeasure to be effective—however, this finding will be challenged by the results of our evaluations presented later in this thesis.

In 2019 Grundmann et al. [GNH19] presented two novel approaches for topology inference, neither of which would exploit `ADDR` message gossip. In experiments on connection inference the more effective of their methods could achieve a precision of 71 % and a recall of 87 % for the cost of 99 transaction fees at the time. Delgado-Segura et al. [DSBPS+19] published their inference technique called *TxProbe* in the same year, which exploits mechanisms regarding the handling of certain transaction records. Their method was able to achieve precision and recall over 90 %. However, as they could not rule out that their technique is disruptive to the Bitcoin system, they used the Bitcoin testnet[2] for their evaluations. Nevertheless, due to their reported success the Bitcoin developers implemented a type of connection called *block-relay-only*, over which no transaction record or network address relay can occur. This countermeasure protects connections of such type from being found using inference methods relying on the gossip protocol for the corresponding message type. In 2021 Grundmann et al. [GBH21] were able to observe a spam wave of `ADDR` messages of unknown origin moving through the network. Using the recorded data they were able to estimate the node degree distribution of part the network, which we will make use of in our research.

Our work aims to answer the question whether the `ADDR` gossip protocol can be exploited using active traffic analysis techniques in order to discover the topology of the Bitcoin P2P network despite current countermeasures implemented in Bitcoin Core. We hope that our findings may serve as a basis for future measurements that seek to assess (1) characteristics of the network related to fairness, robustness and performance and (2) the effectiveness of employed countermeasures. With our connection inference technique we aim to reach precision and recall of at least 40 %, which is what Neudecker et al. [NAH16] have previously achieved using analyses of timing data from gossip-related traffic.

In this thesis we present a method to estimate the number of peers of a target Bitcoin client, i.e. its node degree in the network, as well as a method to infer whether two clients are connected via a *full-relay* connection, i.e. any connection other than one of type *block-relay-only*. Both techniques draw their input data from the same actively collected sample of `ADDR` messages. In order to collect samples, nodes under our own control—so-called *monitor* nodes—have to connect to the target nodes and all other nodes of which we want to determine if they are connected to a specific target. Then, these monitors send `ADDR` messages containing unique *marker addresses* to the targets, which proceed to relay them to their peers. For degree estimation, we consider the fraction of marker addresses that each target directly relays the connected monitor nodes, as this fraction correlates with the target's total number of peers. For inferring the existence of a full-relay connection between a target and another observed Bitcoin client,

[2]The Bitcoin testnet is a network parallel to the Bitcoin network used for testing purposes. It uses a distinct alternative block chain containing transactions of bitcoins that are not designed to carry any value.

we consider the time it took the marker address to travel over two or more relays from the target through the network and finally back to a monitor. This approach is taken from the 2016 paper by Neudecker et al. [NAH16]. However, in our adaptation we use different stochastic relationships to calculate the connection probability and we have to defeat the topology discovery countermeasures implemented since then. In order to validate our methods and not to disturb the Bitcoin network, we wrote a computer simulation of the data collection process and used the generated samples as ground-truth for our evaluations.

This thesis contributes to the body of knowledge surrounding Bitcoin network topology discovery in these points:

- We give a concise summary of previous attempts to infer node degrees and network links and state if and how each of them have been mitigated by specific countermeasures implemented in Bitcoin Core.
- We formally define techniques to infer node degrees and network links in the Bitcoin P2P network using active traffic analysis on recorded `ADDR` messages.
- We evaluate our methods using cross-validation on simulated network traffic, determine inference accuracy and highlight other notable findings.
- We discuss limitations of `ADDR`-gossip-based traffic analysis and outline potential for future research.

The remainder of this thesis is structured as follows. Chapter 2 explains all relevant features of the Bitcoin P2P network as well as mathematical concepts used to define our theoretical framework and techniques. Chapter 3 describe the methodology of our work, including the relay data collection process, formal models of the network and its behaviour, statistical methods for topology inference, and the purpose and function of our simulations. Chapter 4 presents the results of cross-validating our topology inference methods against simulated network traffic. Chapter 5 discusses the significance and limitations of these results and outlines potential for further refinement and evaluations. Finally, chapter 6 concludes by giving a summary of our findings as well as their impact and by pointing out opportunities for future research.

CHAPTER 2

Background

This chapter will present the background information necessary to understand our methods and the context of this research. Section 2.1 helps to motivate this thesis by explaining how Bitcoin works with special emphasis on the relay mechanism responsible for the propagation of network addresses through the network. Section 2.2 deals with concepts in the fields of graph theory and probability theory, particularly probability distributions and statistical inference. Section 2.3 gives an overview of related work surrounding the topic of topology inference with regards to the Bitcoin P2P network and how techniques and countermeasures have evolved over time.

2.1 Bitcoin

Bitcoin is a decentralised digital currency. As such, it allows for electronic payments between two parties without the need for central financial institutions or trusted third party mediators. Furthermore, if used carefully, it can provide anonymity to its users to a certain extent. Bitcoin is implemented as a *peer-to-peer* (P2P) network protocol using cryptographic *proof-of-work* to enable practically irreversible transactions.

These transactions are assembled into *blocks* and recorded in a distributed data structure called the *block chain*, which serves as a public ledger. Any Bitcoin client may choose to maintain an up-to-date copy of the block chain by constantly synchronising its local version with the copies of its peers.

In Bitcoin, transactions are digital records that transfer ownership of bitcoins (BTC) between Bitcoin users. Whenever a Bitcoin user wants to perform a transaction, he or she has to specify (1) a set of recipients along with the amount of BTC each of them is to receive (*transaction outputs*), and (2) a set of references to previous transactions through which he or she has received the BTC that shall be transferred (*transaction inputs*). Notice how each transaction input references a transaction output of an earlier

transaction. The inputs must sum up to at least the sum of the outputs and no input is allowed to reference an output that is already referenced by some other input of a transaction found on the block chain. That is, any output of any transaction can only be used *once*, meaning there is no *double spending*. Furthermore, the inputs must be signed by the sender. Every user owns a key pair, where the public key—also known as the user's *Bitcoin address*—serves as his or her identity in transactions and the private key serves as a means of providing authenticity through signing documents, e.g. transaction inputs. As a result, a Bitcoin user's wealth amounts to the total of BTC he or she has accrued as transaction outputs towards him or her that have not yet been used. Those are called *unspent transaction outputs*.

The next step in completing the transaction is for the Bitcoin client to broadcast the newly created transaction record to its peers. Whenever a client receives a transaction from a peer, it will validate it by checking if the outputs are covered by the inputs and whether the signatures are valid. Only if the transaction is valid, the client will proceed to relay it to its peers, which in turn will validate and broadcast the transaction (except for those clients which have received and broadcast this exact transaction record before). By means of this *flooding* mechanism, the new transaction propagates quickly through (almost) the entire network.

Recent transactions are collected and assembled into blocks by network nodes called *miners*. Each block contains a reference to a previous block, thereby creating a chronologically ordered chain of blocks—the block chain. A block must contain a number, called the *nonce*, such that the hash value of the block is smaller than a certain limit. Finding a right nonce is done by guessing and checking and requires computational power. This system is called *proof-of-work* as anyone can easily compute the block's hash and prove that work has been carried out in order to find a fitting nonce. Once the miner has found a nonce, through which the block's hash value is admissible, it appends the block to its local copy of the block chain and broadcasts it to its peers, which will do the same if the block is valid. In each block, miners are allowed to include a transaction that transfers a predetermined *block reward* in BTC to themselves without transaction input. This process of assembling valid blocks and creating new BTC under the expense of hashing power is called *mining*.

Bitcoin clients view the longest available chain of blocks, which is the chain with the most accumulated proof-of-work, as the true history of events. Note that it is not feasible for a dishonest individual or group to forge an alternative consistent block chain that is longer than any other unless they control more hashing power than the rest of the network combined. This is because even altering one transaction in the block chain alters the enclosing block's hash, which necessitates finding a new nonce. Since each block contains the hash of its predecessor, the forger would have to repeat this process for every following block. When only a minority of hashing power works on an alternative block chain, it will never catch up with the currently longest one, on which most of the network's hashing power is expended.

Miners race each other in finding the next block. If multiple miners each find a different

valid block then the first miner to broadcast its own block has the best chances of reaping the block reward as only one of these blocks will end up in the longest chain and its own block had a head start by being broadcast earliest. Note that a miner implicitly accepts a new block found by another miner by treating it as the latest block in the chain and thus by building upon it. The miner ceases to search for the right nonce for its current block candidate and instead assembles a new block containing the hash of the block that the other miner just published. When a miner does not accept a block published by some other node, e.g. because it is invalid due to containing transactions using bitcoins that have been spent already, it decides to continue where it left off, namely building on the longest chain excluding this block. However, if a miner rejects building upon a *valid* new block, it effectively *forks* the chain by creating a separate branch, called a *fork*. Assuming most other miners (in terms of collective hash power) do *not* reject valid blocks but accept and build upon them, the new fork will not become the longest branch, causing all other impartial clients to disregard its transactions and block rewards. Therefore, miners can only hope to win block rewards by accepting new valid blocks as soon as possible in order not to lose any time in the search for the newest block.

Bitcoin's most popular client implementation is called *Bitcoin Core*[1]. For the remainder of this thesis all Bitcoin client behaviour is assumed to follow the implementation of Bitcoin Core 24.1[2].

2.1.1 P2P Network

The Bitcoin network is a *peer-to-peer* (P2P) network, meaning that there is no need for central nodes providing special services. Each Bitcoin client represents a node in the network having the same privileges like any other node. It does not have to be connected to some central server but instead relies on a set of peers to receive or broadcast new information from or to the rest of the network, respectively. Peers try to keep each other up-to-date regarding, for example, the state of the block chain, new transactions, network addresses in the P2P network, etc. Information propagates through the network by means of *message flooding* or a *gossip protocol*, where each node relays information it has not yet received to a selection of its peers, which in turn will do the same.

Bitcoin Core, which is the most popular Bitcoin client implementation, distinguishes between six types of connections, called *inbound*, *manual*, *feeler*, *outbound-full-relay*, *block-relay-only* and *addr-fetch*. The latter five may sometimes collectively be referred to as *outbound* connections. When a client seeks to build another link to the network, it reaches out to a new network node and tries to establish a connection. If the other node accepts, the client gained a new peer over an outbound connection. When the client accepts an incoming connection request, however, the connection to the new peer is considered to be inbound. The *outbound-full-relay* connection type represents ordinary long-term links without special purpose. As a security measure, Bitcoin Core added

[1] https://bitcoincore.org/
[2] https://github.com/bitcoin/bitcoin/tree/24.x

the *block-relay-only* connection type, which—in contrast to the former type—does not allow the relay of transaction and network address records. All other types, sometimes collectively referred to as *full-relay* connections, allow for such relays in order to facilitate information exchange via gossip. By default a Bitcoin Core client tries to maintain 2 block-relay-only connections, 8 outbound-full-relay connections and a maximum of 125 connections in total.

ADDR Relay Mechanism

Due to Bitcoin's decentralised nature and P2P network, clients cannot simply look up the network addresses of all nodes that are currently online by querying a dedicated server. Instead, peers inform each other about network addresses of active nodes using so-called `ADDR` messages. Each such message contains up to one thousand network addresses of other Bitcoin clients. There are three principal reasons for sending an `ADDR` message:

1. **Advertising one's own address.** As clients want to make themselves known in the network, they periodically send their own network addresses in `ADDR` messages to their peers, which will relay them further into the network.
2. **Responding to a peer's request**. Clients can request a set of known network addresses from peers using `GETADDR` messages.
3. **Address gossip.** In the Bitcoin P2P network there is a relay mechanism for `ADDR` messages in place, which is designed to enable efficient, effective and secure propagation of network addresses.

When relaying contents from an `ADDR` message, clients have to follow a certain procedure to avoid network contention and giving away the identity of their peers as will be discussed in subsection 2.1.2. Algorithm 2.1 showcases an abstraction of a thread processing incoming and outgoing messages for each peer, including the relay procedure, as implemented by Bitcoin Core.

Lines 5 to 15 show the processing of a received `ADDR` message. If a message of such type is received from a peer that is not a block-relay-only peer, then for each address in the message the client will choose up to two peers to which to relay the address in a new `ADDR` message if the following criteria are met:

- The address' time stamp is less than 10 minutes into the past. Note that the data structure for network addresses in the Bitcoin P2P protocol contains a time stamp in order to determine its "age".
- The original `ADDR` message is not the response to a recent `GETADDR` request.
- The original `ADDR` message contains 10 or less addresses.

Algorithm 2.1: ADDR relay algorithm

```
1  while not interrupted do
2      foreach peer do
           // process one received message
3          message ← next message from message queue for incoming messages from
             peer;
4          if message is of type ADDR ∧ peer does not have connection type
             BLOCK_RELAY then
               // relay addresses in message
5              foreach address in message do
6                  save address as address known by peer;
7                  if address is younger than 10 minutes ∧ message is not a reply to a
                     GETADDR request ∧ message contains not more than 10 addresses
                     ∧ address is routable then
8                      n ← if address is reachable then 2 else randomly 1 or 2;
9                      candidates ← peers that are neither the originator of message or
                         have connection type BLOCK_RELAY;
10                     relay_peers ← deterministically random subset of candidates of
                         size min(|candidates|, n);
11                     foreach peer in relay_peers do
12                         push address onto the message queue for peer;
13                     end
14                 end
15             end
16         end
17         process message appropriately to its type;
           // send messages
18         if peer has connection type other than BLOCK_RELAY ∧ peer's Poisson
             process timer expired then
19             update Poisson process timer for peer;
20             message ← new ADDR message containing all addresses queued for
                 peer that are not believed to be known by peer;
21             send message;
22             save all addresses in message as known by peer;
23         end
24         send messages of other types;
25     end
26 end
```

- The address is deemed *routable*, i.e. it is a network address publicly routable on the global internet.

The client will try to choose exactly two peers to which to relay the currently processed address if it is *reachable*, randomly one or two otherwise. A network address is *reachable* to a client if the client is on the network to which the address belongs. For example, IPv6 or Onion addresses are not reachable to IPv4-only nodes. Possible candidates for relay are all peers that are neither the originator nor block-relay-only peers. Of these, one or two are chosen using a source of randomness that stays constant for 24 hours for any given address. Note that because of this, the same one or two relay peers will be chosen for any network address in a 24 hour window[3]. However, once a client sends a network address to a peer, the client will henceforth assume that this peer knows the address and will not send it again. After the appropriate peers have been selected, each address is pushed onto each of these peers' message queues.

Lines 18 to 21 show how queued addresses are collected and sent to peers. No address relay will take place for block-relay-only peers. Bitcoin Core does not immediately relay addresses from `ADDR` messages but retains them such that address relays mimic a Poisson process, where the number of relays for a given peer in a certain amount of time in seconds follows a Poisson distribution with rate parameter $\lambda = 1/30$. As a result there is on average one relay every thirty seconds from one peer to another. The inter-arrival times, i.e. the time spans between relays, follow an exponential distribution with the same rate parameter. This Poisson process is achieved by maintaining a timer for each peer that holds a timestamp indicating when the next `ADDR` message may be sent. As soon as there are network addresses to be relayed and the timer has expired, it is updated by adding a random value x to the current time where x stems from a random number generator producing exponentially distributed numbers with rate parameter $\lambda = 1/30$. Queued network addresses not believed to be known by the peer are then bundled into a new `ADDR` message and sent to the peer by another thread as soon as possible.

2.1.2 Topology Discovery

The *topology* of a network refers to the arrangement of its nodes and connections. A network's topology can be described on multiple levels of abstraction and by different metrics. For example, a network's topology could roughly be known by its class (*ring*, *star*, *mesh* topology, etc.), by its node degree distribution or by the exact adjacency matrix of the underlying network graph.

The Bitcoin P2P network protocol is designed to reveal as little information as possible about the topology of its network. Adversaries could use such information in order to partition the network [NAH15], perform eclipse attacks [DW13, HKZG15, NKMS16] or for deanonymisation [BKP14], i.e. linking public keys from transactions to IP addresses.

[3]The selection may change if the set of peers change but not necessarily.

Bitcoin's flooding and gossip mechanisms have been shown to be instrumental in inferring connections between nodes [BKP14, MLP+15, NAH15, NAH16, DSBPS+19]. As countermeasures, Bitcoin Core implemented exponential delays between relays of network addresses in 2015[4] to mitigate timing-based traffic analysis attacks and added the block-relay-only connections, which do not allow the relay of transactions and network addresses, in order to mitigate the risk of all peers of a node being found by an adversary in 2019[5] (refer back to 2.1.1 for details).

2.2 Mathematical Concepts

2.2.1 Graph Theory

Definitions and terminology within the field of graph theory are not entirely congruent among authors. In the following, we will present formulations that seem to be used in most publications around computer networks and further in modern textbooks on graph theory [DSS10, GYA18, BJG18].

In discrete mathematics a *graph* is given by a set of elements called *vertices* (also called *points* or—especially in the context of computer networks—*nodes*) that are put into pairwise relation by a set of so-called *edges* (also called *lines*). A graph $G = (V, E)$ is defined as an ordered pair consisting of a set of vertices V and a set of edges $E \subseteq [V]^2$, where $[A]^k$ denotes the set of all subsets of set A and natural number k. A *directed graph* or *digraph* has two additional mappings $init\colon E \to V$ and $ter\colon E \to V$ that determine the *initial vertex* and the *terminal vertex* of every edge [DSS10, GYA18]. Edges in directed graphs are also called *arcs* or *directed edges*. Alternatively to using *init* and *ter*, a directed graph can be given by a set of edges $E \subseteq V \times V$, where each edge $e = (u, v)$ is and ordered pair (tuple) of some initial vertex u and terminal vertex v [BJG18]. The latter will be the assumed definition of directed graphs for the remainder of this thesis.

Two vertices u, v are *adjacent* to each other if they are part of the same edge, i.e. if $\{u, v\} \in E$ for undirected graphs or $\{(u, v), (v, u)\} \cap E \neq \emptyset$ for directed graphs. Adjacent vertices are called each other's *neighbours*. A directed edge e with the same initial and terminal vertex is called a *loop*, i.e. $e = (v, v)$ for some vertex v. The *degree* (also called *valency*) of a vertex is equal to the number of its neighbours. In a directed graph the *in-degree* $\deg^+ v$ of a vertex v is equal to the number of edges where v is the terminal vertex, and the *out-degree* $\deg^- v$ is equal to the number of edges where v is the initial vertex. A *walk* of length k in a graph G is an alternating sequence $v_0e_0v_1e_1 \dots e_{k-1}v_k$ of vertices and edges from G with $e_j = v_jv_{j+1}$ for $j < k$. A *path* is a walk in which each vertex is unique, i.e. it occurs exactly once. Walks and paths may be called directed walks and directed paths, respectively, in the context of directed graphs. The *degree sequence* $(d_1, d_2, \dots, d_n)$ of a Graph G with n vertices is the sequence containing the degree of each vertex in G in non-decreasing order. For any given undirected graph

[4] https://github.com/bitcoin/bitcoin/pull/7125
[5] https://github.com/bitcoin/bitcoin/pull/15759

$G = (V, E)$ with $|E| = m$, the probability of any two distinct vertices $u, v \in V$ being adjacent is approximately given by [ZHK+23]

$$P(\{u, v\} \in E) \approx \frac{\deg u \deg v}{\sqrt{(\deg u \deg v)^2 + (m - \deg u - \deg v + 1)^2}}. \tag{2.1}$$

Graphs are a common abstraction for computer networks. The underlying graph of a network is given by the set of network nodes as the graph's set of nodes (or vertices) and by the set of (long-term) connections between nodes as the graph's set of edges. Typically these network graphs are undirected but may instead be directed in case the network protocol defines unidirectional connections. Note that as clients of most computer networks will not hold connections to themselves, analogously their underlying graphs will not contain loops.

2.2.2 Probability Theory

This section will briefly touch on terms and methods from probability theory and Bayesian inference required for understanding the remainder of this thesis. However, the scope of these prerequisites are limited to established concepts commonly discussed in textbooks on probability theory [BT08] and machine learning [GBC16].

Random Variables and Probability

We will first introduce the term *sample space* as any set that contains all possible outcomes of a certain statistical experiment, which can be any observable real-world process. A subset of the sample space is called *event*. An event *occurs* if the outcome of the experiment is an element of the event. The probability for an event A of some sample space $\Omega \supseteq A$ to occur is written as $P(A)$, where P is called a *probability measure*. A *random variable* X is a mapping from a sample space Ω to a *measurable space* E. For the sake of simplicity, we will let random variables always map to $\mathbb{R}$. The *distribution* P_X of X is defined as

$$P_X(A) \coloneqq P(X \in A) \tag{2.2}$$

where $X \in A$ is short for $X^{-1}(A) = \{\omega \in \Omega \mid X(\omega) \in A\}$. Note that we may also write $X = x$ instead of $X \in \{x\}$.

For example, when flipping a coin, one might interpret the outcome as a random variable on $\Omega = \{\mathrm{H}, \mathrm{T}\}$ with $X(\mathrm{H}) = 0, X(\mathrm{T}) = 1$ and thus $P_X(X = \mathrm{H}) = P_X(0) = P(\{\mathrm{H}\}) = 0.5$ as well as $P_X(X = \mathrm{T}) = P_X(1) = P(\{\mathrm{T}\}) = 0.5$.

The *cumulative distribution function* (CDF) of some random variable X is defined as:

$$F_X(x) \coloneqq P(X \leq x), x \in \mathbb{R} \tag{2.3}$$

A random variable X is called *discrete* if its sample space is finite or countably infinite, in which case its CDF is a step function. Its *probability mass function* (PMF) is then given by

$$p_X(x) = P(X = x), \tag{2.4}$$

which describes the probability of occurrence for each individual outcome $\omega \in \Omega$ with $x = X(\omega)$.

A random variable X is *continuous* if its CDF is a continuous function on $\mathbb{R}$. Its *probability density function* (PDF) is then given by

$$f_X(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_X(x), \tag{2.5}$$

which describes the probability density at point x. Note that for any continuous random variable X we have $P(X = x) = 0$. However, probabilities for X taking on a value in a given interval $(a, b]$ can be calculated as follows:

$$P(a < X \leq b) = P(X \in (a, b]) = F_X(b) - F_X(a) = \int_a^b f_X(t)\mathrm{d}t \tag{2.6}$$

Probability Distributions

There are multiple well-studied and common families of probability distributions that random variables may exhibit. For example, if the outcomes for some random variable X are equally likely to occur, it is said to follow a *uniform distribution* written $X \sim \mathrm{U}(a, b)$ with *population parameters* a, b being the limits of the image of X. Let X denote the number of pips shown by a die after it has been rolled then $X \sim \mathrm{U}(1, 6)$ with $p_X(x) = 1/6$.

A discrete random variable X follows a *Poisson distribution* $X \sim \mathrm{Pois}(\lambda)$ if its PMF is given by

$$p_X(k) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{2.7}$$

where k is the number of *arrivals* in a fixed-sized interval and $\lambda > 0$ is the *rate parameter*, which is equal to the PMF's mean, i.e. the expected value of the distribution. A Poisson-distributed random variable models a *Poisson process*. *Arrivals* in such a process are randomly distributed within their space and independent of each other. For example, the number of chewing gums on a stretch of pavement or the number of calls in a call centre within any constant time span may each form a Poisson processes.

The intervals between arrivals, called *inter-arrival times*, follow another fundamental distribution, called the *exponential distribution*. A continuous random variable X follows an exponential distribution $X \sim \mathrm{Exp}(\lambda)$ if its PDF is given by

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0, \\ 0, & \text{otherwise}, \end{cases} \tag{2.8}$$

where x is the interval between arrivals in the Poisson process and $\lambda > 0$ is the *rate parameter*. The mean or expected value of an exponential distribution is $1/\lambda$. Analogous to the above examples for Poisson processes, the distances between chewing gums along a pavement or the times between calls in a call centre may be exponentially distributed.

The sum of $k \geq 1$ independent exponentially distributed random variables each with rate parameter λ constitutes another continuous random variable that follows an *Erlang distribution*. A continuous random variable X follows an Erlang distribution $X \sim \text{Erl}(k, \lambda)$ if its PDF is given by

$$f_X(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}, \tag{2.9}$$

where k is the *shape parameter* and $\lambda > 0$ is the *rate parameter*. The mean or expected value of an Erlang distribution is k/λ. An Erlang distribution with constant shape parameter k may be referred to as an *Erlang-k distribution* with the rate as its only population parameter. For example, the PDF of an Erlang-2 distributed continuous random variable $X \sim \text{Erl-2}(\lambda)$ is given by

$$f_X(x) = \lambda^2 x e^{-\lambda x}. \tag{2.10}$$

Finally, a discrete random variable $X \sim \text{Hyp}(N, K, n)$ follows a *hypergeometric distribution* if its PMF is given by

$$p_X(k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}, \tag{2.11}$$

where $N \in \mathbb{N}^*$ is the *population*, $K \in \mathbb{N}^*, K \leq N$ is the number of *success states*, $n \in \mathbb{N}^*, n \leq N$ is the number of *trials* and $k \in \mathbb{N}^*$, is the number of *successes* among trials. The hypergeometric distribution can be illustrated by the following urn experiment. An urn contains N balls (*population*), of which K are red (*success states*) and $N - K$ are black. If n balls (*trials*) are randomly drawn from the urn without replacement then the number k of red balls drawn (*successes*) follows a hypergeometric distribution.

Point Estimation

In some real-world applications the family of the probability distribution of some random variable of a system may be known (or assumed) but not the exact population parameter(s). When sample data are available, the unknown parameters can be estimated using a

method called *point estimation*. Based on the sample data, a "best guess" for the parameter, called *point estimate* (or simply *estimate*), is calculated.

One of the most basic forms of point estimation is known as *maximum likelihood* (ML) estimation, in which the *maximum likelihood estimate* of the unknown parameter is the value that makes the sample data "most likely", which is to say it maximises a *likelihood* as defined by a *likelihood function*.

Let $X \sim \mathrm{D}(\theta)$ and $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be a sequence of realisations (sample data) of the random variable X. Then the likelihood function for the population parameter $\theta \in \Theta$ for a parameter space Θ is defined as

$$\mathcal{L}(\theta \mid \mathbf{x}) = \prod_{i=1}^{n} f_X(x_i \mid \theta), \tag{2.12}$$

where $f_X(x \mid \theta)$ is the PMF (if X is discrete) or PDF (if X is continuous) of X dependent on the population parameter θ. The maximum likelihood estimate $\hat{\theta}_{\mathrm{ML}}$ is then the population parameter under which the likelihood is maximal:

$$\hat{\theta}_{\mathrm{ML}} = \arg\max_{\theta \in \Theta} \mathcal{L}(\theta \mid \mathbf{x}) \tag{2.13}$$

Note that the likelihood $\mathcal{L}(\theta \mid \mathbf{x})$ does not give the probability of θ being the true value of the unknown population parameter related to sample $\mathbf{x}$. In this case it only provides a means of comparison between population parameters in order to find the most "likely" or "plausible" one given the sample data and assumed distribution family. However, using Bayes' theorem, the so-called *a posteriori probability* or simply *posterior* can be calculated from a *prior* probability distribution on the parameter space and sample data, also called *evidence*. Given the prior PMF g on the parameter space Θ and evidence $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ from the random variable X with probability distribution $f_X(x \mid \theta)$, the posterior probability distribution h is defined as

$$h(\theta \mid \mathbf{x}) = \frac{f_X(\mathbf{x} \mid \theta) g(\theta)}{f_X(\mathbf{x})} = \frac{\prod_{i=1}^{n} f_X(x_i \mid \theta) g(\theta)}{\sum_{\theta' \in \Theta} \prod_{i=1}^{n} f_X(x_i \mid \theta') g(\theta')}. \tag{2.14}$$

Analogously to ML estimation, the *maximum a priori* (MAP) estimate is the value for the parameter θ that maximises its posterior probability given the evidence. It is given by

$$\hat{\theta}_{\mathrm{MAP}} = \arg\max_{\theta \in \Theta} \mathcal{L}(\theta \mid \mathbf{x}) g(\theta). \tag{2.15}$$

Binary Classification

Binary classification refers to the task of labelling elements of a set with one of two labels, thereby dividing them into two *classes*. Typically, the objective is to predict which class any given instance *truly* belongs to. Most commonly, binary classification problems are expressed as yes-no questions concerned with whether an instance of a population is truly of a given type or not. In these cases, the *condition* of an instance is either "positive" or "negative" and a function called the *binary classifier* tries to predict the condition of a given instance. Common examples for binary classification include medical tests and spam detection.

We refer to instances where both condition and prediction are positive as *true positives*. When both condition and prediction are negative, they are referred to as *true negatives*. *False negatives* are positive instances (falsely) predicted to be negative and *false positives* are negative instances (falsely) predicted to be positive.

Commonly used metrics to assess the quality of a binary classifier in terms of the above outcomes are *precision* and *recall*, defined as

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.16}$$

and

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.17}$$

where TP is the number of true positives, FP is the number of false positives and FN is the number of false negatives. Note that TP + FP equals the number of positive predictions and TP + FN equals the number of positive instances. Therefore, the measure of precision is the fraction of positive predictions that are correct and the measure of recall is the fraction of positive instances that are correctly predicted. If both of these measures are valued equally, then their harmonic mean called F_1 *score* defined as

$$\text{F}_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.18}$$

may serve as a single value to represent the quality of a classifier.

2.3 Related Work

A number of methods for topology discovery to various extents have been presented in the past. However, some have been mitigated by Bitcoin Core developers through implementation of countermeasures, and some require transmitting valid transactions, which renders them costly and impractical.

In 2014 Biryukov et al. [BKP14] showed how an attacker can deanonymise Bitcoin users, i.e. link a public key from a recent transaction to the network address of the node that originally issued the transaction. To this end they also established methods to find links between nodes and estimate the number of connections an individual client maintains at a specific point in time. In their paper they showed that links can be found by listening for `ADDR` messages broadcast by nodes which recently joined the network in order to announce their presence. Another method would be to send a set of marker addresses to a node A and then checking—by use of `GETADDR` messages—if the portion of marker addresses known by some other node B corresponds to the number of peers of A. The latter number can be estimated by sending a set of marker addresses to A and recording the number of marker addresses directly relayed back to the attacker. From this number the most likely peer count can be estimated. However, since then the behaviour of Bitcoin Core has changed to a point where these methods are no longer applicable[6,7].

Miller et al. [MLP+15] published a paper on discovering the public Bitcoin P2P network topology and influential nodes in 2015. Using their infrastructure *CoinScope* in tandem with their topology discovery technique *AddressProbe* they showed that only about 2 % of nodes represented three-quarters of the mining power at the time. AddressProbe exploited a mechanism in Bitcoin Core where time stamps of (propagated) addresses enabled the reconstruction of the relay path, ultimately allowing for inferring links between reachable nodes. This behaviour has since been patched[8], making AddressProbe as described in that paper no longer applicable.

In 2016 Neudecker et al. [NAH16] presented a topology inference model for flooding P2P networks. They applied this method to the Bitcoin P2P network by analysing observed propagation delays of `INV` messages. Using an analytical propagation delay model, they formulated the probability distribution of shortest path lengths between two peers v_1, v_2 given a set of observed time spans between sending `INV` messages to v_1 and receiving them from v_2. Thus, under the assumption of correctness of their model, the probability of connection between any two nodes can be calculated. They validated their findings with their own ground truth nodes in the real Bitcoin network and achieved both recall and precision over 40 %. However, these findings only apply to a previous version of Bitcoin Core when message propagation did not mimic a Poisson process[9].

Grundmann et al. [GNH19] described two approaches for topology inference in 2019. The first one exploits the accumulation of transaction records during transaction gossip but proved costly and performed poorly in experimental validation. The second one uses the fact that clients will drop recently received transactions that conflict with earlier ones, i.e. double spending transactions. In an experimental setup the authors could achieve a recall of 87 % and a precision of 71 % for a total cost of 99 transaction fees.

[6] `https://github.com/bitcoin/bitcoin/pull/18991`
[7] `https://github.com/bitcoin/bitcoin/pull/19763`
[8] `https://github.com/bitcoin/bitcoin/pull/5860`
[9] `https://github.com/bitcoin/bitcoin/pull/7125`

In the same year Delgado-Segura et al. [DSBPS+19] presented their topology reconstruction technique *TxProbe*. It leverages the way Bitcoin Core handles *orphan transactions*, which are transactions whose input transactions have not (yet) arrived. The authors reported precision and recall of their technique to be over 90 %. However, as they could not conclusively rule out that TxProbe may disrupt the network, experiments have only been conducted on the Bitcoin testnet. This paper motivated the Bitcoin Core developers to implement the *block-relay-only* connection type, which does not allow for transaction and network address gossip[10].

In 2021 Grundmann et al. [GBH21] used data gathered from a spam wave of `ADDR` messages from an unknown actor in order to calculate the number of neighbours of each connected peer. They based their calculations on the equations described in the 2014 paper by Biryukov et al. [BKP14] and validated their findings using ground truth nodes. This way they were able to reconstruct the node degree distribution of reachable peers in the Bitcoin P2P network. Note that any future such `ADDR` wave will not have the same effect due to a network address rate limit added after this incident[11].

[10] `https://github.com/bitcoin/bitcoin/pull/15759`
[11] `https://github.com/bitcoin/bitcoin/pull/22387`

CHAPTER 3

Methodology

The method of topology inference in the Bitcoin P2P network will be presented in three sections. First, section 3.1 will describe how researchers or adversaries can gather samples of messages on which traffic analysis will be performed. Second, section 3.2 will map the network and its behaviour to a formal model, allowing for a mathematical description and analysis of node degree estimation and connection inference. Third, section 3.3 will explain the purpose, design and functionality of the computer simulation that has been written in order to simulate the process of data collection and topology inference.

3.1 Data Collection

Since our technique for topology discovery will be a kind of active traffic analysis using only the Bitcoin P2P protocol, we will have to connect our own clients to the existing network, which have to be capable of at least the relevant subset of the protocol. The data to be analysed consist solely of sent and received `ADDR` messages. Nodes controlled by the researcher or adversary performing the task of data collection are called *monitor nodes*. These are responsible for sending fake but reachable[1] so-called *marker addresses* to the *target* node, which is the Bitcoin network node of which peers and the number thereof are to be inferred.

We call the sending of a marker address by a monitor to a target an *injection*. After such injection, the target will relay the marker address to two peers as a part of the gossip protocol. The address may thus multiply and spread through the network along what we will call *relay paths* where each node in a path is known as a *hop*. Eventually, a marker address may *recur* when a Bitcoin node receives this address and relays it to a monitor. Every recurrence yields knowledge of the first (the target) and last hop as well as the delay between injection and recurrence since we know what unique marker

[1] as defined in 2.1.1

address has been injected to which target and which Bitcoin node has finally sent it back to a monitor. Note that when the target relays an injected marker address directly back to another monitor node, it is both the first and last hop in the relay path.

Monitor nodes could be configured to accept inbound connections in order to attempt to discover at least part of the peers of an otherwise unreachable Bitcoin node. However, it cannot be guaranteed that a node will open a full-relay connection to a monitor. While a Bitcoin Core client announces whether it seeks to establish a block-relay-only or a different type of connection, monitor nodes are assumed to ignore all incoming connection requests in this thesis.

3.2 Mathematical Description

3.2.1 Network Model

The Bitcoin P2P network can be modelled as a directed graph $G = (V, E)$, where the set of vertices V represents the set of reachable[2] network nodes and the set of directed edges E represents the set of full-relay connections between pairs of network nodes. It holds that $(u, v) \in E$ if and only if node u maintains an outbound-full-relay connection to node v and consequentially node v maintains an inbound connection to node u. Since our method of inferring connections between nodes cannot determine their directions, we will use the underlying undirected graph as the Bitcoin network model, i.e. where $\{u, v\} \in E$ if and only if u maintains any type of full-relay connection to v for all pairs of nodes $u, v \in V$.

Recall that a standard Bitcoin client will relay addresses received through `ADDR` messages to peers over potentially any type of connection except for the block-relay-only connection type. Given that our probing and inference method relies on precisely this network address relay mechanism and block-relay-only connections play no part in network address gossip, there is currently no hope of finding block-relay-only links. Therefore, these connections are not part of the network model and are not depicted in the graph abstraction of the Bitcoin P2P network.

From the definition of the network graph model it follows that the degree $\deg v$ of a node v is equal to the number of its full-relay connections minus the number of inbound connection where the peer holds a block-relay-only connection to v. However, as monitor nodes and their connections will be introduced in the course of data collection, these have to be considered as well. Formally, the set of network nodes V is extended by the set of monitor nodes and the set of network connections E is extended by all connections that involve a monitor node. The number of monitor nodes to which any Bitcoin node v is connected will be denoted by $\deg^m v$. Note that $\deg v$ will continue to refer to the degree of v within the graph excluding monitor nodes and edges, because ultimately only this degree is what attackers and researchers are trying to infer and not the number

[2] *reachable* in the sense that their network addresses are known and that they are not configured to reject incoming connection requests

of connections they themselves established. However, as will prove practical in later sections, we will introduce the *relay degree* $\deg^r v = \deg v + \deg^m v$ of a node v denoting the number of all potential candidates for network address relay, namely the number of peers excluding block-relay-only peers.

While useful for the purposes of this thesis, our network model comes with several limitations. Firstly, it is consistent and entirely static, whereas the real Bitcoin P2P network is of dynamic nature with network nodes joining and leaving constantly. However, research has shown that the churn rate among reachable nodes is moderately high and relatively stable at around 5 to 6 % per day [ECP21]. Secondly, tying in with the previous point, the short-lived feeler and addr-fetch connections are not differentiated from outbound-full-relay connections. Typically, when it comes to the Bitcoin network topology, attackers and researchers are mostly concerned with long-standing connections, i.e. outbound-full-relay, block-relay-only and manual connections as well as their symmetric inbound counterparts. Lastly, we assume every node in the network to behave like Bitcoin Core 24.1 in all aspects relevant to this study.

3.2.2 Relay Model

In the attempt to find both the number and the identity of peers of a targeted Bitcoin network node, the `ADDR` relay mechanism as implemented by Bitcoin Core serves as the central source of information. Whenever a monitor node sends an `ADDR` to a targeted Bitcoin node, there are two types of random variables that can be observed: 1. the number of addresses that the target relays directly to other monitors and 2. the delay between injection and recurrence of each marker address. Note that some marker addresses may never recur and that the aforementioned delay is only relevant if a monitor receives the marker address by a node other that the target.

Degree Estimation

Probabilistically finding the degree of a node in the network graph—in other words finding the number of peers of a reachable Bitcoin client—can be performed by point estimation. To this end, an attacker gathers a sample of data by repeatedly sending `ADDR` messages containing unique marker addresses to the targeted node and observing part of its relay behaviour. The number of peers of this client is a parameter of the distribution of the values in the collected sample. Finally, the attacker can estimate the degree of the node, in other words the number of peers, using ML or MAP estimation on the sample.

Let v be the target node and let it be connected to at least two monitor nodes, i.e. $\deg^m v > 1$. Recall that upon receiving a message of type `ADDR`, a Bitcoin Core client will take part in network address gossip by relaying the contained addresses to a subset of its peers, provided the preconditions as described in 2.1.1 are met. The implemented relay mechanism causes the client to randomly choose $\min(\deg^r v - 1, 2)$ full-relay peers for relay of each address contained in that message excluding the originator of the message.

Note that the number of connections $\deg v$ maintained by any Bitcoin Core instance v is never meant to be below 8. When a Bitcoin Core process starts, it tries to establish 8 outbound-full-relay connections as soon as possible. Whenever an outbound-full-relay peer disconnects, the client tries to find another node to which such a connection can be established. It can also reasonably be assumed that any node accepting two monitor connections has at least one other full-relay connection (and likely many more). Therefore, for the purposes of this thesis $\min(\deg^r v - 1, 2) = 2$ can reasonable be assumed for any reachable node v even if only two monitors are connected, i.e. $\deg^m v = 2$.

After the client has chosen the relay peers for a certain address, it proceeds to schedule and send the address enclosed in an `ADDR` message unless it thinks the chosen peer knows of the address already. The latter can be the case if the client has recently received that address from the chosen peer or if the client has recently sent that address to it.

Assume one of the monitor nodes injects a sequence of `ADDR` messages containing fresh network addresses into a target node v. The higher the fraction of monitors among the full-relay peers of the target, i.e. the higher the ratio $\deg^m v : \deg^r v$, the higher the average portion of monitor clients among the relay peers chosen by v. Put more precisely: Let the discrete random variable X denote the number of monitor clients among the peers chosen for relay by v for an address that is to be relayed. Then X follows the hypergeometric distribution

$$X \sim \mathrm{Hyp}(\deg^r v - 1, \deg^m v - 1, \min(\deg^r v - 1, 2)), \tag{3.1}$$

where the distribution's *population* $\deg^r v - 1$ is the number of eligible peers, the number of *success states* $\deg^m v - 1$ is the number of monitors connected to v excluding the injecting monitor, and the number of *trials* $\min(\deg^r v - 1, 2)$ is the number of chosen relay peers, which is 2 in all relevant cases. Therefore, the probability mass function for X parameterised by the degree of the target $\deg v = d$ (for $d \in \mathbb{N}^*, d > 1$) is given by

$$p_X(k \mid d) = \frac{\binom{\deg^m v - 1}{k}\binom{d}{2-k}}{\binom{\deg^r v - 1}{2}}, \tag{3.2}$$

where $\deg^r v = d + \deg^m v$ is dependent on the assumption $\deg v = d$. With a sample $S = (k_0, k_1, \ldots, k_n)$ of n observations, where each element is a realisation of X, i.e. the number of marker addresses directly relayed to another monitor following an injection, the ML estimate $\hat{d}_{\mathrm{ML}}$ is given by

$$\hat{d}_{\mathrm{ML}} = \arg\max_{d \in \mathbb{N}} \sum_{i=0}^{n} \log p_X(k_i \mid d). \tag{3.3}$$

Under the assumption of a prior distribution of node degrees in the Bitcoin network, the MAP estimate can be calculated instead. Let $p_d(d)$ be the probability mass function of the prior distribution of node degree d then the MAP estimate $\hat{d}_{\mathrm{MAP}}$ is given by

$$\hat{d}_{\text{MAP}} = \arg\max_{d \in \mathbb{N}} \prod_{i=0}^{n} p_X(k_i \mid d) p_d(d). \tag{3.4}$$

The prior degree distribution may be taken from the work of Grundmann et al. [GBH21] (see 2.3).

Edge Prediction

Predicting connections between any two reachable clients is done similarly to estimating a client's number of peers: First sample data are collected by injecting marker addresses and listening for their recurrences. Then ML or MAP estimation is performed using the sample data as evidence. However, in this case timing data are extracted from the sample and recurrences are only accepted from nodes other than the target node itself. Also, instead of *point estimation*, the task of finding out whether $\{u, v\} \in E$ or $\{u, v\} \notin E$ for the network graph $G = (V, E)$ is more commonly called *binary classification* and the function trying to predict if $\{u, v\} \in E$ on the basis of collected data samples is called a *Bayes classifier*.

In order to maximise the number of identified peers of a target node, the monitor nodes have to be connected to as many reachable nodes as possible, since without any prior knowledge any reachable node could be a peer of the target. Every time a monitor sends a marker address to a target, the latter has the option to choose a monitor or a non-monitor peer for each of the two relays. When it chooses a monitor, this information can be used for degree estimation as described in 3.2.2 but not for edge prediction. If it chooses a non-monitor peer, there are again two options. The peer could either relay the marker address to a monitor or relay it further into the network. If the peer chooses the second option, the marker address may take a path of several nodes through the network before finally ending up at a monitor if it recurs at all. In both cases the delay between injection and recurrence of this marker address can be recorded for each pair of target node and final hop and used as evidence for the classifier. In theory if target u and node v are connected, a certain fraction of marker addresses are bound to be relayed from u over v to a monitor. The delays exhibited by relays along this path follow a known distribution that could not be observed if u and v were not connected.

As previously mentioned, a Bitcoin Core instance does not immediately relay network addresses from received `ADDR` messages. Instead, it holds a timer for each peer and only if the timer expires, scheduled network addresses are bundled into an `ADDR` message (or multiple messages if more than 1,000 addresses need to be sent) and relayed. After every such expiry, the timer is updated by a number drawn from an exponential distribution with mean $\lambda = 30$ seconds. This causes the delays for individual relays between any two peers to follow the same exponential distribution.

Let u be the target node and v be one of its peers, i.e. $\{u, v\} \in E$ for network graph $G = (V, E)$. Further, let monitor node m be connected to target u and monitor node

m' be connected to node v, i.e. $\{(m,u),(m',v)\} \subset E$. We will denote the fact that an injection has caused a relay path (m,u,v,m') (called *direct path*) by $u \to v$ and we will denote the fact that an injection has caused a relay path with at least one extra hop between u and v (called *indirect path*) by $u \nrightarrow v$.

Let X be the random variable denoting the delay δ between injection into target u and recurrence from node $v \neq u$ of a marker address within the network graph $G = (V, E)$. The delay observed from a direct path, in other words the delay between the target node receiving a marker address, relaying it to a peer and that peer relaying it to a monitor, is the sum of two such exponential distributions known as an *Erlang-2* distribution. Therefore, X under the condition $u \to v$ follows the Erl-2(30) distribution with probability density function

$$f_X(\delta \mid u \to v) = \lambda^2 \delta e^{-30\delta}. \tag{3.5}$$

The probability density function for X under the sole condition of the target and the final hop being connected is the sum of the probability density for the delay under condition of a direct relay path multiplied by the prior probability of the relay path being direct and the probability density for the delay under condition of an indirect relay path multiplied by the prior probability of the relay path being indirect:

$$f_X(\delta \mid \{u,v\} \in E) = f_X(\delta \mid u \to v)P(u \to v) + f_X(\delta \mid u \nrightarrow v)P(u \nrightarrow v) \tag{3.6}$$

Given a set of observed delays between a target and a potential peer, the likelihood of these nodes being full-relay peers is determined by the equality

$$\mathcal{L}(\{u,v\} \in E \mid \Delta) = \prod_{\delta \in \Delta} f_X(\delta \mid \{u,v\} \in E) \tag{3.7}$$

and analogously the likelihood of them not being full-relay peers by

$$\mathcal{L}(\{u,v\} \notin E \mid \Delta) = \prod_{\delta \in \Delta} f_X(\delta \mid \{u,v\} \notin E). \tag{3.8}$$

Using these likelihoods we can now calculate the posterior probability for connection under the assumption of some prior probability $P(\{u,v\} \in E)$ for nodes u and v to be connected using

$$P(\{u,v\} \in E \mid \Delta) = \frac{\mathcal{L}(\{u,v\} \in E \mid \Delta)P(\{u,v\} \in E)}{P(\Delta)}, \tag{3.9}$$

where

$$P(\Delta) = \mathcal{L}(\{u,v\} \in E \mid \Delta) P(\{u,v\} \in E) + \mathcal{L}(\{u,v\} \notin E \mid \Delta) P(\{u,v\} \notin E). \tag{3.10}$$

Finally, we can define the Bayes classifier that labels a pair of nodes given the corresponding data set as either "positive" (i.e. connected) or "negative" (i.e. disconnected). Additionally, we parameterise it by a *confidence* or *certainty threshold* t, which may let us trade precision for recall and vice versa:

$$C(\Delta_{u,v}; t) = \begin{cases} \text{“positive”} & \text{if } P(\{u,v\} \in E \mid \Delta_{u,v}) > t \\ \text{“negative”} & \text{otherwise} \end{cases}. \tag{3.11}$$

Note that while a threshold of 50 % should theoretically minimise the chance of misclassification, it might not lead to an optimal balance between precision and recall as will be shown in Chapter 4.

If no prior probability for connection is known, we can simply assume the existence of a connection to be as probable as its absence and hope that a large enough sample Δ can be gathered to make accurate predictions nonetheless. This is in line with the frequentist interpretation of statistics and leads to what we will call the *frequentist classifier*. If, however, we can assume a prior connection probability and incorporate it into the posterior calculation, then we might achieve better prediction quality while requiring less data. Using this informed prior distribution within the Bayesian interpretation of statistics, we arrive at the true *Bayes classifier*. Note that strictly speaking our frequentist classifier is a Bayes classifier as well but we chose this terminology to refer to the fact that it does not make use of an informed prior probability.

The probability densities $f_X(\delta \mid u \nrightarrow v)$ and $f_X(\delta \mid \{u,v\} \notin E)$ as well as the probabilities $P(u \to v)$ and $P(u \nrightarrow v)$ will not be analytically derived but extracted from simulations described in 3.3. The prior edge probability $P(\{u,v\} \in E) = 1 - P(\{u,v\} \notin E)$ is approximately given by [ZHK+23]

$$P(\{u,v\} \in E) \approx \frac{\deg u \deg v}{\sqrt{(\deg u \deg v)^2 + (|E| - \deg u - \deg v + 1)^2}}. \tag{3.12}$$

The degrees of u and v can either be inferred as described in 3.2.2 or assumed to be the average degree determined by Grundmann et al. [GBH21].

Before moving on to the next section, it should be noted that confounding factors like CPU time and network latency are not taken into account. Instead, the entire time span between sending an `ADDR` message and receiving a marker address from that message is assumed to be wholly due to delays deliberately introduced through the implementation of the relay mechanism. While part of these confounders could possibly be accounted for in the classification process, we will view them as negligible considering how short these delays typically are compared the client's intentionally caused delays.

3.3 Simulation

In the course of this thesis a simulation of the Bitcoin network's behaviour regarding the ADDR gossip protocol has been developed. Specifically, it simulates the data collection process described in 3.1 and provides samples structurally identical to what one might expect from an experimental setup involving the actual Bitcoin network. All the relevant data structures and functions from the C++ source code of Bitcoin Core have been adapted and transferred into the source code of the simulation. The software is written in Python 3.11 using the popular packages NumPy, SciPy, Pandas and NetworkX.

The purpose of this endeavour is threefold:

1. It is used to generate empirical probability distributions for $f_X(\delta \mid u \nrightarrow v)$ and $f_X(\delta \mid \{u, v\} \notin E)$ as well as the empirical prior probabilities for $P(u \to v)$ and $P(u \nrightarrow v)$ in order to perform edge prediction.

2. It provides a way to evaluate and adapt our methods without having to interact with and potentially disturb the real Bitcoin network.

3. It serves as a proof of concept for our presented method.

First, a directed graph data structure representing the Bitcoin P2P network is generated. Its vertex set represents all Bitcoin clients and monitor nodes and its edge set represents outbound-full-relay and block-relay-only connections between the network nodes. Although the simulation can be performed on almost any directed graph without loops, we evaluated our methods on two base graphs of particular interest as described in 3.3.1. Then ADDR gossip is initiated by having the monitor nodes inject marker addresses into the network. Each client emulates the relay behaviour of Bitcoin Core and thus a cascade of message relays along the edges of the network is triggered. During this stage all message data are recorded for later evaluation. Once the simulation has concluded, the raw data are compiled and stored to disk. These traffic data can then be used for node degree estimation and edge prediction as if collected from the real Bitcoin network. The predictions are validated against ground truth knowledge on the simulated network topology.

Note that this approach inherits all limitations discussed in the prior section of this chapter. Most notably, the network is assumed to be completely static, i.e. the set of nodes and the set of edges remains constant, and no message exchange is simulated that is not originally caused by a monitor node. Furthermore, the topology and behaviour of the simulated network may considerably deviate from the real Bitcoin network. Either refining the network model and with it the simulation or further analysing the implications of these simplifications is outside the scope of this thesis and left for future research.

3.3.1 Graph Generation

For the purposes of this thesis, ADDR relay simulations have been performed on the basis of two different graph generation procedures. Although these methods only differ in the number of unreachable nodes and the distribution of connections between the reachable and unreachable portion of the network, they lead to noticeably different results as will be discussed in chapter 4.

The employed graph generation procedures are based on two different sources of prior knowledge on the Bitcoin network. One of them (used in the *random attachment model*, see section 4.1) replicates the number of reachable and unreachable nodes as reported by Bitnodes[3], whereas the other (used in the *degree distribution model*, see section 4.2) tries to match the node degree distribution within the reachable portion of the network as has been reported by Grundmann et al. [GBH21] in 2021.

Both graph generation procedures start by creating the reachable portion of the network as a *random uniform k-out graph* without loops where $k = 10$ under the constraint that $\deg v \leq 125$ for all reachable nodes v. The value $k = 10$ and the constraint $\deg v \leq 125$ are chosen in analogy to Bitcoin Core, which requires maintaining exactly 2 block-relay-only and exactly 8 outbound-full-relay connections, and allowing no more than 125 connections in total. The number of reachable nodes is set to 17,000, which is roughly what Bitnodes reports at the time of this writing [bit]. 10 reachable nodes are chosen uniformly at random to represent the target nodes, of which degrees and neighbours are to be inferred.

The next step is to add unreachable nodes to the current graph of reachable nodes. This is done in one of two ways. The first one is to add 28,000 nodes—as is again reported by Bitnodes—and then randomly pick 10 reachable nodes as outbound peers for each unreachable node. These kinds of graphs define what will be called the *random attachment model*. The second way to add and connect unreachable nodes is to first determine the number of unreachable nodes under the assumption of each one having exactly 10 outbound peers and requiring the degree distribution among reachable nodes as estimated by Grundmann et al. [GBH21], and then to randomly choose the outbounds for each unreachable node without breaking these premises. Graphs computed with this methods define the *degree distribution model*.

Next, two outbound connections of each node will be determined to be of type block-relay-only. Finally, 20 monitor nodes are added to the graph and each of them will be connected to as many reachable nodes as possible without exceeding the connection limit in the simulated Bitcoin clients. The number of connections that each monitor node can hold is not limited in this simulation.

[3] https://bitnodes.io/

CHAPTER 4

Results

This chapter will present the results from our simulations using two different network graph models, called *random attachment model* and *degree distribution model*, which will be discussed separately in sections 4.1 and 4.2, respectively. Our degree estimation and edge prediction methods have been evaluated under each model using leave-one-out cross-validation (LOOCV) on 10 samples. During LOOCV each sample is chosen to be the *validation set* once, leaving 9 samples as *training sets*. The latter provide relay duration distributions for both connected and disconnected pairs of nodes, such that the likelihood and posterior probability for connection between target and some monitored node given their set of relay durations can be computed. These values represent the classifier's *confidence* in two nodes being adjacent given the collected evidence. Note that degree estimation does not require a training phase since the relationship between the observed relay behaviour and the degree of a given node is assumed to be exactly as described in 3.2.2. Results are averaged over all validations for each model.

4.1 Random Attachment Model

In the *random attachment model* the simulation is run on a network graph consisting of 17,000 reachable and 28,000 unreachable nodes connected at random. Refer to 3.3.1 for details on the generation procedure. This type of graph represents a simple and predictable yet likely naïve abstraction of the Bitcoin P2P network. In the following we will outline the composition of the generated network, statistics about the simulated network traffic, and present the cross-validation results.

4.1.1 Network and Traffic Statistics

Figure 4.1 shows the average degree distribution among all reachable nodes and target nodes. Because the reachable portion of the graph is generated as a random uniform

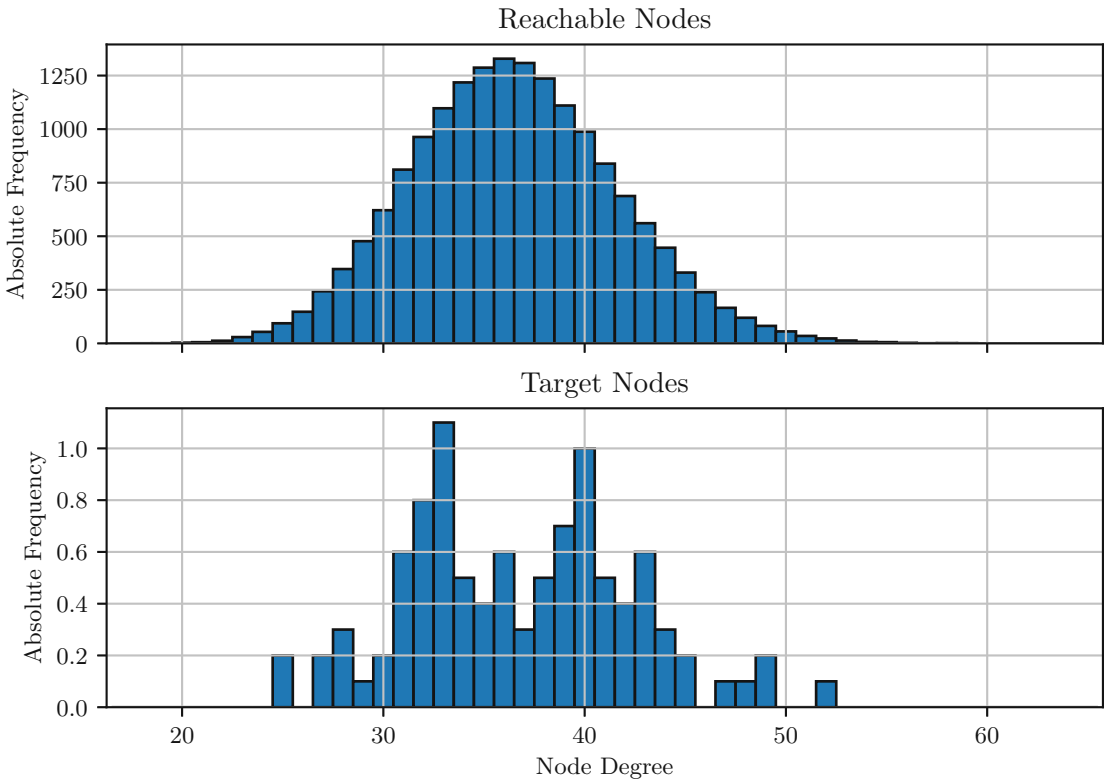

Figure 4.1: Degree frequency distribution among reachable and target nodes

10-out graph, the corresponding node degree distribution resembles a balanced bell shape with mean and standard deviation of around 36.47 and 5.15, respectively. The highest recorded node degree was 65, which is why all 20 monitors could connect to every reachable node without exceeding the connection limit of 125. Although the shape of the rather uneven degree distribution among the target nodes differs significantly from the previous bell-curve distribution, the target node degrees exhibit similar statistics with mean and standard deviation of 36.80 and 5.55, respectively.

Recall that a Bitcoin Core client refuses to relay the addresses in a previously received `ADDR` message if the peer which sent the message has exceeded the rate limit or if the encompassing `ADDR` message contained more than 10 addresses. These mechanisms are relevant to our research as they restrict the `ADDR` message gossip necessary for data collection. In order to avoid network congestion, wasting marker addresses and possibly skewing training data, a dedicated monitor node injects marker addresses into all targets in a round-robin fashion, waiting 0.75 seconds between any two consecutive injections. This delay has been chosen because each node is assumed to require 30 seconds per relay to one full-relay peer on average. Since each node is also assumed to have at least 8 full-relay peers and to relay each address to 2 full-relay peers in parallel, the minimum required injection delay for a single target amounts to $\frac{2 \cdot 30}{8} = 7.5$ seconds. Interspersing

Table 4.1: Number of received and dropped marker addresses

	Received	Rate Limited	Payload Too Large
Per Client	1,086	0.049	0.117
Total	48,888,846	2,210	5,247

Table 4.2: Number of injected and recurred marker addresses

Period [min]	Injected markers	Recurred markers
15	1,200	1,295,602
55	4,400	7,901,036
95	7,600	14,100,711
135	10,000	20,065,056

injections for 10 targets thus yields the aforementioned delay of 0.75 seconds.

Table 4.1 shows the average number of addresses transmitted to network clients and how many of those were not relayed due to either the rate limit or message size limit with regards to `ADDR` gossip. With less than one thousandths of addresses left unrelayed, it can be said that in these simulations network traffic contention was low. However, since no "background" `ADDR` gossip was simulated and instead only the one introduced by monitor nodes using marker addresses, the inter-injection interval of 0.75 seconds cannot be deemed sufficient to prevent reaching the `ADDR` message throughput capacity of the real Bitcoin network.

In order to assess how the amount of available data affects the quality of degree estimation and edge prediction, each validation was performed four times, using either only the first 15, 55, 95 or 135 minutes of measurement data. Table 4.2 lists the average number of injected and the average number of recurred marker addresses within each period. We can infer that a single injected marker address may multiply during `ADDR` gossip so that on average around 2,000 copies travel through the network and end up at monitor nodes. After the initial 15 minutes of data collection, each additional 40 minutes yielded between roughly 66 to 60 million additional recurrences, diminishing with increased simulation time.

Note that not every marker address which found its way back to a monitor node corresponds to exactly one data point for both degree estimation and edge prediction. In the case of degree estimation, each injection maps to exactly one data point, namely the number of monitor nodes—either 0, 1 or 2—among the peers which the target determined to be the relay recipients. In the case of edge prediction, every recurrence of a marker address where the node that relayed the marker to a monitor is not also the target into which the marker has originally been injected corresponds to one data point, namely the delay between injection and recurrence of this marker address. Therefore no marker

Table 4.3: Number of datapoints for degree estimation and edge prediction

	Degree Estimation		Edge Prediction	
Period [min]	Per Instance	Total	Per Instance	Total
15	120	1,200	7	1,294,677
55	440	4,400	46	7,898,189
95	760	7,600	82	14,095,966
135	1,000	10,000	118	20,058,859

address recurrence can be part of both degree estimation and edge prediction input data. Table 4.3 gives the average number of data points for both inference techniques by measurement period. In the context of degree estimation the word "instance" refers to the target nodes, while in the context of edge prediction it refers to pairs of nodes where one of them is a target node and the other is any monitored client.

Part of the purpose of this `ADDR` gossip simulation was to study the distribution of time spans between injection of a marker address into a target peer and recurrence of the same address via some other observed node dependent on whether these two nodes are neighbours through a full-relay connection or not. Figure 4.2 shows the Gaussian kernel density estimates (KDEs) on different subsets of the relay duration data gathered from the simulations. The plot at the top shows the KDEs of the delays separated by whether they stem from pairs connected by full-relay connections or not, whereas the bottom plot shows the KDEs of the delays recorded from full-relay connected pairs, split by relay path length. Recall that a direct relay path contains exactly two relays and implies a full-relay connection while an indirect relay path does not.

Notice how the relay durations produced by full-relay connected pairs exhibit a discernibly different distribution than the one produced by pairs that are either disconnected or connected via a block-relay-connection. Comparing direct and indirect recurrences from full-relay connected pairs of nodes, it becomes intuitively clear that the difference we observed in the densities above are due to the portion of relay paths taking the direct route back to a monitor node without any nodes in between the observed pair. As expected, the KDE of durations from direct paths approach the probability density of the Erlang-2 distribution. On average, about 20 % of observed relay paths from full-relay connected pairs of nodes were direct. Indirect paths from full-relay connected nodes and paths from disconnected or block-relay-only connected nodes seem to produce effectively indistinguishable delay distributions. These empirical probability density functions allow for the calculation of the likelihood of full-relay connection between a pair of nodes given a sequence of observed relay durations.

An interesting observation that can be made on the simulated relay data is that the mean relay path durations do not scale linearly with their corresponding path lengths. Since a relay path of length n contains n consecutive relays that each follow an exponential

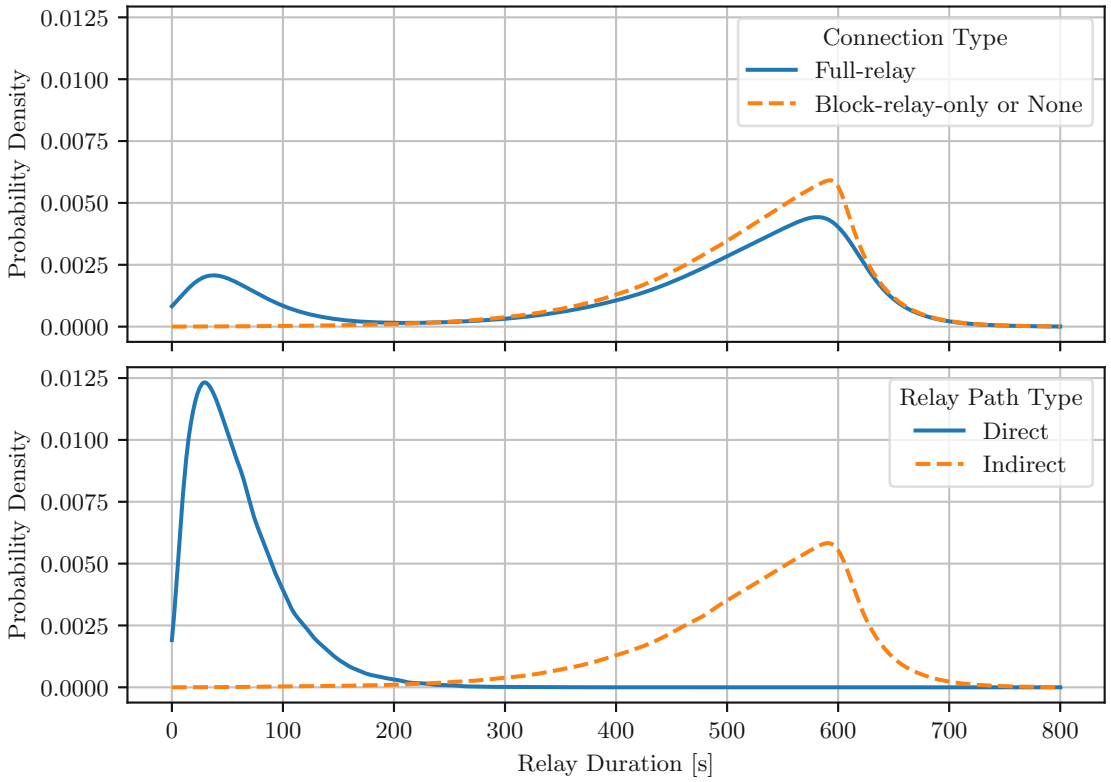

Figure 4.2: Gaussian kernel density estimates for total relay durations

distribution with $\lambda = 30$ seconds, one might expect the mean total duration of a relay chain of length n to be the expected value of the Erlang-n distribution. However, as can be seen in figure 4.3, the expected mean duration for paths of length n seem to increasingly overestimate the observed value for increasing n. This can be explained by the fact that relay paths with shorter relay times "crowd out" relay paths with longer ones. Once a marker address has arrived at a Bitcoin client, there is no more chance for the same marker address to be relayed further if it arrives again at the same client over a path containing longer delays later because in this simulation Bitcoin clients do not relay the same address twice. In other words, relay chains with longer relay times are likely to be extinguished early because the marker addresses have already permeated through the network over relays that have been much quicker solely by chance. Ultimately, what this observation proves is that relays are not truly independent of each other as assumed in our inference techniques. However, since our techniques are mainly reliant on relay paths of length 1 or 2, this circumstance is effectively of no relevance.

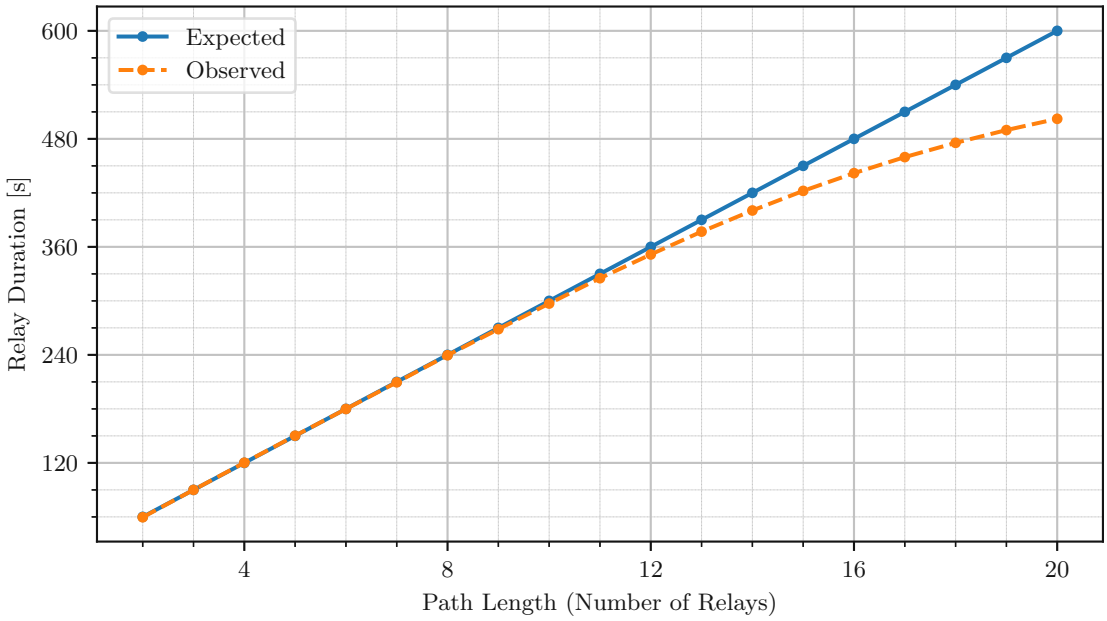

Figure 4.3: Empirical mean relay durations compared to expected values of corresponding Erlang distributions

4.1.2 Degree Estimation

Table 4.4 shows the mean and standard deviation of absolute[1] and relative errors for each degree estimator by measurement period, averaged over all 10 evaluations. After only 15 minutes of measurement amounting to 120 data points per target, both estimators produced a mean error within one node degree. This mean error converges towards 0 with increasing amounts of data, meaning neither estimator seems to be biased towards systematically over- or underestimating. The ML estimator, which disregards knowledge on the prior node degree distribution in the network, consistently outperformed the MAP estimator in terms of mean error likely due to the fact that the degree distribution of the randomly selected sets of targets did not end up being very close to the degree distribution of the remaining reachable portion of the network graph (see figure 4.1). However, the ML estimates do showcase a higher average standard deviation from the true degrees than the MAP estimates. Again, with increasing amounts of data the standard deviation of errors seem to decrease, ending up at around 1.37 for MAP estimation and 1.53 for ML estimation as well as at a standard deviation of relative errors of around 0.05 using the entire data set containing 10,000 data points from 135 minutes of probing.

[1] In the sense of "not relative", also simply "error". Not to be confused with the mean or standard deviation of the *absolute values* of the errors

Table 4.4: Degree estimation errors

Period [min]		Absolute Mean	Absolute STD	Relative Mean	Relative STD
15	MAP	-0.690000	3.061266	-0.012269	0.101907
	ML	0.010000	4.217746	-0.000065	0.138399
55	MAP	-0.530000	1.988669	-0.012755	0.065817
	ML	-0.250000	2.305326	-0.008799	0.074859
95	MAP	-0.300000	1.422783	-0.006959	0.048555
	ML	-0.150000	1.667752	-0.005455	0.055897
135	MAP	-0.320000	1.370973	-0.008008	0.048355
	ML	-0.130000	1.525029	-0.004784	0.050213

4.1.3 Edge Prediction

Figure 4.4 shows the average precision, recall and F_1 score of edge prediction depending on the certainty threshold above which an instance is labelled positive, i.e. above which a pair of nodes are predicted to be peers over a full-relay connection. Three features are immediately apparent:

1. With limited data the Bayes classifier, which takes the prior connection probability into account, performs much better in terms of F_1 score at moderate thresholds around 0.5 compared to the frequentist classifier. The latter requires much less supporting evidence before labelling an instance positive, which trivially improves recall but comes at the cost of precision due to the increasing number of false positives.

2. With increasing amounts of data the recall score tends to stay high for both classifiers even with strict thresholds close to 1. This means that existing connections between monitored clients are likely to be found given sufficient data. In other words, existing connections are unlikely to be falsely omitted.

3. With increasing amounts of data the recall curve decreases steeply just below the threshold of 1 while the precision curve increases steeply just above the threshold of 0. Precision remains nearly linear with only a small positive slope in the threshold range between 0.01 and 0.99, after which it increases very steeply again until the threshold value 1. This is because with growing evidence the classifier tends to map negative instances (disconnected node pairs) to ever smaller certainties and positive instances (connected node pairs) to ever greater certainties, such that relatively few instances are assigned certainty values closer to 0.5 than to either 0 or 1. Table 4.5 summarises this effect. The reason neither classifier can achieve their maximum precision until using certainty thresholds extremely close to 1 is due to false positives

Table 4.5: Mean connection certainty values

	Bayesian		Frequentist	
	Negative	Positive	Negative	Positive
Period [min]				
15	0.001626	0.754702	0.216450	0.952677
55	0.001670	0.989744	0.009530	0.998680
95	0.002079	0.999366	0.004382	0.999726
135	0.002070	0.999976	0.003560	1.000000

with extremely high connection probability estimates. These are negative instances that seem to produce deceptively "positive looking" data samples, i.e. there are observed pairs of disconnected clients that produce an unexpectedly high number of short delays. Interestingly, this problem cannot be solved by collecting more data points since those will only cause the calculated probability for connection to rise instead of fall.

Figure 4.5 shows the same plots from figure 4.4 but limited to the evaluations using the full 135 minutes of data collection and thresholds above $1 - 10^{-14}$. For each measurement period and classifier table 4.6 shows (1) the threshold at which the F_1 score becomes maximal, (2) precision and recall at that threshold, and (3) the maximum value for the F_1 score. We can see that with increasing thresholds up to 1 precision continues to increase, i.e. negative instances are correctly identified, whereas recall decreases only slightly, meaning higher thresholds come at only a slight cost of falsely omitting positive instances.

Like mentioned before, the sharp increase in precision is due to some disconnected node pairs consistently producing short delay values, thus mapping to a high connection probability estimate. This fact combined with the low prevalence of 0.093 % has significant detrimental impact on the precision metric.

In order to better understand these false positives we can analyse the relay paths produced by the simulations in terms of their hop counts and durations. Figure 4.6 shows their frequency distributions for negative and positive instances as well as for negative instances falsely labelled positive by the Bayes classifier even using thresholds greater than or equal to the value optimising for F_1 score. Notice how the delay distributions for negative and positive instances closely resemble the KDEs computed from the training sets shown in figure 4.2. However, we can also see that the delay distribution of some of the negative instances much rather resembles that of the positive ones, resulting in false positives and thus a reduction in the measure of precision. The corresponding distribution of path lengths exhibits a noticeable spike at length 3, which is non-existent in the path length distribution of the remaining negative instances. This spike at path length 3 causes a

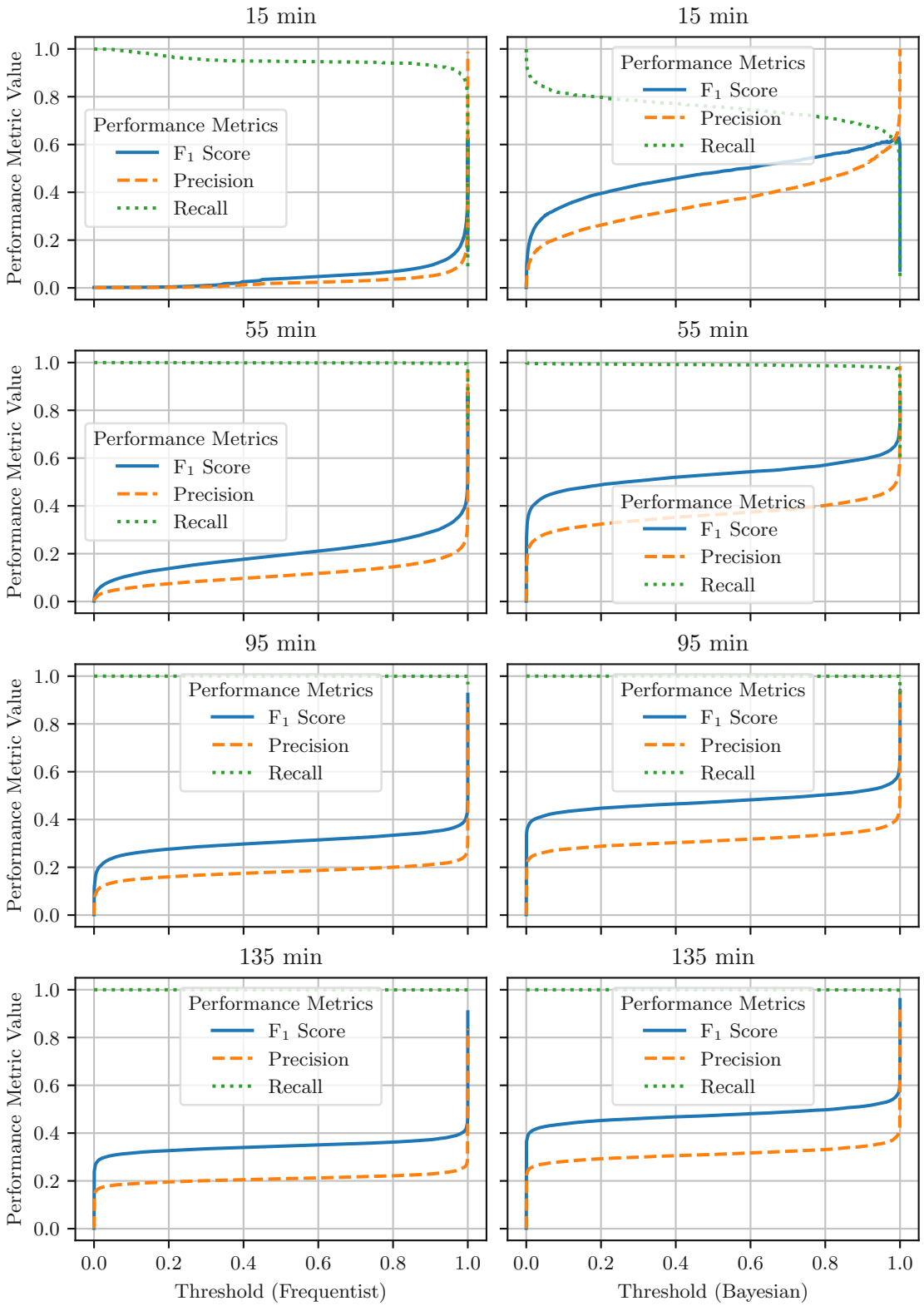

Figure 4.4: Performance metrics for edge prediction

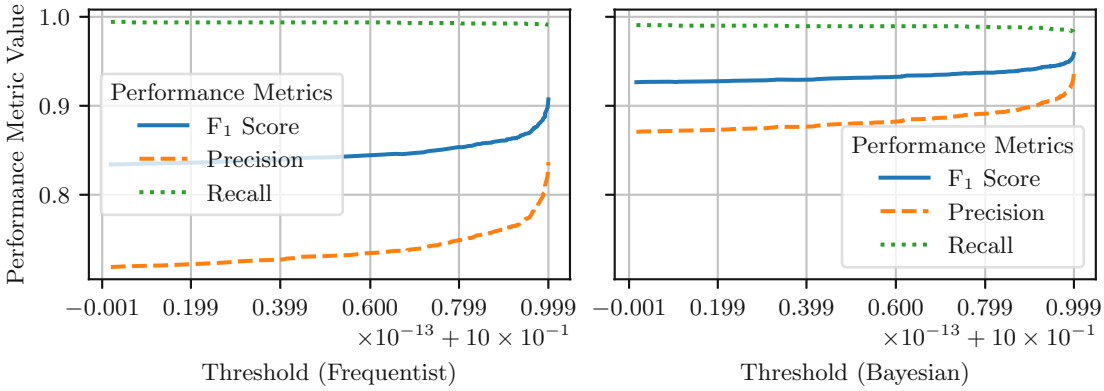

Figure 4.5: Performance metrics for edge prediction around maximal F_1 score

Table 4.6: Performance metrics for edge prediction at maximal F_1 score

Period [min]	Classifier	Threshold	F_1 Score	Precision	Recall
15	Bayesian	0.996031	0.626973	0.681302	0.581360
	Frequentist	1.000000	0.627036	0.694521	0.572338
55	Bayesian	1.000000	0.875263	0.903255	0.849929
	Frequentist	1.000000	0.874042	0.904185	0.846845
95	Bayesian	1.000000	0.940702	0.953230	0.928806
	Frequentist	1.000000	0.923672	0.885743	0.965375
135	Bayesian	1.000000	0.958481	0.937635	0.980690
	Frequentist	1.000000	0.907222	0.836741	0.991276

similar delay distribution as the spike at path length 2 (direct relays implying full-relay connection) of relay paths from positive instances, ultimately confusing the classifier.

4.1.4 Computational Performance

Ten simulations were executed in parallel on a virtual machine with an Intel®Xeon®Gold 6230 2.10 GHz CPU and 512 GB of RAM. Network generation took 16 seconds on average. Each simulation completed in a mean duration of 35 minutes and performed 18,052 virtual message passings per second on average.

4.2 Degree Distribution Model

The *degree distribution model* is similar to the previous *random attachment model* in all aspects except for the underlying network graph. In this model the network also consists of 17,000 reachable nodes as well as 20 monitors and 10 targets. However, with

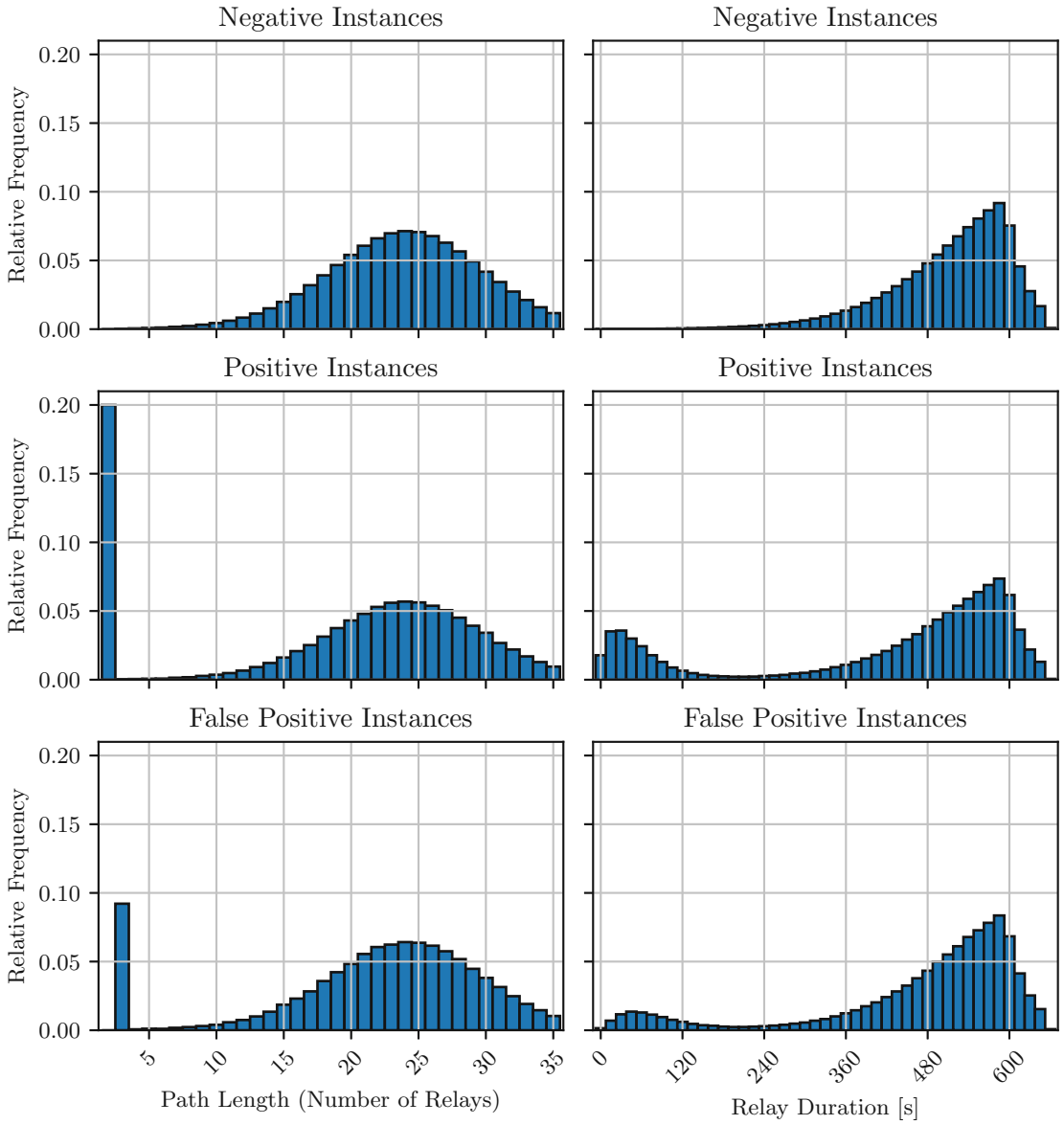

Figure 4.6: Frequency distributions of path lengths and durations of recorded relays

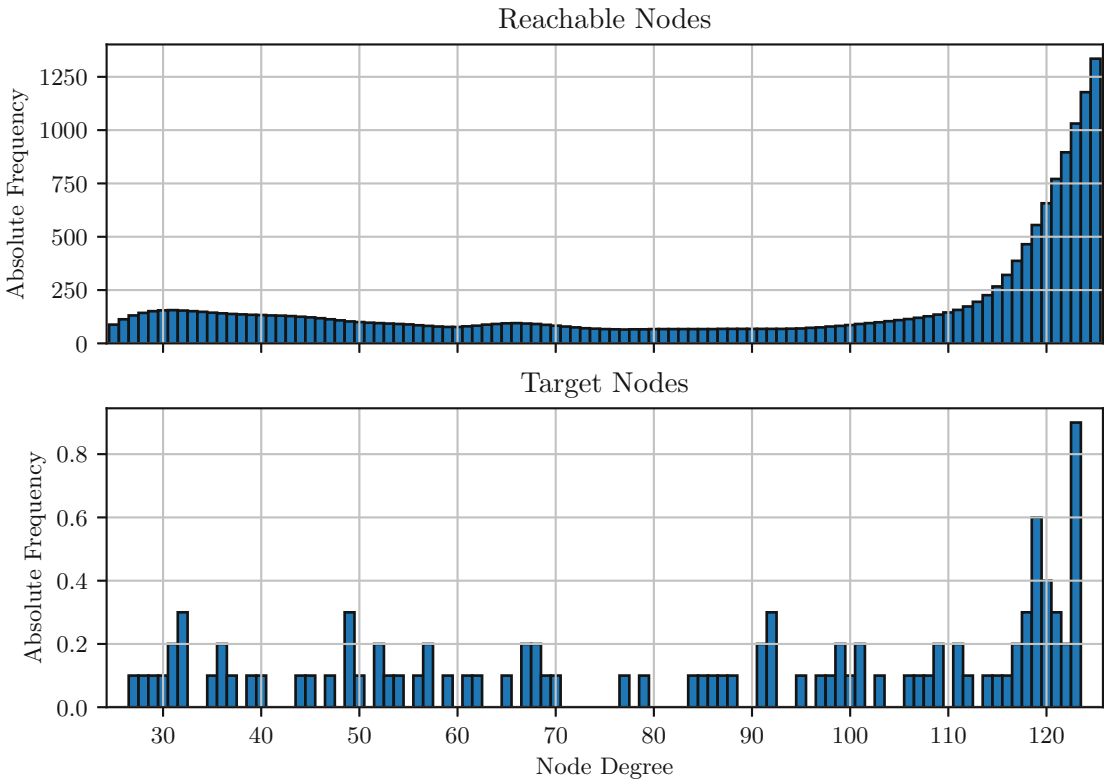

Figure 4.7: Degree frequency distribution among reachable and target nodes

an average number of 123,247 unreachable nodes the network graphs end up considerably larger. For details on the network generation procedure for this model, refer back to 3.3.1.

4.2.1 Network and Traffic Statistics

Figure 4.7 shows the average degree distribution among all reachable nodes and target nodes. Due to the number of reachable nodes already having 125 connections, only 14,487 of them would accept connection requests from monitors and end up being monitored.

With a total of 444,971,598 marker addresses receptions within the entire network as can be seen in table 4.7, the degree distribution model generated much more network traffic than the previous random attachment model, where only a total of 48,888,846 address receptions were recorded. Furthermore, this time around 7 % of received addresses were not relayed due to either the rate limiting or `ADDR` message payload size limit.

As was shown with the random attachment model, table 4.8 presents the average number of injected and recurred marker addresses recorded over the first 15, 55, 95 and 135 minutes of simulated network activity. Although the total number of marker address receptions is more than 7 times greater than in the previous model, monitor nodes in

Table 4.7: Number of received and dropped marker addresses

	Received	Rate Limited	Payload Too Large
Per Client	3,173	0.016	223
Total	444,971,598	2,210	31,215,495

Table 4.8: Number of injected and recurred marker addresses

Period [min]	Injected markers	Recurred markers
15	1,200	2,761,227
55	4,400	13,008,569
95	7,600	22,779,752
135	10,000	31,691,324

Table 4.9: Number of datapoints for degree estimation and edge prediction

	Degree Estimation		Edge Prediction	
Period [min]	Per Instance	Total	Per Instance	Total
15	120	1,200	19	2,760,759
55	440	4,400	89	13,007,116
95	760	7,600	157	22,777,322
135	1,000	10,000	218	31,688,157

the degree distribution model only received around 1.6 times the number or marker addresses.

Table 4.9 shows more specifically the number of usable data points for degree estimation and edge prediction. Note how each injection again leads to one data point for degree estimation but we see a 1.8-fold increase in the mean number of data points per instance for edge prediction.

Figure 4.8 shows the Gaussian KDEs of the empirical relay distributions. The top plot shows the curves for full-relay connected and disconnected pairs of nodes while the bottom plot shows the curves for the durations of relay paths of length 2 (direct relay) and of paths having lengths greater than 2 (indirect relay) for full-relay connected pairs of nodes. Compared to the KDEs from the random attachment model the difference between the delay distribution of connected nodes and that of disconnected nodes is much less pronounced. This is largely due to the overall higher node degrees in the reachable portion of the network, causing fewer relays to take place directly form target to observed node and finally to a monitor node. About 3.5 % of relays recorded from connected pairs were direct as opposed to 20 % in the previous model.

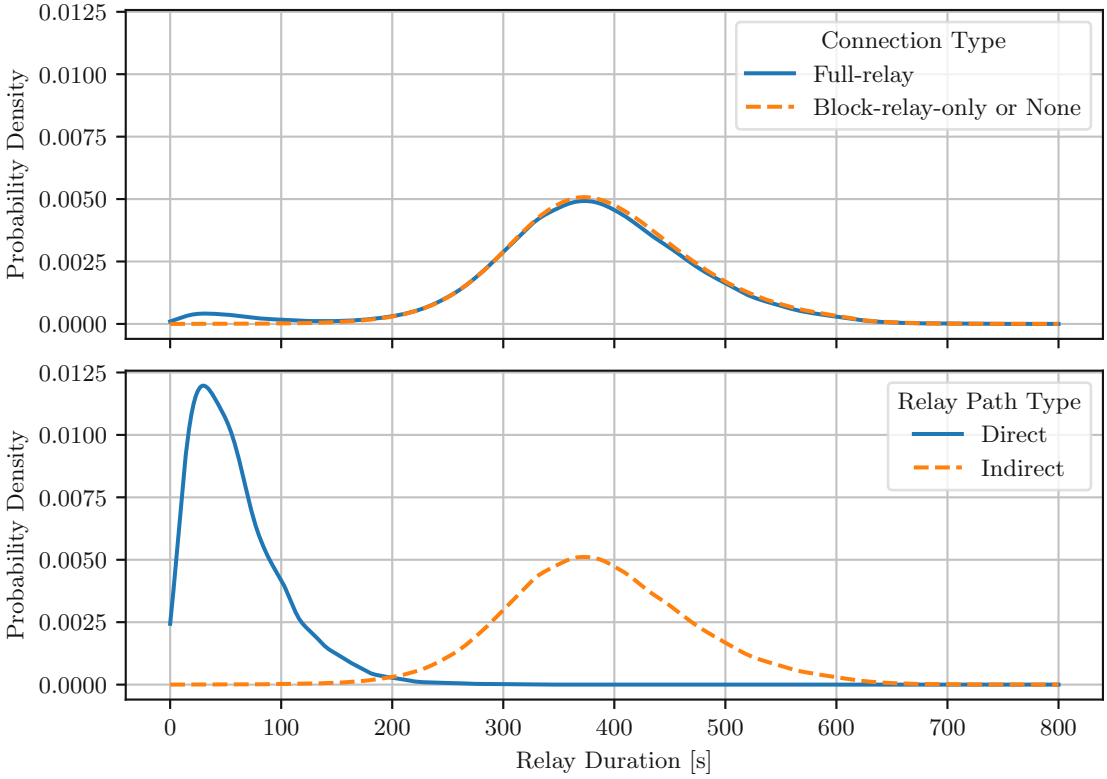

Figure 4.8: Gaussian kernel density estimates for total relay durations

As with the relay data from the simulations in the random attachment model, we can observe for all recorded relay chains that the longer they are, the more they fall short regarding total relay duration compared to the expected value of the corresponding Erlang distribution. This time this effect is much stronger, as can be seen in figure 4.9.

4.2.2 Degree Estimation

Table 4.10 shows the mean and standard deviation of absolute and relative errors for each degree estimator by measurement period. Due to the higher dispersion of node degrees in the reachable portion of the network and thus among the target nodes, degree estimation becomes a significantly more difficult task as is evident in the much higher standard deviations of errors compared to the results from the random attachment model. The mean errors seem to converge to 0 with increasing amounts of data, suggesting neither estimator is biased, although we will not give an analysis of the statistical significance of these findings. Unlike the results from the previous model, where both estimators performed similarly after 135 minutes of measurement, with a standard deviation of about 3.6 the MAP estimator performed noticeably better than the ML estimator with a standard deviation of about 7.5. While the standard deviation of absolute errors of the MAP estimator increased by a factor of 2.7 compared to the random attachment model,

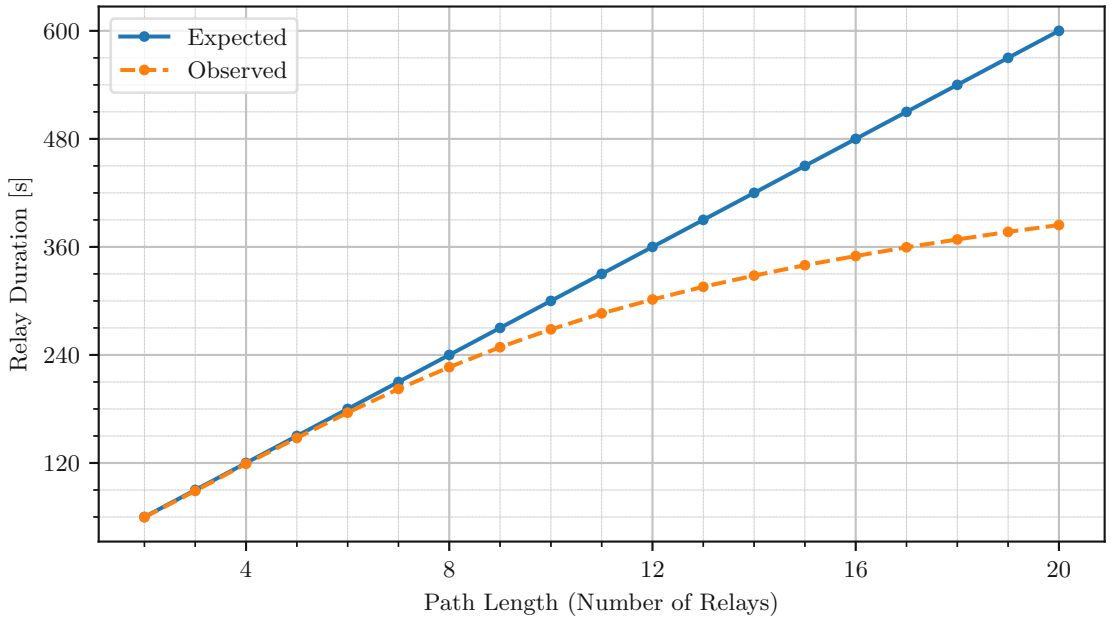

Figure 4.9: Empirical mean relay durations compared to expected values of corresponding Erlang distributions

Table 4.10: Degree estimation errors

		Absolute		Relative	
		Mean	STD	Mean	STD
Period [min]					
15	MAP	0.210000	10.154902	-0.000826	0.142112
	ML	-2.510000	15.089685	-0.033807	0.177979
55	MAP	-0.350000	6.458835	-0.006063	0.087424
	ML	-1.440000	9.801629	-0.019148	0.114840
95	MAP	-0.240000	3.946647	-0.006624	0.058200
	ML	-1.250000	7.937623	-0.016967	0.091475
135	MAP	-0.020000	3.647560	-0.001712	0.053106
	ML	-0.230000	7.504723	-0.004176	0.085736

the standard deviation of relative errors only increased by a factor of about 1.1. In the case of the ML estimator, we see an increase of 4.9 and 1.7, respectively. The reason for the milder increase in dispersion of relative errors than in the dispersion of absolute errors when comparing this model to the previous one is that the target nodes in the degree distribution model have on average 2.3 times as many peers with 5.9 times higher standard deviation than in the random attachment model. This makes the task of point estimation generally more difficult when measured in terms of mean errors.

Table 4.11: Mean connection certainty values

	Bayesian		Frequentist	
	Negative	Positive	Negative	Positive
Period [min]				
15	0.000668	0.189075	0.383759	0.699533
55	0.000923	0.455799	0.167384	0.812279
95	0.001180	0.602290	0.114599	0.871433
135	0.001231	0.661752	0.086628	0.883968

4.2.3 Edge Prediction

In order to evaluate the quality of edge prediction and find the optimal threshold parameter, metrics of precision, recall and F_1 score have been calculated and are presented in figure 4.10. The following observations can be made:

1. Similarly to the evaluation using the random attachment model, the Bayes classifier performs much better than the frequentist classifier at moderate thresholds.

2. Increasing amounts of data substantially improve overall recall. However, in this model at comparable thresholds both classifiers perform significantly worse than in the previous model. After 135 minutes of measurement, the Bayes classifier's recall drops to around 0.7 at a relatively low threshold of 0.2, meaning that using this already lenient threshold only 70 % of connections between targets and the remaining monitored clients could be identified. Note, however, that recall does not drop below 46 % until very strict thresholds above 0.999975 are used.

3. Again similarly to the performance evaluation of the random attachment model, precision and recall of the Bayes classifier change sharply around thresholds close to 0 and 1 whereas much softer change can be seen for a large range of threshold values in between. Table 4.11 shows that generally an increase in evidence allows the classifiers to "divide" negative and positive instances more and more clearly. The exception is that the Bayes classifier increases the mean certainty value for negative instances with an increasing number of data points although this value should decrease. This phenomenon might be caused by underestimating the prior probability of connection, such that even disconnected pairs of nodes produce delays that tend to increase the posterior probability of the existence of a full-relay connection. Despite this unexpected observation, the mean posterior probability for negative instances amounts to only 0.1231 % at most.

Analogous to the previous evaluation, we present classifier scores symmetrically around the threshold where F_1 is maximal in figure 4.11 for each classifier using all measurement data. Table 4.12 lists the numerical values of the scores at said thresholds for varying

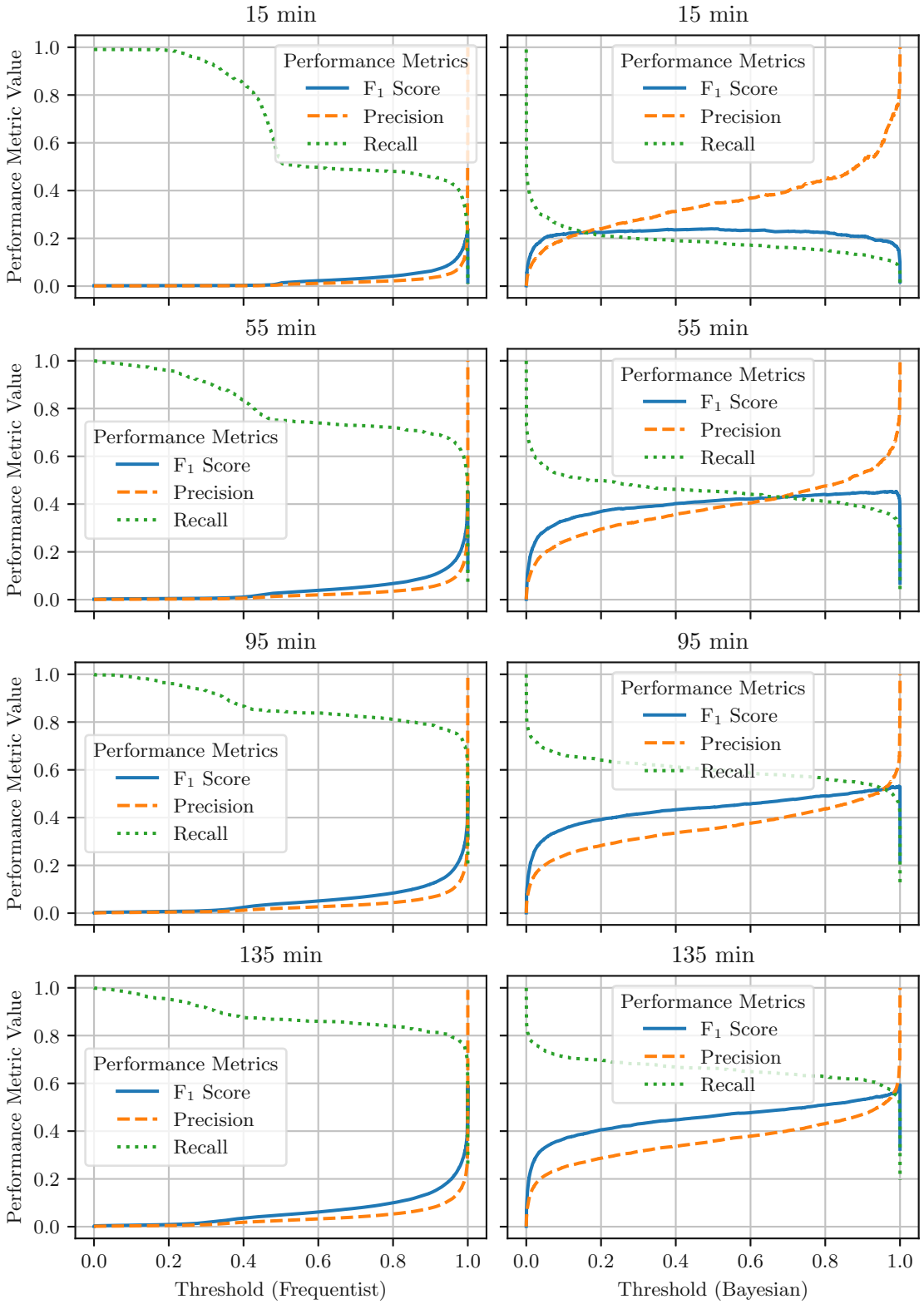

Figure 4.10: Performance metrics for edge prediction

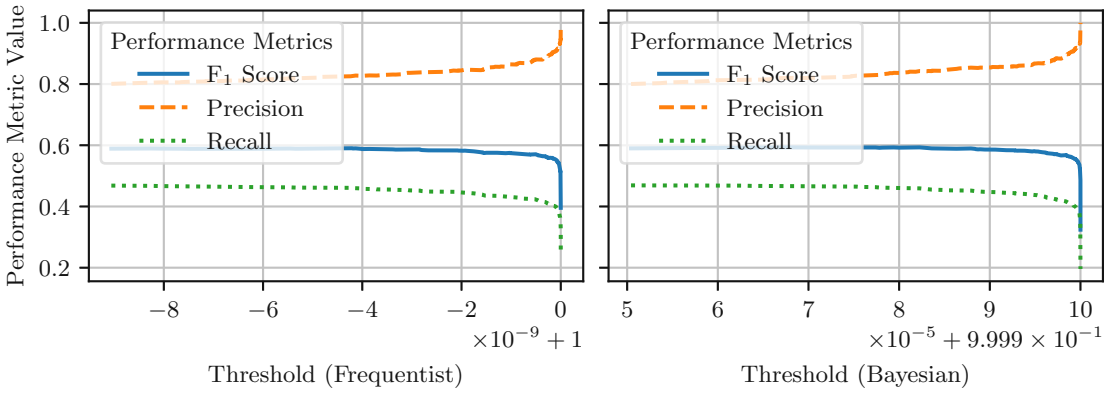

Figure 4.11: Performance metrics for edge prediction around maximal F_1 score

Table 4.12: Performance metrics for edge prediction at maximal F_1 score

Period [min]	Classifier	Threshold	F_1 Score	Precision	Recall
15	Bayesian	0.494815	0.240866	0.347054	0.185127
	Frequentist	0.999841	0.240252	0.355509	0.181984
55	Bayesian	0.972392	0.453151	0.619442	0.358529
	Frequentist	0.999997	0.451626	0.666394	0.343143
95	Bayesian	0.995591	0.531097	0.622063	0.464717
	Frequentist	1.000000	0.531013	0.658161	0.448002
135	Bayesian	0.999975	0.594079	0.826156	0.465765
	Frequentist	1.000000	0.590355	0.824362	0.461642

measurement periods. Comparing to table 4.6 from the section on the evaluation in the random attachment model it can be seen that with regards to F_1 score edge prediction seems much less feasible in this model. Here both classifiers performed worse even when using all available data (on average 218 delays per node pair) compared to in the other model using only data of the first 15 minutes of measurement (on average 7 delays per node pair). Presumably, the most important factor causing this drop in F_1 score is the much lower rate of observed direct relays between connected node pairs (3.5 % as opposed to 20 %). To a lesser degree, the slightly lower prevalence of full-relay connected nodes (0.084 % as opposed to 0.093 %) may contribute as well.

Figure 4.12 shows the frequency distribution of path lengths (i.e. number of relays between target and monitor) and of total relay durations for negative and positive instances as well as for those falsely labelled positive by the Bayes classifier using the entire data set. For classification the threshold at which the F_1 score becomes maximal has been used. Effectively all the observations made on figure 4.6 in section 4.1 apply here as well.

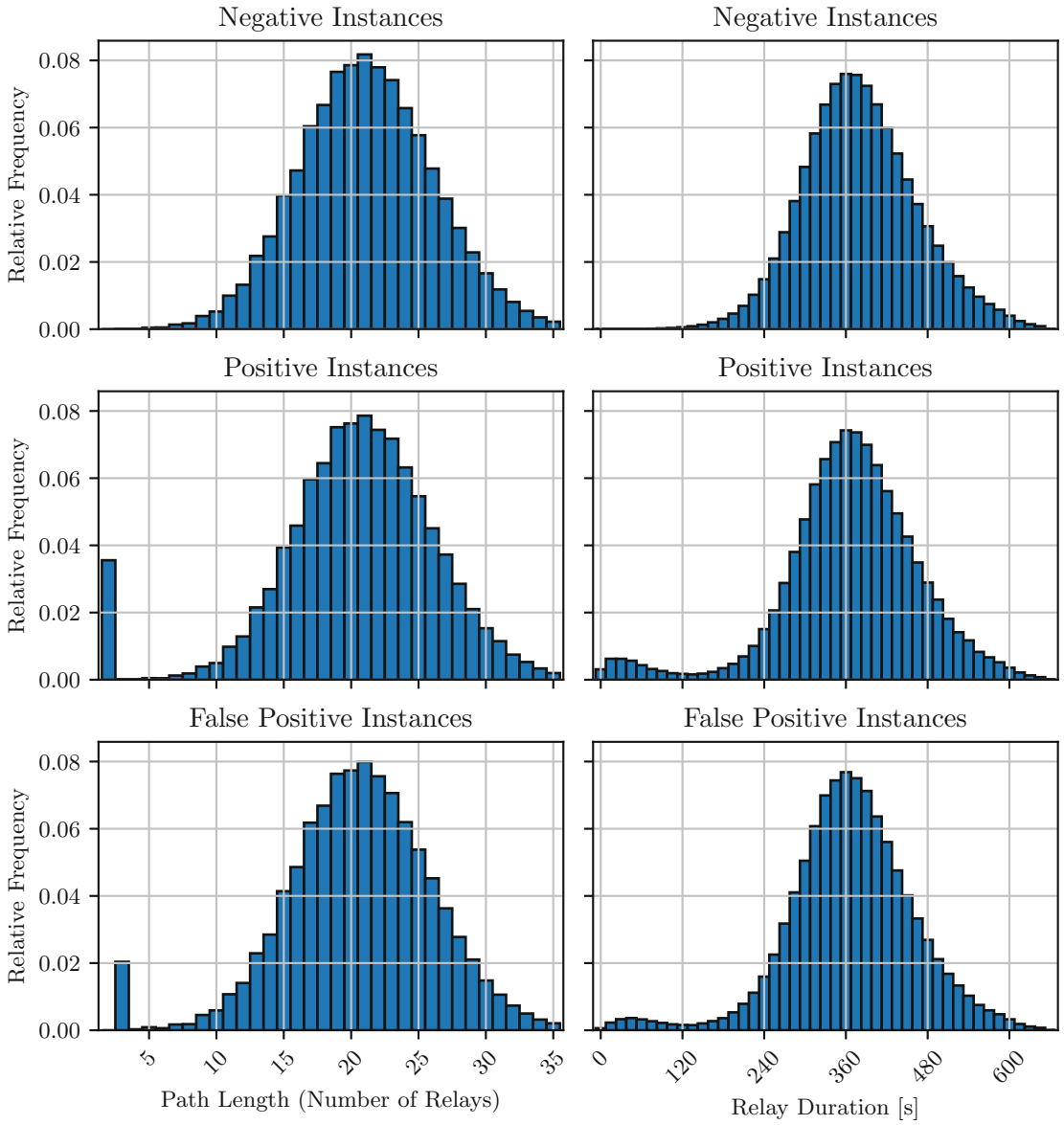

Figure 4.12: Frequency distributions of path lengths and durations of recorded relays

4.2.4 Computational Performance

As before, ten simulations have been carried out in parallel on the same virtual machine. This time, network generation was 2.6 times slower, taking 42 seconds on average. With 373 minutes on average, simulation times were over 10 times longer. The rate of processed messages per second dropped by 41 % down to 10,633 on average due to the increase in network size.

CHAPTER 5

Discussion

Detailed knowledge on the topology of the Bitcoin P2P network could provide valuable insights for researchers on the health and robustness of the network and help developers make informed design decisions. However, it could also enable adversaries to partition the network [NAH15], perform eclipse attacks [DW13, HKZG15, NKMS16] or deanonymise [BKP14] users. Previously proposed topology inference techniques are either no longer applicable due to changes in the Bitcoin reference client [BKP14, MLP+15, NAH16], incur monetary costs [GNH19] or may possibly be disruptive to the network [DSBPS+19]. This thesis set out to develop a topology discovery technique for the Bitcoin P2P network using active traffic analysis. In particular our methods exploit the gossip protocol used for propagating network addresses of Bitcoin clients. By sending marker addresses to target nodes, observing part of the resulting relay behaviour and performing statistical analysis, both the target's number of peers and their addresses in the network can be inferred to varying degrees of accuracy. The main findings of our evaluations are discussed below.

5.1 Validation and Analysis

5.1.1 Degree Estimation

Our evaluations suggest that estimating the degrees of individual Bitcoin nodes by exploitation of the `ADDR` gossip protocol can be performed with good accuracy using relatively few resources. In the more complex and likely more realistic *degree distribution model* the standard deviation of relative errors was 18 % with ML estimation and 14 % with MAP estimation after 15 minutes of monitoring amounting to 120 data points per target (one injection providing one data point). These measures of dispersion drop to 9 % and 5 %, respectively, when 1,000 data points from 135 minutes of measurement are used. While it is clear that knowledge of the prior degree distribution, which is used in

the MAP estimator, leads to more accurate results using less data, even the uninformed ML estimator shows considerable performance. Although these results are not detailed enough to allow for extrapolating the quality of our technique with growing data sets, it may be reasonable to assume that even longer measurement periods lead to more accurate predictions. Note that this degree estimation technique—as opposed to our edge prediction technique—scales with the number of targets, such that estimating the degree of an individual Bitcoin node only requires a few monitor connections to this node.

5.1.2 Edge Prediction

Edge prediction proved to be a much more difficult task. In the somewhat idealised *random attachment model* our technique proved reliable regarding correctly predicting full-relay connections where such are indeed present, i.e. both the frequentist and the Bayes classifier showed a recall of 100 % using only 55 minutes of measurement data amounting to an average of 46 data points per node pair (one data point being one marker address recurrence) at precision values above 20 %. For almost the entire certainty threshold range, the frequentist classifier's F_1 score stayed above 0.3 and that of the Bayes classifier above 0.4. As noted in section 4.1, the reason for the precision curves remaining stagnant until extremely high certainty thresholds is due to the combination of the low prevalence of full-relay connections and some disconnected pairs producing deceptively positive looking relay durations (see figure 4.4). The precise reason for the latter phenomenon remains unexplored. The most plausible explanation so far seems that the arrangement of connections (including monitors) around these node pairs is simply such that the probability of getting relay paths of length 3 is increased, while the probability of getting longer ones is decreased. These situations may arise if the following conditions are met:

- The target has few reachable peers such that it is relatively likely to relay a marker address to one of its unreachable peers.
- Each of the target's unreachable peers has a large number of (necessarily reachable) peers that are connected to a large number of monitors.

For node pairs that fulfil these criteria, monitors will record samples containing a disproportionate number of relay paths of length 3. These cause a distinctive peak in the delay distribution which is very similar to the one found in the delay distribution from truly connected pairs, ultimately confusing the classifier. Unfortunately, gathering more data from these instances cannot solve the problem of misclassification because they would only continue to produce the same deceptive relay duration samples. In theory, however, connected and disconnected pairs do seem to be linearly separable, as choosing just the right threshold leads to an F_1 score of about 0.91 (precision = 0.84, recall = 0.99) for the frequentist classifier and an F_1 score of about 0.96 (precision = 0.94, recall = 0.98) for the Bayes classifier. Finally, these results show that while prior knowledge on the

edge probability is valuable when there is little data, the qualities of the two classifiers converge with increasing amounts of data.

When looking at the edge prediction results for the degree distribution model using the Bayes classifier, we observe the more typical trade-off between precision and recall in binary classification problems. At a certainty threshold of 0.2, the classifier surpasses an F_1 score of 0.4, which continues to slowly increase. The frequentist classifier performs poorly except at very high thresholds even if the input comprises the entire data set. Nonetheless, we see that for optimal thresholds the frequentist classifier, reaching an F_1 score of about 0.59 (precision = 0.82, recall = 0.46), performs practically the same as the Bayes classifier, reaching approximately the same F_1 score (precision = 0.83, recall = 0.47). However, it must be noted that the threshold intervals in which the classifiers perform their best are extremely narrow for both classifiers and for both network models and would be unknown in a real world setting. We can observe similarly deceptive negative instances causing false positives even for very high thresholds as has been discussed above for the random attachment model.

5.1.3 Independence of Relay Processes

The simulations show that `ADDR` relays are not actually independent stochastic processes. Due to the fact that Bitcoin clients do not relay the same network address to the same peers in a certain time frame, relay chains containing longer delays are relatively unlikely to accumulate more hops because each potential future recipient is likely to have already received this address via some relay chain containing shorter delays. Therefore, estimating the relay path length k given its total delay δ by determining which k minimises the absolute difference between δ and the mode of the Erlang-k distribution will lead to underestimation (refer back to figures 4.3 and 4.9). While these findings may be relevant for future research, this circumstance does not practically affect our techniques.

5.2 Limitations

Limitations apply to our methodology by design and to the results discussed above due to simplifying assumptions and abstractions in our models.

- There is no hope of uncovering connections of type *block-relay-only* as Bitcoin Core excludes these connections from `ADDR` message gossip entirely.
- Because our techniques are based on active traffic analysis, they require maintaining connections to the target nodes and in the case of edge prediction also to their peers. Since potentially any two nodes of which at least one is reachable could be peers and since `ADDR` messages can only be collected from monitored clients, monitor nodes must connect to as many Bitcoin clients as possible if all connections (with the exception of block-relay-only connections) between observable clients are to be found. This is problematic because unless unreachable clients try to establish

full-relay connections to monitor nodes of their own accord, unreachable nodes cannot be targeted and unreachable peers of target nodes cannot be discovered. Thus, large parts of the Bitcoin network topology remain unknown even if inference accuracy were perfect. One could make the case that any particularly interesting subset of the network is likely to be mostly comprised of reachable nodes but those might not accept incoming connection requests from monitor nodes either due to exhaustion of open connection slots; or they might accept only few monitors as new peers, such that only few marker address recurrences take place, which drastically limits inference quality.

- Our simulations did not account for any network traffic other than the `ADDR` gossip caused by marker address injections from monitor nodes. When applying our technique to the real Bitcoin network, it would likely be necessary to increase the interval between injections in order to keep traffic congestion low. Although background `ADDR` gossip activity in the Bitcoin network could be estimated by recording samples using regular Bitcoin Core instances, any interaction with the Bitcoin network was deemed outside the scope of this thesis and left for future research.

- While the Bitcoin network graph is subject to constant change due to nodes leaving and joining, our methods assume a perfectly static graph. The longer the `ADDR` probing procedure takes, the more likely it is that a target's degree or set of peers changes, causing inconsistent evidence and skewed results.

5.3 Implications

The results of our evaluations provide proof of concept and suggest that both inferring the number of peers as well as the existence of connections between given Bitcoin clients leveraging the `ADDR` gossip protocol is feasible with substantial accuracy despite implemented countermeasures. Although limitations apply and immediate success in exhaustively mapping the Bitcoin network's topology using our methods and models are unlikely, refining these techniques and conducting investigations in controlled realistic settings seems to be a promising endeavour. Furthermore, if the network's true topology is to remain hidden, Bitcoin developers may be required to add additional countermeasures to existing protocols and to Bitcoin Core.

5.4 Future Research

The above discussion leads to several questions and tasks for future research. The next logical step would be to design and implement a framework for controlling ground-truth Bitcoin Core instances and monitor nodes in order to collect ground-truth data and evaluate our methods on them. Using this framework, the effects of background network traffic and network churn could be more closely studied, as well as statistics on `ADDR` relays

including delay distributions under varying conditions. By running *super nodes*, which advertise their own addresses and accept all incoming connection requests, one might be able to maintain connections to a considerable portion of the otherwise unreachable Bitcoin network, making more complete analyses possible. Finally, by refining models and statistical methods—possibly by taking other protocols or even side-channels into account—inference accuracy may be further improved. In particular, finding a way to distinguish connected pairs from disconnected ones that produce samples containing unusually high proportions of relay chains of length 3 as discussed in section 5.1.2 would drastically improve classification precision.

Future studies interacting with the real Bitcoin network may have to deal with several challenges. Depending on the scope of measurements, a large pool of marker addresses must be available. Recall that every injected marker address must be unique for the duration of a measurement so that it can be uniquely traced. Additionally, marker addresses should be unique in the entire network in order not to inhibit `ADDR` gossip, since a node may dedicate the same peer for relaying the same address within a 24-hour time frame while refusing to relay it more than once. Due to the fact that Bitcoin Core deems any two IPv4 addresses to be different as long as they are associated with different ports, a single IPv4 address may provide one marker address per available port, i.e. potentially 65,536 marker addresses. However, for repeated very large scale measurements, this may in fact not be enough. Furthermore, maintaining a large number of ground-truth and monitor nodes may incur significant bandwidth and computational overhead. Another aspect to consider would be that super nodes may over time be detected, raise suspicion and ultimately lead to network nodes avoiding and rejecting them. Finally, experimenting on the Bitcoin network may incur ethical problems and care must be taken not to be disruptive and cause harm to individual network nodes or the network as a whole.

CHAPTER 6

Conclusion

Being able to observe the topology of the Bitcoin P2P network would on the one hand enable adversaries to perform certain attacks on individual clients or the network at large but would on the other hand aid in assessing the network's resilience, fairness and performance. Previous topology inference approaches have either been mitigated by countermeasures implemented in the Bitcoin reference client or are otherwise impractical.

In this thesis, we have developed active traffic analysis techniques exploiting the gossip protocol around `ADDR` messages, which are used to publish and propagate network addresses of Bitcoin clients, in order to infer (1) the number of peers of an observed target client and (2) the existence of a full-relay connection from a target to some other observed client. In validations using simulated network traffic our degree estimation method proved to be nearly unbiased while having produced mean errors showing a standard deviation of 1.5 in an idealised model and mean errors showing a standard deviation of 3.6 in a more realistic one. When using relative errors instead, these metrics come out to be approximately 5 % in both models. Using conservative classifier sensitivities, our connection inference method was able to predict network links with a precision of 40 % and a recall of 99.8 % in the idealised model and both precision and recall of 56 % in the more realistic one. Using the ideal classifier sensitivities with regard to the generated data sets, precision and recall come out to be 94 % and 98 % in the former model and 83 % and 47 % in the latter model.

The effectiveness of our approach is limited by the fact that connections of type block-relay-only prohibit any `ADDR` gossip, thereby making themselves completely invisible to our analyses. Also, it is not entirely clear how these simulation-based evaluations would carry over to attempts in the real Bitcoin network. Nonetheless, our results suggest the possibility of success in real settings. Therefore, Bitcoin developers are advised to consider adapting current countermeasures or implementing new ones if they want to ensure that the Bitcoin network topology remains obscured.

Our evaluations show that topology finding is theoretically possible with substantial accuracy. In order to definitely prove the effectiveness of our methods, experiments within the real Bitcoin network using controllable ground-truth nodes are required. Additionally, future research on identifying patterns in `ADDR` message traffic might further increase the quality of connection inference.

List of Figures

List of Tables

Bibliography

[bit] https://bitnodes.io/nodes/all/. (accessed May 17, 2023).

[BJG18] Jørgen Bang-Jensen and Gregory Gutin. *Classes of directed graphs*, volume 11. Springer, 2018.

[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.

[BT08] Dimitri Bertsekas and John N Tsitsiklis. *Introduction to probability*, volume 1. Athena Scientific, 2008.

[coi] https://coinmarketcap.com/currencies/bitcoin/. (accessed Dec. 21, 2023).

[Del21] Merlin Delcid. El Salvador buys bitcoin as the digital currency becomes legal tender. https://edition.cnn.com/2021/09/06/business/bitcoin-price-el-salvador-intl-hnk/index.html, 2021. (accessed Dec. 21, 2023).

[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 *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*, pages 550–566. Springer, 2019.

[DSS10] Reinhard Diestel, Alexander Schrijver, and Paul Seymour. Graph theory. *Oberwolfach Reports*, 7(1):521–580, 2010.

[DVGKS22] Alex De Vries, Ulrich Gallersdörfer, Lena Klaaßen, and Christian Stoll. Revisiting bitcoin's carbon footprint. *Joule*, 6(3):498–502, 2022.

[DW13] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, 2013.

[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.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GBH21] Matthias Grundmann, Max Baumstark, and Hannes Hartenstein. Estimating the peer degree of reachable peers in the bitcoin p2p network. *arXiv preprint arXiv:2108.00815*, 2021.

[GNH19] Matthias Grundmann, Till Neudecker, and Hannes Hartenstein. Exploiting transaction accumulation and double spends for topology inference in bitcoin. In *Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers 22*, pages 113–126. Springer, 2019.

[GYA18] Jonathan L Gross, Jay Yellen, and Mark Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 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.

[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.(2015). *Cited on*, page 54, 2015.

[NAH15] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. A simulation model for analysis of attacks on the bitcoin peer-to-peer network. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1327–1332. IEEE, 2015.

[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*, 2008.

[NKMS16] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack.

In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 305–320. IEEE, 2016.

[ZHK+23] Michael Zietz, Daniel S Himmelstein, Kyle Kloster, Christopher Williams, Michael W Nagle, and Casey S Greene. The probability of edge existence due to node degree: a baseline for network-based predictions. *bioRxiv*, pages 2023–01, 2023.