University of Melbourne

School of Mathematics and Statistics

# The Bitcoin P2P Network Topology: a Review of Inference Methods and Machine Learning Approach

*Itsi Weinstock*

supervised by
Professor Peter Taylor

A thesis submitted for the degree of
Master of Science

October 11, 2019

# Abstract

Blockchain technology, pioneered with the implementation of the Bitcoin digital currency, allows for the maintenance of a decentralised ledger of transactions that is run on a distributed network of users, free of centralised authority and trusted third parties. Specifically in Bitcoin, this network is a peer-to-peer network operating a gossip protocol, designed to ensure the efficient spread of information through the network and consensus on the ledger. Knowledge of the topology of this network allows for the analysis of the health of the ecosystem and studies into its properties, but also risks the ability of users to be deanonymised and targeted by other attacks applicable to these networks. In this context, we explore methods that have been made to discover this topology.

Through this thesis, we explore in depth the various techniques developed by researchers. New methods are regularly being developed due to the subsequent reactions of the Bitcoin developer community. Most methods rely at least in part on idiosyncrasies of the Bitcoin protocol, which its developers update to prevent such inference following the release of new research. We make recommendations on directions of future research given these circumstances, and provide a proof of concept through the development of nonparametric models to learn the topology of P2P networks running gossip protocols, and validate these models on a simulation.

# Acknowledgments

I wish to sincerely thank Peter Taylor for his enthusiastic and thoughtful supervision. I am also extremely grateful to Rhys Bowden for his welcome guidance and expertise. This would not have been possible without their critical feedback and necessary direction.

I would like to thank Aapeli Vuorinen for his tireless help with my endless questions, for his belief and support, and most of all the friendship we have built through our time in this course.

I owe specific mentions and thanks to my dear friends Jacinta Cooper, for her heartfelt moral support during this degree, and Daniel Beratis, for his selfless second pair of eyes and rigorous sense of written style.

I would like to thank all my friends, my family, and the wonderful communities I have been a part of over this significant chunk of my life.

**I dedicate this work**:

to my grandparents — sorry for not calling for Shabbas, I had to write a thesis;

to my father — yes I'm a professor now;

and to my mother, who I miss very much.

# Contents

# Chapter 1

# Introduction

You arrive at a very secretive and gossipy village. Each person in the village has a house that they never leave. In each house is a phone, which the residents use to call each other. The average citizen has a few friends who they call to relay the latest rumours, which is the only reason anyone ever calls anyone else. Whenever someone receives a new piece of gossip from a friend, they quickly call up their other friends and tell them the news. There's just one quirk that they all follow: no one ever tells any of their friends who their other friends are. Now you've arrived and you've been given a phone. You can phone up anyone you like and make them your friend so they'll start telling you the town gossip. You have one mission: without asking anyone directly, figure out who is friends with who.

This village is an example of a *peer-to-peer* (P2P) network, specifically a P2P network using a *gossip protocol*. In these networks, new information spreads not through some central authority, but rather by individuals relaying information to each other until it reaches everyone. In this thesis, we discover what we can about the P2P network at the foundation of Bitcoin. Primarily we look at the methods that researchers have used to find the *topology* of the network, that is which computers are talking to each other. Knowledge of this topology has a degree of impact on the anonymity of Bitcoin, so the Bitcoin developers administering it regularly patch the underlying protocol in order to make the task harder; quickly rendering the methods of these researchers useless. In this pattern, we will follow the arms race between researchers and developers to discover the topology or stop it from ever being discovered.

## What is Bitcoin?

Blockchain technology was introduced by Satoshi Nakamoto in 2008 in his original Bitcoin white paper [1], which he first implemented with the Bitcoin Core client in

2009. The paper outlines a solution to creating a digital currency that doesn't allow for someone to spend the same money more than once — known as the *double spend problem* — while not relying on a trusted third party such as a bank to maintain and check transactions against a centralised database. Instead, Bitcoin proposes a novel technology for maintaining a *decentralised* database known as *blockchain* — as it is comprised of a chain of *blocks* — as well as an algorithm, using which competing entities on the network can reach a consensus on how to progressively add transactions to that database. The technology has since been applied to many other areas including finance, insurance, voting and contract management [2, 3].

When you want to make a payment with Bitcoin, you must broadcast your transaction onto the public Bitcoin P2P network, made up of *nodes*; entities that run a program that validates transactions and passes them onto their neighbours in the network. When joining the network, a node selects their neighbours, *peers* in our terminology, by choosing semi-randomly from the nodes that are already part of the network.

Several entities on this network known as *miners* collect transactions into *blocks*. The miners race against each other to find a solution to a computationally difficult problem, thus forming a proof-of-work: a "probabilistic proof" that the network together had to perform a certain amount of computational work to find this particular block. The first miner to find a solution *mines* the block, allowing it to be propagated into the network. The successful miner collects two rewards: a quantity of Bitcoins for contributing a block to the blockchain; and a quantity of Bitcoins equal to the sum of the transaction fees of transactions contained within the block. Transaction fees are commissions paid by the payer of the transaction to incentivise the inclusion of that transaction in blocks, which is necessary due to competition arising from the a size limit on blocks. Each block must reference the previously mined block, providing a linking succession of blocks known as the *blockchain*. The difficulty of the mining problem is regularly adjusted with the aim of making the average inter-arrival time of blocks ten minutes.

The main problem in this thesis is to infer the topology of this P2P network; determining the peers of any given node or the topology of the whole network. Members of the network act individually, but in general, nodes do not give out information about who their peers are. Key to inferring the structure is understanding how information is transmitted between peers. The network uses a *gossip* protocol: each node shares its messages with its neighbours who in turn pass them on to their neighbours; akin to a rumour spreading like gossip in the village. This same modelling has been used to study viruses spreading between cells and epidemics spreading

between people or geographically, so it is also known as an epidemic protocol[4].

## Why find the topology?

Aside from being a mathematically interesting topic of study, there are various reasons why we would want to understand methods of finding the topology of the Bitcoin P2P network. Peer links are the only way that peers learn of transactions and blocks. This method of information propagation is vital to understanding how attacks may be possible in the Bitcoin ecosystem. Knowledge of the topology would give insight into the health of the ecosystem. Bitcoin and other P2P based systems offer and rely on decentralisation, so the network topology contains the information necessary for this to be monitored. Centralisation would allow for attackers to ease the implementation of denial of service (DoS) attacks — for instance, targeting nodes in the minimum cut of the network; amongst other issues. A description of attacks common to P2P Networks such as DoS flooding, eclipse and Sybil attacks can be found in [5].

Göbel, Keeler, Krzesinski, and Taylor [6] developed models for the effect of propagation delay on the evolution of the Bitcoin blockchain in the context of a selfish mining strategy [7]. An adversarial miner upon mining a block could chose to not propagate it into the network and start work on a block linking to this hidden block. Any extra time the miner has solely working on the next block gives an advantage in finding it over other miners in the network, corresponding to an increase in expected income. The miner could release the first block after some time, or when they become aware that another conflicting block is in the network so they can flood the network with their own block. The selfish miner runs the risk of losing the income from their first block if they can't flood the network fast enough. Inferring the topology of the P2P network was left as future work so implementations of simulation and analytical models could be optimised to improve the success of this strategy.

Biryukov, Khovratovich, and Pustogarov [8] demonstrated a method for deanonymising users that connect to the P2P network that relied on knowing the connections of any given targets. Specifically they showed a way to correlate a user's IP address with their Bitcoin address, the account or *wallet* that is described in their transactions in the blockchain. The researchers claim this method works for users behind NATs and firewalls, clients using anonymity services such as Tor, and even in cases where users generate several wallets. It has a success rate of 11% to 60% without false positives depending on how stealthy the attacker wishes to be. At the time the paper was written, they estimated the cost of the attack to be 1500 Euros per month.

Lei [9] demonstrated a method for conducting double spend attacks — applicable in fast payment scenarios where the victim must provide a service before they are fully confident that a transaction has been entered into the consensus blockchain. If the victim relies on knowledge of a transaction being broadcast into the network, then an attacker can trick the victim by sending one transaction to a small set of nodes connected to the victim, giving them confidence in the transaction; simultaneously sending a transaction spending from the same input (see Section 3.1.2) to a large set of different nodes. Given that the latter transaction is sent to more nodes, it is more likely to be accepted into the consensus blockchain than the former. Knowledge of the topology increased the success probability from 12% to 60%.

Due to the attacks possible with knowledge of the network topology, developers working on Bitcoin clients (programs that run the software for Bitcoin on users' computers) regularly update the protocol so that attacks are harder to mount. This has led to a phenomenon of researchers spending significant time developing new methods to infer the topology, followed by developers quickly changing small parts of the protocol to make their research redundant. The questions that arises is whether there exists a general approach to determining topology in P2P networks that would work no matter the changes in protocol; in Bitcoin and other blockchain or P2P applications.

## Thesis Outline

We begin in Part I by introducing concepts that will be necessary to understanding the body of the thesis. In Chapter 2 we cover mathematical definitions and outcomes followed in Chapter 3 by an in-depth explanation of the Bitcoin ecosystem.

In Part II we embark on a chronological investigation of methods used in the inference of the Bitcoin P2P network topology. In Chapter 4 we define this core problem of thesis. Chapter 5 provides a thorough examination of the literature focusing on five key papers that introduced new methods, within the context of the Bitcoin developers actively seeking to impede the feasibility of their techniques. In Chapter 6 we provide a meta-analysis of the research area, and motivate suggestions for new directions in research that focus on passive, statistical methods.

Part III consists of a proof of concept in developing these methods, describing nonparametric algorithms. In Chapter 7 we construct a simulation of a P2P network running a gossip protocol, and show that information of the edges in the network can be found in the structure of a correlation matrix of delay times. Finally, in Chapter 8 we use this structure to conduct a machine learning experiment on topology inference, achieving 55% recall and 86% precision in reconstructing edges.

# Part I

# Preliminaries

# Chapter 2

# Mathematical Preliminaries

In this chapter we cover some fundamental mathematical theory including definitions and results used in the later chapters. These will be necessary for understanding the contents and analyses of the methods used to discover network topology, which will be discussed from Part II onward. Readers unfamiliar with Bitcoin, blockchain technology and cryptography will require an understanding of Section 2.2 on cryptography to follow the Bitcoin preliminaries in Chapter 3. We will also briefly cover selected optimisation algorithms that will be necessary for the nonparametric methods developed in part III.

## 2.1 Random graphs

Networks are formalised by the mathematical notion of graphs. This will provide a language with which we can naturally model the nodes and connections underlying the Bitcoin network. The methods we will look at will make certain assumptions about the networks under investigation. As we don't know their structure, we often assume that these graphs come from some probabilistic distribution, or that they are the outcome of some random process. We will present the method introduced by Erdos and Renyi in [10] to generate a random graph, and explore some of its properties.

Those familiar with graph theory may want to skip onto the discussion of random graphs in Section 2.1.2.

### 2.1.1 Graphs

**Definition 1** (Graphs, vertices and edges)**.** *A* ***graph*** $G$ *is a pair* $G = (V, E)$ *where*

- $V$ *is a set whose elements are called* ***vertices****:* $v \in V$*;*


(a) directed graph (b) undirected graph

Figure 2.1: Graph representations

- *$E$ is a set of pairs of vertices called **edges**: $(v_1, v_2) \in E$ where $v_1, v_2 \in V$.*

**Definition 2** (Directed and undirected graphs)**.** *A graph $G = (V, E)$ is called a **directed** graph if the edges are ordered pairs*

$$(v_1, v_2) \neq (v_2, v_1).$$

*A graph is **undirected** if the edges are unordered pairs*

$$(v_1, v_2) = (v_1, v_1).$$

We can pictorially represent graphs using circles to represent vertices, and arrows or lines between pairs of vertices as edges, if they are directed or undirected graphs respectively. In Figure 2.1 we have representations for $G = (V, E)$ where $V = \{v_1, v_2, v_3\}$ and $E = \{(v_1, v_2), (v_1, v_3), (v_2, v_3)\}$.

**Definition 3** (Connected graphs and paths)**.** *We say a graph $G = (V, E)$ is **connected** if for any pair of vertices $v_1, v_n \in V$, there exists at least one set of edges $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$ called a **path** from $v_1$ to $v_n$ where $v_i \in V$ and $(v_i, v_{i+1}) \in E$ for $i \in \{1, 2, \ldots, n-1\}$.*

For this thesis, we will assume that all graphs we encounter are connected. We will also assume that graphs do not allow for multiple edges between the same vertices, and do not allow for loops — edges that are a pair of the same vertex $(v_1, v_1),\ v_1 \in V$.

**Definition 4.** *For a graph $G = (V, E)$:*

- *the **order** of $G$ is $|V|$, the number of vertices;*
- *the **size** of $G$ is $|E|$, the number of edges;*

- *for an edge $e = (v_1, v_2) \in E$, we refer to $v_1$ and $v_2$ as the **endpoints** of $e$;*
- *for vertices $v_1, v_2 \in V$, if the edge $e = (v_1, v_2) \in E$ then we say $v_1$ and $v_2$ are **neighbours**. If $G$ is directed we say $v_2$ is an **outgoing** neighbour of $v_1$ and $v_1$ is an **incoming** neighbour of $v_2$;*
- *for vertices $v_1, v_2 \in V$, if the edge $e = (v_1, v_2) \in E$ then we say $e$ is **incident** on $v_1$ and $v_2$. If $G$ is directed we say $e$ is an **outgoing** edge of $v_1$ and an **incoming** edge on $v_2$;*
- *the **degree** $\deg(v)$ of a node $v \in V$ is defined as*

$$\deg(v) = \sum_{v_i \in V} \mathbf{1}\{(v, v_i) \in E\};$$

*the number of edges incident on $v$, where $\mathbf{1}\{\cdot\}$ is the indicator function*[1].

**Definition 5** (Trees)**.** *A **tree** is an undirected graph $G = (V, E)$, where every pair of nodes $v_i, v_j \in V$ is connected by exactly one path.*

**Definition 6** (Cycles)**.** *A **cycle** is a path from a vertex $v \in V$ to itself that contains at least one edge, and no edge is repeated.*

**Theorem 7.** *A graph $G = (V, E)$ is a tree if and only if it contains no cycles.*

*Proof.* See [11]. □

**Lemma 8** (Handshaking lemma)**.** *For an undirected graph $G = (V, E)$,*

$$\sum_{v \in V} \deg(v) = 2|E|.$$

*Proof.* See [12]. □

**Corollary 9.** *In an undirected graph $G = (V, E)$, the mean degree $\overline{\deg}$ of the vertices in $G$ is*

$$\overline{\deg} = \frac{2|E|}{|V|}. \tag{2.1}$$

**Lemma 10.** *For an undirected graph $G = (V, E)$*

1. *$|E| \leq \binom{|V|}{2}$;*
2. *for any $v \in V$, $\deg(v) \leq |V| - 1$.*

[1] $\mathbf{1}\{A\} = 1$ if $A$ and 0 otherwise

*Proof.*

1. The most number of edges is all the possible pairs of $|V|$ vertices, $\binom{|V|}{2}$;
2. each vertex can be neighbours with all vertices in $V$ except itself.

□

### 2.1.2 Erdos-Renyi random graphs

Erdos and Renyi outline in [13] a method of generating random undirected graphs $G = (V, E)$ with a fixed order $n := |V|$ and fixed size $N := |E|$, where the vertices $v_1, \ldots, v_n \in V$ are labelled so they are not exchangeable. As there are $\binom{n}{2}$ possible edges, the total number of possible graphs is $\binom{\binom{n}{2}}{N}$, and we wish to choose one uniformly at random. We present two methods for constructing such a graph, and prove their equivalence.

**Method 1**: We construct all $\binom{\binom{n}{2}}{N}$ graphs and pick one at random, each with probability $1/\binom{\binom{n}{2}}{N}$.

**Method 2**: We define a stochastic process that forms a graph in the following way. We have a vertex set $V$. At time $t = 1$, there are $\binom{n}{2}$ possible pairs of vertices, and we choose one $e_1$ uniformly at random. At time $t = 2$ we now have $\binom{n}{2} - 1$ possible pairs, and we select one $e_2$ uniformly at random. We continue in this way until time $N$ edges are selected. At some time $t = k + 1$ we have $\binom{n}{2} - k$ possible edges to pick from which are different from the $e_1, \ldots, e_k$ edges already selected. We select one possible edge with equal probability $1/\left(\binom{n}{2} - k\right)$.

**Theorem 11.** *Methods 1 and 2 are equivalent.*

*Proof.* Consider a vertex set $V$ where edges between vertices are selected independently at random with some probability $p$. The number of edges follows a binomial distribution $|E| \sim \text{Binomial}(\binom{|V|}{2}, p)$, and in the case that there are $N$ edges selected, each edge has an equal likelihood of occurring with probability $p^N(1-p)^{\binom{n}{2}-N}$. And so each graph of order $n$ and size $N$ is equally probable. In the case that we pick $N$ given edges at random with equal probability, we can consider it as an outcome of this method, which also creates a graph uniformly at random from the set of graphs with order $n$ and size $N$. In the method considered above, the stochastic process selects each edge independently with a probability of $\frac{N}{\binom{n}{2}}$, and so it creates a random graph uniformly in the same way. □

**Definition 12** (Erdos-Renyi graphs)**.** *We call a graph constructed by methods 1 or 2 an* ***Erdos-Renyi graph*** *of order $n$ and size $N$.*

**Theorem 13.** *For an Erdos-Renyi graph $G = (V, E)$ of order $n$ and size $N$, the mean degree of the vertices $\overline{\deg}$ is*

$$\overline{\deg} = \frac{2N}{n} \tag{2.2}$$

*and for a vertex $v \in V$, the distribution of its degree is given by*

$$P(\deg(v) = k) = \binom{n-1}{k} \left(\frac{\overline{\deg}}{n-1}\right)^k \left(1 - \frac{\overline{\deg}}{n-1}\right)^{n-1-k}. \tag{2.3}$$

*Proof.* Equation 2.2 comes from Equation 2.1. For a vertex $v \in V$, there are $n-1$ other vertices that $v$ can share an edge with. Given that each edge incident to $v$ has a probability $\frac{N}{\binom{n}{2}}$ of being selected, and that is independent of all other edges, then the degree of vertex $v$ is given by a binomial distribution, $\deg(v) \sim \text{Binomial}(n-1, \frac{N}{\binom{n}{2}})$. Now note that $\frac{N}{\binom{n}{2}} = \frac{2N}{n(n-1)}$ and substitute in Equation 2.2 to get the result for Equation 2.3. □

Note that the degrees of vertices are not independent of one another, as there is a fixed number of edges $N$.

## 2.2 Cryptography

Bitcoin is often referred to as a *cryptocurrency*, due to the elements of cryptography used in the underlying technology. To aid us in understanding, we will need knowledge of definitions and outcomes from this field. This will include the $P$ versus $NP$ problem, public key cryptography and hash functions.

### 2.2.1 $P$ and $NP$

The $P$ versus $NP$ problem is a major unsolved problem in theoretical computer science. It poses whether a large class of extremely hard problems can be solved quite quickly.

**Definition 14.** *A problem is a **decision problem** if it can be answered yes or no.*

**Definition 15.** ***P** is the set of decision problems that can be solved by a deterministic polynomial-time Turing machine.*

**Definition 16.** ***NP** is the set of decision problems that can be solved by a nondeterministic polynomial-time Turing machine; alternatively it is the class of problems able to be verified by a deterministic polynomial-time Turing machine.*

In simpler terms, $P$ is a group of problems that take an amount of time to solve that is proportional to a polynomial function of the size of the problem. These are generally thought of as easy problems that our computers are quite good at solving quickly. One example is the problem of deciding whether there exists a lowest common divisor of two numbers that isn't 1. It's quick to find an answer, and quick to check that an answer is correct.

$NP$ is a class of much harder problems. They don't need to necessarily be *solvable* in any polynomial order of time, but they can be *verified* in a polynomial order of time. An example decision problem is whether it's possible for the University of Melbourne to schedule its classes for all of its students in such a way as to guarantee that no one has over an hour to wait between classes and no student has a time clash with two classes at the same time. It's extremely hard to say if that's possible, but if given a solution, you'd be able to check very quickly by checking each student's timetable.

Clearly $P \subseteq NP$, but it is an open question whether $P = NP$ or not. It could be proven to be true if someone could demonstrate an algorithm that solves an NP problem[2] in polynomial time. If that were the case, then many problems we consider extremely difficult would become very easy to solve. In particular, if $P = NP$, then there exist trivial ways to break modern cryptography. If someone were able to show that $P = NP$ then most of our communication systems would collapse quite quickly for this reason. However, most in the field are confident that $P \neq NP$. But until that is proven, the practice will remain in computer science academic papers of leaving a footnote[3].

### 2.2.2 Public-key cryptography

Consider that I want to send you an important letter in the mail. When you read it, you might get suspicious that it wasn't from me, because anyone could have sent you that letter and written my name on it. To remove any doubt, I can sign that letter, and you can verify that the signature is correct (assuming signatures are unforgeable). We now outline a mathematical algorithm for digitally signing messages in an analogous way.

The sender randomly generates a number $k_{pr}$, and keeps it secret. This is her private key, not to be shared with anyone. She uses a function $K$ to generate her public key $k_{pu}$,

$$k_{pu} = K(k_{pr}). \tag{2.4}$$

[2] actually a problem in a related class called $NP - hard$.
[3] provided that $P \neq NP$.

She broadcasts the public key into a public place, so anyone can use it. Wanting to send a message, she uses a function $S$ to generate a *signature* $sig$ for the data, using her private key,

$$sig = S(\text{data}, k_{pr}). \tag{2.5}$$

Now, she sends the message to a recipient containing the data and the signature. In order to check that the data did actually come from her, the recipient checks the data and signature with a verification function $V$ and her public key $k_{pu}$,

$$ver = V(\text{data}, sig, k_{pu}). \tag{2.6}$$

By convention, $ver = 1$ if the signature for the data was signed with the private key $k_{pr}$ that generated the public key $k_{pu}$, and $ver = 0$ otherwise. With data representations $d_1, d_2$ and private keys $k_1, k_2$, we can summarise this as

$$V(d_1, S(d_2, k_1), K(k_2)) = 1 \iff d_1 = d_2 \text{ and } k_1 = k_2.$$

As long as we have functions $K$, $S$ and $V$ that interact in this way, then this is a way of guaranteeing that anyone we send messages to can verify that it came from us. We also require that it's intractable to find the inverse of the functions, otherwise it would be possible for anyone to find the private key, and so anyone could sign the message.

We can find functions that have these properties. In the Bitcoin protocol, their derivations come from methods utilising finite fields constructed over elliptical curves, specifically Koblitz curve secp256k1 [14]. These methods require that $P \neq NP$.

### 2.2.3 Cryptographic hash functions

Figure 2.2: Examples of the SHA-256 cryptographic hash function

```
> sha1('BITCOIN', algo='sha256')
[1] "a0505c4cd14f38f2a9b23da0fe09ad8e12ea920918a9df2cbd0d395fe21f39c0"
> sha1('BITCOIM', algo='sha256')
[1] "2a193d4a715ec57c2f4b89130a62379e020f11d29ed4640384bba038c517d30a"
```

The key idea behind a cryptographic hash function [15] is that it takes a data string of any size, and maps it to an output space of fixed size (such as a binary number in some range), in a way that seems like a uniform distribution, but always gives the same output for the same input. Cryptographic hash functions, specifically SHA-256 [16], are extensively used in the Bitcoin protocol.

**Definition 17** (Hash functions)**.** *A* ***hash function*** *is any deterministic function that maps data of arbitrary size to data of fixed size, called the* ***hash*** *of the data.*

**Definition 18** (Cryptographic hash functions)**.** *A* ***cryptographic hash function*** *is a hash function $H$ with the following properties:*

- *a small change in the data $d$ should yield a completely different hash $H(d)$, such that $d$ and $H(d)$ appear uncorrelated;*
- *$H$ is non-invertible. Given an arbitrary hash $h$, it should be intractable to find any $d$ such that $H(d) = h$ (the only way to do so should be a brute force approach of trying many different data arguments until the given hash is found);*
- *$H$ is collision-resistant. It should be intractable to find two inputs $d$ and $d'$ such that $H(d) = H(d')$ when $d \neq d'$.*

Given these properties, we can map every input in a way that appears uniformly at random to the output space. These functions can not exist if $P = NP$, so their existence relies on the assumption that $P \neq NP$. In the case of SHA-256, every input is mapped seemingly uniformly at random to a 256-bit binary number and it is intractable to find any input that maps to any given 256-bit binary number. These properties are essential for the proof-of-work process, as we can set a difficulty that defines the probability that any hash is successful, and any user can quickly verify that the hash is successful.

## 2.3 Accuracy metrics

Our stated goal in this thesis is to study ways to infer the topology of a network. Given a graph $G = (V, E)$, infer the edge set $E$. There are many possible guesses for $E$, so it may be better to think of this rather as a binary classification problem on the edges. That is, for some edge $e$, does $e \in E$? There are only two options, true or false. We conduct this analysis on every possible edge of which there are $\binom{|V|}{2}$, and construct our best guess for the edge set $E' = \{e : \text{we believe } e \in E\}$. Based on $e$'s membership of $E$ and $E'$, we can place it in one of four accuracy categories which describe the *confusion matrix* in Figure 2.3.

The accuracy of a prediction can be described by the total number of edges in each category. There are a number of useful interpretations that can be made using combinations of these totals. Different metrics are useful depending on the problem.

| Prediction | Truth $e \in E$ | $e \notin E$ | |
|---|---|---|---|
| $e \in E'$ | **True Positive** | **False Positive** | predicted Positive |
| $e \notin E'$ | **False Negative** | **True Negative** | predicted Negative |
| | actual Positive | actual Negative | |

Figure 2.3: Confusion matrix

In the medical field, it is common to use sensitivity and specificity [17], defined as

$$\text{sensitivity} = \frac{\text{True Positives}}{\text{actual Positives}}, \qquad \text{specificity} = \frac{\text{True Negatives}}{\text{actual Negatives}}.$$

Consider a diagnostic test to predict whether someone has a disease. Sensitivity would give the proportion of the population with the disease that the test captures. Specificity would give the proportion of healthy people that are identified correctly.

In the field of machine learning, it is common to use true positive rate (TPR) and false positive rate (FPR) [18], defined as

$$\text{TPR} = \frac{\text{True Positives}}{\text{actual Positives}}, \qquad \text{FPR} = \frac{\text{False Positives}}{\text{actual Negatives}}.$$

TPR gives the probability that a true case is captured. FPR gives the probability of a false alarm. Often in machine learning approaches, a hyperparameter can be tuned to adjust the prediction. As it does this, the TPR and FPR are altered. These values can be plotted as the Receiver Operating Characteristic (ROC) curve as seen in Figure 2.4. Quality of algorithms are often evaluated by comparing the area under this curve.

In the literature on Bitcoin topology inference we review in Chapter 5, we find that the researchers consistently use recall and precision,

$$\text{recall} = \frac{\text{True Positives}}{\text{actual Positives}}, \qquad \text{precision} = \frac{\text{False Positives}}{\text{predicted Positives}}.$$

It may have become apparent that sensitivity, TPR and recall are identical. It seems useful to know the proportion of total positives that are captured in the prediction.

The first paper [20] quotes numbers from their confusion matrix directly. The second [21] uses recall and precision. The papers that follow use the same, most likely so there is a common comparison point. These metrics are fairly uncommon in the broader literature, so it is interesting to see how these practices sustain themselves in a narrow field. It also likely indicates that the researchers in general haven't

[4] from [19]

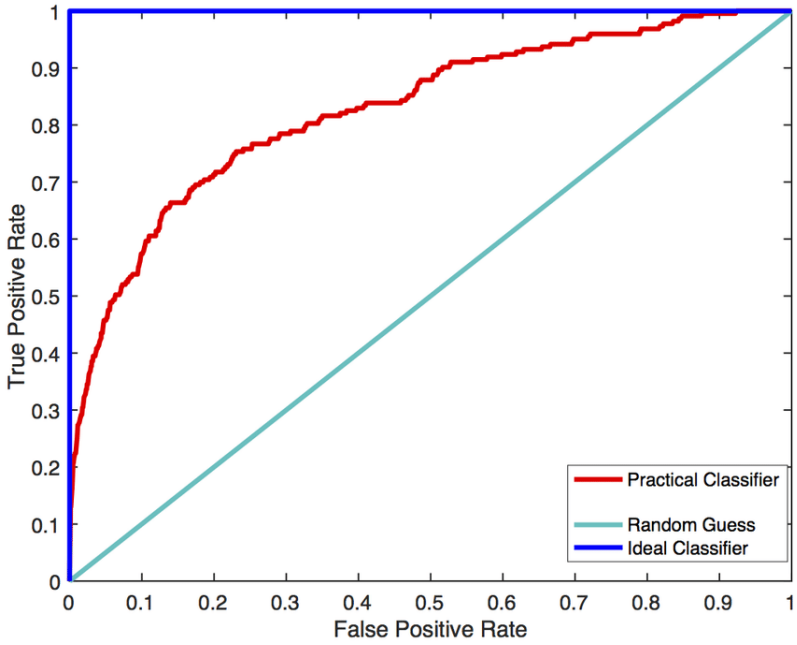



Figure 2.4: ROC curve[4]

looked at other fields for inspiration, which will become a recurring theme in this thesis.

One argument is hinted at in [21] as to why it is useful. If an adversary needs to know the peers of a given node to perform an attack such as a DoS, then they aren't so concerned with false positives. They would wish to maximise their recall to ensure that all the peers are found, and then perhaps place some bound on the number of false peers as a secondary goal by raising the precision.

The F1 score is sometimes used to combine recall and precision. It is defined as their harmonic mean

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

It can be useful to distil the confusion matrix into a scalar value such as F1. Given a vector of parameters $\boldsymbol{\beta}$ in a model for a classification problem, we can calculate F1($\boldsymbol{\beta}$), and so use optimisation techniques to find the values of $\boldsymbol{\beta}$ that maximise F1. We will use this approach in part III.

## 2.4 Optimisation algorithms

In Part III, we will utilise algorithms from the field of mathematical optimisation. Specifically, two algorithms using *local search* heuristics called *coordinate ascent* and *simulated annealing*.

### 2.4.1 Local search

**Definition 19.** *An **optimisation** problem is one where we have a set of feasible solutions $\chi$, a cost function with a scalar output $f : \chi \to \mathbb{R}$, and a goal $\in \{\min, \max\}$. We aim to find the solution $x \in \chi$ such that for any $y \in \chi$, $f(x) \leq f(y)$ if goal $= \min$ and $f(x) \geq f(y)$ if goal $= \max$.*

When optimisation problems are hard, for instance when they're in NP, we usually require a *heuristic* solution: a practical method that aims to find a good solution that isn't necessarily optimal, but still has a cost that is a good proportion of the optimal.

**Definition 20.** *In an optimisation problem, two feasible solutions $x_1, x_2 \in \chi$ are called **neighbours** if $x_1$ can obtained from $x_2$ by a small change in specification called a **local transformation**. The **neighbourhood** $N(x)$ of a feasible solution $x_i \in \chi$ is the set of all neighbours of $x_i$.*

In a local search heuristic, we acknowledge that there are often too many feasible solutions to analyse at any given time. We restrict our focus to the immediate area around our current solution and use a method to pick a new solution, then shift our focus to the immediate area around that solution. We chose this area with a local transformation that defines the neighbourhood of any given solution. By repeating this process, the aim is to steadily head in the direction of the optimal solution, or at least a good solution.

**Definition 21.** *In an optimisation problem, A **local optimum** is a solution $x \in \chi$ such that for any neighbour $y \in \mathcal{N}(x)$, the cost $f(x) \leq f(y)$ if goal $= \min$ and $f(x) \geq f(y)$ if goal $= \max$.*

A local optimum is the best solution in the small area an algorithm looks at, but that may be missing better solutions outside of the neighbourhood. There is no guarantee that local optima are good solutions to optimisation problems. They can be hazardous for local search heuristics due to their restricted neighbourhoods.

### 2.4.2 Coordinate ascent

Consider some set of feasible solutions $\chi \in \mathbb{R}^n$ in a maximisation problem. We define a local transformation of $x \in \chi$ as a change in only one of its coordinate values. The scope of that change defines the neighbourhood of a solution focusing on a specific coordinate value, $\mathcal{N}_i(x)$. For instance, given $x$ has coordinate in $i$ $x_i$, we could define the neighbourhood $\mathcal{N}_i(x) = \{y \in \chi | y_i \in [x_i - 1, x_i + 1]\}$. We

cycle through the dimensions $i \in \{1, \ldots, n\}$ changing the $i$th coordinate to the best solution in the neighbourhood, all other coordinates being kept fixed. We continue this process until we complete an entire cycle through the coordinates without any of them changing. The explicit description is explicitly given in the appendix in Algorithm 4.

### 2.4.3 Simulated annealing

Methods such as coordinate ascent tend to get trapped in local minima. Often the analogy is used (in the case of minimisation) that we need to sometimes climb hills to get to deeper valleys. Given that coordinate ascent never decreases the cost of the solution, it will never attempt to perform this kind of exploration that is sometimes necessary to find better solutions.

Simulated annealing [22] is an optimisation heuristic designed to perform exploration of the solution space in order to avoid local optima. It is based on a physical analogy from metallurgic annealing where metals are brought up to a certain heat to break their crystal structure, then through a controlled cooling process, recrystallise into a more uniform and lower energy state, giving desirable properties such as increased hardness and ductility. Applying this to optimisation problems, we initially bring the process to a high initial "temperature" $T_i$, which controls how likely the search algorithm is to move to worse solutions in the solution space. We pick a solution in the current solution's neighbourhood, if the solution is better, we move to it, and if it is worse, we still move to it with positive probability that depends on the current temperature. As the algorithm progresses, the temperature is lowered by a cooling schedule $s(T)$, reducing the probability that we move to worse solutions. By using this method, the heuristic has a positive probability of leaving local minima. By the time it reaches its final temperature $T_f$ after $I$ iterations, there is a very low probability of moving to worse solutions, so we end up in a local optima in hopefully a better region of the solution space. There is a large body of literature surrounding simulated annealing, including on variants of the basic algorithm and on the best selection of hyperparameters and a cooling schedule to fit various problems.

# Chapter 3

# Bitcoin Preliminaries

In this chapter we discuss what Bitcoin is, the problem it attempts to solve, and how it achieves that. We will describe the Bitcoin ecosystem including its users, how they communicate, the rules they follow, and why users follow those rules.

Bitcoin is an electronic payment system, theorised in 2008 as a white paper outlining its theory [1], and first launched in 2009. While there had been many previous attempts to create digital currencies based on cryptography, Bitcoin was the first to successfully work and provide a proof of concept, which it achieved through the introduction of blockchain technology. A blockchain provides a solution to the *double-spend problem*, while allowing a completely decentralised design and incentivising all parties to participate in the system. Every participant keeps a copy of a ledger of all past transactions and follows a set of rules such that a consensus on the ledger is reached as new transactions are added.

Bitcoin is comprised of two parts: the theory of the Bitcoin blockchain and its implementation. A *protocol* is an implementation that dictates the specifics of how users communicate with each other, the data structures that are used to store information, and how the information is processed. Although all users must use the same communication methods and data structures to make the system work, different nodes may follow different rules for processing the information. However in practice, most users utilise a Bitcoin *client*: software that manages all of this for them. While in theory there are any number of protocol that could be implemented, at the time of writing, over 98% of nodes that are publicly reachable use an implementation of a client called Bitcoin Core[1], and so we will refer to the Bitcoin Core implementation as the *Bitcoin protocol*.

[1] `https://bitnodes.earn.com/nodes/`

## The double-spend problem

In a digital currency system, we want to facilitate transactions; for instance a payer giving one coin to a payee. Using public-key cryptography, it is easy to verify that the payer has authorised the transaction. A payer can sign a transaction with their private key, and the payee can verify that the signature and transaction came from the payer. However, the payee cannot guarantee that the same coin hasn't been spent somewhere else already, and won't be spent again. Normally this problem is solved by introducing a trusted (and usually centralised) third party, such as a bank. Being able to address this without introducing trust in a third party had been a long standing problem. This is the double-spend problem that the original white paper [1] addresses.

The solution proposed is to be aware of all transactions. This is achieved by transactions being publicly broadcast, and a system for all participants to arrive at a consensus on the state of a ledger, discarding any invalid transactions such as double-spends. All then that is needed for the payee to have confidence in the transaction is proof that the transaction is part of the consensus, and that the consensus can't be changed in the future.

## General approach for a P2P network

When a node wishes to make a transaction, spending some money, they broadcast a message detailing the transaction to their neighbours, which eventually floods the network. Each node checks that the transaction doesn't spend money that has already been spent somewhere else, and adds it to a personal collection of transactions called a block.

Nodes can send their blocks onto the network once they verify it. The process of finding a solution to the verification problem is difficult, and requires a lot of work to solve. However if a node solves it, they are rewarded. A verified block represents a manifestation of the amount of work it took solve the problem. It is a *proof-of-work*. The verified block is sent onto the network, where the other nodes quickly check it for verification, and it is accepted as the next block on the history of accepted blocks, called the *blockchain*. Each node tries to get their blocks onto the blockchain so they can collect the reward, and so an important feature of blocks is that they refer to the previous block in the chain. Apart from being a record of all transactions, the blockchain represents the total proof-of-work of solving the problem of verifying all its constituent blocks. The longer the chain, the more work it took to create that chain.

The guiding principle of the network arriving to a consensus of accepted transactions is to trust the largest proof-of-work; trust the longest chain. This solves the double-spend problem as each block contains no transactions that double-spend any other transaction in the blockchain, and due to the amount of work that would need to be proved, it is highly unlikely that anyone can double-spend any transaction in the future, provided that no individual actor holds a significant proportion of the network's total computing power, as we will discuss in Section 3.1.6.

## 3.1 The Bitcoin blockchain

Here we will describe in detail the theory underpinning the Bitcoin blockchain including the structure of the ecosystem, the the distributed ledger of transaction, how the ledger evolves and the principles underlying this process. Readers familiar with Bitcoin may want to skip to Section 3.2 for a specific protocol details that are utilised in Chapter 5.

### 3.1.1 P2P network

A key feature of the Bitcoin network is that it is decentralised, in that users do not need to trust a central organisation or entity. A computer connected to the Bitcoin network is called a *node*. Two nodes that are connected, sharing information, are called *peers*. A peer-to-peer network is distinguished from a client-server network where clients communicate directly with trusted, centralised servers; for instance a customer requesting a money transfer from a bank. Most nodes are established by users wanting to use the Bitcoin currency and nothing else, and are known as *partial nodes*. We will only study the P2P network of the *full nodes*: nodes that participate fully in the ecosystem by verifying transactions and blocks, and transmit information further into the network; requiring them to keep a full up-to-date copy of the blockchain. At the time of writing, there are estimated to be 9694 publicly accessible full nodes [23]. From here we mean full nodes when we say nodes.

It is important to note that when transactions are passed across this network, the details of the payers and payees are not in reference to their IP addresses[2] but rather to their Bitcoin addresses; hashes of their respective public keys. We will refer to an individual's Bitcoin address as their *pseudonym*, and their IP address as their *address*. The problem of deanonymisation in Bitcoin is the matching of addresses with pseudonyms: finding out which computers are performing which transactions,

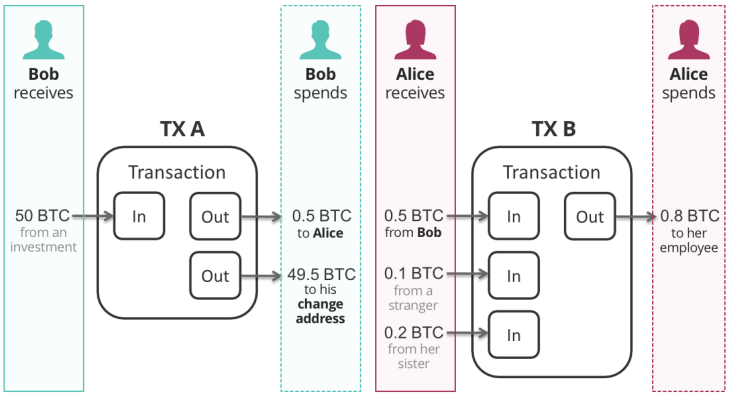
[2]Internet Protocol address: the unique identifier for a device connected to the internet

and how much Bitcoin currency they have.

### 3.1.2 Transactions

Transactions are records that reassign Bitcoins from one pseudonym to another. They consist of inputs, references to the payer's funds received from previous transactions; and outputs, a list of pseudonyms and funds that the payer wishes to send those funds to. The input of one transaction is the output of a previous transaction. Each output is signed by the payer using their private key, such that anyone can verify that the payer wished to spend it.

Figure 3.1: Transaction examples[3]




It is important to understand that a traditional notion of a bank account does not exist in this system. A user does not refer to a pool of funds that they have access to. Rather, there is a collection of unspent transaction outputs known as `UTXO`s in the system, and if an `UTXO` refers to a user, then the user can spend it by creating new outputs with the original output as an input. Note that you cannot spend part of an input. The whole input must be spent, however a user may refer to themselves in the new output they create, thus preserving their funds.

### 3.1.3 Verification and conflicting transactions

Transactions are sent through the P2P network, and are verified by nodes before being passed to their peers. The verification has two parts:

1. Verification of the signature of each input, using the public key associated with the pseudonym of the payer of each input;

[3]from `https://freedomnode.com/guides/17/how-bitcoin-works`

2. Verification of the validity of the transaction, checking that the sum of inputs is greater than the sum of outputs, and that each input and that each input is an UTXO. If the sum of outputs is greater than the sum of inputs, then this is interpreted as a transaction fee which is reassigned at the discretion of whoever mines the block that contains the transaction, usually to the miner themself.

For a given $\texttt{UTXO}_i$, if two transactions have $\texttt{UTXO}_i$ as an input, then we call those transactions *conflicting*. If a set of transactions all have an input $\texttt{UTXO}_i$ as an input, we call that set *mutually conflicting*. If more than one of these transactions is included in the blockchain, then this would be a double-spend. The structure of transactions and verification problems were made to address this problem. Once a node receives a transaction that spends $\texttt{UTXO}_i$, that node no longer recognises $\texttt{UTXO}_i$ as unspent. So if it receives another transaction spending $\texttt{UTXO}_i$, then it will not recognise it as only spending from the collection of all UTXOs, and so it will be *dropped* by that node: The node will not store it and not propagate it into the network.

### 3.1.4 Blocks

Nodes work on creating blocks, collections of transactions that haven't appeared in previous blocks. This creation process is known as Bitcoin mining, with a node participating in the process referred to as a miner. Blocks always refer to a single previous block, forming a chain of blocks, which is where we get the term blockchain. It is usually in the miner's interest to link their block to the most recent block on the longest chain they are aware of. If they don't, the block risks becoming a fork in the blockchain that will become ignored by future users as it is not part of the longest chain. In this way, the blockchain acts as a ledger of all accepted transactions.

The details of the structure of a block can be found in [24]. Key to understanding their use is that they contain:

- a list of transactions;
- an extra *coinbase transaction* with no inputs that consists of a block reward, newly introduced Bitcoins that are given to the miner. Initially 50 Bitcoins, this reward halves every 210,000 blocks (roughly every four years) and will be set at 0 once 21 million Bitcoins have been mined (created in coinbase transactions);
- a timestamp of the creation of the block;
- a hash referring to a previous block;

- a 32-bit nonce which is a number used in the mining process. Miners adjust this number in order to find a successful hash.

### 3.1.5 Mining blocks

To mine a block, a miner must solve a computationally difficult problem to demonstrate a proof-of-work. In return, they receive any transaction fees and the coinbase transaction as a reward. This incentivises the miners to include transactions with the largest fees, and in turn incentivises payers to include higher fees in order to have their transactions included in the blockchain.

A hash of a block $H_i$ can be computed in the following way:

$$H_i = \text{hash}(H_{i-1}, \text{transactions, timestamp, nonce})$$

where arguments of the hash are concatenated, and the hash takes the form of a binary number. The miner must find a valid hash by repeatedly adjusting the nonce, such that $H_i < d_i$. The difficulty threshold $d_i$ of finding a valid hash is determined by the current mining difficulty: a 256-bit binary number adjusted every 2016 blocks (roughly two weeks) in order to keep the mining rate at approximately one block every ten minutes. There is no better way to solve this problem apart from a brute force approach, assuming that $P \neq NP$.

Using this method, the amount of work gone into finding a block is quantifiable. The current[4] threshold for a valid hash is[5] $d_i = 2.11 \times 10^{54}$. Miners repeatedly create hashes by adjusting the nonce until they find a successful hash. This is equivalent to i.i.d. sampling without replacement of a Bernoulli variable with success probability $p = \frac{2.11 \cdot 10^{54}}{2^{256}} = 1.82 \times 10^{-23}$. This process can be modelled as a geometric distribution, which gives probabilities for the number of trials until the first success. The expected value of a geometric distribution is $\frac{1}{p}$, which would correspond to $5.48 \times 10^{22}$ hashes. To give this some perspective, the universe is currently $4.35 \times 10^{17}$ seconds old. The number of hashes on average to mine one block corresponds to an enormous amount of computational power. The energy usage is currently estimated in [25] to be 73.121 TWh per year. The key principle here is that an adversary would have to prove a similar amount of work to these levels in order to overcome the consensus. Interestingly and perhaps concerningly, the $CO_2$ emissions of the Bitcoin ecosystem are comparable to that of Denmark.

[4] `https://www.blockchain.com/charts/difficulty`

[5] The *difficulty* defined $D_i$ in the Bitcoin protocol is different. Here we use the target $d_i$ which is calculated as $d_i = d_{\max}/D_i$, where $d_{\max} = 2^{224}$ is the maximum threshold. The Bitcoin difficulty at the time of writing is $D_i = 1.28 \times 10^{13}$.

Once a block has been mined, the miner can broadcast it onto the network by sending it to its peers. Nodes validate the block by checking the hash (thereby ensuring the proof-of-work), verifying the individual transactions and checking for double-spends.

It is a common tactic for many computers to be used by an individual or group to find blocks. These are called *mining pools*, and they usually don't interact directly with the network, but rather through communicate with the network through a gateway node that otherwise functions as a normal node. We can just model that gateway node as holding the computational power in that instance and ignore the mining nodes in the pool.

### 3.1.6 Block acceptance

When a full node receives a new block, the rule they follow is to accept the block if it forms part of the longest blockchain, representing the largest combined proof-of-work. Where two blocks are both part of separate, but equally long chains, the rule is to accept the block that they received first. It is possible that the blockchain will fork, caused by two or more blocks being mined before any are fully propagated. This is an artefact of the propagation delay, which is studied in [26]. At that point, the different parts of the network mine new blocks linking to the block they have accepted, until one side of the fork forms a longer chain than the other, and for a long enough time for that information to spread through the network. For this reason, it is recommended that a user should not have confidence in a transaction being part of the consensus until a sufficient time has passed (around 1-2 hours) such that it is very improbable that the block the transaction is contained in is part of a fork that may become redundant.

To understand why this is a stable consensus, consider an adversarial miner, who we assume only has a small fraction of the computational power of the whole network, who wishes to alter the consensus. That is, they wish to form a blockchain long enough that all the nodes switch from accepting the consensus blockchain to the adversarial one. This adversarial miner would have to link a block to some previous block in the consensus blockchain, and then work at linking more blocks to that fork until it is longer than the consensus chain. All the while, every other miner in the network is working on making the consensus chain longer.

Nakamoto analyses the probability of the adversarial miner catching up in the original paper [1] with a proportion $p$ of computational power of the network, average time to find a node $T_0$, length of the consensus chain past the fork $n$ and length of adversarial chain past the fork $m$. The time to find $n$ blocks on the consensus chain

takes on average $\frac{nT_0}{1-p}$. He uses that mean to make a assumption that simplifies a model for the number of blocks that the adversary will find in the same time, which he calculates to be Poisson distributed with mean $\frac{np}{1-p}$. Using this, and modelling the difference between the length of the consensus and adversarial chains as a continuous-time Markov birth-death process, he arrives at a probability of $\min(1, (\frac{p}{1-p})^{n-m})$.

Rosenfeld [27] corrects this analysis by noting that this assumption isn't needed for an analytic solution. As the probability that the attacker successfully mines a block is described by a Bernoulli random variable with probability $p$, He models $m$ more accurately as the number of successes until $n$ failures (the consensus chain finding a block). This is described as a negative-binomial random variable: $m \sim$ NegativeBinomial($n$,$p$). He then calculates that if someone waits for $n$ blocks before considering a transaction confirmed, the probability that the adversary can perform a double spend is $\min\left(1, 1 - \sum_{m=0}^{n} \binom{m+n-1}{m}((1-p)^n p^m - (1-p)^m p^n)\right)$.

For values of $p$ over 0.5, the adversary will eventually catch up with probability 1. The reccomended wait time to confirm a transaction is generally at least 1 hour, corresponding to at least 6 blocks on average. Here, the probability of catching up with roughly 40% of the computing power is roughly 40%, and drops quickly with less probability. This would mean for those with less than 40% of the network computing power would be better off contributing to the consensus chain if they are seeking to maximise their expected reward. However this strategy still has positive probability if one wishes to re-spend a specific transaction.

However assuming only a small percentage of total network computational power, the probability of an adversary being successful with this attack is minuscule as they will never be able to demonstrate a proof-of-work that is greater than the rest of the system combined. It is intractable to alter the consensus blockchain for this reason, giving rise to two main properties of Bitcoin:

- All parties trust and are incentivised to add to the longest blockchain.
- All past transactions are immutable. Once a transaction is part of the consensus, it is intractable to cancel it.

## 3.2 The Bitcoin protocol

In this section we will outline some features of the Bitcoin protocol that are necessary to understanding some tactics common to the inference methods used in Chapter 5.

The code for Bitcoin Core is publicly available. It has a large developer community, with changes being approved once issues are discussed and code is reviewed and tested before being implemented.

### 3.2.1 Nodes

Nodes hold a list of the IP addresses of their peers. By default they manage 8 outgoing connections and up to 117 incoming connections; up to a total of 125. Most information is transmitted in both directions along these connections, with outgoing and incoming not being differentiated. in the discussion of nodes and full nodes in Section 3.1.1, we can differentiate full nodes (active participants in the ecosystem) as those nodes that accept incoming connections.

### 3.2.2 Node discovery and new peers

A list of "good" nodes is kept, and hardcoded into the Bitcoin Core client. Upon joining the network, a new node will randomly select some of them as their outgoing connections. They will then begin adding nodes to their list of known addresses called `addrMan` (not necessarily connecting to them) through the use of `ADDR` messages, which is described in detail in Section 5.1.1. When needing to form new connections, a node randomly selects from their list. The intention of this is to construct a random graph, which likely lead to healthy properties and consistency in the network.

### 3.2.3 Propagating transactions

Nodes keep a data structure called `memPool` that records all valid transactions the node has received but that are not yet included in blockchain. Propagation of transactions between peers is a three step process, with initial communication of transactions as hashes in order to reduce bandwidth.

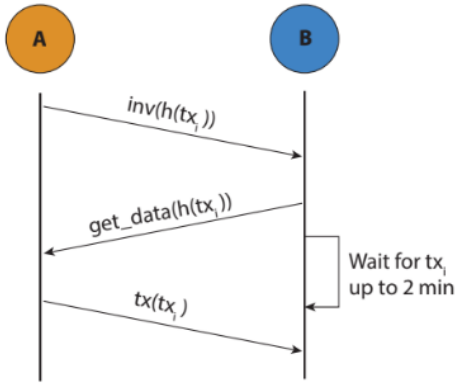



Figure 3.2: Three step process for forwarding transactions from A to B.[6]

[6]from [28]

1. `INV`: When a node receives or creates a new transaction, it records it in its `memPool`. Periodically, the node sends a hash of these new transactions to each of its peers with an `INV` message.

2. `GETDATA`: When a peer receives an `INV` message, it checks the hashes of the transactions in its own `memPool`. If an item of the `INV` message is not in its `memPool`, then the peer will respond with a `GETDATA` message containing the hashes of the transactions that it does not already have.

3. `TX`: Upon receiving a `GETDATA` message from its peer, a node will send the transactions that are requested to that peer.

A node may receive multiple `INV` messages of the same transaction from different peers in a short space of time, and so they create a queue of those peers. The node sends a `GETDATA` message to the first node in the queue, then waits two minutes to receive a `TX` message before moving down the queue and sending another `GETDATA` request. The decision to include this was likely made because to account for time taken to send large transactions.

### 3.2.4 `InvBlock`

Miller and 6 other authors describe a clever technique in [20] that takes advantage of this three step process to block any given node from receiving a transaction that you make for a period of time.

Consider three nodes, $v_i, v_j$ and $v_k$ who are all peers, and two transactions $tx_1$ and $tx_2$, that $v_i$ is unaware of. If $v_j$ sends $\texttt{INV}(tx_1)$ to $v_i$, who responds with $\texttt{GETDATA}(tx_1)$ back to $v_j$. After one minute, $v_i$ has still not received $\texttt{TX}(tx_1)$ from $v_j$, but now $v_k$ sends a message $\texttt{INV}(tx_1, tx_2)$ to $v_i$. Two minutes have not elapsed yet, so $v_i$ cannot request $tx_1$ from $v_k$, so just sends $\texttt{GETDATA}(tx_2)$ to $v_k$, who responds quickly with $\texttt{TX}(tx_2)$. Once the two minutes since sending $\texttt{GETDATA}(tx_1)$ to $v_j$ has elapsed, $v_i$ goes to the next node in the queue for $tx_1$, which is $v_k$, and sends them $\texttt{INV}(tx_1)$, which is resolved with $\texttt{GETDATA}(tx_1)$ and $\texttt{TX}(tx_1)$ messages.

This might seem like a needlessly complex arrangement, as in an alternate protocol, $v_i$ might be allowed to send $\texttt{GETDATA}(tx_1, tx_2)$ to $v_k$ after the `INV` message was received from them. However if $v_j$ had eventually sent $\texttt{GETDATA}(tx_1)$ within the two minutes, then the full message $\texttt{TX}(tx_1)$ would have been sent from both $v_j$ and $v_k$ to $v_i$, which is something we want to avoid as `TX` messages are much larger so we want to avoid taking up bandwidth across thousands of needless `TX` messages. There might be other ways to design this in a way that doesn't leave a 2 minute gap,

but this method works and doesn't interfere with what is a very complex system of many simultaneous transactions being broadcast.

The researchers proposed the following `InvBlock` technique. Consider that we have some node $v_a$ that we control, and we generate a transaction $tx_a$ that we want to send into the network. Suppose that there is some node $v_z$ and we want to guarantee that it doesn't know about $tx_a$ for a period of time, at most two minutes. We send `INV`($tx_a$) to $v_z$, then wait for a responding `GETDATA`($tx_a$) message. Then we send $tx_a$ in the normal three step process to all our other peers. Given that $v_z$ has already sent the `GETDATA` message to $v_a$, it will not send another to any other node for at most 2 minutes after we received that message. So as long as we don't send them `TX`($tx_a$), we can guarantee that they don't know about $tx_a$, and so within that time period we can control when they find out about it.

It is interesting to note that this from this pragmatic method to reduce bandwidth on the network, these researchers were able to find a significant vulnerability in the system.

### 3.2.5 The Bitcoin testnet

The Bitcoin testnet is a second implementation of the Bitcoin blockchain where it is agreed that the currency has no value. The Bitcoin protocol is only slightly different, with details found in [29]. Its purpose is to exist for testing protocol changes and other experimentation.

# Part II

# Topology Discovery

# Chapter 4

# Problem Definition

At any given time, there exists a network of full nodes that are running Bitcoin. This network changes as nodes enter or leave the network and make new connections. At any given time, we wish to know the topology of this network. We can model the nodes as a set of vertices $V$, and the connections as directed edges $E$. An unknown proportion of the nodes are publicly *reachable*, meaning that we can find their addresses and connect to them. We assume that reachable nodes make up the vast majority of the network, and denote their set $V_p$. We may set up our own nodes, denoted by the set $V_a$, and connect them to the reachable nodes $V_p$ to receive messages and take messages. We denote the edges incident to vertices in $V_a$ as the set $E_a$.

We can formally describe the problem of finding the Bitcoin P2P network topology as: Given $V_p$, $V_a$ and $E_a$, find the graph $G = (V, E)$.

## Concessions

While the Bitcoin P2P network is best modelled by a directed graph due to its distinction of incoming and outgoing peers, for most outcomes outlined in the introduction it may be sufficient just to find which edges exists, and not necessarily in what directions they point. For this reason, it will be sufficient to model the graph as undirected, and if we can ascertain edge directions then that is bonus information. The vast majority of the time, the edges do not distinguish between incoming and outgoing peers. For instance, the propagation of transactions and blocks occur in both directions on an edge. However in some cases, it is important to distinguish direction, for instance in the case of address propagation, as described in Section 5.1.1.

We may also have to limit our capacity for complete knowledge of the network at any given time period. Most methods will measure over a period of time $[t_1, t_2]$, in

which nodes may join or leave the network, and new edges may be formed because of that. So we may have to limit out maximum knowledge to those nodes in $V$ that were in the network for the entire time $[t_1, t_2]$.

It may also be sufficient to restrict ourselves to finding the graph of reachable peers $G_p = (V_p, E_p)$, where $E_p$ is the set of edges between the reachable peers.

## Tools

We have a number of tools that are available to us to help us in finding the topology. These are wide and varied. For instance, we could try to email users to ask them to send us information on their connections as part of a research project. However we want to discuss reliable methods, which will mostly be restricted to information that peers will send us that conforms with the Bitcoin protocol. This can take the form of transaction messages, block messages, address requests, and a plethora of other messages part of the protocol including messages establishing connections, synchronising systems, and many others. We can also measure the times that we receive these messages, which will be the focus of the statistical models.

## A note on transaction fees

Most of the techniques in this thesis will focus on transactions. If we wish to send transactions we will encounter transaction fees. This may place a financial limitation on methods that actively send transactions.

As noted in Section 3.1.2: for any transaction, the difference between the sum of input funds and sum of output funds is the transaction fee. If that transaction is included in a block, then the miner of that block designates where those funds go, most likely to themself.

Before propagating a transaction, nodes check that transactions fees meet a transaction fee. This is likely a countermeasure to preventing the flooding of the network with functionally empty transactions. Bitcoin Core has a default setting of $10^{-5}$ Bitcoin per kB. Given an average transaction size of 500 bytes and the current exchange rate, each transaction costs about 0.043 USD: [30].

Transactions have a size in memory, so we can define a transaction's fee/kB. Given that blocks have a limited size, miners choose available transactions to populate the block based on maximising the total transaction fee, likely including transactions with high fee/kB values[1].

[1] Actually this is an example of a knapsack problem from the field of combinatorial optimisation [31]. The problem is NP-hard. However, it is much more likely that large fee/kB transactions will be included in the solution and heuristic solutions will likely include them anyway.

The question arises of if there exists a "no-man's-land" of fee/kB that is high enough to be propagated but low enough to never be included in a block, so never pay the fee. I will leave that question, or the problem of finding the probability of a transaction being included in a block given the minimum transaction fee to future research.

# Chapter 5

# A Review of Prior Research

In this section we will conduct a literature review on the topic of finding the Bitcoin P2P network topology. By exploring these papers chronologically, we will see how methods have built on each other. This is also the best lens through which to observe the recursive reactions of the researchers and developers, who sit on opposite sides of the problem. The developers want to prevent inference of the topology to prevent attacks and preserve the anonymity of its users. The researchers wish to study the network for varied reasons including monitoring the health of the network, but also in doing so provide knowledge of weaknesses to the developers so they can improve the protocol. By studying the methods and how the developers react, we will be able to gain some clarity on the best future path for approaching this problem.

The methods fall into two major categories. *Idiosyncratic* methods take advantage of some quirk in the Bitcoin protocol. *Statistical* methods use statistical inference on timing measurements.

We can classify methods as *node-specific* or *network-wide*, depending on whether they target a specific node to determine their peers, or infer the whole network topology simultaneously. Of course a method that is node-specific can be made network-wide by repeating it on every node, but it is a useful distinction to make. In general network-wide methods can't be scaled down to infer the peers of an individual node.

We can also classify the methods as *active* or *passive*, depending on whether we propagate transactions into the system or not. If a method involves sending messages to nodes that don't propagate into the network further than the recipients, then we classify it as a passive approach; otherwise we would have to classify all methods as active due to various messages that need to be sent between nodes, for instance when forming a communication connection to become peers.

The chronology starts with a paper released in 2014 [8] by Biryukov, Khovra-

tovich and Pustogarov which introduced the possibility of deanonymising users (linking addresses with pseudonyms) given knowledge of a node's peers. Towards the end of the paper, they propose a topology inference method based on how nodes learn of new addresses, which was node-specific, passive, idiosyncratic, and quite invasive. The first paper we look at took their concept and greatly refined it, providing the first full implementation and a snapshot of the Bitcoin P2P network.

## 5.1 2015: Miller and 6 other authors

### [20] Discovering Bitcoin's Public Topology and Influential Nodes

In [20] Miller and 6 other authors developed a passive, network-wide, idiosyncratic method of discovering the P2P network by exploiting a vulnerability in a way that nodes stored known addresses of other nodes. The algorithms they developed were extremely effective.

They developed *CoinScope*, a software for large-scale Bitcoin experiments, *AddressProbe*, an algorithm to discover the P2P network topology of Bitcoin, and a decloaking method for discovering influential nodes that are well connected to hidden mining pools. At the time, they discovered that only 2% of nodes were responsible for relaying about 75% of the mining power.

Due to the success of these methods, and a worry that this would lead to deanonymisation of the network, the protocol was changed to make the AddressProbe algorithm infeasible in 2015[1].

### 5.1.1 Addresses and timestamps

The algorithm AddressProbe that they created relied on an old methodology the Bitcoin protocol used to update timestamps associated with known addresses. Recall from Section 3.2.2 that nodes keep a list `addrMan` of known addresses. Note that this list contains all the node's peers, as well as any other nodes that it is aware of. Along with each address is an associated timestamp, which is used to keep track of when that address was last seen and allows nodes to keep track of nodes that drop out of the network. Each node also keeps a list of addresses they know their peers have in their `addrMan`, which is purged every 24 hours. A node can send a `GETADDR` request of addresses to a peer, who can repply with an `ADDR` message containing up to 1000 entries from its `addrMan` . Before the 2015 patch, a set of

[1] https://github.com/Bitcoin/Bitcoin/commit/9c2737901b5203f267d21d728019d64b46f1d9f3

rules were followed to update the timestamps in a node's `addrMan`. While normally peers send messages in both directions symmetrically, here is a case where nodes distinguish between incoming and outgoing peers.

**Timestamp updating rules**

1. For outgoing peers, update the timestamp every time you receive any message;
2. For incoming peers, set the timestamp when the connection is made;
3. When learning any address through `ADDR` messages (including addresses of outgoing and incoming peers), age the address by adding a two hour penalty to the associated timestamp. If the address is already known, only update the timestamp if the new timestamp is 20 minutes younger than the old one, and age it two hours.

Normally, it is only permitted to send `ADDR` messages as a response to a `GETADDR` message from a peer. To facilitate the propagation of new addresses through the network, it is permitted to send an `ADDR` message without receiving a `GETADDR` message in two situations.

**Unsolicited `ADDR` rules**

1. When receiving a new peer connection, send an `ADDR` message to a randomly chosen peer, with information of only the new peer;
2. When receiving an `ADDR` message with fewer than 10 entries, send the same `ADDR` message to two randomly chosen peers, as long as you believe those peers already have that information in their `addrMan`. Also do not age the timestamp.

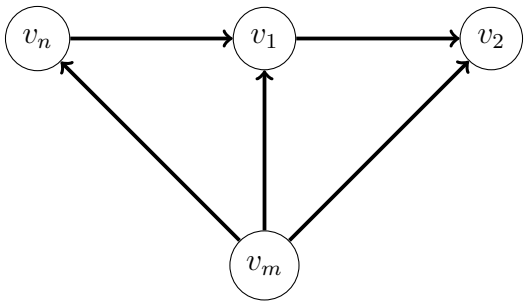
This way new node addresses flood the network, and when existing nodes become peers the timestamp of those peers is updated to those nodes' other peers by propagating some distance into the network.

### 5.1.2 Inference

Consider a node $v_1$ and its outgoing peer $v_2$. A new node $v_n$ joins the network, and connects to $v_1$ at some time $t$. $v_1$ sends an unsolicited `ADDR` message to $v_2$ containing address $v_n$ and timestamp $t$. From the timestamp updating rules, we have:

- $v_n$'s timestamp of $v_1$ will always be (nearly) current as long as the connection is maintained;
- $v_1$'s and $v_2$'s timestamps of $v_n$ will stay at $t$, unless they hear of a timestamp for $v_1$ more recent than $t$+20mins from another node.

We set up a node $v_m$, and connect to all three nodes. We send `GETADDR` messages to all nodes in the network, and record all the timestamps in their response `ADDR` messages. We have no knowledge of the time $t$ or the connections, and we wish to infer the connections. If a timestamp is less than 2 hours old, we call it *current*. If a node $v_i$ has a timestamp for $v_j$ that is different from every other node's timestamp of $v_j$, we say $v_i$'s timestamp of $v_j$ is *unique*. Consider the following three outcomes:




1. If $v_n$ sends $v_m$ a timestamp of $v_1$, it will be current;
2. If $v_n$ sends $v_m$ a timestamp of $v_1$, it will be unique, as $v_n$ is constantly and independently updating its timestamp of $v_1$;
3. If $v_1$ and $v_2$ send $v_m$ timestamps of $v_n$, it is possible that they will be the same. For instance if they are both still set to the connection time $t$.

From these three outcomes, we can deduce three general inference rules for any two nodes $v_i$ and $v_j$.

**Inference rules**

1. If $v_i$'s timestamp of $v_j$ is not current, then there is no connection from $v_i$ to $v_j$. This comes from the contrapositive of outcome 1;
2. If $v_i$'s timestamp of $v_j$ is current and unique, then there is a connection from $v_i$ to $v_j$. This comes from considering both outcomes 1 and 2;
3. If $v_i$'s timestamp of $v_j$ is current but not unique, then we are unsure whether there is a connection between $v_i$ and $v_j$. This comes from outcome 3, considering that both $v_1$ and $v_2$ report the same timestamp.

### 5.1.3 AddressProbe

Miller et al. represent this information in the following matrix. We indicate the age of the timestamp with "ts". $v_i \rightarrow v_j$ indicates $v_i$ has an outgoing connection to $v_j$, $v_i \not\rightarrow v_j$ if $v_i$ does not have an outgoing connection to $v_j$, and $v_i \stackrel{?}{\rightarrow} v_j$ if we are not sure. Note that if we just desire the undirected graph, we only need one $\rightarrow$ connection in a table entry.

| $v_i$'s ts of $v_j$ | $v_j$'s ts of $v_i$: ts $\geq$ 2hr | $v_j$'s ts of $v_i$: ts $<$ 2hr & unique | $v_j$'s ts of $v_i$: ts $<$ 2hr & not unique |
|---|---|---|---|
| ts $\geq$ 2hr | $v_i \not\rightarrow v_j$<br>$v_j \not\rightarrow v_i$ | $v_i \not\rightarrow v_j$<br>$v_j \rightarrow v_i$ | $v_i \not\rightarrow v_j$<br>$v_j \stackrel{?}{\rightarrow} v_i$ |
| ts $<$ 2hr & unique | $v_i \rightarrow v_j$<br>$v_j \not\rightarrow v_i$ | $v_i \rightarrow v_j$<br>$v_j \rightarrow v_i$ | $v_i \rightarrow v_j$<br>$v_j \stackrel{?}{\rightarrow} v_i$ |
| ts $<$ 2hr & not unique | $v_i \stackrel{?}{\rightarrow} v_j$<br>$v_j \not\rightarrow v_i$ | $v_i \stackrel{?}{\rightarrow} v_j$<br>$v_j \rightarrow v_i$ | $v_i \stackrel{?}{\rightarrow} v_j$<br>$v_j \stackrel{?}{\rightarrow} v_i$ |

All that is left to do given this analysis is to send multiple `GETADDR` requests to all nodes on the network. The researchers built an infrastructure called CoinScope to manage these experiments in collecting the data.

### 5.1.4 Accuracy

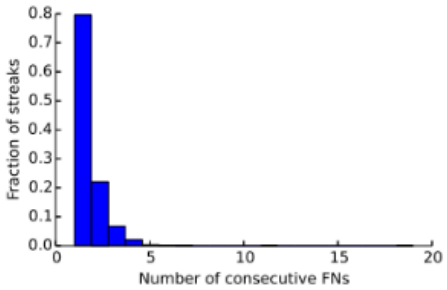
The researchers set up five validation nodes, and ran experiments every 2 minutes for 18 days. The accuracy statistics reported in Figure 5.1 quote directly from the confusion matrix, and take the average values across all the experiments with 95% confidence intervals. True negatives are not reported as they are not of as much interest and can be calculated from the other values and number of total reachable nodes.

| GT Node | Num. true pos. | Num. false pos. | Num. false neg. |
|---|---|---|---|
| A | $15.12 \pm 1.84$ | $0.08 \pm 0.03$ | $5.02 \pm 0.69$ |
| B | $8.29 \pm 1.10$ | $0.41 \pm 0.17$ | $2.13 \pm 0.36$ |
| C | $8.28 \pm 1.13$ | $0.29 \pm 0.14$ | $2.22 \pm 0.37$ |
| D | $7.63 \pm 2.12$ | $0.02 \pm 0.04$ | $2.92 \pm 0.95$ |
| E | $6.52 \pm 0.81$ | $0.20 \pm 0.13$ | $1.64 \pm 0.27$ |

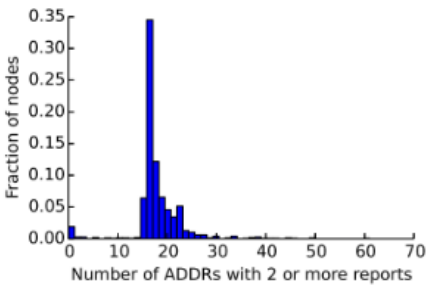
Figure 5.1: Accuracy statistics for AddressProbe

While these are initially good statistics, the researchers explored how they could reduce the number of false negatives. They concluded that it was likely because of

under-scraping: not collecting all the addresses from each node in each as nodes send at most 1000 entries selected at random in `ADDR` messages. Figure 5.2 shows that when combining the results from consecutive experiments, the number of false negatives could be dramatically reduced; and the number of `ADDR` messages required to scrape all the addresses from the nodes. From this they concluded to combine results from multiple experiments, and collect 24 `ADDR` messages per node, not exceeding this to limit bandwidth. The researchers did not report results from combined experiments, which would have been useful to see if this would preserve the good true and false positive rates. However, we can assume that the results would have shown these methods to be very accurate.




(a) Distribution of persistence of false negatives across subsequent experiments.




(b) Distribution of `ADDR` messages required from each node.

Figure 5.2: Study of under-scraping.

### 5.1.5 Topology results

There were some interesting results in the 2015 snapshot of the inferred topology. While most nodes had a degree in the range of 8-12, which matched the expectations of the Bitcoin protocol, there were some significant outliers. They found that extremely high degree nodes with connections in the multiple thousands persisted over the 18 days of experimentation.

Most strikingly they found that, even when excluding these outliers, the Bitcoin P2P network is most probably *not* a random graph. They tested various properties of random graphs and found that the observed graph was statistically significantly different. While there are certainly random elements to the way that connections are formed as described in Section 3.2.2 it does not truly form a random graph as intended.

### 5.1.6 Discussion

AddressProbe is clearly a very powerful method, even detecting directed edges. Moreover, it is fairly non-invasive and quick to perform. This is likely why the Bitcoin Core developers changed the method for updating timestamps to render this algorithm useless. It is unclear why the original timestamp updating rules were created like that in the first place. It could be, as for many other features that are exploited in other methods, a relic of how the protocol was first established. It could have seemed like a reasonable way to update timestamps, without considering this vulnerability.

We can point out some flaws in this method that will lead to incorrect inference. This method will give false positives of $v_i \to v_j$ when

- the connection from $v_i$ to $v_j$ was recently broken;
- knowledge of a new connection $v_j \to v_i$ is recent, but hasn't propagated far into the network. $v_j$ would update its timestamp of $v_i$ quickly, leaving a timestamp of $v_i$ in $v_j$'s `addrMan` that is potentially unique. This would signal $v_i \to v_j$ when it is not necessarily true;

and false negatives, arising from missed $v_i \to v_j$ recognition, because

- nodes may evict addresses from their `addrMan` lists, as its size is finite;
- `ADDR` messages only give a random subset of addresses in the `addrMan` of the node that is being requested, as discussed in the previous section.

Further, this method is limited in that we can only detect the outgoing connections from the reachable peers in $V_p$ to the unreachable peers in $V \setminus V_p$. While it may detect if these peers exist, we cannot detect incoming connections from unreachable peers to the reachable peers.

### 5.1.7 Influential nodes

Miller et al. also developed two very similar algorithms: Candidate Selection (CS) and Influence Validation (IV) to analyse which nodes in the Bitcoin P2P network were the origin of new blocks, called *influential* nodes. These nodes are likely to be gateway nodes to mining pools, as they represent significant amounts of computational power.

CS and IV took advantage of two features of the Bitcoin protocol. The first that nodes drop conflicting transactions, that is transactions that spend from the

same `UTXO` as one they have already recorded in their `memPool`. The second is the `InvBlock` technique described in Section 3.2.4.

In these algorithms we partition nodes into n sets $V_1, \ldots, V_n$. We generate a set of n mutually conflicting transactions $T = \{tx_1, \ldots, tx_n\}$. That is, all transactions $tx_i \in T$ spend from the same `UTXO`, so at most one transaction in $T$ can be accepted in any given node, and eventually end up in the blockchain. For each node in a set $V_i$, we use the `InvBlock` technique to prevent them from seeing all transactions in $T$ but $tx_i$ for 2 minutes. We do this because when we send out transactions, they are not received immediately[2], and we don't want transactions to designated for one set ($tx_1$ for $V_1$) to end up anywhere in another set. Within two minutes, once we are sure that each set $V_i$ is isolated from all transactions except $tx_i$, we send all nodes in each set $V_i$ their designated transaction $tx_i$. After the two minutes has elapsed, each node in set $V_i$ will not accept any other transaction $tx_j \in T \setminus \{tx_i\}$, as it conflicts with the transaction $tx_i$ which they have stored in their `memPool`. One transaction $t_B \in T$ is eventually included in a block which is mined by a miner in $V_B$ and included in the blockchain. We are unsure exactly which node in $V_B$, so we increment a "win" score of all the nodes in $V_B$ by one.

Initially in CS, this process is repeated over many randomly generated sets. The influential nodes end up having disproportionately high scores, as they are included in winning sets more often because of their contribution of computational power to those sets. In IV, the influential nodes are designated their own sets of size one, with all non-influential nodes placed in one large set. This method verifies the proportion of computational power in each influential node identified in CS, by running this algorithm for a long time over those sets.

The researchers showed that 2% of nodes won 189 out of 258 trials. This gives a 99% confidence interval between 66% and 80% for the percentage of computational power in the 2% most influential nodes. Transactions that are propagated to this small set of nodes are therefore much more likely to be included in the blockchain. The researches also noted that the influential notes did not contain any notable features in terms of their topology. For instance having a very high degree or forming exclusive communities.

[2]There is delay known as *latency* between any two nodes for any message to send

## 5.2 2016: Neudecker, Andelfinger, and Hartenstein

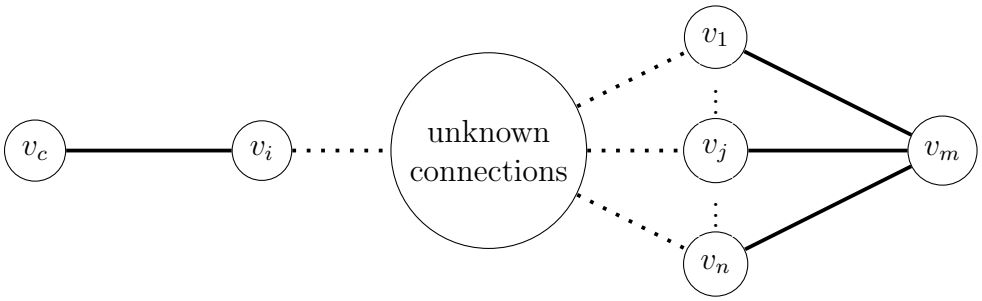
### [21] Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network

AddressProbe having been disabled by the Bitcoin protocol developers, researchers started looking into statistical methods of inferring the Bitcoin P2P network topology that didn't rely as heavily on idiosyncrasies in the Bitcoin protocol that could be disabled by the developers. Neudecker, Andelfinger and Hartenstein developed a active, node-specific, statistical method for discovering the Bitcoin P2P network topology by modelling propagation delay. They were able to achieve precision and recall of approximately 40%. However, the modelling they used was still in some way idiosyncratic, in that their assumptions were based on the known distributions that nodes use to delay sending transactions. Since they developed this method, the delay distributions built into the protocol were changed to limit the amount of information that could be gained by this kind of analysis.

The method is based on a Bayesian analysis of measurements of the arrival times of transactions at every node, given the transaction came from the node whose peers we wish to infer. Given these measurements, the researchers calculated a likelihood of delay times given a minimum path length. They also constructed a prior distribution of minimum path lengths between two nodes given the assumption of an Erdos-Renyi random graph. Combining these gave a posterior distribution for the minimum path length given the distribution times. A minimum path length of one would correspond to them being peers.

However, when describing their model, they did so in frequentest terms. We will describe the model with these terms in the explanation as they did, using terms such as maximum likelihood estimation, and then correct them in the discussion.

Figure 5.3: Experiment setup.




They set up the experiment as in Figure 5.3 they established a creation node $v_c$

and monitor node $v_m$, and connected both to all known nodes. By sending a new transaction from $v_c$ to a target node $v_i$, they created a situation where $v_i$ was a known source node of a transaction in the network. Then they could make measurements at the monitor node $v_m$.

### 5.2.1 Timing measurements of transactions

For a transaction $tx_k$ originating at source node $v_i$, the monitor node receives the message from each node $v_j$ at reception time $tr_{kij}$. The researchers were able to accurately measure the *latency* between the monitor node and all other nodes. Latency is the delay time to send data between two points over the internet. Latency was subtracted from the reception times to give sending times $ts_{kij}$. The sending time from the source node was subtracted from these to give the propagation delay measurements

$$\delta_{kij} = ts_{kij} - ts_{kii}.$$

Here $\delta_{kij} = \emptyset$ if message $tx_k$ did not have source node $i$. From this the set

$$\Delta_{v_i,v_j} = \bigcup_k \{\delta_{kij}\} \cup \{\delta_{kji}\}$$

contains all delay measurements between two nodes $v_i$ and $v_j$ where a transaction originated at one of them.

### 5.2.2 Propagation delay model

Neudecker et al. propose the following analytical model for propagation delay in a network using a gossip protocol. Assume an undirected graph $G = (V, E)$ where $V$ is the set of nodes and $E$ the set of edges. Define a random variable $x(e)$ such that $x(e) = 1$ if the edge $e \in E$ exists and 0 otherwise. Assume that the graph's edges can be modelled as an Erdos-Renyi random graph, which define probabilities for each edge $P(x(e) = 1)$ existing independently. We measure time in discrete units (milliseconds).

For any two nodes, there exist paths which are sequences of edges $(e_1, \ldots, e_n)$ that connect them (where edges are not repeated). Each path has a specific length $l = |\{e_1, \ldots, e_n\}|$. Define the set of paths with length $l$ as $c(l)$, called the *class* of paths of length $l$. Each class has an associated discrete random variable $D_l$ modelling the delay distribution. The probability mass function $P(D_l = t)$ defines the probability of receiving a message at time $t$ after it is initially sent along a path of length $l$ at time 0. Assume we know the distribution of $D_1$.

Now define $z_l$ as the actual number of paths of length $l$ between two nodes, and $z_l^*$ as the maximum possible number of paths of length $l$ between two peers.

$$z_l^* = \underbrace{\binom{|V|-2}{l-1}}_{\text{choices of nodes}} \cdot \underbrace{(l-1)!}_{\text{arrangement of nodes}} . \tag{5.1}$$

We can define the probability of the existence of a specific path in $c(l)$ as $p_l := P(x(e) = 1)^l$. So the probability that there are $k$ paths in lass $c(l)$ between two nodes is

$$P(z_l = k) \approx \binom{z_l^*}{k} p_l^k (1-p_l)^{z_l^* - k}. \tag{5.2}$$

This is an approximation because the existence of paths that contain the same edges are not independent. However for $l = 1, 2$ this equation is exact. Consider two nodes $v_1$ and $v_2$. There is only one possible path of length 1 between two nodes. Given $n$ nodes, there are $n-2$ possible paths of the form $\{(v_1, v_i), (v_i, v_n)\}$, of which none of them share edges as there is only one intermediary node. However this cannot be said for path lengths greater than 2. For instance consider with 5 nodes the paths $\{(v_1, v_2), (v_2, v_3), (v_3, v_5)\}$ and $\{(v_1, v_2), (v_2, v_4), (v_4, v_5)\}$ which share the edge $(v_1, v_2)$.

Now consider a given class $c(l)$ between two nodes. Assume that $z_l = n$. Sending a message between the two nodes, we draw $n$ samples from $D_l$, $(D_l^1, \ldots, D_l^n)$. Define $\hat{D}_l = \min_i D_l^i$, the fastest propagation time over all the $n$ paths. The conditional distribution of $\hat{D}_l$ is given by

$$P(\hat{D}_l = t | z_l = n) = \underbrace{P(D_l = t)}_{\text{1 path takes } t} \cdot \underbrace{P(D_l \geq t)^{n-1}}_{\text{other paths take longer}} \cdot n. \tag{5.3}$$

Applying the law of total total probability we have for a class $c(l)$

$$P(\hat{D}_l = t) = \sum_{n=1}^{z_l^*} \Big( P(\hat{D}_l = t | z_l = n) P(z_l = n) \Big) \tag{5.4}$$

which defines the distribution of the fastest time delay for a message to pass across all paths of length $l$.

Now to find the fastest delay over all paths, defined by a random variable $D$, we have

$$P(D = t) = \sum_{l \geq 1} \left( P(\hat{D}_l = t) \prod_{l' \neq l} P(\hat{D}_{l'} \geq t) \right). \tag{5.5}$$

However for a given pair of nodes, there exists a shortest path length $C_{\min}$, and so

the distribution of the delay should only take into account lengths that are $C_{\min}$ and longer, with the probability contribution of shorter paths set to 0. We calculate the probability of $k$ paths in $c(l)$ given at least one path exists

$$P(z_l = k | z_l > 0) = \frac{P(z_l = k, z_l > 0)}{P(z_l > 0)} = \frac{P(z_l = k)}{1 - (1 - p_l)^{z_l^*}} \tag{5.6}$$

We also create the prior distribution for $C_{\min}$ before we use our data.

$$\begin{aligned} P(C_{\min} = l) &= P(z_l > 0) \cdot \prod_{i<l} P(z_i = 0) \\ &= (1 - (1 - p_l)^{z_l^*}) \cdot \prod_{i<l} (1 - p_i)^{z_l^*} \end{aligned} \tag{5.7}$$

From here we can calculate $P(D = t | C_{\min} = l)$ by using Bayes' theorem, constructing the joint distribution $P(D = t, C_{\min} = l)$ by disregarding $P(\hat{D}_i = t)$ for all $i < l$ and calculating $P(\hat{D}_l = t)$ using $P(z_l = k | z_l > 0)$ rather than $P(z_l = k)$ in the relevant steps.

The researchers validated their model for $D$ on two simulation sets: a random network with 1000 peers, an average of 8 connections per peer and uniformly distributed latency $D_1$ between 50 and 100ms; and a random network with 6000 peers, an average of 16 connections and uniformly distributed latency $D_1$ between 100 and 300ms. Distributions for $D_2, D_3, \ldots$ were calculated as convolutions of $D_1$. The results are shown in Figure 5.4. They found that their model matched the empirical distribution extremely well for small delays, but then diverged. They explained this as an effect due to the assumption of independent existence of paths, which is true for $l = 1, 2$ (and so matches small time delays) but not true for longer paths. The problem of fixing this model to solve it exactly for longer path lengths was left for future research.

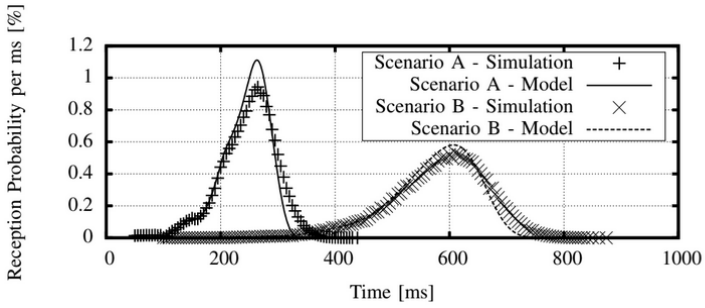



Figure 5.4: Models and true distributions $P(D = t)$ on simulations.

### 5.2.3 Inference of shortest path length

Given the above model, the researchers calculated the maximum likelihood estimate (MLE) $\hat{l}$ of $C_{\min}$, the shortest path length between two nodes $v_1, v_2$ as follows:

The likelihood is calculated as

$$L(C_{\min} = l|\Delta_{v_1,v_2}) = P(C_{\min} = l) \cdot \prod_{\delta \in \Delta_{v_1,v_2}} P(D = \delta|C_{\min} = l) \tag{5.8}$$

with the MLE taken at the maximum:

$$\hat{l} = \arg\max_{l} L(C_{\min} = l|\Delta_{v_1,v_2}). \tag{5.9}$$

The researchers claim convergence of $\hat{l}$ to $C_{\min}$, and provide a certainty distribution for $l$

$$P(C_{\min} = l|\Delta_{v_1,v_2}) = \frac{L(C_{\min} = l|\Delta_{v_1,v_2})}{\sum_{i \geq 1} L(C_{\min} = i|\Delta_{v_1,v_2})} \tag{5.10}$$

which assigns probabilities to each path length being the shortest given the data. Where the highest probability is assigned to $l = 1$, this would correspond to the two nodes being peers.

In a simplified simulation of the Bitcoin network, this method achieved precision of near 100% and recall of 90% within 12 observations.

### 5.2.4 Model of delay in the Bitcoin P2P network

Neudecker et al. developed a parametric model to construct delay distributions on the Bitcoin P2P network. To do this they approximated three distributions: latency between nodes based on geographical distance; delay between a client receiving and sending information broken into intentional and unintentional parts; and node degree distribution which they achieved by minimising square error between simulated and measured propagation time. The delay distributions $P(D = \delta|C_{\min} = l)$ were then calculated by simulation.

Figure 5.5 shows in the dotted lines examples of $P(D = \delta|C_{\min} = l)$, the condition delay distributions given minimum path lengths. This is determined by the three distributions mentioned above. The value of certainty of the MLE $P(C_{\min} = \hat{l}|\Delta_{v_1,v_2})$ is given as the solid line. Note that the certainty in our solution drops as there is more overlap in the delay distributions. Intuitively, this is because if two distributions are very similar, it is harder to know which distribution the observations come from. These distributions can be made to overlap through altering the Bitcoin protocol to include random delays in a node propagating transactions to its peers. This delay

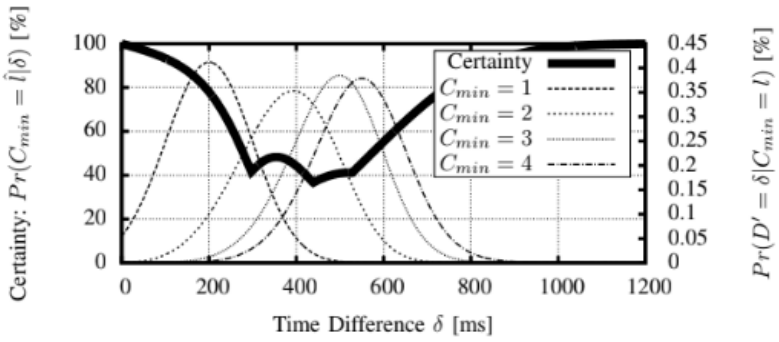



Figure 5.5: Conditional delay distributions and certainty w.r.t. observed delay.

purposefully introduced in the protocol is known as *trickling*, and it is done so that these delay distributions overlap as more, so the certainty in a solution decreases. This is how the protocol developers are able to limit the leaking of information by introducing delay into the system.

### 5.2.5 Accuracy

The researchers created two nodes with 50 peers each, and used them to create transactions. One monitor node collected timing measurements from all other nodes. For a pair of peers $v_1, v_2$, $n$ observations were collected in the set $\Delta_{v_1,v_2}$. Given the high certainty of distribution in lower time delays, the smallest element of $\Delta_{v_1,v_2}$, $\delta_{\text{min}}$ was taken. Using the theoretical distribution derived from the model and some threshold hyperparameter $s$, if

$$P(D > \delta_{\text{min}}|C_{\text{min}} = 1)^n > s$$

then an edge between $v_1$ and $v_2$ is predicted.

The researchers found that changing $s$ tweaked the precision and recall values for this method, with values of 40% achievable in both with an appropriate selection of $s$. They also found that higher recall and precision were achieved with more observations, up to 6 observations per pair, after which no improvement was achieved.

### 5.2.6 Discussion

While this method wasn't hugely successful in terms of accuracy, it did provide a proof of concept of using timing measurements to reconstruct the graph. 40% precision and recall is still significantly better than random guessing. However, this method still rests to some degree on idiosyncrasies in the Bitcoin protocol, namely

the purposeful trickling delays that nodes employ during transaction propagation. The researchers showed themselves that this method could be made redundant by optimising the trickling mechanism, which has since occurred so this method no longer works well.

An assumption made in the propagation delay model is that the network is a random graph. However Miller et al. showed in [20] that the topology of the Bitcoin P2P network (at the time of their experiments) was not random, even when disregarding anomalous high degree outliers.

Throughout the entire methodology, there is an assumption of independence of paths and connections given the timing data. While this may be true for paths that are of length 1 or 2 in the context of ignoring all others, there is no doubt that information to be gained by considering other connections in the network as paths and therefore timings are not be independent in general.

There were several limitations to the model of the Bitcoin P2P network that they proposed. They acknowledge this about the estimation of latency based solely on geographical distance:

> "The model does not account for latency introduced by temporal behavior such as link saturation. It also cannot make any statements on the connection quality between the specific user and its ISP[3], as the empirical model relies solely on the distance. Furthermore, the distance between two peers on the earth's surface may not reflect the routed distance, e.g., through submarine optic fiber cables."

Further, in the inference of node degree distribution, the researchers proposed a specific parameterisation that minimises square error between propagation delay in a simulated network and the observed propagation delay, as the real node degree distribution is unknown. This parameterisation minimises error that is itself partially explained by any incorrect parameterisation of the estimation of latency and client delay, rather than just modelling node degree. This method cannot be used to predict node degrees, and does not reflect the anomalous behaviour observed by some nodes in the paper [20].

A significant issue with their analysis, as mentioned in the introduction of this section, is that they employ a frequentest approach to a Bayesian model. The only mention of Bayesian terminology is in describing the distribution of $C_{min}$ as a prior. he likelihood is calculated as per Equation 5.8. However this equation includes the prior distribution of $C_{\min}$, and so this quantity is proportional to the posterior

[3]internet service provider

distribution $P(C_{\min} = l|\Delta_{v_1,v_2})$, which makes $\hat{l}$ (calculated in 5.9), not the MLE as they have suggested, but rather the *maximum a posteriori* (MAP) which is the mode of the posterior distribution.

Readers familiar with Bayesian statistics will understand that this is not just an abuse of terminology, but also requires a different interpretation of the results. Under a Bayesian framework, we create a prior belief for the path length (here the distribution of path lengths in an Erdos-Renyi graph), and update that belief with data that we collect, which here is the timing measurements. The result is a probabilistic distribution for the minimum path length, which should not be used to generate point estimates as is done in this analysis by claiming that information for the minimum path length is given by the MAP value. The MAP is rarely used in analysis because point estimation is not useful in a Bayesian framework. There are many ways to derive a best guess for the minimum path length, but it requires more thought than taking the mode of its distribution. This could possibly be reflected in the accuracy of the results.

## 5.3 2018: Grundmann, Neudecker, and Hartenstein

### [32] Exploiting Transaction Accumulation and Double Spends for Topology Inference in Bitcoin

Given that the timing analysis method was made infeasible by the change of trickling mechanisms, Grundmann, Neudecker and Hartenstein proposed two new methods for finding the topology of the Bitcoin P2P network. The first is an active, network-wide, idiosyncratic approach based on the trickling mechanism nodes use to delay sending transactions to their peers. While theoretically possible, it is far too expensive to run, and at an affordable cost they were only able to achieve a recall of 10%. The second method is a passive, node-specific, idiosyncratic method based on the way that nodes drop conflicting transactions. It was much more successful with 87% recall and 71% precision.

### 5.3.1 Transaction queues

Because of analyses similar to the previous paper, Bitcoin clients include *trickling* mechanisms; purposefully delaying transaction messages in order to impede timing inference. In the Bitcoin protocol, every node keeps an outgoing queue for each peer. New transactions are added to every queue whenever they are received or

created by the node. The queues of transactions are broadcast to the respective peers independently at times following an exponential distribution. The next time a queue is sent is at

$$t = \texttt{current\_time} - \log(1 - \frac{X}{2^{48}}) \cdot \texttt{average\_interval\_seconds} \cdot 1,000,000 + 0.5,$$

where all measurements are in microseconds, $X$ is drawn from a uniform distribution on $[0, 2^{48} - 1]$ and `average_interval_seconds` is 2,000,000 for outgoing connections and 5,000,000 for incoming connections. So reception of transactions can be modeled as a Poisson process. The maximum number of transactions that are sent is 35. A new sending time is calculated each time the transactions are sent to a peer.

### 5.3.2 Inference by transaction accumulation

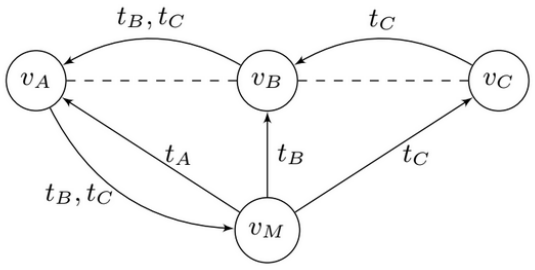



Figure 5.6: Edge inference using transaction accumulation[4]

Consider a setup as in Figure 5.6. We have a monitor node $v_m$ connected to three nodes $v_a, v_b$ and $v_c$ that we want to examine. Assume that these nodes comprise the entire network. We create three non-conflicting transactions $t_a, t_b$ and $t_c$ to sent simultaneously from $v_m$ to only $v_a, v_b$ and $v_c$ respectively. Consider the scenario where $v_b$ is connected to $v_a$ and $v_c$, but $v_a$ and $v_c$ are not connected.

We examine the situation in which node $v_a$ relays $t_c$ to $v_m$. As $v_a$ and $v_c$ are not peers, $v_c$ must first send $t_c$ to $v_b$, after which $v_b$ can relay $t_c$ to $v_a$, then $v_a$ sends $t_c$ to $v_m$. When $v_b$ sends $t_c$ to $v_a$, it also sends $t_b$ which is earlier in the queue (or $t_b$ is sent as part of an earlier queue). So either $v_a$ sends $t_b$ to $v_m$ in an earlier message, or $t_b$ and $t_c$ are sent in the same message. There is no situation in which $t_c$ is the sent alone as the earliest message. Taking the contrapositive gives us our first rule:

**If node $v_a$ sends an `INV` message with $t_b$ alone as its first `INV` message, then $v_a$ and $v_b$ are peers.**

[4]from [32]

From the above scenario, notice that $t_b$ is always sent in $v_a$'s first message to $v_m$, giving us our second rule:

**If node $v_a$'s first `INV` message to $v_m$ has multiple transactions, then at least one of the nodes associated with those transactions is a peer of $v_a$.**

There are clear limitations to this method in practice. If, for instance, we do not connect to node $v_b$ (as we may miss one of thousands of nodes), and so do not send transaction $t_b$, then we will send transaction $t_c$ to node $v_c$, which will then be sent to $v_b$, then $v_a$, then back to $v_m$. We may register that $v_a$ sends transaction $t_c$ alone and conclude that $v_a$ and $v_c$ are peers when that is not the case.

We also cannot guarantee that we can send all the messages simultaneously and that we have perfect knowledge of latency, so even if $v_m$ does connect to $v_b$, there is no guarantee that $t_b$ will reach $v_b$ from $v_m$ before $t_c$ from $v_c$; which may lead to more false positives if $v_a$ receives $t_c$ before $t_b$.

Consider also the situation where $v_a, v_b$ and $v_c$ are all peers. Because of the Poisson process that transaction propagation follows, $t_b$ may travel from $v_b$ to $v_c$ to $v_a$ before it travels directly from $v_b$ to $v_a$, so it may take multiple repetitions before we can deduce that $v_a$ and $v_b$ are peers.

Given around 10,000 reachable nodes [23], and that there is a minimum transaction fee necessary for nodes to propagate a transaction, this method is very expensive. The researchers propose **Variant DS$_i$**, where $i$ sets of conflicting transactions $T_1, \ldots, T_i$ are created. Within each set $T_j$, all transactions spend from the same $\texttt{UTXO}_j$, so they are conflicting. Between the sets, $t_j \in T_j$, $t_k \in T_k$, $t_j$ and $t_k$ are not conflicting, as they spend from different $UTXO$s. All transactions are still unique, but the experimenter only has to pay for $i$ transactions per observation instead of 10,000. However this also means that nodes will drop most transactions they receive, which makes inference much harder and causes false negatives.

The authors ran variant $\text{DS}_3$ was run on the Bitcoin testnet, connecting to about 520 nodes and used 5 nodes as validation targets. The researchers were able to achieve a precision of 67% and recall of 10% after 50 observations. They noted that due to the small sample size, these are only rough estimates of the recall and precision of the method. This result is equivalent to there being 2600 possible edges, of which 20 are real, and them finding 2 real edges and 1 false edge. This is clearly not a good method.

### 5.3.3 Inference by double spends

Grundmann, Neudecker and Hartenstein developed a second method due to the poor performance of the first, and because thy desired a node-specific method as it would

allow for more accurate validation. Further, they reasoned it would be the method that an adversary would prefer for an attack on a specific node.

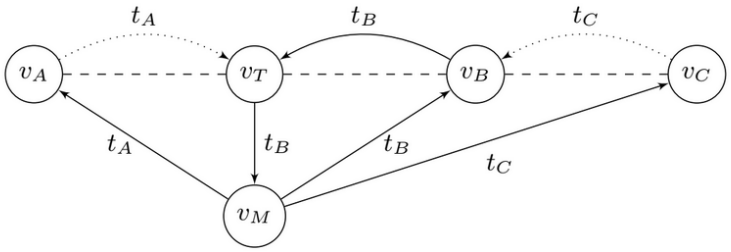



Figure 5.7: Edge inference using double spends[5]

Consider our monitor node $v_m$, nodes $v_a, v_b$ and $v_c$ as well as a target node $v_T$ connected as in Figure 5.7. We connect $v_m$ to all nodes and simultaneously send mutually conflicting transactions $t_a, t_b$ and $t_c$ to $v_a, v_b$ and $v_c$ respectively.

Node $v_c$ will send $t_c$ to $v_b$, which will drop it as it conflicts with $t_b$. Node $v_T$ will either receive $t_b$ or $t_a$ first, and drop the other one. Node $v_T$ will then send that transaction to $v_m$. This gives us the rule: **the target node will send one transaction corresponding to one of its peers.** If we repeat this for many observations, we should be able to infer all the peers of $v_T$.

There are similar limitations to this method. If for some reason we do not connect $v_m$ to $v_b$, then $t_c$ may be sent from $v_c$ to $v_T$ to $v_m$, which would give us a false positive that $v_T$ and $v_c$ are peers. If we do not send all transactions simultaneously, there may be a situation where $v_c$ sends $t_c$ to $v_b$ before it receives $t_b$ from $v_m$, which may also lead to a false positive. The researchers propose three variants of this basic algorithm to address these issues.

### Variant *Count*

In the false positive cases mentioned above, the transaction $t_c$ must be propagated at least twice before it reaches $v_T$. This is a slower process than $t_a$ being propagated directly to $v_T$ from $v_a$. So we would expect $v_T$ to receive $t_a$ more often than $t_c$. This would be reflected in an experiment with many observations, where we would receive a positive signal for $v_a$ being a peer more often than $v_c$. So we require that the number of times that a node is identified as a peer to be greater than some threshold count.

[5]from [32]

### Variant *Ignore*

In the case that transactions are not able to be sent from $v_m$ simultaneously, we may have situations where nodes drop the transactions they are sent, as they conflict with other transactions that have been forwarded to them before their allocated transaction can reach them. In the above scenario, suppose transaction $t_c$ is sent from $v_m$ to $v_c$, and $v_c$ forwards $t_c$ to $v_b$ before $v_b$ receives $t_b$ from $v_m$. Node $v_b$ would drop transaction $t_b$, and forward $t_c$ onto $v_T$. In the case that $v_T$ sends transaction $t_c$ to node $v_m$, we would also have that node $v_b$ sends transaction $t_c$ to $v_m$. So in this variant, we ignore the inference (that $v_T$ and $v_c$ are peers) if we receive the corresponding transaction from any other node but $v_T$.

### Variant *Suppress*

Using the same example, if we have inferred that $v_T$ and $v_b$ are peers, then we may wish to create a method from preventing the same inference in subsequent observations. This is because there is at most one peer that can be infered per observation, which would lead to these repetitions being slow, expensive or even pathological in the case that certain nodes follow protocols for forwarding transactions that are much faster than other (or if they have negligible latency with the target node $v_T$).

We wish to suppress peer $v_b$ that has already been inferred, which would require:

1. that node $v_T$ drops transaction $t_b$ in all further iterations of the experiment;
2. that node $v_b$ is prevented from forwarding on any other transactions that we introduce—to prevent false positives.

The researchers suggest the following method to meet these requirements. Consider two UTXOs, $\texttt{UTXO}_1$ and $\texttt{UTXO}_2$. Node $v_m$ sends the following transactions simultaneously:

1. transactions $t_i$ with shared (and so conflicting) input $\texttt{UTXO}_1$ individually to all respective nodes $i$, $i \neq b, v$;
2. transaction $t_v$ with input $\texttt{UTXO}_2$ to node $v_T$;
3. transaction $t_b$ with inputs $\texttt{UTXO}_1$ and $\texttt{UTXO}_2$ to node $v_b$.

The first set of transactions follows the original method. Now node $v_T$ receives $t_v$ which conflicts with $t_b$ as they both spend from $\texttt{UTXO}_2$, so node $v_T$ will always drop $t_b$. This meets the first requirement. Transaction $t_b$ also conflicts with all other transactions $t_i$ as they spend from $\texttt{UTXO}_1$, so node $v_b$ will always drop other transactions $t_i$, and so never forward them on to $v_T$. This meets the second condition.

### 5.3.4 Accuracy

The researchers validated this method on the Bitcoin testnet; fully connecting the monitor node to all 500 nodes and inferring peers of five validation nodes using 50 iterations in each experiment. Using a combination of Suppress and Ignore, they were able to achieve 60% recall and 97% precision. Using just the Supress variant, they were able to achieve 87% recall and 71% precision, only paying 99 transaction fees.

### 5.3.5 Discussion

The transaction accumulation method didn't provide very good results, however it showed that at least some information could be gained in this way. The double spending method showed significantly better results.

It should be noted that both methods could be rendered unusable with small changes to Bitcoin protocol. For instance, if transactions were not sent in the order that they are received, but randomly, then the first method could not be used. If nodes still propagated conflicting transactions rather than dropping them (and leaving it up to block miners to ensure no double spends), then the second method would not work.

The researchers acknowledged in both methods that if we cannot synchronise the timing of sending transactions to their respective nodes, then we may infer false positives. As noted in other papers, there is a way around this using the `InvBlock` technique. In the basic double spend example, if we were to send `INV` messages to the nodes containing all transactions except their designated one, then never respond to the resulting `GETADDR` requests, those nodes would not be able to receive those transactions for at most 2 minutes, allowing us to synchronise the timings of sending the designated transactions.

## 5.4 2018: Delgado-Segura and 6 other authors

#### [28] TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions

Soon after, the next paper by Delgado-Segura and 6 other authors (4 of who were authors of the first paper [20] in this chapter) described TxProbe, an active, idiosyncratic method to determine the topology that is designed to be network-wide, but can easily be adapted to be node-specific.

It is extremely effective, with experiments yielding 100% precision and ∼95% recall. However it is also extremely invasive. A full implementation would involve the disabling of important functionalities of the Bitcoin protocol for the duration of an experiment, which they estimated to take 8.5 hours. Specifically, it makes use of the way the protocol handles *orphan transactions*, and in doing so disables their use in any node they are investigating. For this reason, the researchers only validated TxProbe on the testnet, and avoided any tests on the main Bitcoin P2P network. If an adversary were to try to perform a network-wide probe of the network, it would also be very easy to spot.

### 5.4.1 Orphan transactions

Consider a scenario where there are two transactions $tx_1$ and $tx_2$, where $tx_2$ spends the output from $tx_1$, and they are created in quick succession. Due to propagation delays on the network, there is a positive probability that some node $v$ will receive these transactions in the wrong order. It receives $tx_2$ first, but can't treat it as a normal transaction. It doesn't spend an `UTXO` (that would be $tx_1$'s output, but $v$ doesn't know about that yet), but it also isn't a double spend, because it doesn't conflict with any transaction in its own `memPool` list of transactions it has seen, or the blockchain history. So $v$ classifies $tx_2$ as an orphan transaction. It doesn't propagate it further into the network, but does store it in a data structure called `MapOrphanTransactions`, so that in the case its *parent* transaction does arrive, it doesn't need to re-request it from the network and use up bandwidth. In the event that the parent transaction $tx_1$ does arrive at $v$, it can verify that $tx_2$ does spend a legitimate output, so removes $tx_2$ from `MapOrphanTransactions` and propagates both transactions further on into the network.

There are two properties of the protocol's method of handling orphan transactions that are key to the TxProbe algorithm:

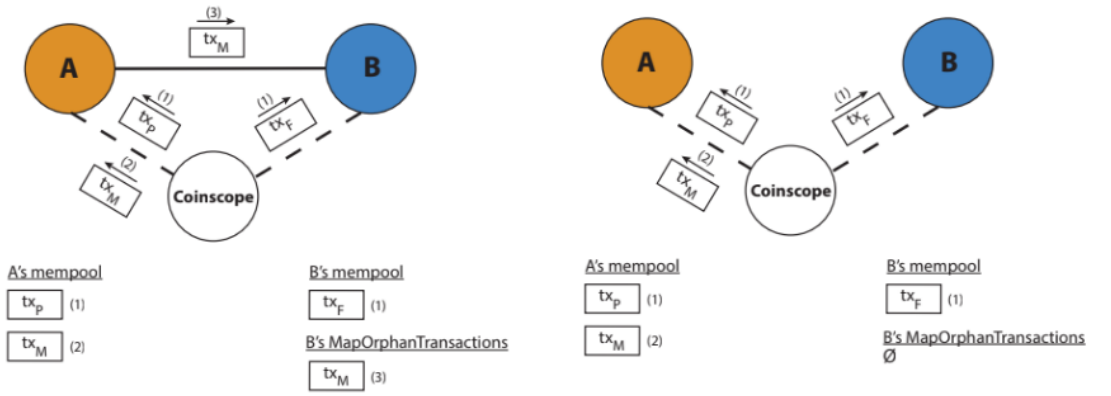
1. when a node receives a transaction that it clasifies as an orphan, it will not forward the transaction onto its peers, in case it is not legitimate;

2. when a node has a transaction in its `MapOrphanTransactions` structure, then it will not request that transaction in `GETDATA` messages when receiving `INV` messages containing a hash of that transaction. This is to preserve bandwidth.

### 5.4.2 Inference using orphan transactions

Consider a network comprised of a monitor node M that is connected to two nodes A and B which are peers, as in Figure 5.8. Node M wishes to infer if A and B are

peers. Create two conflicting transactions: the parent transaction $tx_p$ and the flood transaction $tx_f$. We simultaneously send $tx_p$ from M to A and $tx_f$ from M to B. They will both drop the other transaction if they receive it from the other node. We then create a marker transaction $tx_m$ that spends an input that is the output of the parent transaction $tx_p$. As A has the parent transaction recorded in its `memPool`, it will regard the marker transaction $tx_m$ as a normal transaction, and send it onto B. Node B does not have $tx_p$ stored in its `memPool`, and so will regard $tx_m$ as an orphan transaction, and not forward it onto M. However, if M sends an `INV` message containing $tx_m$'s hash to B, B will not request $tx_c$ with a `GETDATA` message, as it is already in B's `MapOrphanTransactions` list. If A and B are not peers, B would not have $tx_m$ stored as an orphan transactions, and so will request $tx_m$ in the same scenario.




(a) Basic positive edge inferring technique between two nodes.

(b) Basic negative edge inferring technique between two nodes.

Figure 5.8: Basic inference principle with orphan transactions[6].

When any other nodes are present in the network, this method may lead to false positives. Consider if A and B are not peers, but they are both peers with some node C. Transactions $tx_p$ and $tx_m$ may be sent from M to A to C, and so B may receive $tx_m$ as an orphan node from C, and will not request $tx_m$ when sent the `INV` message from M. This would lead us to conclude that A and B are peers when it is not the case. So we need this method to have three properties.

1. isolation: $tx_p$ should not propagate from the node we send it to;

2. synchronicity: in this example, A must receive $tx_p$ at the same time that B receives $tx_f$, otherwise one could forward their transaction to the other, confounding the inference rule;

[6]from [28]

3. scalability: using this basic method with three transactions, we can only infer the neighbourhood of one node. Many transactions would be needed to infer the entire topology. We wish to find a way to achieve this more efficiently.

### 5.4.3 Displacing orphan transactions

This section describes the invasive property of TxProbe.

The size of `MapOrphanTransactions` is by default limited to 100 transactions at any given time. This is likely so old orphan transactions, which for whatever reason never had a parent transaction propagated, are eventually forced to be removed from `MapOrphanTransactions`. For this experiment using thousands of transactions, we must be able to guarantee that any transactions created are able to be stored, and so we need a method of emptying the `MapOrphanTransactions` structure of any node that we are examining.

The Bitcoin protocol has a strange quirk that allows us to do this. When the number of transactions in `MapOrphanTransactions` exceeds the limit, some transactions are randomly evicted, but they are not evicted uniformly. The node generates a random number `randomhash`, then evicts the transaction whose hash is the closest number that is larger than `randomhash`. This process is repeated until the number of orphan transactions is within the limit. It is unclear why this method is used in the protocol, rather than a method that would select transactions uniformly. This procedure is easily exploitable.

We can create 100 mutually conflicting transactions whose hashes are very small (by repeatedly resigning them), so we can displace all stored orphan transactions in a given node with our new ones. We can check that all our transactions are there by sending an `INV` message with all those transactions. If the node does not send any `GETDATA` messages back, then we know that they have them all stored as orphan transactions, populating the entire `MapOrphanTransactions` list. Now, we send the parent *cleanser* transaction of those orphan transactions to the node. The node now sees the orphan transactions as regular transactions, chooses one of them to be legitimate (as they are are mutually conflicting), and propogates the two further into the network and empties their `MapOrphanTransactions` list. We are now guaranteed an empty list, which we can fill as part of an experiment.

Given that TxProbe requires thousands of orphan transactions as part of this experiment, this process must be continuously repeated, 100 orphan transactions at a time until all necessary transactions have been used. So for the entire duration of the experiment, the `MapOrphanTransactions` structure on every reachable node in the network is effectively disabled, as no new transactions will be able to be kept

in it without quickly being removed. The protocol allows for orphan transactions as they are necessary to the healthy functioning of the network. Without them, all transactions that arrive at a node in the wrong order will be discarded. This may lead to many transactions not being able to be included in the blockchain, or at best take up far too much bandwidth as transactions will have to be sent repeatedly to each node before all transactions arrive in the correct order.

### 5.4.4 TxProbe

Delgado-Segura et al. developed the TxProbe algorithm to have the isolation, synchronicity and scalability properties. It requires several iterations of the following procedure.

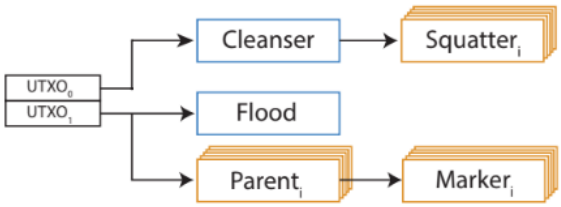



Figure 5.9: Transactions created for TxProbe[7].

1. **Partition the network:** We label one group of nodes the *source* set $\{a_1, \ldots, a_n\}$, having size $n$, and the others the *sink* set $\{b_1, b_2, \ldots\}$. The aim is to infer all connections between source and sink nodes.

2. **Create cleanser transactions:** We create one transaction $tx_{cleanse}$ with one output, and use that output as the input of $n$ mutually conflicting squatter transactions $tx_{s_1}, \ldots, tx_{s_n}$ spending from $\texttt{UTXO}_0$ which are individually resigned until they have sufficiently small hashes.

3. **Send squatter transactions:** We send all squatter transactions to all nodes in the sink set. As these nodes are not aware that they are conflicting yet, they are all accepted as orphan transactions, and displace all current orphan transactions.

4. **Cleanse `MapOrphanTransactions`:** We send $tx_{cleanse}$ to all sink nodes. The squatter transactions cease being orphan transactions. One squatter transaction will be accepted, and the others will be discarded as double spends. The `MapOrphanTransactions` structure in each of the sink nodes is now empty.

[7] from [28]

5. **Create inference transactions:** We create $n+1$ mutually conflicting transactions: $n$ parent transactions $tx_{p_1}, \ldots, tx_{p_n}$ and 1 flood transaction $tx_f$. We also create $n$ marker transactions $tx_{m_1}, \ldots, tx_{m_n}$, each having spending the output of their respective parent transaction (for example: $t_{m_2}$ has input that spends the output of $t_{p_2}$).

6. **Isolate the network:** We wish to ensure that each parent transaction $tx_{p_i}$ is isolated to its respective source node $a_i$, and that the flood node is isolated to the sink set. Consider the `InvBlock` procedure from Section 3.2.4. We connect our monitor node $m$ to all nodes, and send out an `INV` message to all nodes containing hashes of all the $tx_{p_i}$ and the $tx_f$ transactions depending on which we want to block in which node. Each node will respond with a `GETDATA` message that $m$ will ignore. All nodes will then wait at least 2 minutes before sending a `GETDATA` message to any other nodes for the same transactions. This isolates the transactions as required.

7. **Send inference transactions:** Now we have a window of slightly less than 2 minutes to send the transactions. We send the flood transaction $tx_f$ to nodes in the source set, and wait a few seconds for it to propagate. We then send the parent transactions $tx_{p_i}$ to their respective source set nodes $a_i$, wait a few seconds, then send the corresponding marker transactions $tx_{m_i}$.

8. **Request the marker transactions:** Each marker transaction $tx_{m_i}$ will be seen as a regular transaction by its source node $a_i$, so $a_i$ will propagate $tx_{m_i}$ to all of its peers. We have guaranteed that no other nodes have seen $tx_{p_i}$, and so all peers of $a_i$ will recognise $tx_{m_i}$ as an orphan transaction, and not propagate it further. We wait a few seconds after the previous step for $a_i$'s to propagate the $tx_{m_i}$'s, then send `INV` messages to all sink nodes containing hashes of all the marker transactions. We collect all the `GETDATA` message responses.

9. **Inferring links:** If a sink node sends a `GETDATA` request containing the hash of a marker node $tx_{m_i}$, then it does not have that transaction in its `MapOrphanTransactions` structure, and so does not have a link with $a_i$. If a sink node does not send a `GETDATA` message containing the hash of $t_{m_i}$, then it is a peer of $a_i$.

### 5.4.5 Partitioning the network

Each iteration of the TxProbe procedure gives the links between the source set and sink set, but no internal links within those sets. So a number of iterations are

required, such that every pair of nodes is separated at least once. The source set can contain at most 100 nodes, as this is the limit of orphan transactions that any one node can store. The researchers propose the following partitioning method for $N$ reachable nodes:

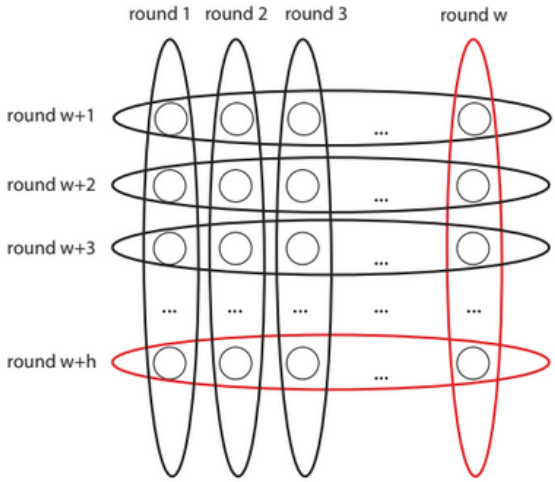



Figure 5.10: Partitioning method[8].

Create a grid of all known nodes. The width is $w = \min\left(\lceil\sqrt{N}\rceil, 100\right)$. The height is $h = \lceil\frac{N}{w}\rceil$. In each iteration we assign all nodes in a row or column to the source set, until all rows and columns are exhausted.

If $h > 100$, then each source set associated with a column must be separated into several iteration such that each node is included at least once and no iteration contains more than 100 nodes. Here the total number of iterations is given by $h - 1 + \lceil\frac{h}{w}\rceil \cdot w$.

As currently $N < 10000$, then $h < 100$ and so the total number of transactions is $h + w - 2$ (as the last row and column are not necessary).

### 5.4.6 Time and transaction costs

Delgado-Segura et al. found that each iteration took roughly 2.5 minutes per iteration. On the Bitcoin testnet with $\sim$ 1000 nodes, it would be 2.6 hours. On the main Bitcoin network with $\sim$ 10,000 nodes, this would take 8.5 hours.

In each iteration, we must spend transaction fees on $tx_{cleanse}$, one $tx_{s_i}$, and either the flood transaction $tx_f$ or a parent-marker pair $tx_{p_i}$, $tx_{m_i}$. Given a source set of size $n$, there is a $\frac{n}{N}$ probability of a parent-marker pair being accepted in the blockchain, and $\frac{N-n}{N}$ for the flood transaction. So the expected number of transactions for an iteration is $3 + \frac{n}{N}$. Given $\sim$ 10,000 nodes, the researchers estimated the total cost

[8]from [28]

to be between $5.7 \times 10^{-3}$ and $7.6 \times 10^{-3}$ BTC, currently between 46.42 and 61.93 USD.

### 5.4.7 Accuracy

The researchers set up 5 nodes for validation on the Bitcoin testnet. They excluded any nodes that they weren't able to `InvBlock` in a trial run, any nodes that at some point held the wrong transaction (not $tx_f$ or $tx_{p_i}$ when they were meant to) and any that disconnected from the monitor node during the experiment.

Over 40 trials of the experiment, TxProbe achieved 100% precision with a 95% confidence interval for recall between 93.86% and 95.45%.

Interestingly, they also found that the Bitcoin testnet was not a random graph.

### 5.4.8 Discussion

The accuracy of this method can be attribute to its robustness to false positives. As orphan transactions don't propagate through the network, it is obvious to detect when nodes are peers. The flood transaction prevents false positives as it travels to all nodes in the network, even unobserved ones. This way, the parent transactions can't be propagated to unobserved nodes, and so false inferences can't be made. This is a property that the other methods in this paper don't posses, and makes this method very powerful.

The researchers do acknowledge how damaging the method is. Running one test on the Bitcoin mainnet would disable the use of orphan transactions for normal users for over 8 hours, which could significantly interfere with transaction propagation times and reliability.

An improvement to the method may be made by observing that marker nodes propagate in much the same way through the source nodes as through the sink nodes, and so it would be possible to infer links within the source set by cleansing all `MapOrphanTransactions` structures and sending `INV` messages to them as well as the sink set. As the nodes of the source set are not aware that the marker transactions have inputs that are double spends of their own parent transactions, they would be accepted as orphan transactions, allowing for the same inference. This would improve the method by limiting the number of iterations to $\lceil \frac{N}{100} \rceil$, as it would be sufficient to include each node in the source set at least once.

Alternatively, this method could be re-framed as a node-specific inference technique. If we have a singleton source set of one node, then we only need to generate one orphan transaction, so we don't need to empty any `MapOrphanTransactions`

structures. Now we can view it as a fairly reliable, unintrusive and quick way of finding all the peers of a given node. While this doesn't meet the goal of finding the entire network topology, this would be a very effective method of mounting an attack requiring the knowledge of a node's peers.

TxProbe cleverly exploits the idiosyncracies of the Bitcoin protocol, but for this reason it also rests on them heavily. It relies on the way orphan transactions are stored and propagated, how `MapOrphanTransactions` evicts entries, the ability to conduct `InvBlock`, and the way conflicting transactions are dropped. If any of these were altered, the method would become unusable. This is a new and overly effective and damaging technique. We should not expect to see the protocol developers allowing this to continue far into the future.

## 5.5 2019: Daniel, Rohrer, and Tschorsch

### [33] Map-Z: Exposing the Zcash Network in Times of Transition

In an attempt to create a technique that would be immune to changes in protocol, Daniel, Rohrer and Tschorsch created a passive, network-wide, statistical method for inferring the ZCash P2P network. It is based on the earlier statistical method by Neudecker et al. [21], and applied to block arrival times on the ZCash network.

Zcash [34] is a blockchain based digital currency that was developed to improve upon Bitcoin's security features. It features a method of zero-knowledge proofs [35] that allow for anonymous transactions that are still publicly verifiable. Otherwise, the protocols are very similar.

Apart from the network, the key differences in their method are that it focuses on block arrival times instead of transactions; and that a passive measuring approach was taken, so using this method does not require generating transactions. In doing so, the researchers restricted their focus to the peers of nodes that mine blocks or introduce them to the network from mining pools. This is a useful area of study as a significant aspect of the health of the network is ensuring that nodes generating blocks are adequately connected, so they can receive transactions and broadcast blocks effectively.

### 5.5.1 Inference of shortest path length

Similarly to in Section 5.2.2, we define random variables $D$ the arrival time of a block sent from node $v_1$ to $v_2$, $C_{\min}$ the shortest path length between those two nodes, $\Lambda$ the latency between two peers, and $\delta$ the processing delay at a node, which

is the time between a node receiving and sending a block. We assume that $v_1$ is the source of the block, a mining node, and that all nodes and pairs of nodes draw from the same distributions $d$ and $\lambda$. So we have

$$P(D = t|C_{\min} = l) = (\Lambda^{*l} * \delta^{*l})(t) \tag{5.11}$$

where $*$ denotes convolution, and $a^{*b}$ denotes $a$ to the $b$ convolution power. Using Bayes theorem we have

$$P(C_{\min} = l|D = t) = \frac{P(D = t|C_{\min} = l)P(C_{\min} = l)}{P(D = t)}. \tag{5.12}$$

We can find $P(D = t)$ from the law of total probability $P(D = t) = \sum_l P(D = t|C_{\min} = l)$. We can find $P(C_{\min} = l)$ by assuming that the network is an Erdos-Renyi random graph with mean degree $\overline{\deg}$ and total number of nodes $n$. The researchers interpreted this as a geometric distribution.

$$P(C_{\min} = l) = \left(1 - \frac{\overline{\deg}}{n-1}\right)^{l-1} \cdot \frac{\overline{\deg}}{n-1}. \tag{5.13}$$

We model the latency $\Lambda$ and node processing delay $\delta$ with normal distributions. $\Lambda(x) \sim N(s; \mu_\Lambda, \sigma_\Lambda^2)$; $\delta(x) = N(x; \mu_\delta, \sigma_\delta^2)$. From this we can construct the probabilities

$$P(t - \epsilon < D \leq t + \epsilon|C_{\min} = l) = \int_{t-\epsilon}^{t+\epsilon} N\left(x; l(\mu_\Lambda + \mu_\delta), l(\sigma_\Lambda^2 + \sigma_\delta^2)\right) dx \tag{5.14}$$

where $\epsilon$ is a tolerance variable on measurement error.

Daniel et al. parameterised $\Lambda$ based on publicly available global latency measurements. Parametrisation of $\delta$ is compared experimentally between previous research [36] and experimental research on the average processing time of Zcash nodes.

### 5.5.2 Accuracy

The researchers set up one monitor node and one node for validation of its connections. They used an error tolerance of $\epsilon = 5ms$ and assumed a range of distances between nodes of 1 to 9, inferring the distance with the highest mean probability as the the likeliest distance. Using empirical measurements of processing time, they were able to achieve a recall of 82.5% and precision of 50%.

### 5.5.3 Discussion

This method provides advantages and disadvantages over method provided by Neudecker et al. [21], and has similar issues. This method is more accurate, however its scope is limited to measuring connections with mining nodes. The ability to passively measure timings makes it cheaper, less intrusive, and robust to countermeasures against active measurements such as trickling.

Similarly to the previous method, an Erdos-Renyi random graph for the network is assumed to construct a prior for the minimum paths between nodes. We have seen experimentally that at least the Bitcoin P2P network and testnet are not random graphs [20, 28], and that the inference used is only an approximation for path lengths greater than 2. It also assumes independence of paths, which does not capture information that may be useful to creating more accurate inference.

Using an Erdos-Renyi random graph model, they calculate the distribution of the shortest path lengths to be geometric, with probability parameter $\frac{\overline{\deg}}{n-1}$, which would model the number of iterations of independent Bernoulli trials until the first success. This interpretation is incorrect. We can speculate how they could arrive at this conclusion, but it would rely on the assumption that we can draw independently the probability that we can draw from samples of degree distributions independently. However, degree distributions of nodes are not independent in a Renyi-Erdos random graph, as knowledge of the degree of some nodes will inform the distribution of the degrees of other nodes, due to either a fixed number of edges or an independent probability of edges existing. Further, we can simulate the distribution for $C_{\min}$. We can construct an Erdos-Renyi random graph with say, 10,000 vertices and 40,000 edges (similar to what we expect in the Bitcoin P2P network). Applying Dijkstra's Algorithm [37] to find the shortest path lengths, we can see the plot of an example histogram in Figure 5.11. Clearly this does not come from a geometric distribution, which is characterised by nonnegative monotonically decreasing probabilities.

A key assumption the researchers make is that both latency and processing delay are normally distributed, which they provide no evidence for. While the parameterisation for processing delay is experimentally optimised, latency is again entirely modelled by geographical distance which may be too simple a generalisation.

The validation they use may not be a very good estimate. They only use one monitor node and one validation node, which is a small sample size.

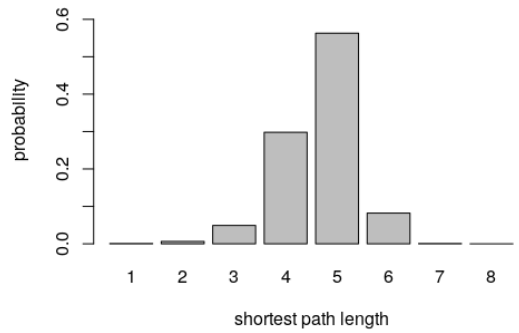



Figure 5.11: Simulated shortest path length distribution for an Erdos-Renyi graph

## 5.6 Similar and ongoing research

### 5.6.1 2018: Deshpande, Badis and George

#### [38] BTCmap: Mapping Bitcoin Peer-to-Peer Network Topology

These researchers perform a crawl of the Bitcoin nodes, finding the addresses stored in each node's `memPool` structure. Based on this, they recreate the Bitcoin P2P network using the algorithms that nodes follow to create their connections. While this is not a measurement of the exact topology of the Bitcoin P2P network, it does provide a possible probabilistic outcome given the methods that Bitcoin nodes use, and so they use repeated measurements of these resulting networks to characterise various important traits of the real network.

### 5.6.2 2019: Ben Mariem

#### [39] Vivisecting Blockchain P2P Networks

In a his thesis investigating properties of blockchain P2P networks, Ben Mariem outlines a passive, network-wide, statistical method for inferring the Bitcoin P2P network. It is based on the literature surrounding infection cascades, which is discussed in Section 6.1.1. The steps he outlines are:

1. Collect a transaction's arrival times from every reachable peer at a monitor node $v_m$;
2. Infer the source $v_0$ of the broadcast and it's initial broadcast time $t_0$;
3. For each node $v_i$, infer the propagation time along the path from $v_0$ to $v_m$ through $v_i$;

4. Reconstruct the tree that the describes the path the transaction travelled through to reach every node, representing the transaction's exact broadcast through the network;

5. After collecting many of these tree, reconstruct the topology as the union of their edge sets.

A proof of concept of this algorithm is part of his ongoing work.

### 5.6.3 Other blockchain networks

Research is being performed on other blockchain implementations. However they are studied to a far lesser degree than Bitcoin. These papers look at idiosyncratic methods for discovering the topology of the Etherium [40] and Monero [41] P2P networks respectively.

# Chapter 6

# Discussion

Having explored these methods of inferring the Bitcoin P2P network topology, we now take a step back and provide a meta-analysis of this research topic. Understanding the trends and results should give us some insight into the evolution of this problem.

In trying to formulate recommendations for where the research should proceed, it is important to understand the interaction between the researchers in these papers and the developers of the protocol. While the topic of privacy was addressed in Nakamoto's original paper [1] via the suggestion of keeping public keys anonymous, some have argued [42] that anonymity was never a significant design goal of Bitcoin. Other cryptocurrency implementations such as ZCash and Monero have appeared for this very reason, with anonymity built in as a primary goal. In any case many believe incorrectly that transactions in Bitcoin are anonymous, and the developers of the Bitcoin protocol attempt to make it as anonymous as possible. Knowledge of the topology can undermine this attempt, and facilitate various attacks as discussed in the introduction.

Broadly speaking, researchers in this topic area have tried to develop techniques for two reasons. Primarily, they wish to investigate the network, it's properties, and the health of the system as it's own goal, to develop knowledge on the complex Bitcoin ecosystem especially as gains relevancy in many societies. However, there is also a second motive of wanting to inform the protocol developers of vulnerabilities. For example, Neudecker et al. [21] show how trickling has impeded their inference, and include a section on the analysis of trickling parameterisation, showing how it can be used to prevent similar attacks. This is analogous to the practice of penetration testing in security systems, where attacks are performed benevolently to evaluate the security of the system.

We will no doubt see this interplay of conflicting priorities play out in future

papers and protocol updates. However, we can observe a trend hinting at movements towards methods that require less assumptions on their models, and therefore less reliance on idiosyncracies of the protocol, as seen in the most recent papers by Daniel, Rohrer and Tschorsch [33] and Ben Mariem [39].

## 6.1 Fields of research

While the methods reviewed differ in many ways, they do share a significant commonality: they have not looked far into other fields for inspiration on how to tackle the problem of topology inference. To some extent, these researchers may have been trying to reinvent the wheel, a problem arising in many fields. There is an extremely large literature surrounding topology inference. In this section we will discuss some approaches in related fields that may be useful.

### 6.1.1 Infection cascades

The method of information propagation used by the Bitcoin P2P network is known by several names including an *epidemic protocol*. We can think of transactions in Bitcoin as contagions that spread like a virus, infecting nodes and moving through their connections. This modelling has been used to study virus propagation in the real world, but also as the basis for communication on various other implementations of P2P networks.

In the broad class of stochastic models known as infection cascades, we have a graph $G = (V, E)$ and an initial vertex $v_0 \in V$ is randomly chosen as the source of a contagion. At some time, the vertex can pass the contagion onto a neighbour, who is infected with some probability according to the *weight* of the edge. This process continues throughout the graph until every node is infected or the contagion ceases spreading. In the case that no vertex can be infected more than once, there cannot be a cycle in any path of the contagion, and so the process maps out a tree consisting of infected nodes and the edges that the contagion crossed.

For instance, the common Susceptible-Infected-Recovered (SIR) model used in mathematical epidemiology [43] can be applied to an infection cascade. We model vertices as susceptible, with one initial infected node. Infected vertices either spread the infection to their neighbours with some probability, or enter the recovered state, where they no longer infect neighbours and are immune to future infection. This can happen in either in discrete time or continuous time where infected vertices attempts at some random times to infect its neighbours or recover.

We can model the Bitcoin P2P network as a simpler Susceptible-Infected (SI) model in continuous time, where the source of the transaction is the initial infected vertex, and the weight of all edges are 1. This corresponds to a guarantee of infection on all edges - a guarantee that if a node sends a transaction to a neighbour who hasn't heard of it, they'll request that transaction.

Using this framework, we can explore the literature on inferring epidemic graphs in continuous SI systems. Here it is noted that even with perfect knowledge of infection times, the problem of finding the mostly likely graph to explain those times is NP-hard [44]. The majority of models focus on perfect knowledge of infection times [45, 46, 47], with extended research into inference from individual time snapshots of the state of the contagion, or time series data [48]. However, where we use monitor nodes to detect transaction arrival times, we cannot find the exact times that nodes receive a transaction, as random propagation effects take place between when a node receives a transaction and when it is forwarded onto us from that node. Recently, some headway has been made into inference from noisy time observations [49], but only cases of a discrete time cascades.

It would be useful to see if some previous modelling could be applied to noisy time observations in the continuous case. Alternatively, results have been achieved using knowledge of the times of infection of a subset of nodes [50]. We could interpret our monitor timing measurements rather as exact times on new nodes. For every node $v_i \in V$, we construct a node $u_i$ and the edge $(u_i, v_i)$. Rather than a noisy time measurement of $v_i$, we interpret this is an exact time measurement of $u_i$, and no measurement of $v_i$, and use the theory available on sparse timing measurements.

### 6.1.2 Network tomography

Network tomography is the study of network properties by observing end-point data, which is the data retrievable at the nodes of a system. This is also what we are attempting to achieve. This field is largely driven by efforts to characterise the internet. The internet has many functionalities and properties: in general, the internet is characterised as a client-server network [51], but in other parts we can observe peer-to-peer behaviour, even if not operating a gossip protocol. For instance, consider *torrenting* [52], the practice of efficient replication and distribution of files across computers on a P2P network using the internet. Here there have been studies into network topology using timing delay measurements [53].

Group communication over the internet is commonly performed through a communication method known as *multicast*, where a single computer wishes to send information to a group of destination computers simultaneously. In these networks,

the messages have to travel via a group of intermediary nodes such as other entities on the internet like as routers or switches. There is significant literature surrounding topology in multicast systems using delay or similar measurements [54, 55, 56]

### 6.1.3 Machine Learning

Classification problems are a very common topic of study in the field of machine learning. While graphs frequently investigated, it is usually in the context of predicting new edges given pre-existing knowledge of the topology, such as on predicting new friends in a social network [57]. A seminal paper by Vert and Yamanishi [58] introduced the problem of supervised network inference. However, the large bulk of literature comes from computational biology in modelling protein and gene networks, and are not necessarily applicable to this problem.

## 6.2 Moving forward

We categorise the methods used in Chapter 5 in Figure 6.1. It is hard to place a value judgement on whether research should aim to be node-specific or network-wide, as they both have their uses. A successful node-specific method would expose vulnerabilities for attacks and allow for more accurate validation, whereas network-wide approaches provide information about the entire network, which can be used to study its properties. Even considering the Map-Z method in [33], a focus on just the topology surrounding mining nodes can have its use.

Figure 6.1: Categorisation of methods

| | passive | active |
|---|---|---|
| idiosyncratic | AddressProbe | Grundmann et al.<br>TxProbe |
| statistical | Map-Z<br>Ben Mariem | Neudecker et al. |

However in considering the direction of future research, it may be prudent to develop methods that are both passive and statistical. Active approaches may be very expensive given the need for transactions, as in Grundmann et al.'s method of transaction accumulation [32], or highly invasive, as in TxProbe [28]. The passive methods discussed did not encounter these issues, as they simply connect to the network and take measurements. Idiosyncratic methods have had the most success, with both AddressProbe and TxProbe being highly accurate, but their success rests on minute details of the underlying protocol for which they are built. While it is

useful to find vulnerabilities in the protocol, it is highly likely that the protocol developers will alter the protocol to stop methods from being used for too long. Statistical methods still encounter this problem, as in [21], but this is due to the statistics resting on known distributions that are part of the protocol or implementation. A statistical method further removed from the protocol, as in Map-Z [33], is more likely to stand the test of time.

Ideally, we would like a statistical model based on transaction arrival times that does not make assumptions about the network it is studying. If we can create this, the model would be impervious to changes in protocol, and would be generalisable to other P2P networks where we study timing delays. We explore this possibility in Part III.

# Part III

# Simulation and Learning

# Chapter 7

# Bitcoin Simulation

In considering the problem of inferring the topology of a P2P network employing a gossip protocol, we have now motivated a model that passively measures message time arrivals and that should not be based on any further assumptions from the specific behaviour of nodes on that network. Over the next two chapters, we investigate using nonparametric techniques how information is stored in the structure of a delay correlation matrix, and then based on this present a proof of concept for learning the edges of a network using a training-validation approach.

## 7.1 Intuition

We have an undirected P2P network running a gossip protocol, where a node forwards on a message to a given peer at a time given by some unknown probabilistic distribution. We consider two nodes $v_1$ and $v_2$, which are connected via many different paths in the network. We expect that the shorter the minimum path length, and the greater the number of paths, the shorter the time should be on average for a message to be sent between them. In particular, if the two nodes are peers, we expect that the time at which they receive messages should be similar. Either $v_1$ sends the message to $v_2$ or vice versa, in which case we only incur a delay of one path length, or they receive the messages from other peers at a similar time, the difference between those times being short enough to be less than the delay incurred by the one path between them.

We cannot say that the reverse is true, however. Given two nodes receive a message at similar times, we cannot say they are peers: is possibly just coincidental. However, if they are not peers, we would not expect this coincidence of similar arrival times to consistently continue. Therefore any given node's timing measurement should be more highly correlated with its peers than with nodes further away, given a

large number of measurements. Similar approaches have been taken to infer network topology in computational biology [59] and multicast network tomography [54].

## 7.2 Delay correlation

Given a graph $G = (V, E)$, set of $n$ nodes $v_i, \ldots, v_n \in V$ and a monitor node $v_m$ connected to all of those nodes, for a message $tx_k$, the monitor node $v_m$ will receive the message $tx_k$ from node $v_i$ at arrival time $tr_{ik}$. We take the first time we receive the message

$$tr_k^* = \min_{i \in \{1,\ldots,n\}} (tr_{ik}), \tag{7.1}$$

and subtract that from each arrival time to give the delay

$$t_{ik} = tr_{ik} - tr_k^*. \tag{7.2}$$

We define a non-negative random variable $X_i$ to describe the delays $t_{ik}$ at node $v_i$. We desire a *nonparametric* model, not making any assumptions on the distribution of $X_i$. We model the delays $t_{ik}, k \in \{1, \ldots, m\}$ as independent outcomes $X_i$. This independence makes a simplifying assumption that the messages themselves trace out independent trees on the underlying true graph. We note that this may not be true in general, as messages may interfere with each other, (for instance if a node is slowed by having to process multiple messages simultaneously). Given $m$ messages, we can represent this information in the measurement matrix $T$,

$$T = \begin{bmatrix} t_{11} & t_{12} & \ldots & t_{1m} \\ t_{21} & t_{22} & \ldots & t_{2m} \\ \vdots & & \ddots & \vdots \\ t_{n1} & t_{n2} & \ldots & t_{nm} \end{bmatrix}, \tag{7.3}$$

where the delay times of each node $v_i$ are given by the row $i$ of $T$.

A simple measure of dependency between two delay random variables $X_i$ and $X_j$ is the Pearson correlation $\rho_{ij}$, which gives the linear dependency of the two random variables. If for $i \in \{1, \ldots, n\}$ we have $E[X_i] = \mu_i$ and $var(X_i) = \sigma_i^2$, $\rho_{ij}$ is given by

$$\rho_{ij} = \frac{E[(X_i - \mu_i)(X_j - \mu_j)]}{\sigma_i \sigma_j}. \tag{7.4}$$

We define the sample correlation $r_{ij}$ between the observation sets $\{t_{i1}, \ldots, t_{im}\}$

and $\{t_{j1}, \ldots, t_{jm}\}$ as

$$r_{ij} = \frac{\sum_{k=1}^{m}(t_{ik} - \bar{t}_i)(t_{jk} - \bar{t}_j)}{\sqrt{\sum_{k=1}^{m}(t_{ik} - \bar{t}_i)^2 \sum_{k=1}^{m}(t_{jk} - \bar{t}_j)^2}}, \tag{7.5}$$

where for any $i \in \{1, \ldots, n\}$, $\bar{t}_i = \frac{1}{m}\sum_{k=1}^{m} t_{ik}$. Note that $r_{ij} = r_{ji}$, and $r_{ii} = 1$. In general, we cannot say that the sample correlation converges to the correlation, but we make this simplifying assumption so that, with more messages, we assume higher accuracy of our estimated correlation. We can represent the sample correlations in the matrix $R$,

$$R = \begin{bmatrix} 1 & r_{12} & \ldots & r_{1n} \\ r_{21} & 1 & \ldots & r_{2n} \\ \vdots & & \ddots & \vdots \\ r_{n1} & r_{n2} & \ldots & 1 \end{bmatrix}, \tag{7.6}$$

Now we have $\binom{n}{2}$ distinct $r_{ij}$ entries in $R$, corresponding to one correlation measurement for each edge. The question now is whether we retrieve the correct classification of edges from the correlations, with no further assumptions on the underlying graph. Is all the information we need stored in $R$? We now attempt to find some algorithms $f$ such that $E \approx f(R)$.

## 7.3 Simulation setup

In order to answer this question, we will perform some experiments on a simulation of the Bitcoin P2P network. In the paper by Neudecker et al. [21], a statistical delay model was tested on a simulation with 100% recall and 90% precision, which became only 40% recall and precision on the real network. Grundmann et al.'s [32] idiosyncratic double spend method achieved 96% recall and 94% precision in simulation, which became 87% recall and 71% precision on the real network. Similarly, in Daniel et al.'s [33] statistical mehtod, 100% recall and 50% precision decreased to 82.5% recall and 50% precision when tested on the real network over one measurement.

While simulation results may be misleading, it is important to keep in mind that in these papers, the methods they used and the simulations they developed came from the same modelling of the network, built on knowledge of the Bitcoin protocol. Therefore the accuracy in the simulations would have been reported as higher than in the real network due to the shared assumptions of the simulation and methods; methods not necessarily shared in the real network and method. For instance, Neudecker et al. assume a Erdos-Renyi graph for both their model and simulation, which gave misleading figures when the model was tested on the simulation. The

accuracy dropped in part due to the Bitcoin P2P network not being a true random graph.

If we use different assumptions for our simulation and model, then we may be able to improve our model to create a more general method for approaching many different networks. In our model, we have assumed that the sample correlation $r_{ij}$ converges to the real correlation $\rho_{ij}$, and that $\rho_{ij}$ contains information on the edges. We will not use these assumptions in the simulation. We acknowledge the common assumptions for later discussion:

1. the network is an undirected graph;

2. nodes relay messages to peers after receiving them, at times according to some probabilistic distribution;

3. messages propagating through the network are independent events.

### 7.3.1 Simulated gossip protocol

We simulate a P2P network by generating an Erdos-Renyi random graph $G = (V, E)$ with $n$ vertices $v_i \in V$ and $N$ edges $e_i \in E$. In order to account for various trickling, latency, and other delays both intentional an unintentional, we generate delay distributions. Each node $v_i$ is given a relay delay $\Lambda_i \sim \text{Exponential}(\lambda_i)$. We also assign each node $v_i$ a processing delay $\Pi_i \sim \text{Uniform}(a, \pi_i)$ where $\pi_i \sim \text{Pareto}(\gamma, \delta)$. $\lambda_i \sim \text{Gamma}(\alpha, \beta)$ and $\pi_i$ are drawn once for each node at the start of the simulation. We can parameterise delay in the simulation by $a, \alpha, \beta, \gamma$ and $\delta$. In particular $a$ lets us control the minimum time between a message arriving at a node and the relaying of a message to a peer.

When a node $v_i$ receives a message, it draws a relay delay for each of its peers $v_j$ from $\Lambda_i$. Each of those peers then adds a delay drawn from $\Pi_j$ before it can send the message onto its own peers.

Miller et al. [20] found that about 2% of nodes were responsible for 75% of the mining power. We want to model messages generally, whether they are transactions or blocks, so we allow for this heavy-tailed behaviour. We wish for a distribution where $d_n$ proportion of nodes account for $d_s$ proportion of message sources.

Consider $\xi \sim \text{Pareto}(1, c)$. We define and calculate the discretised truncated

Pareto distribution for random variable $D$,

$$\begin{aligned} P(D = i|n) :&= P(i \leq \xi \leq i+1 | \xi \leq n+1); \\ &= \frac{P(i \leq \xi \leq i+1, \xi \leq n+1)}{P(\xi \leq n+1)}; \\ &= \frac{P(i \leq \xi \leq i+1)}{P(\xi \leq n+1)} \mathbf{1}\{i \leq n\}; \\ &= \frac{1}{1 - \left(\frac{1}{n+1}\right)^c} \left[\frac{1}{i^c} - \frac{1}{(i+1)^c}\right] \mathbf{1}\{1 \leq i \leq n\}. \end{aligned} \tag{7.7}$$

where the value for $c$ is calculated numerically as the solution to

$$d_s = P(D \leq d_n \cdot n | n) = \frac{1}{1 - \left(\frac{1}{n+1}\right)^c} \left[1 - \frac{1}{(d_n \cdot n + 1)^c}\right] \tag{7.8}$$

Now we can simulate the cascade of a message $tr_k$ through the graph $G$. We pick a node $v_i$ according to distribution $D$, with its arrival time set to 0, and all other arrival times set at $\infty$. In each iteration we pick the node with the lowest arrival time that hasn't propagated the message before. We draw new arrival times for each of its peers $v_j$ as an outcome of $\Lambda_i + \Pi_j$, updating the arrival time if it is less than the one stored. Finally, for each node $v_i$, we add to the arrival time a sample from $\Lambda_i$ representing the time $tr_{ik}$ that the monitor node receives the message from $v_i$. Across multiple cascades, we record all the $tr_{ik}$ values and build the correlation matrix $R$.

## 7.4 Edge detecting algorithms

Now we can start developing nonparametric algorithms to reverse engineer the problem, determining $E$ from $R$. We essentially use all of test $E$ as both a training set to develop these algorithms, and the validation set to test them. The goal here is to see if information about $E$ can be recovered from $R$. Of course a trivial method would be to enumerate all possible edge combinations, then select values in $R$ that correspond to the correct edges: on testing, we would have all the right edges. However we wish to see if there is a structured approach we can take that can give us structural insights about $R$.

### 7.4.1 MaxCorr

Consider this very basic approach in Algorithm 1: for each node $v_i$, we find the maximum correlation in its row $i$ in $R$, and take the corresponding column as a peer. At best this approach will predict $n$ edges, giving a maximum recall of $\frac{n}{N}$ or $\frac{2}{\overline{deg}}$, which bounds the F1 score from above at $\frac{4 \cdot \text{precision}}{2 + \text{precision} \cdot \overline{\text{deg}}}$ for all values of $\overline{\text{deg}} \geq 2$.

### 7.4.2 ThreshCorr

Improving on this in Algorithm 2, we set a threshold value $r^*$, and for all correlations $r_{ij} > r^*$, we take $r_{ij}$ to be an edge. Another way of thinking about $r^*$ is that it defines the number $k^*$ edges to include. We take all $\binom{n}{2}$ distinct values $r_{ij}$, and order them by size, taking the $k^*$ edges with the highest corresponding correlations. This method is not bounded in recall or precision. It is preferable to work with $k^*$ rather than $r^*$, as $k^*$ is an integer. This allows us to fully compute the various accuracy statistics as functions of $k^*$, as opposed to $r^*$, on which we are not sure which values the accuracy functions will change. Through computing values of an accuracy statistic, we can optimise the choice of $k^*$.

### 7.4.3 EdgeCorr

Taking this idea of thresholds further in Algorithm 3, we consider for each node $v_i$ some threshold $r_i^*$. For all entries in the $i$th row of $R$, include all edges that have corresponding correlations $r_{ij} \geq r_i^*$. Again, this can also be though of as a number $k_i^*$ of edges to include for each node, taking the $k_i^*$ edges corresponding to the largest correlation values. As there will be $n$ threshold values, and for each node between 0 and $n-1$ edges to include, there are $n^n$ combinations of parameters. This is an intractable number to enumerate for any problem that we wish to consider. We must therefore employ optimisation algorithms. We will use coordinate ascent (CA) and simulated annealing (SA) as described in Section 2.4.

## 7.5 Results

We ran the simulation with $n = 500$ nodes, average degree $\overline{\text{deg}} = 8$ ($N = 2000$ edges) for 10,000 messages. Given that in the last seven days, at the time of writing, an average of 3.7 transactions per second entered the Bitcoin network [60], this would correspond to 45 minutes of passive listening time. We set $(a, \alpha, \beta, \gamma, \delta, d_n, d_s) = (20, 20, 5, 1.25, 20, 0.1, 0.5)$, with $n = 500$ nodes, $N = 2000$ edges. We ran 10,000 messages.

We will compare the algorithms based on recall, precision, and F1 score. Recall that the F1 score is

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{recall} \cdot \mathrm{precision}}{\mathrm{recall} + \mathrm{precision}}, \tag{7.9}$$

with $0 \leq \mathrm{F1} \leq 1$. An algorithm with an F1 score of 1 classifies perfectly. We use F1 scores here predominantly as it lets us define an objective function to maximise, measuring the accuracy of a model with a scalar value that takes many factors into account.

#### `MaxCorr`

First we consider `MaxCorr`, picking the edge with the highest correlation for each node. We compared this classification with the true network, resulting in a recall of 22.6% and a precision of 97.4%, with an F1 score of 0.37. This is very close to the theoretical limit of recall at 25%. This method does not capture a good proportion of the edges, but is highly accurate for those that it does predict. This shows that correlation is an extremely good predictor when the correlation is relatively high for the node.

#### `ThreshCorr`

Implementing `ThreshCorr`, we calculate the F1 score for each value of $k^*$, which is the model that predicts the $k^*$ edges with the highest correlations. In Figure 7.1 we choose a computationally feasible range of values of $k^*$ between 0 and 8000, and pick the model with the highest F1 score. The highest range of F1 scores is reached between 1000 and 3000 nodes, which we can compare with the total simulated edges $N = 2000$. As for larger numbers of nodes, computing all $\binom{n}{2}$ $k^*$ may be infeasible, but we can assume that this method will give best results close to $N$. We note the presence of local maxima throughout the plot, especially surrounding the global optima. This will be necessary to consider for the next threshold methods. The best model that this method would be able to predict given this data set is at $k^* = 1371$ with 39.5% recall, 57.6% precision, and 0.46 F1 score.

#### `EdgeCorr`

In implementing `EdgeCorr`, we require optimisation of $n = 500$ threshold values $k_i^*$, one for each node. There is a discrete solution space of $500^{500} \approx 10^{1350}$ solutions to explore. We first use CA. We consider the outcome of `ThreshCorr`, which gives a best guess of edges between 1000 and 3000, corresponding to average node degree between 4 and 12. We pick a starting solution of all $k_i^*$ at 8, and for a solution

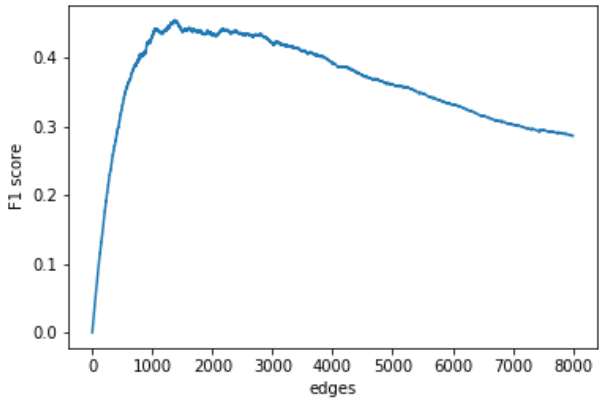



Figure 7.1: `ThreshCorr`: F1 score against $k^*$ edges included.

$x = (x_1, \ldots, x_{500})$ define the coordinate neighbourhood as all solutions within a reasonable distance of 3 from the given coordinate value, with all other coordinates being kept equal, $\mathcal{N}_i(x) = \{y \in \mathbb{N}^{500} | y_i \in [x_i - 3, x_i + 3], y_j = x_j, j \neq i\}$. Recall that this algorithm ceases when it goes through $n$ iterations without changing.

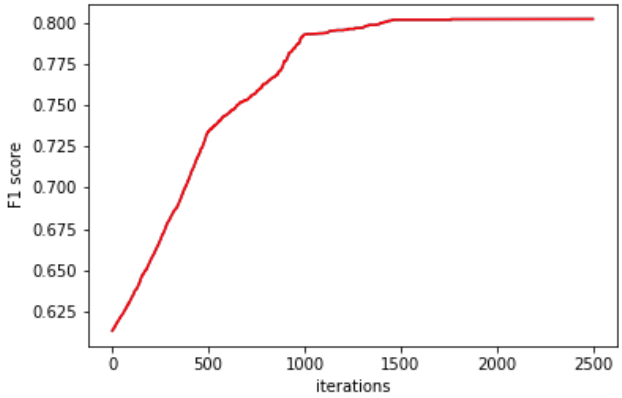



Figure 7.2: `EdgeCorr`: F1 score against iterations of CA.

In Figure 7.2, we observe how CA progresses over its iterations, rising monotonically to a local minimum at F1 = 0.80, defining a model given by the resulting set of $k_i^*$ values with a recall of 79.4% and precision of 81.0%. It is interesting to observe that the starting solution of all $k_i^* = 8$ gives a better F1 score than the optimal model using `ThreshCorr`. We can conjecture that some nodes have lower correlations on average than others, and that it is the correlations that are highest (relatively) within that node that is the better predictor, rather than highest correlations in general.

We also perform selection of thresholds with the SA algorithm. Given the low

"energy" needed to pass through the solution space, we set the initial highest temperature at $T_i = 0.0005$. We set the final temperature to just above 0, and choose a simple cooling schedule. Given that we want $I$ iterations, we linearly increment the temperature at $s(T) = T - \frac{T_i}{I}$. We define the same neighbourhood as in CA, however in this case instead of cycling through the coordinates and finding the best solution within a distance of 3, in each iteration we pick a coordinate at random and choose a random change between -3 and 3.

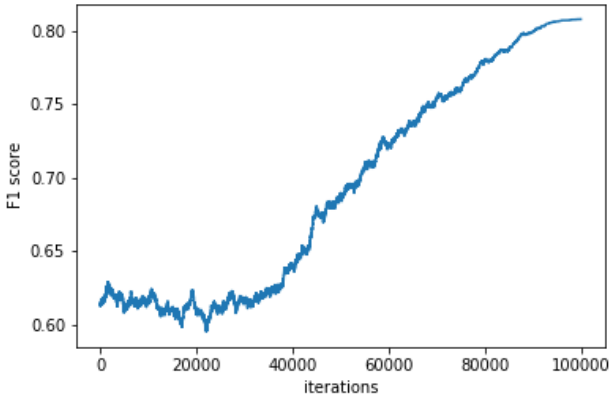



Figure 7.3: `EdgeCorr`: F1 score against iterations of SA.

In Figure 7.3, we can see how SA progresses through its iterations. SA benefits from longer run-time [61], and so we ran it for 100,000 iterations. We can see the characteristic fluctuations of the curve as it accepts worse solutions, far more at the start and then decreasingly as it progresses. We eventually arrived at a model with 79.6% recall and 82% precision, with an F1 score of 0.81.

## 7.6 Discussion

Figure 7.4: Table of accuracy results

| | recall | precision | F1 |
|---|---|---|---|
| `MaxCorr` | 22.6% | 97.4% | 0.37 |
| `ThreshCorr` | 29.5% | 57.6% | 0.46 |
| `EdgeCorr` with CA | 79.4% | 81.0% | 0.80 |
| `EdgeCorr` with SA | 79.6% | 82.0% | 0.81 |

Considering the results in Figure 7.4, we can conclude that through exploring a variety of methods that information of the edge set of this simulated P2P network is contained within its correlation matrix $R$ derived from timing measurements.

To at least some extent, this information is retrievable through considering the ordering of correlation values. Of particular interest is the result for `MaxCorr` which demonstrates extremely high precision in predicting a proportion of the edges in a remarkably simple way. This could be combined with other models to improve overall accuracy.

The method used in `ThreshCorr`, while not the most striking of the results, is interesting in that it only uses one value $k_*$ to classify edges and has a result that is far better than random. Perhaps there is some operation that could be performed on the correlation matrix, even using a different dependency metric from the Pearson correlation, that would allow for edges to be separated more cleanly. Or perhaps the sample Pearson correlation is not the best dependency metric

In foreseeing the results of applying this methodology to real networks, we must challenge the common assumptions that were used to create both this simulation and model. First we assumed that the network is undirected, which is not true of the Bitcoin P2P network as the delay distributions for incoming and outgoing peers are different, as discussed in Section 5.3. Secondly, we assumed arbitrary distributions as part of the process of message propagation, whereas the protocol includes trickling countermeasures that optimise obfuscation of timing analyses. However, the authors of Map-Z claimed that their passive method was immune to trickling. Thirdly, we assumed independence of messages, which may not be true in general as there are peer processing delays and bandwidth delays to consider when multiple messages are simultaneously propagating through the network.

# Chapter 8

# Learning on Message Time Correlation

Here we formalise the methods from Chapter 7 into a machine learning approach. In the last chapter, we used full knowledge of the graph edges to test to what extent it is possible to recover information from the structure of the correlation matrix. In this chapter, we test to what extent it is possible to learn the edges of a graph using only timing measurements, providing a proof of concept for applying passive, nonparametric statistical models to this research area.

## 8.1 Experiment setup

We set up our experiment similarly as in Chapter 7, with $n = 500$ network nodes plus an additional 5 testing and 5 validation nodes, average degree $\overline{\deg} = 8$ ($N = 2040$ edges), $k = 10,000$ messages and $(a, \alpha, \beta, \gamma, \delta, d_n, d_s) = (20, 20, 5, 1.25, 20, 0.1, 0.5)$. However, we now also create 5 testing nodes and 5 validation nodes, connected to the network as part of the random graph. Whereas in the last chapter we used knowledge of the whole network to optimise classification variables, here we use a standard machine learning approach. We train the methods using knowledge of the connections of the 5 training nodes, then apply the models to infer the connections of yet unseen validation nodes. This will test the prediction power of the developed models, and give approximations for the accuracy across the whole network.

## 8.2 Results

We implemented the same four methods of testing across this experiment. On the training data (the edges of the training nodes) the methods optimised to achieve

the results in Figure 8.1. Out of 2525 possible edges, there were 38 true edges to deduce.

| | recall | precision | F1 |
|---|---|---|---|
| `MaxCorr` | 21.1% | 88.9% | 0.34 |
| `ThreshCorr` | 47.4% | 52.9% | 0.50 |
| `EdgeCorr` with CA | 78.9% | 96.8% | 0.87 |
| `EdgeCorr` with SA | 81.6% | 86.1% | 0.84 |

Figure 8.1: Optimal models on testing data.

Taking the same predicted edge sets, the models achieved the results in Figure 8.2 based on the edges of 5 unseen validation nodes with 42 true edges out of 2525 possible edges.

| | recall | precision | F1 |
|---|---|---|---|
| `MaxCorr` | 11.9% | 100% | 0.21 |
| `ThreshCorr` | 23.8% | 76.9% | 0.36 |
| `EdgeCorr` with CA | 0.00% | 0.00% | 0.00 |
| `EdgeCorr` with SA | 54.8% | 85.7% | 0.60 |

Figure 8.2: Models applied to validation data.

The obvious feature here is the drop of accuracy to nothing in the coordinate ascent variation of `EdgeCorr`. This is not a mistake. With a total of 2040 edges in the graph, this method only predicted 64, consisting of 31 predictions accounting for the results on the testing set, and another 33 false positives. This is consistent with observations in toy problems in developing these algorihtms, where the CA performed worse as the number of nodes increased. We can conjecture that the greedy nature of CA causes it to drastically overfit, whereas the exploratory nature of SA is much better suited for arriving at solutions that are reflective of the network as a whole.

The other feature to note is the change in accuracy for `MaxCorr`. This method doesn't use any training data; these results can therefore be interpreted as reflective of this method's accuracy. We should interpret the others methods similarly reflective as outcomes of the true distribution of their accuracy. Further experiments should be used to give confidence intervals in a full analysis, however we only show here a proof of concept.

## 8.3 Discussion

Here we provided a proof of concept of the possibility of developing passive, nonparametric statistical approaches to P2P gossip protocol topology that are independent from most assumptions built into the network. Specifically, we used measurements of delay times to create a correlation matrix describing the dependency of nodes. We found that taking the maximum correlation of each node to predict edges was extremely precise, and should be considered as part of other models. We also found that by learning the individual number $k_i^*$ of the highest correlated potential peers for each node $v_i$, we could predict with accuracy as compared with other methods in the literature, provided a good optimisation algorithm is selected.

These methodologies are by no means state of the art. They are merely well-motivated inferences based on simple modelling, with which to prove the possibility of broadening the research area of P2P gossip protocol topology inference. With more considered modelling approaches and well-designed hypothesis testing, we believe these methods can be far outperformed.

# Conclusion

In this thesis we explored methods of inferring the Bitcoin P2P network topology. We conducted a thorough review of the literature, exploring the techniques used based on statistical inference and the idiosyncratic properties of the Bitcoin protocol. Through an meta-analysis of the methods, taking into account the desire of the Bitcoin protocol developers to prevent inference, we motivated recommendations towards researching passive, statistical techniques based on timing measurements of transactions or blocks. Following this, we developed nonparametric statistical models, creating three algorithmic methods as a proof of concept. Through testing via simulation, we were able to show that information of network edges could be deduced from timing measurements of message propagation, free of other assumptions about the network.

# Appendix A

# Algorithms

**Algorithm 1:** `MaxCorr`

**Input :** nodes $v_i \in V$; correlations $r_{ij} \in R$; $i, j \in \{1, \ldots, n\}$

**Output:** Set of edges $E'$

**for** $i \in \{1, \ldots, n\}$ **do**
  $\text{col} = \arg\max_{j=\{1,\ldots,n\}, j \neq i}\{r_{ij}\}$
  $e_i = (v_i, v_{\text{col}})$
**end**
$E' = \bigcup_{i \in \{1,\ldots,n\}}\{e_i\}$

**Algorithm 2:** `ThreshCorr`

**Input :** nodes $v_i \in V$; correlations $r_{ij} \in R$; threshold $r^*$; $i, j \in \{1, \ldots, n\}$

**Output:** Set of edges $E'$

$E' = \emptyset$
**for** $i \in \{1, \ldots, n-1\}$ **do**
  **for** $j \in \{i, \ldots, n\}$ **do**
    **if** $r_{ij} \geq r^*$ **then**
      put $(v_i, v_j) \in E'$
    **end**
  **end**
**end**

**Algorithm 3:** `EdgeCorr`

**Input :** nodes $v_i \in V$; correlations $r_{ij} \in R$; thresholds $r_i^*$; $i, j \in \{1, \ldots, n\}$

**Output:** Set of edges $E'$

```
1 E' = ∅
2 for i ∈ {1, ..., n} do
3 |   for j ∈ {1, ..., n} \ {i} do
4 |   |   if r_ij ≥ r_i^* then
5 |   |   |   put (v_i, v_j) ∈ E'
6 |   |   end
7 |   end
8 end
```

**Algorithm 4:** Coordinate ascent

**Input :** initial $x_0 \in \mathbb{R}^n$; cost $f$; feasible set $\chi$; neighbourhood $\mathcal{N}_i(x)$

**Output:** Solution $x$

```
1  x ← x_0
2  i ← 1
3  while x has changed in the last n iterations do
4  |   x* ← arg max_{x_j ∈ N_i(x)} f(x_j)
5  |   if f(x*) ≥ f(x) then
6  |   |   x ← x*
7  |   end
8  |   if i = n then
9  |   |   i ← 1
10 |   else
11 |   |   i ← i + 1
12 |   end
13 end
```

**Algorithm 5:** Simulated annealing (maximisation)

**Input :** initial $x_0 \in \mathbb{R}^n$; cost $f$; feasible set $\chi$; neighbourhood $\mathcal{N}(x)$; initial temperature $T_i$; final temperature $T_f$; cooling schedule $s$.

**Output:** Solution $x$

$x \leftarrow x_0$
$T \leftarrow T_i$
**while** $T > T_f$ **do**
    randomly pick $y \in \mathcal{N}(x)$
    $\delta = f(y) - f(x)$
    **if** $\delta > 0$ **then**
        $x \leftarrow y$
    **else**
        sample $u$ from Uniform[0,1]
        **if** $u < \exp\left\{-\frac{\delta}{T}\right\}$ **then**
            $x \leftarrow y$
        **end**
    **end**
    $T \leftarrow s(T)$
**end**

# Bibliography


[1] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[2] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, Vignesh Kalyanaraman, et al. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2(6-10):71, 2016.

[3] George Foroglou and Anna-Lali Tsilidou. Further applications of the blockchain. In *12th Student Conference on Managerial Science and Technology*, 2015.

[4] Praneeth Netrapalli and Sujay Sanghavi. Learning the graph of epidemic cascades. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 211–222. ACM, 2012.

[5] Sergi Delgado-Segura, Cristina Pérez-Solà, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joan Borrell. Cryptocurrency networks: A new p2p paradigm. *Mobile Information Systems*, 2018, 2018.

[6] Johannes Göbel, Holger Paul Keeler, Anthony E Krzesinski, and Peter G Taylor. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Performance Evaluation*, 104:23–41, 2016.

[7] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.

[8] 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. ACM, 2014.

[9] Matthias Lei. Exploiting bitcoin's topology for double-spend attacks, 2015.

[10] Paul Erdös and Alfréd Rényi. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[11] Pearson discrete maths solutions. `http://www.maths.lse.ac.uk/Personal/jozef/MA210/08sol.pdf`.

[12] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

[13] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[14] Kristian Bjoernsen. Koblitz curves and its practical uses in bitcoin security. *order (ε (GF (2k)*, 2(1):7, 2009.

[15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Security evaluation of sha-224, sha-512/224, and sha-512/256. 2015.

[16] Nicolas T Courtois, Marek Grajek, and Rahul Naik. Optimizing sha256 in bitcoin mining. In *International Conference on Cryptography and Security Systems*, pages 131–144. Springer, 2014.

[17] Rajul Parikh, Annie Mathai, Shefali Parikh, G Chandra Sekhar, and Ravi Thomas. Understanding and using sensitivity, specificity and predictive values. *Indian journal of ophthalmology*, 56(1):45, 2008.

[18] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[19] Hongge Chen et al. *Novel machine learning approaches for modeling variations in semiconductor manufacturing*. PhD thesis, Massachusetts Institute of Technology, 2017.

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

[21] 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.

[22] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[23] Bitnodes. `https://bitnodes.earn.com/`.

[24] Bitcoin wiki: Protocol documentation. `https://en.bitcoin.it/wiki/Protocol_documentation#Block_Headers`.

[25] Digiconomist. Bitcoin energy consumption index. `https://digiconomist.net/bitcoin-energy-consumption`, 2019.

[26] Aapeli Vuorinen. The blockchain propagation process: a machine learning and matrix analytic approach. https://bitcoin.aapelivuorinen.com/, 2019.

[27] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.

[28] 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. *arXiv preprint arXiv:1812.00942*, 2018.

[29] Testnet. `https://en.bitcoin.it/wiki/Testnet`.

[30] Transaction confirmation. `https://en.bitcoinwiki.org/wiki/Transaction_confirmation`.

[31] Gerhard J Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial optimization—eureka, you shrink!*, pages 185–207. Springer, 2003.

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

[33] Erik Daniel, Elias Rohrer, and Florian Tschorsch. Map-z: Exposing the zcash network in times of transition. *arXiv preprint arXiv:1907.09755*, 2019.

[34] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[35] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 781–796, 2014.

[36] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16. ACM, 2016.

[37] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[38] Varun Deshpande, Hakim Badis, and Laurent George. Btcmap: Mapping bitcoin peer-to-peer network topology. In *2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, pages 1–6. IEEE, 2018.

[39] Sami Ben Mariem et al. Master thesis: Vivisecting blockchain p2p networks. 2019.

[40] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. Measuring ethereum network peers. In *Proceedings of the Internet Measurement Conference 2018*, pages 91–104. ACM, 2018.

[41] Tong Cao, Jiangshan Yu, Jérémie Decouchant, Xiapu Luo, and Paulo Veríssimo. Exploring the monero peer-to-peer network. *IACR Cryptology ePrint Archive*, 2019:411, 2019.

[42] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer, 2013.

[43] William Ogilvy Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character*, 115(772):700–721, 1927.

[44] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4):21, 2012.

[45] Bruno Abrahao, Flavio Chierichetti, Robert Kleinberg, and Alessandro Panconesi. Trace complexity of network inference. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 491–499. ACM, 2013.

[46] Tiago P Peixoto. Network reconstruction and community detection from dynamics. *arXiv preprint arXiv:1903.10833*, 2019.

[47] Caitlin Gray, Lewis Mitchell, and Matthew Roughan. Bayesian inference of network structure from information cascades. *arXiv preprint arXiv:1908.03318*, 2019.

[48] Alfredo Braunstein, Alessandro Ingrosso, and Anna Paola Muntoni. Network reconstruction from infection cascades. *Journal of the Royal Society Interface*, 16(151):20180844, 2019.

[49] Jessica Hoffmann and Constantine Caramanis. Learning graphs from noisy epidemic cascades. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(2):40, 2019.

[50] Xiao Han, Zhesi Shen, Wen-Xu Wang, and Zengru Di. Robust reconstruction of complex networks from sparse data. *Physical review letters*, 114(2):028701, 2015.

[51] Hiroshi Nishida and Thinh Nguyen. Optimal client-server assignment for internet distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 24(3):565–575, 2012.

[52] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. A performance study of bittorrent-like peer-to-peer systems. *IEEE Journal on selected areas in communications*, 25(1):155–169, 2007.

[53] Peng Qin, Bin Dai, Guan Xu, Kui Wu, and Benxiong Huang. Taking a free ride for routing topology inference in peer-to-peer networks. *Peer-to-Peer Networking and Applications*, 9(6):1047–1059, 2016.

[54] Nick G Duffield and F Lo Presti. Multicast inference of packet delay variance at interior network links. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, volume 3, pages 1351–1360. IEEE, 2000.

[55] Nick G Duffield, Joseph Horowitz, and F Lo Prestis. Adaptive multicast topology inference. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 3, pages 1636–1645. IEEE, 2001.

[56] Brian Eriksson, Gautam Dasarathy, Paul Barford, and Robert Nowak. Toward the practical use of network tomography for internet topology discovery. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[57] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[58] Jean-Philippe Vert and Yoshihiro Yamanishi. Supervised graph inference. In *Advances in neural information processing systems*, pages 1433–1440, 2005.

[59] Sinisa Pajevic and Dietmar Plenz. Efficient network reconstruction from dynamical cascades identifies small-world topology of neuronal avalanches. *PLoS computational biology*, 5(1):e1000271, 2009.

[60] Transaction rate. `https://www.blockchain.com/charts/transactions-per-second`.

[61] Lester Ingber. Simulated annealing: Practice versus theory. *Mathematical and computer modelling*, 18(11):29–57, 1993.