TU WIEN Informatics

Ermittlung von IP-Siblings im Bitcoin-Netzwerk mittels Clock Skew

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Philipp Nowak
Matrikelnummer 11776858

an der Fakultät für Informatik

der Technischen Universität Wien

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

Wien, 5. Februar 2021

Philipp Nowak Edgar Weippl

TU WIEN Informatics

IP Sibling Detection in the Bitcoin Network using Clock Skew

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software & Information Engineering

by

Philipp Nowak
Registration Number 11776858

to the Faculty of Informatics

at the TU Wien

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

Vienna, 5th February, 2021

Philipp Nowak | Edgar Weippl

Erklärung zur Verfassung der Arbeit

Philipp Nowak

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

Wien, 5. Februar 2021

Philipp Nowak

Danksagung

Auch wenn diese Arbeit selbstständig verfasst wurde, wäre sie in dieser Form ohne die Beiträge verschiedener Menschen nicht möglich gewesen.

Für die direkte Unterstützung möchte ich mich bei meiner Betreuerin Johanna Ullrich bedanken, ohne deren wertvolle Anmerkungen und Lösungen dieses Projekt über einen Entwurfsstatus wohl nicht hinausgekommen wäre. Auch von großer Bedeutung war ihre ausgeprägte Geduld, die es mir ermöglichte, diese Arbeit neben weiteren Lehrveranstaltungen und meiner beruflichen Tätigkeit ohne zu viel Druck fertigzustellen – Ohne diese Flexibilität wäre das Resultat sicherlich nicht in der vorliegenden Qualität möglich gewesen. Dank schuldig bin ich auch allen Wissenschaftlerinnen und Wissenschaftlern, die die in dieser Arbeit zitierten Quellen verfasst haben. Insbesondere betrifft das Marco Starke, dessen Forschung ein wichtiges Fundament bildet.

Bedanken möchte ich mich auch bei vielen weiteren Akteuren, die nicht zur konkreten Sachmaterie, sondern viel mehr zu den notwendigen Umständen beigetragen haben. Das bezieht sich insbesondere auf meine Familie und Freunde und deren emotionale Unterstützung. Besonders meine Eltern haben mir meine Ausbildung in dieser Form erst ermöglicht. Einen wertvollen Beitrag leisteten auch mein Arbeitgeber sowie viele Kolleginnen und Kollegen, durch deren Flexibilität es mir möglich wurde, neben dem Studium zu arbeiten.

Besonders in der Sitation einer globalen Pandemie war für mich auch das Internet von großer Bedeutung. Ohne die schier unbegrenzte Kreativität so vieler Menschen, neue Verwendungsweisen dieser Technologie zu finden, könnte ich es mir nicht vorstellen, über so eine lange Zeit physisch isoliert zu bleiben. Für mich persönlich besonders wichtig war in diesem Zeitraum die Musik und die diversen Möglichkeiten, die viele Künstlerinnen und Künstler finden konnten, auch ohne physische Konzerte Inhalte bereitzustellen.

Acknowledgements

Despite this work being an independent product of my own work, it would not have been possible without the valuable support of various different people.

For direct contributions, I would like to express my gratitude to my advisor Johanna Ullrich, without whose valuable ideas and solutions, this project might not have ever left a draft status. In particular, her great patience was important, which enabled me to complete this thesis alongside other courses and my professional activities. Without this flexibility, the quality at hand would not have been achievable. Additionally, I am thankful to all scientists whose important research is cited in this work. One major foundation is provided by the efforts of Marco Starke's Master's thesis.

Invaluable contributions were also made by various other individuals, which were not directly targeted at the subject matter, rather allowed the necessary environment for this work. I am deeply appreciative of the emotional support provided to me by family and friends. Especially my parents have played a crucial role in enabling my education. Further, my employed and so many colleagues have provided vital flexibility, which allows me to pursue my studies while working.

Particularly in this global pandemic we are finding ourselves in right now, the internet has been irreplaceable to me. I cannot imagine maintaining the necessary degree of isolation without the virtually unlimited creativity people have found in using this technology. During these times, music specifically was indispensable to me, with so many artists finding ways of providing reassurance even without physical concerts.

Kurzfassung

Durch die Einführung von IPv6 zeigen sich in verschiedenen Bereichen neue Sicherheitsherausforderungen. In dieser Arbeit wird eine konkrete Anwendung, die Kryptowährung Bitcoin, untersucht. Betrachtet wird dafür das Szenario einer *Network Partitioning Attack*, die die durch unterschiedliche Protokolle (IPv4, IPv6) verursachte natürliche Grenze ausnutzen soll. Dabei wird *Clock Skew* verwendet, um verschiedene IP-Adressen derselben Bitcoin-Node zu erkennen. Dieses Prinzip basiert auf fertigungsbedingten Ungenauigkeiten in der Frequenz von in Computern verarbeiteten Uhren. Die Messung wird mittels der *TCP Timestamps Option* durchgeführt.

Um die Anwendung existierender Clock-Skew-Methoden auf Bitcoin zu evaluieren, werden neuartige Metriken präsentiert, die (negative) Zuordnung zusammengehöriger IP-Adressen basierend auf Applikationsverhalten von Bitcoin erlauben. Zusätzlich werden Konzepte für weitere Methoden aufgezeigt.

Für die Datensammlung wird auf Basis existierender Forschung eine Python-Plattform entwickelt, die zusammengehörige IP-Adressen im gesamten öffentlichen Bitcoin-Network erkennen soll. Da sich keine offensichtlichen Vergleichsdaten gewinnen lassen konnten, wird die Qualität der Zuordnungen anhand einer Sammlung unterstützender Metriken evaluiert, die die vorgestellten Bitcoin-Methoden beinhaltet.

Für die neuartigen Bitcoin-Metriken wird festgestellt, dass sie die Bewertungsraten deutlich erhöhen, mit guter Zuverlässigkeit. Existierende Methoden auf Basis von Clock Skew zeigen bei vorgefilterten Bitcoin-Datensätzen keine überzeugende Zuverlässigkeit. Der Angriff wird letzlich als unter gewissen Voraussetzungen theoretisch möglich, aber praktisch undurchführbar eingestuft.

Abstract

New security challenges are presented in various areas by the introduction of IPv6. In this work, one particular application, the Bitcoin cryptocurrency, is explored with regards to a network partitioning attack based on the natural boundary imposed by operating on multiple protocols (IPv4 and IPv6). The proposed method is based on clock skew sibling detection, which utilises frequency differences in computers' internal clocks to identify remote actors. The measurement is conducted using the TCP Timestamps Option.

To evaluate the application of existing clock skew methods to Bitcoin, novel metrics are presented which allow a (negative) sibling decision to be made based on application-layer Bitcoin behaviour. Ideas for further work are also proposed.

For data collection, a Python platform is developed based on existing work, which attempts to detect Dual-Stack nodes over the entire public Bitcoin network. Since no obvious ground truth could be located, performance of clock skew metrics is evaluated against a collection of supporting metrics including the proposed Bitcoin methods.

The proposed metrics are found to improve classification rates significantly, with good accuracy. Existing methods show non-convincing performance over a pre-filtered Bitcoin dataset. The attack is ultimately deemed theoretically possible with various limitations, but practically infeasible.

Contents

Kurzfassung xi

Abstract xiii

Contents xv

1 Introduction 1

2 Background 3
- 2.1 Clock Skew Measurements . . . 3
- 2.2 Clock Skew for IP Siblings . . . 4
- 2.3 The Bitcoin Cryptocurrency . . . 5
- 2.4 The Bitcoin Protocol . . . 6
- 2.5 Attacks on Bitcoin . . . 8

3 Methodology 15
- 3.1 Measurement Platform . . . 15
- 3.2 Harvesting . . . 17
- 3.3 Evaluation Process – Clock Skew . . . 22
- 3.4 Validation of Sibling Classification . . . 27
- 3.5 Further Possible Verifying Metrics . . . 32
- 3.6 Expected Challenges . . . 36

4 Evaluation 39
- 4.1 Performance of Starke's Algorithm . . . 40
- 4.2 Performance of Bitcoin Metrics . . . 42
- 4.3 Detailed Analysis of a Single Measurement . . . 44
- 4.4 Parameter Choices . . . 48

5 Conclusion 53
- 5.1 Interpretation . . . 53
- 5.2 Future Work . . . 55

List of Figures 57

Bibliography **59**

Appendices **63**
Appendix A. DNSSEC Status of Bitcoin DNS Seeds 65
Appendix B. Python Script for `ADDR` mismatch 66
Appendix C. Negative Impact on Fingerprinting of Address Caching 68

CHAPTER 1

Introduction

Since its initial standardisation in 1998, the adaption of IPv6 has been slowly going forward.[1] It has long been known that the amount of IPv4 addresses cannot sustain the growth of the internet, and adaption is expected to increase. The new protocol itself, and its deployment alongside the existing IPv4 infrastructure leads to new security challenges. [CLAB]

This includes the Bitcoin cryptocurrency which supports both protocols. As a Peer-to-Peer technology, Bitcoin relies on users being able to communicate with other nodes to verify transactions. Especially to establish a shared view of the blockchain, it relies on every node being able to receive updates from the network.

If this is not the case, various attacks are possible including parties spending their funds twice, at least until the split is resolved. However, a single node relaying messages between partitions is sufficient to overcome this situation. By design, once a peer becomes aware of such a *double-spend* transaction, one of the two is rejected, following a standard algorithm.

Currently, the Bitcoin network is composed of both IPv4 and IPv6 peers.[2] These two protocols are technically independent of each other, however some peers – *Dual-Stack nodes* – support both and relay messages for nodes which only support one of them.

In this work, we consider a possible network partitioning attack that takes advantage of this natural boundary. One of the main goals is to investigate whether this advantage is sufficient to make such an attack easier in comparison to existing methods. Communication crossing the natural protocol boundary is enabled by Dual-Stack nodes and consequentially inhibited by removal of all such nodes, e.g., by Denial-of-Service attacks.

[1]The initial specification for the IPv6 protocol was RFC 2460, officially a Draft Standard, which was replaced by the Internet Standard https://tools.ietf.org/html/rfc8200 as recently as 2017.

[2]Additionally, peers may connect via the Tor network, which is not directly considered in this work.

For detection of such Dual-Stack peers, existing methods for sibling detection, with a focus on *clock skew* are employed.

Clock skew describes the fact that clocks used in modern computers have slightly incorrect frequencies due to inaccuracies in manufacturing. This causes their notion of time to deviate from the correct value over time. Research has shown that – even in presence of synchronisation with true time – clock skew is measurable for a remote adversary. From repeated measurements, a single target device can be identified out of a small candidate group with reasonable accuracy. Candidates may be multiple addresses thought to be assigned to the same device, which in confirmed state are referred to as *IP siblings*.

Similar work has been done for general-purpose servers [BB] [SGRC] and for routers [Sta], often producing general-purpose algorithms, which are re-used in this work. In particular, the proposed measurement platform is based on a previous implementation created by Marco Starke in 2019 [Sta].

In particular, these questions are of interest:

1. Are existing techniques of IP sibling detection using clock skew measurements suitable for the Bitcoin network?
2. Can such techniques be augmented by utilising properties specific to the Bitcoin protocol or network?
3. Is it feasible to run this detection in a non-disruptive way, ideally behaving like a legitimate Bitcoin client?
4. What portion of the Bitcoin network is currently operated by Dual-Stack nodes, i.e. nodes that are connected via IPv4 and IPv6?
5. Is it feasible to split the Bitcoin network into an IPv4- and IPv6-only part by attacking these Dual-Stack nodes? Could such a situation be abused?
6. How long does it take to detect a newly-joined IP sibling with this method?

In order to answer these questions, this work is structured as follows. The theoretical background of involved technologies and techniques introduced in existing work are explored in chapter 2. Building on this foundation, chapter 3 proposes methods for data collection, whose results are obtained and analysed in chapter 4. Finally, chapter 5 draws conclusions with regard to the questions and proposes possibilities for future work.

CHAPTER 2

Background

As a foundation for our work, the theoretical background of different fields is explored in this chapter. It starts with an explanation of clock skew and shows how clock skew is utilised for IP siblings. For application-specific methods, it also describes the functionality of Bitcoin and relevant parts of its network protocol. Further, a range of existing attacks on the cryptocurrency is described.

2.1 Clock Skew Measurements

Most modern work on clock skew measurement operates in an adversarial setting, ultimately based on [KBC]. In contrast, earlier work [Pax], assumes control of both parties. Some necessary clock-skew-related terms are borrowed from [KBC]:

resolution r.	Smallest increment unit of a clock.
frequency f.	Number of increments per time unit (inverse of resolution).
reported time $R(t)$.	Clock value at some point in true time.
offset $\mathit{off}(t)$.	Difference between value and true time.
skew $s(t)$.	Derivative of the offset, i.e. the rate at which the inaccuracy changes. This is usually reported in parts per million (ppm), equivalent to $\mu s/s$.

Note that, despite these definitions referring to a true measure of time, the reference clock is ultimately arbitrary. In this work, measurements are considered relative to the measurement node's clock. Earlier work usually assumes the skew to be constant, but more recent work lifts this assumption [SGRC].

It may seem intuitive that regular adjustment of the target's clock via NTP would be able to obscure clock skew. However, this protocol is built for large-scale accuracy, while time scales required in networking are usually small. Hence, timestamps provided for operation of networking protocols are often sufficiently accurate for clock skew measurements even in presence of NTP synchronisation. [KBC] [PF] Further, as demonstrated in [ZM], it is even possible to measure clock skew by means of second-resolution timestamps when synchronising probes to the target's clock frequency.

For the general case, the first widely-known algorithm to determine clock skew was proposed by Paxson in 1998 [Pax]. He demonstrates that skew trends can be observed by plotting the one-trip times for packets from both sides. For automated analysis, he proposes a statistical test based on cumulative minima. Directly applied, this leads to many false positives, so in addition to the algorithm itself, a complex set of heuristics is proposed.

However, Paxson's algorithm requires control of both peers. This restriction is lifted by Kohno et al. in 2005 [KBC], who show applicability of an algorithm taken from Moon et al. [MST] in two experiments, which operates on timestamps from a single side of the connection. It is based on Linear Programming, finding the line such that the sum of distances between it and the observed offsets is minimal. The slope of this line then approximates the linear clock skew. They further show that remote clock skew is consistent even when changing location and network conditions.

In an adversarial setting such as theirs, usually, transport protocols are exploited for the timestamps. A popular choice here is TCP, partly because of its wide usage and applicability behind firewalls. The TCP timestamp extension as defined in RFC 7323 [BBJS] is utilised. This option is usually enabled for protocol performance. The provided timestamps may be (and usually are) rather granular, with the RFC mandating frequencies between 1 ms and 1 second. Some other protocols that have been explored are ICMP (often blocked by firewalls, affected by NTP) and HTTP (granular only to the second) [KBC] [ZM].

2.2 Clock Skew for IP Siblings

Beside a multitude of other applications, utilisation of clock skew to detect IP siblings is the subject of various existing research, started in 2015 by Beverly and Berger [BB]. The focus of this work lies on determining if the infrastructures serving IPv4 and IPv6 are different. This is investigated using active TCP measurements. Before applying clock skew, data is pre-filtered by comparing TCP options.

For the actual analysis, the algorithm from [MST] is used with thresholds inferred from a small ground truth of 61 hosts. Additionally, Beverly and Berger propose a new pairwise point distance algorithm. This finds the closest IPv6 packet for each IPv4 packet, measured by arrival time, and then takes the median of all these pairs' TCP timestamp differences.

A different approach is taken by Scheitle et al. in their 2017 paper [SGRC]. They operate on a large scale with a focus on servers. For test data, address pair candidates are learned from DNS, with a proposed algorithm being validated against two ground truth sets of known siblings mainly from the RIPE Atlas and NLNOG RING projects, with a total size of 682 nodes – significantly larger than previous work.

For actual evaluation, various features are used to train a machine-learning algorithm on this ground truth data. These include TCP options (order, presence), clock-frequency- and skew-related metrics, dynamic range, and, notably, a spline approximation of variable clock skew, whereas previous research is limited to linear skew. Sanity checks used for pre-filtering here include comparison of clock frequencies.

For the decision itself, Scheitle et al. propose two decision-tree algorithms, one hand-tuned and one machine-learned. The former respects a multitude of different verifying and falsifying features, taking into account non-linear skew. The latter, on the other hand, solely depends on a single feature, Δ_{tcpraw}, which is the difference of the initial TCP Timestamp values. This algorithm is recommended for simplicity, with both algorithms being expected to generalise well.

Expanding upon these results, in his 2019 Master's thesis [Sta], Starke analyses IP siblings in the context of network routers. Routing devices are discovered by tracing routes to known servers and scanned for open ports to obtain TCP timestamp information. To accomplish that, a ground truth of 851 servers based upon that of Scheitle et al., along with their network paths, is utilised. Final targets are constructed from the Alexa and Cisco Umbrella Internet top lists, combining IPv4 and IPv6 hosts assigned to the same DNS domain.

Existing measurement techniques are augmented by considering TCP timestamp randomisation, SYN cookies, and short run times. For the algorithm itself, the machine learning technique from Scheitle et al. is expanded and found to use more features than the original work. Further, only features that represent differences of the IP versions are considered, to prevent decisions based solely on a single node of the pair. Finally, good results are obtained also with few timestamps and low run times.

2.3 The Bitcoin Cryptocurrency

Bitcoin is a virtual currency based on a cryptographic model and a Peer-to-Peer network proposed by [Nak]. The foundation is the concept of an *electronic coin*. This is a chain of digital signatures, where each signature transfers currency to the next owner by signing their public key and the previous transaction in the chain. Validity may be verified by any third party via each transaction's signature, signed with the public key designated in the previous one.

The issue pointed out by [Nak] with this model is that the recipient is unable to check for double-spending of the coin without knowledge of all transactions. Therefore, it is proposed to publicly announce every transaction. If an electronic coin is spent twice, the earlier transaction is agreed upon as valid. However, to determine the order of transactions, a trusted notion of time is necessary. This might be some agreed-upon timestamp server. Introduction of such party violates the desired principle of an untrusted Peer-to-Peer network.

This goal, however, is still achievable by introduction of a proof-of-work scheme, as consequently used in Bitcoin. Each block gets a random nonce, and hashes are computed (the *work*) until a special schema is matched. The block then can no longer be changed without re-doing the work. Since blocks are chained by inclusion of the previous block's hash, necessary effort for a forgery increases significantly over time. [Nak] proposes that the network exchanges transactions and blocks, the longest known block chain being the one agreed upon. Each node verifies that new blocks follow certain rules, and expresses acceptance by working on, and eventually publishing, a new block based on the accepted transaction.

Honest participation in this proof-of-work scheme is incentivised by permitting the creator to add a special transaction to each block, creating a new electronic coin for them. Eventually however, the system is proposed to be financed solely from transaction fees. Each transaction may define multiple inputs and outputs, allowing electronic coins to be split and parts to be used for purposes such as these fees.

2.4 The Bitcoin Protocol

Another important foundation of this work is the Bitcoin protocol as well as the underlying transport protocol TCP. Since TCP is widely-known, detailed discussion is deferred to existing educational material.

For our purposes, the following information is necessary. TCP is a transport-layer protocol transferring information in packets with ordering guarantees. Connections are set up in a three-way handshake starting in a *SYN* packet. The TCP Timestamps Option permits inclusion of each party's notion of time in packets to improve performance, with timestamps being between millisecond and second granularity. This widely-used extension is the basis of most earlier work measuring clock skew. An additional detail is that TCP packets, particularly SYN, may specify various options, commonly abbreviated *TCPopts*, that control the behaviour of the protocol. A single host will usually send relatively consistent options, with the order and values sometimes being operating-system-specific. [noa] [BBJS]

A relatively recent development in TCP timestamps is the recommendation to start timestamps at a random offset per connection. [BBJS] This behaviour makes them more difficult to analyse. An implementation has been merged into the Linux kernel (and enabled by default) in 2016, so it is reasonable to assume that a large portion of servers active at the time of writing implement this feature.[1] At least on Linux machines,

[1]Initial implementation: `https://github.com/torvalds/linux/commit/95a22caee396cef0bb2ca8fafdd82966a49367bb`
Consistent timestamps per $(host, port)$ tuple: `https://github.com/torvalds/linux/commit/28ee1b746f493b7c62347d714f58fbf4f70df4f0`

timestamps will be consistent per remote boot and (*source address*, *destination address*) tuple. Notably, these addresses are distinct for each sibling IP, so randomised offsets need to be considered when comparing multiple measurements of the same host.[2]

Since it relies on TCP for low-level data transfer, the Bitcoin protocol is specified on a *message* level. As pointed out by previous works such as [BP], no binding specification exists for the Bitcoin protocol. Rather, the Bitcoin developers believe the actual behaviour of the reference client to be the binding documentation. This makes reasoning about the protocol on a theoretical level more difficult.[3]

However, a description of the protocol exists on the official website, providing the foundation for the remainder of this section.[4] Each message has a header, specifying the network used, a command, the size and a checksum of the payload.

The interaction starts by the connecting party sending a `VERSION` message. Relevant information transmitted here is the protocol version number, a bitmask specifying services provided by the sender, the current Unix epoch time, a fixed-form user agent string[5], and the current heigh of the sender's accepted blockchain. In response to this, a `VERACK` and the other party's version message is sent. Once the connecting party confirms with another `VERACK`, the connection is established.

Actual data such as blocks, block headers, and transactions is transmitted mostly via *inventories* announcing their existence and hashes, with the actual payload requested by the recipient in another packet only if they are not yet aware of the information. This avoids sending redundant (already-known) data over the network.

Another important packet pair is `GETADDR` / `ADDR`. These allow to request known addresses from a peer, which will announce a subset of its address database including a timestamp, the service flags, the IP address and the port. Additionally, a peer may advertise addresses unsolicitedly. The reference client maintains addresses in two tables, *New* and *Tried*, with the former storing newly-received peers and the latter storing these where a connection has already succeeded [HKZG].

A more comprehensive discussion of the network protocol may, among others, be found in [PISP].

[2]https://github.com/torvalds/linux/blob/325d0eab4f31c6240b59d5b2b8042c88f59405b5/net/core/secure_seq.c#L69

[3]https://developer.bitcoin.org/reference/intro.html – Introduction to Bitcoin's developer documentation (accessed 2021-01-14) Additionally, some parts of the protocol are standardised via *Bitcoin Improvement Proposals* by the community: https://github.com/bitcoin/bips (accessed 2021-01-14)

[4]https://developer.bitcoin.org/reference/p2p_networking.html (accessed 2021-01-14)

[5]Example for the reference client: `/Satoshi:0.20.1/` – https://github.com/bitcoin/bips/blob/master/bip-0014.mediawiki (accessed 2021-01-14)

2.5 Attacks on Bitcoin

Due to its rising popularity in recent years, numerous attacks targeting the Bitcoin protocol and network have been proposed. The software constantly evolves to prevent known attacks and conceal more information to limit attack surface. As a result, not all of the presented attacks remain feasible in the original way described in their original papers.

Existing literature, referenced below, describes different classes of attacks on the cryptocurrency. For the purpose of this work, we focus on network-level attacks, but other types also exist [SM] [KAC].

These network-based attacks can be roughly classified as follows:

1. Majority attack: Controlling the majority of computation power allows manipulation, i.e., ignoring certain transactions [SM]
2. Deanonymisation attacks on specific clients, correlating multiple identities [Pus] [BP]
3. Eclipse attack, i.e. isolating a single node from the network flow and presenting tampered information to it [HKZG]
4. Partitioning attacks separating whole groups of nodes from the remaining network [NAHa] [AZV] [TCM+]
5. Network structure disclosure [NAHb] [MLP+] [DSBPS+] [Noo]

A high-level survey of security issues with Bitcoin and especially its network protocol is performed by Tapsell et al. [TAM]. They argue that the TCP-based, unencrypted protocol has some inherent suboptimal security properties. Due to missing authentication and encryption, messages may be freely modified (only protected by an unauthenticated checksum), intercepted, replayed and fabricated. As we will see later, this enables many attacks.

Further, they argue that the initial connection process is not actively secured. If a Bitcoin peer does not have knowledge of any reachable peers, hard-coded DNS names are queried for so-called “seed addresses” which are initial peers that provide further address information via the Bitcoin protocol. Control or spoofing of these DNS names allows an attacker to trick honest clients into joining a malicious network. At the time of their research, none of the DNS seeds were using DNSSEC, which has not significantly improved since.[6]

Looking at the protocol itself, they argue that the `GETHEADERS` and `MEMPOOL` messages may be used for Denial-of-Service reflection attacks, due to the fact that they produce

[6]At the time of writing, only a third of mainnet DNS seeds had DNSSEC deployed correctly. See Appendix A for details.

significantly more output than input. They however acknowledge that there are more powerful vectors in other applications that are easier to exploit. As a mitigation, a nonce in the `VERSION/VERACK` handshake is suggested.[7]

In the survey provided by Soni and Mahehwari [SM], an overview of Bitcoin attacks is provided, including some network-level vectors. We refer to that publication for description of non-network-level attacks and the relatively trivial majority attack.

2.5.1 Deanonymisation Attacks

In his 2015 paper, Pustogarov [Pus] proposes an attack relying on monitoring of clients' entry nodes, i.e. the peers that a victim is directly announcing transactions to. These entry nodes are detected by analysing over the entire network which nodes advertise transactions of interest first. The first few of these are expected to include the victim's entry nodes. If the victim keeps this same set and announces another transaction, an attacker may be able to correlate these as belonging to the same identity. This attack requires significant effort (connections to all nodes) and does not claim particularly high success rates under realistic conditions.

A different yet more practical deanonymisation attack is proposed by Biryukov and Pustogarov [BP] in the same year. They show a fingerprinting technique dubbed ”address cookies“ which relies on announcing distinct sets of non-genuine peer addresses to a target via `ADDR` messages. These “cookies” are later retrieved using a series of `GETADDR` requests. They show eight subsequent requests to be sufficient in practice. They also provide a patch to ignore `GETADDR` from outgoing connections, preventing their attack for clients.[8], which has since been integrated into Bitcoin Core.

This attack is further complicated by a June 2020 patch[9], which caches `GETADDR` responses per *listening* address. That particularly means that (a) it is no longer easily possible to scrape the list of peers known to a node and (b) provided addresses will be different and hard to relate over multiple address families (e.g. IPv4, IPv6, Tor), effectively preventing the attack in the other direction (which our work is interested in) as well. Luckily for this work, the change has not been released at the time of writing. The latest version as of 2021-01-14 is 0.20.1, which does not include the patch. Thus, attacks of this fashion are still feasible until the release of the next version.

Another interesting idea proposed by [BP] is that, exploiting Bitcoin's own DoS prevention, a Tor exit node can be banned from making connections to a Bitcoin peer. This works by using the Tor exit node to connect to Bitcoin and sending malformed messages, which causes the IP to be banned from that peer for some time. Repeating this process, all Tor exit nodes can be banned. An attacker providing their own exit node now has control

[7]This should not be confused with the nonce already present in the `VERSION` message, which is used to prevent connections to self. This provides no inherent security properties, due to not being reflected in `VERACK`.

[8]https://github.com/bitcoin/bitcoin/pull/5442

[9]https://github.com/bitcoin/bitcoin/pull/18991

over all Tor $\rightarrow$ Bitcoin traffic. This excludes Bitcoin nodes operating as Tor hidden services, but a method to circumvent that is shown as well.

2.5.2 Eclipse Attack

Security of the Bitcoin protocol as defined by Nakomoto relies on perfect information, as Heilman et al. recognise in their 2015 paper [HKZG]. They propose an attack that inhibits this by occupying all peers of a victim. An adversary that achieves this goal is subsequently able to filter the target's view of the blockchain by omission of undesired messages. Another possibility is to force the victim to work on an attacker-controlled version of the chain, helping to mount other attacks.

Feasibility of such attack is illustrated by two adversary profiles presented by [HKZG]: One controlling an Autonomous System (AS) used to route traffic on the public internet (e.g. 32 IPv4 /24 prefixes) and a botnet with 4600 nodes – these both achieve success rates greater than 80%.

The attack itself works by filling the *Tried* address table of the victim with adversarial addresses and the *New* table with garbage. Now, by waiting or repeatedly forcing the victim to restart (multiple packet of death vulnerabilities existed at the time), all outgoing connections are occupied and the goal is achieved. According to [TCM$^+$], the presented form of the attack is effectively prevented in recent versions of Bitcoin Core.

2.5.3 Partitioning Attacks

A related but different class of attacks are partitioning attacks. Here, the adversary aims to split a set of bitcoin nodes into separate partitions. As noted by Neudecker et al. in 2015 [NAHa], this is not as powerful as the Eclipse Attack, since there are still honest clients communicating with each other that need to be collectively misled, as opposed to a single node.

The partitioning attack proposed by [NAHa] assumes a botnet as adversary. This network connects to Bitcoin with many peers that only announce attacker addresses. It then analyses the network topology to obtain a minimum vertex cut and performs a Distributed Denial of Service (DDoS) attack on these nodes, causing the connectivity between partitions to be lost – This is very similar to the attack that our work targets. The authors acknowledge that only a very large botnet or nation-state actor could realistically mount such an attack, requiring simultaneous DDoS against $\approx$1500 nodes at the time.

Partitioning attacks, however, can also be performed without Denial of Service, as Apostalaki et al. show in their 2017 work [AZV]. They take the role of an AS operating on the routing layer. It is noted that a full partitioning attack is hard to achieve since mining pools are sometimes also connected via private networks (e.g. via the Stratum protocol), and these nodes are non-trivial to isolate. Also, partitions usually resolve quickly, with full connectivity being restored after just a few hours. Hence, they propose a related class of attacks – delay attacks – which focus not on a complete partition, but sparsening and delay of information propagation. An AS may perform such an attack by having connections routed to them and, for example, dropping transaction announcements, which were waited on for 20 minutes at the time. A caveat noted by the authors is that, effectively, this attack requires cooperation between multiple ASes.

This is expanded upon by Tran et al. [TCM+] in 2020. They note that the route manipulation used by Apostalaki et al. is easy to detect and multiple parties actively monitor Internet routes for such behaviour. The proposed *Erebus* attack, however, is not trivially detect- or preventable, but takes 5-6 weeks to execute.

The base principle of this attack is that an AS may effectively forge (unencrypted) traffic as well as intercept responses for any IP address whose routing path passes it, being able to simulate a Bitcoin server running at such *shadow IP*. This cannot be trivially detected by either the legitimate owner of the address or the connecting party.

Due to the fact that a single AS may forward traffic from a multitude of different prefixes, a multitude of different shadow IPs will be available, which the victim will not recognise as related. It only considers the AS that the actual target IP resides in for grouping. An adversary still needs to fill the *New* table with its 65 335 slots that are only replaced if the previous address is marked as "terrible". Addresses are moved to the *Tried* table via a "feeler connection", which is started every two minutes. Now, the adversary may force the victim to restart at convenient times to take over its outgoing connections and reach the attack table.

The authors propose four countermeasures, two of which have been integrated into the Bitcoin source code since.[10]

2.5.4 Discovering Network Structure

Another adversarial activity is to discover the actual structure of the Bitcoin Peer-to-Peer network, i.e. which nodes are connected to which, how many connections specific nodes have, and how information is propagated through the connectivity graph. While not in itself an attack, this can be the foundation for further operations that undermine privacy or result in an informational advantage required for other attacks.

One of the earlier but still relevant works in this field was conducted in 2015 by Miller et al. [MLP+]. Their aim is to discover influential nodes, which they accomplish by observing timestamps on `ADDR` responses, and especially an implementation detail that leaked whether a node was currently connected based on its timestamp. This leak has since been fixed [MCR+].

However, Miller et al. introduce further techniques, including the idea to distinguish nodes by their mempool, i.e. open transactions, which may not conflict. By sending

[10] As outlined on the paper's website: https://erebus-attack.comp.nus.edu.sg/ (accessed 2021-01-14)

mutually-conflicting transactions to every single node, sets may be coloured and single nodes identified. This however does not work in practice, so an assisting method is proposed.

“InvBlocking” utilises the fact that, once a transaction is announced via `TX` and a corresponding `GETDATA` is sent, the peer waits for some time before requesting this transaction from another node. With this, there is enough time to deliver the corresponding conflicting transactions to each node individually. We note that this may also be used to identify a single node over multiple sibling IPs, announcing a specific fake transaction only to it.

A similar approach, based on orphan transactions, is proposed by Delgado-Segura et al. in 2018 [DSBPS+] and named “TxProbe”. Bitcoin separately stores orphan transactions, i.e. where at least on of the parents is missing, similar to the mempool. The identifying behaviour here is that orphan transactions, while not reflected in `MEMPOOL` responses, are not requested via `GETDATA` once an `INV` is received for their hash.

The crucial advantage of using orphan transactions in contrast to the method from Miller et al. is that these are not propagated to peers, due to the fact that they cannot be validated. A mechanism is proposed that clears out the orphan transaction set in preparation for TxProbe.

The actual method works as follows: For every node, a “marker” and “parent” transaction is created. The parent must remain with the node itself, as it locally validates the marker, which peers should see as orphan. Thus, nodes are invblocked on all others’ parents. Finally, all markers are sent to each node. From the resulting `GETDATA` request, we infer which markers the peer already knew. This directly corresponds to the set of nodes from which it got markers, i.e. it has a direct connection to.

The authors estimate this method to take around eight hours on the Bitcoin main network. They however acknowledge that cleaning out the orphan transaction set, which is necessary for multiple iterations of TxProbe, is a destructive action on real-world data. As such, it is not recommended to perform this attack on the main network. Also, it is noted that for each invocation, three or four non-conflicting transactions will eventually be accepted, resulting in transaction fees.

Multiple existing techniques in different fields are compared by Noort in his 2016 thesis [Noo]. He highlights issues in relation to IP siblings, which might cause wrong conclusions with existing methods. For detection of IP siblings – relevant for our work – he proposes a novel mechanism. Bitcoin nodes randomly decide for each *transaction* ($\neq$ peer-transaction tuple) whether to immediately send it (broadcast) or to apply a random delay (trickle).

Now, we can remember for each transaction which behaviour was chosen. This will be consistent for a sibling’s IPs, especially if repeated for multiple transactions. The author observes that after only ten iterations, the false positive rate will be as low as 1%, resulting in a resource-friendly yet high-accuracy method for sibling detection.

Finally, a slightly different approach is proposed by Neudecker et al. in 2016 [NAHb]. As a Global Passive Adversary, their method analyses propagation behaviour of transactions over the whole network. In particular, it observes timing delay in network messages and uses a simulation model to conclude whether two peers are connected.

CHAPTER 3

Methodology

In this section, a methodology is proposed to measure clock skew and various auxiliary properties from all Bitcoin nodes over a longer period, allowing a sibling decision to be made for each possible pair of nodes. An existing measurement platform provided by Starke [Sta] is modified to fit the requirements of this work.

The process starts with obtaining a list of currently active public (termed *reachable* in various previous works) Bitcoin nodes using the API provided by `Bitnodes.io`. This service was also used to conduct most recent works, see Chapter 2 (e.g. [TCM$^+$] [HKZG]). (PREPARATION)

Next, for each node, the remote's TCP clock value is periodically obtained over a specified measurement period using the TCP Timestamps Extension [BBJS] alongside Bitcoin node metadata provided by its application-layer protocol. (HARVESTING)

In a final step, various metrics proposed in previous work as well as novel Bitcoin-specific values are calculated and sibling decisions made using different algorithms. Before computing these metrics, nodes are pre-filtered based on metadata from the `VERSION` message to reduce the amount of candidate pairs. Finally, metrics for groups of algorithms are computed in order to validate their accuracy as applied to Bitcoin nodes. (EVALUATION)

The modified platform is published under the GNU General Public License version 3 to match the licensing requirements set forth by the earlier work it is based upon. Its source code can be found at `https://github.com/literalplus/ipsiblings`.

3.1 Measurement Platform

The original Python platform proposed by Starke was optimised for evaluation of the routing path to servers and hence required modification. Further, extensibility was limited due to structural choices made in the initial implementation, possibly due to time constraints. Such reason is hinted for example by To-Do comments indicating that the implemented evaluation solution is not ideal and might be improved in further work.

Further, the existing platform has two evaluation implementations, for full (ten hours) and low (significantly shorter) runtime, respectively, and two ways of obtaining a candidate pair. A multitude of features such as tracerouting, TCP timestamp harvesting, a CDN filter and automatic configuration of operating system settings are provided.

However, the code is only split into relatively few modules, each comprising only a single file with multiple thousands of lines. The structure is further complicated by opaque dependencies between modules, weak encapsulation making it difficult to change fundamental implementation choices such as the ways of obtaining targets without touching dependent code. In particular, a single yet not generalised data flow is used for both possibilities, managed by the 930-line main module conditionally executing similar code paths.

Thus, it is necessary to adapt the platform to this work's requirements and provide the extensibility and encapsulation necessary to generalise for further work without major structural changes. As practical part of this work, the platform proposed by Starke [Sta] is adapted to match this architecture.

On a high level, execution is split into four parts, each having its own module:

1. Preparation – Obtain target nodes from a provider (Bitnodes.io, existing list on filesystem)
2. Harvesting – Periodically call all selected providers (TCP Timestamps, Bitcoin Protocol) until the requested time has passed, saving results in provider-specific formats
3. Evaluation
    a) *Batching* – Split candidates into consistent batches to enable multi-process parallel evaluation and support memory-constrained environments
    b) *Filtering* – Use Bitcoin protocol metadata to remove candidate pairs that cannot be siblings due differing Bitcoin protocol versions
    c) *Properties* – Compute different metrics based on data from the Harvesting stage, with dependency management between properties and caching of results
    d) *Evaluators* – Decide sibling status of a candidate based on properties
    e) *Candidates* – Export metrics provided by properties and evaluator status for each candidate, enabling detailed analysis if necessary
    f) *Stats* – Aggregate candidate results per batch and provide statistics to enable algorithm evaluation and cross-validation. For non-parallel batching, aggregate statistics from all batches.

Further, there are several cross-cutting concerns:

1. Configuration – Control of parameters via an abstract interface, currently sourced from command-line arguments, but decoupled from concrete source
2. Bootstrap & Wiring – Manage data flow between stages and initialise dependencies
3. OS Settings – Optimise operating system properties for measurement
4. Model – Shared data model
5. Logging – Observability

A visual representation of the platform's structure may be observed in Figure 3.1. The concrete implementation makes this architecture extensible by calling pluggable providers via standardised interfaces for each stage. Further providers may be added without modifying callers, for example to measure another application or to evaluate via a novel statistical metric.

During the first phase, Preparation, targets are obtained. This works via the API provided by `Bitnodes.io`. This service continuously captures snapshots of all reachable Bitcoin nodes and stores some interesting information such as domain and the Autonomous System number (used to identify which organisation owns the related IP address). From the latest snapshot, IP address and port are captured. Due to the focus of this work on IPv4 and IPv6 nodes, `.onion` addresses are discarded, as they are only reachable via the Tor network.

This information is stored to disk for later retrieval and passed on to the next stage. Alternatively, previously-obtained Bitnodes targets may be loaded from such a file in this stage.

The remaining phases are described in the following sections of this chapter.

3.2 Harvesting

The Harvesting stage is characterised by its two providers: Bitcoin and TCP Timestamps. These are repeatedly called in their respective configured intervals until the measurement duration has elapsed. After a grace period to enable open connections to finish, results are collected and forwarded to the next stage.

What is notable about this stage is that it makes heavy use of multi-processing and inter-process communication tools such as queues provided by the `multiprocessing` Python module. Future provider implementations may expand on this by orchestrating multiple measurement nodes as seen in [Noo].

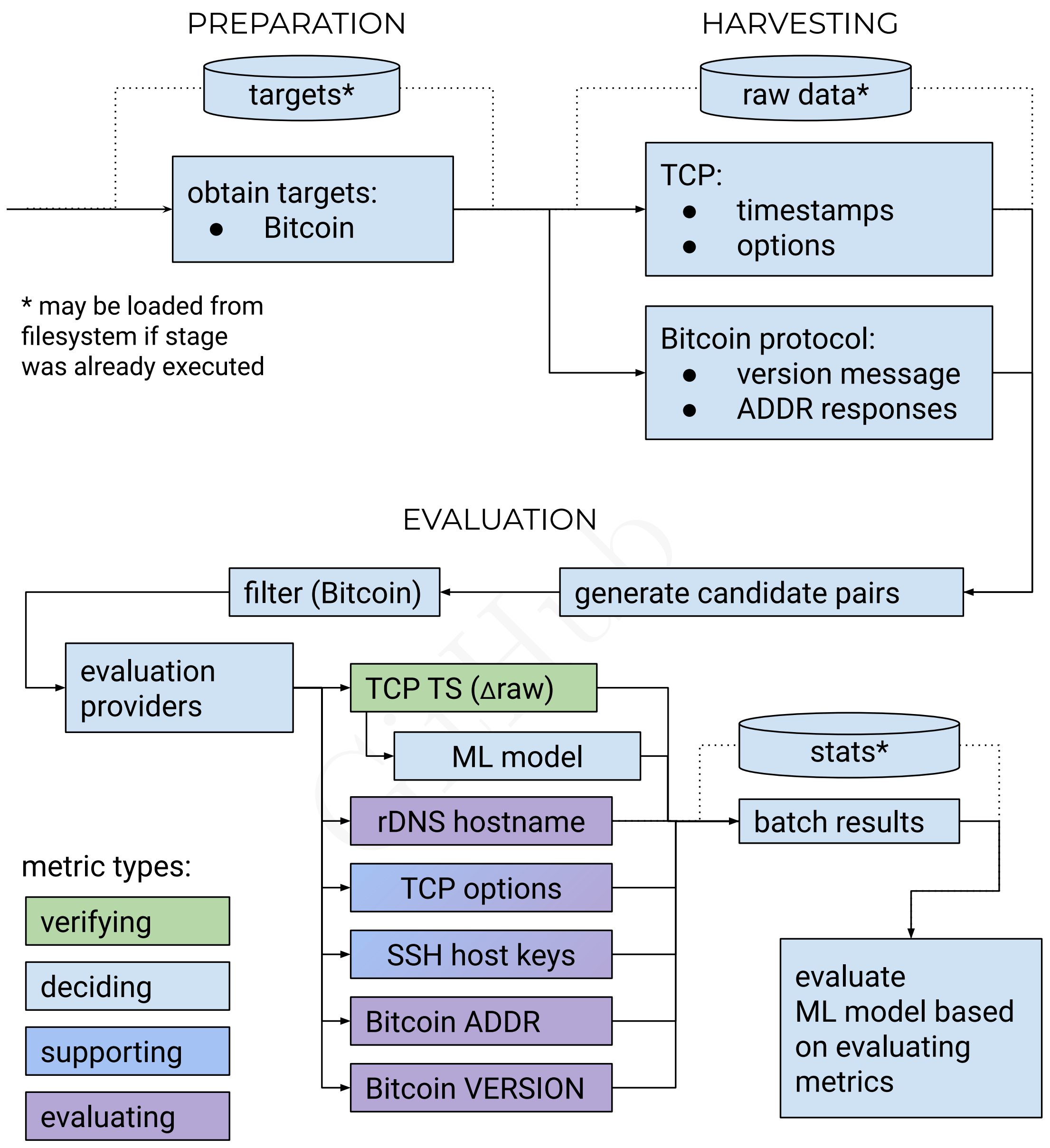

Figure 3.1: Diagram modelling data and execution flow, and the evaluation process.

3.2.1 Clock Skew Measurement

For measurement of clock skew, the existing implementation provided by Starke [Sta] is mostly re-used and adapted to fit the modified execution model. This implementation uses the *Scapy* Python library[1] to send and receive Layer 2 packets. This allows it to read and modify the TCP header and hence collect timestamp information via the TCP Timestamps option, which is described in section 2.4.

Specifically, raw TCP SYN packets are sent to the target host and port as obtained by Preparation, with the TCP Timestamps option set, indicating to the remote that we want to receive timestamps. Since even the initial SYN response already should contain the TCP Timestamps option [BBJS], we do not need to complete the full TCP Handshake, making this a performant single-shot technique.

As mentioned by [Sta], this may be adversely affected by a security mechanism known as *SYN Cookies*. That is, to save memory, connection metadata of not-yet-accepted connections still in the handshake protocol may be represented in various TCP header fields. However, this is usually only used in low-resource environments.

If SYN Cookies are employed, the TCP Timestamp option and especially the TSval field may be used to store data as well, providing invalid timestamps. That would break the measurement. A solution might be to fully establish a connection and send measurement packets after the handshake. That however consumes additional resources and requires a rudimentary implementation of the application-layer protocol. Due to the supposed rarity of such configurations, handling of this case is omitted from this work, as in the original implementation.

The fixed destination ports are blocked using a firewall, preventing the operating system's TCP stack from processing measurement replies. Packets blocked in this manner are still observable over a raw socket, as provided by Scapy. If not blocked, the TCP Stack would recognise the SYN response not belonging to a tracked TCP connection and react accordingly. This is done in the OS Settings using `iptables`. [Sta]

Another issue predicted by [Sta] is remote nodes adding a random offset to TCP Timestamps, since they only need to be consistent relative to each other. The author also proposes a solution via their machine learning model. This setup is replicated by falling back to the machine learning model if the remote node appears to be employing random timestamp offsets.

As outlined in section 2.4, the Linux kernel now uses random offsets by default, however retaining a consistent offset per $(address, port)$ tuple. This allows the timestamp series to be consistent for a single IP version. This does not apply across IP versions as the addresses differ.

Each received packet is associated with its respective target, storing the provided remote timestamp and local reception time. Further, the first received set of TCP options is

[1] `https://github.com/secdev/scapy`

saved for later use as a supporting evaluation feature. The concrete execution flow for the TCP Timestamp harvest provider is shown in Figure 3.2. This shows the dispatcher in the Main Thread, the harvester object, and the harvest-related threads. Packets are periodically sent by the Harvester Thread, and responses received by the Receiver Thread. Results are passed to the Main Thread via a queue, which is periodically processed, in a round-robin fashion over all harvesters.

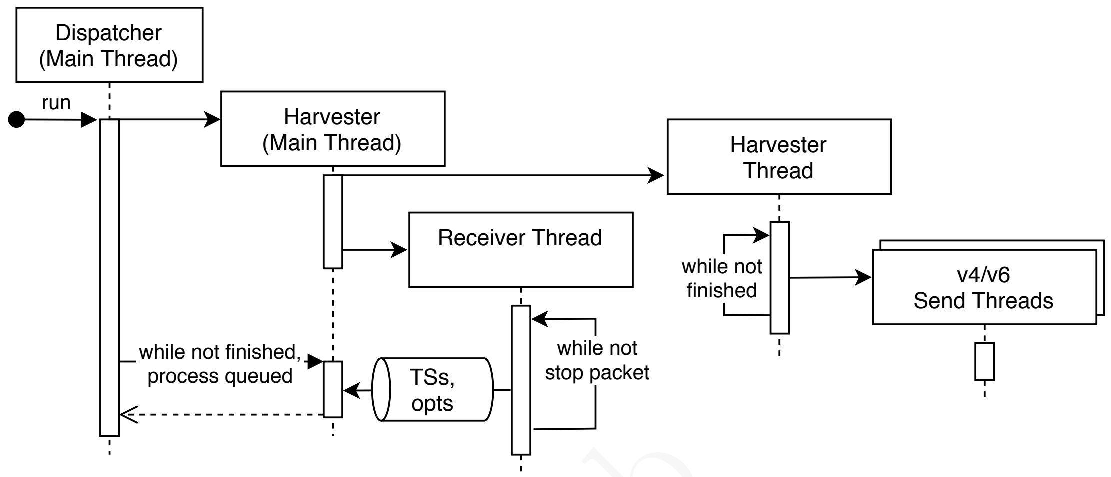

Figure 3.2: Diagram modelling execution and data flow for the TCP Timestamp harvest provider. Notation is loosely based on that of a UML Sequence Diagram.

3.2.2 Bitcoin Protocol

The second harvester provided in the platform is for the Bitcoin application-layer protocol. This provider is original work independent of [Sta]'s implementation. Again, target data obtained in the Preparation stage is utilised.

It periodically connects to all targets collecting a `VERSION` packet and requesting an `ADDR` packet. Protocol parsing is done via the *Python-Bitcoinlib*[2] Python library, albeit the deserialisation flow is expanded to accommodate non-blocking sockets, which are used to allow multiple concurrent connections without maintaining a separate thread for each one.

This provider utilises the same basic data flow as the TCP Timestamp provider, however extended for stateful connection handling. This flow is shown in Figure 3.3.

As visible in the diagram, for each run, a queue of connection targets is filled. The ConnectionHandler Threads, in each execution, open a new connection up until a limit of 101 simultaneous connections per thread. The reason for this limit is to prevent resource exhaustion in the measurement node, especially considering possible starvation of other harvest providers.

[2]https://github.com/petertodd/python-bitcoinlib

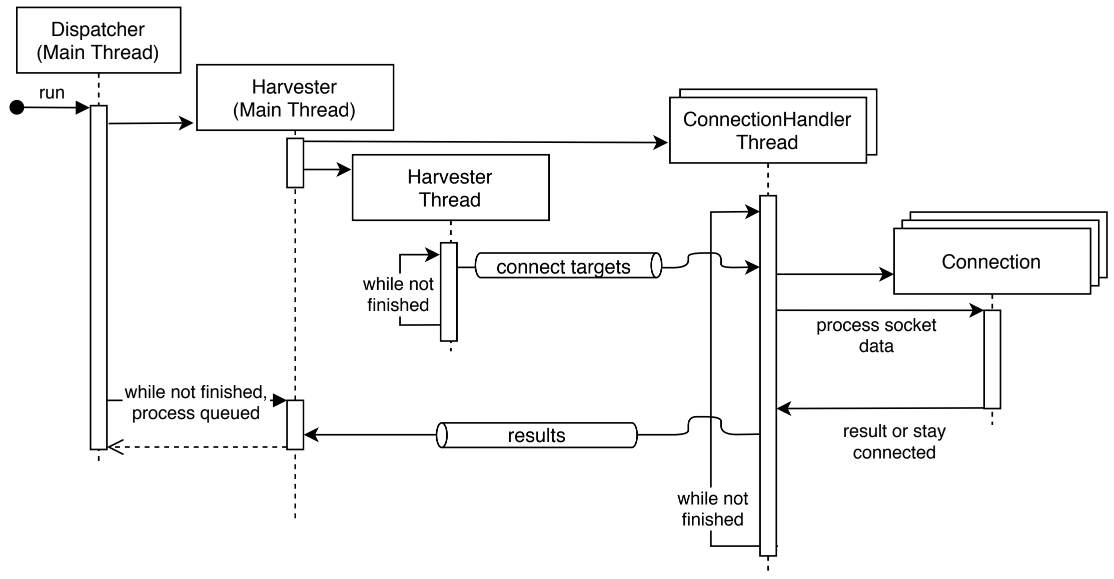

Figure 3.3: Diagram modelling execution and data flow for the Bitcoin harvest provider. Notation is loosely based on that of a UML Sequence Diagram.

After connection creation, a ConnectionHandler thread processes any readable sockets using a `select` syscall with a 100ms timeout. Parallelism is accommodated by utilisation of non-blocking sockets. Any connections with data waiting are notified, with the Connection class handling the actual protocol. If all necessary data for a connection has been received, it is closed and the collected data passed on to the result queue. Otherwise, the connection is retained in anticipation of further information.

Finally, not shown in the diagram for simplicity, any connection that has been open without capturing a "useful" packet (as marked in the list below) for some time is closed. During experimentation, a timeout value of 150 seconds has been deemed sufficient. Most manually-tested peers respond to a `GETADDR` request within 30 seconds, with some occasionally taking up to 100 seconds. It is expected that most peers behave according to the protocol and provide a timely response. Thus, a higher value is chosen, minimising the number of missed responses.

Packet handling on a protocol level works as follows:

- Upon connection, a `VERSION` packet is sent.
- A `VERSION` packet ("useful") is recorded and responded to with a `VERACK` packet.
- `PING` is responded to with a `PONG`.
- `VERACK` ("useful") is recorded.

- ADDR with more than ten addresses is recorded and the connection closed. All necessary data has been collected.
- ADDR packets with less than ten addresses (“useful”) are usually either the peer advertising itself or forwarded advertisements from other nodes, which are of no value to us.
- The common messages GETHEADERS, INV, ALERT, and GETBLOCKS are ignored.
- Other packets are logged.
- Finally, if more than six packets have been received in total, a GETADDR is sent. Technically, this might be done directly after reception of VERACK, but experiments in practice have shown that some peers do not immediately respond to such requests. The Bitcoin Core source code[3] at the time of writing however does not provide an obvious explanation for such behaviour, and this might be a misconception.

It is important to note that peers will routinely forward and send information, so the threshold of six packets is usually reached within a few seconds. Further, even leaving aside resource constraints, connections cannot be left open. This is due to (a) the volume of data received producing unnecessary load and (b) Bitcoin Core[3] only accepting one GETADDR request per connection.

Due to the connection limit and occasionally long response times from peers, this provider takes around 30 minutes to scrape all known nodes on our measurement node. Hence, the interval for this must be chosen higher than that to prevent congestion. This might be improved upon by increasing the connection limit, heuristics for when not to expect a response, or better hardware. Additionally, due to each node sending up to (and usually exactly) 1 000 addresses, relatively to the TCP Timestamp provider, a lot of data is generated, reaching about 1.6 GB for a ten-hour measurement run with 40 minute interval.

3.3 Evaluation Process – Clock Skew

The final stage is Evaluation. Here, the results from Harvesting are combined into sibling candidate pairs, filtered, evaluated using various evaluation providers, and finally statistics about each evaluator’s decisions are recorded. This process allows us to evaluate the applicability of Starke’s [Sta] clock skew sibling detection algorithm by correlation against other algorithms that are known to produce accurate results.

First, sibling candidate pairs are generated by combining each IPv4 target which each possible IPv6 sibling. The amount of candidates, which is in the millions in our measurements, is reduced by filtering based on the Bitcoin application-layer data collected

[3] https://github.com/bitcoin/bitcoin/blob/1dfe19e2840b16c014674a890ba58a43fe039687/src/net_processing.cpp#L3509

during the harvesting stage. In particular, nodes with different Bitcoin protocol versions are assumed not to be siblings.

While this filtering makes sibling evaluation more tractible, it is also acknowledged that not evaluating the filtered candidates might hide good performance of Starke's algorithm on these obvious negatives.

Parallelisation[4] and operation in memory-constrained environments is made possible by dividing the generated candidate pairs into batches deterministically, and processing them separately.

The decision itself is performed by evaluation providers, which may result in one of multiple sibling statuses: Positive (sibling), Negative (non-sibling), Indecisive (cannot decide), Conflict (two providers resulted in positive and negative, respectively), and Error (unable to evaluate due to a invalid, inconsistent, or missing data).

Each evaluation provider may decide based on a variety of *properties*, which are described in the following two sub-sections, along with evaluation providers. Properties may express dependencies upon each other, which are automatically resolved when requesting computation of one, propagating any errors that occur.

3.3.1 Clock Skew Properties

For clock skew, a focus is laid on existing properties already used by Starke [Sta] and Scheitle et al. [SGRC]. Examination of new features exceeds the scope of this work. While based on the code provided by Starke and ultimately Scheitle et al., most properties have been optimised to better utilise vector computation features of the NumPy Python library [HMvdW$^+$].

All properties in this section are taken from [Sta], whose implementations are based on [SGRC]. Because of this, these two works are not individually cited for each property. Detailed discussion of the properties is deferred to the original work.

In the following, data series are referred to as follows:

timestamp series.
A list of TCP Timestamp values (*TSval*) and their respective reception times.

normalised timestamp series.
A timestamp series, with each value and reception time relative to the first observed one, respectively, and TSval overflows normalised.

offset series.
A normalised timestamp series, with TSval converted to seconds using the estimated frequency.

[4]Parallel processing of batches is not fully implemented, due to existing performance being sufficient for this work. It is possible to select the exact range of batches to process in a run, but aggregation of statistics is only supported if running the whole evaluation in one go. Since batches are independent of each other, full parallelisation is possible.

PPD series.
For each IPv4 datapoint of an offset series, the offset difference to the closest IPv6 datapoint in time (*pairwise point distance*), as proposed by [BB]

The properties computed from these series, for clock skew, are as follows:

NormSeriesProperty [BB]

Input: IPv4 and IPv6 timestamp series
Output: Two normalised timestamp series

Detects and removes overflow of the timestamp value field, removes constant offset such that values and reception times are relative to the first received timestamp. Removes the first data point $(0, 0)$.

FrequencyProperty [KBC]

Input: IPv4 and IPv6 normalised timestamp series
Output: *per family:* Frequency (raw and rounded), R^2 value
global: Difference of raw frequencies and R^2 values

Applying linear regression to each normalised timestamp series yields an approximation of its slope, which corresponds to the frequency of the remote clock relative to the frequency of our own clock (here: as TSval change per second).

FirstTimestampDiffProperty

Input: *FrequencyProperty*, raw timestamp series
Output: Difference between the protocol versions' first remote and reception timestamps

This represents how far apart the series start. The difference between the initial remote and reception timestamps between the protocol families is taken. Remote timestamps are converted to seconds using the frequency. Now both in the same unit, remote and reception timestamp differences are subtracted.

OffsetsProperty [BB] [KBC]

Input: IPv4 and IPv6 normalised timestamp series, frequencies
Output: Two offset series

Remote timestamp values are converted to seconds and rounded to six decimal places. Each of these is taken relative to its reception time, converted to milliseconds, and again rounded.

DenoiseProperty

Input: IPv4 and IPv6 offset series
Output: Two denoised offset series

Performs noise removal by aggregating offsets into 120-second buckets. For each bucket, the minimum value is taken, to accommodate the additive nature of network noise described by most earlier works, e.g. [Pax].

MeanOutlierRemovalProperty

Input: IPv4 and IPv6 denoised offset series
Output: Two mean outlier-removed offset series

Performs outlier removal using a symmetric 97% z-confidence interval around the mean offset.

PpdProperty [BB]

Input: Two outlier-removed offset series
Output: A single PPD series, dynamic range of PPDs

Computes the pairwise point-distance as defined above, as well as thresholds for symmetric 95.5% z-confidence intervals around mean and median PPD.

PpdOutlierRemovalProperty

Input: PPD series, IPv4 and IPv6 mean outlier-removed offset series
Output: Two PPD outlier-removed offset series

Performs further outlier removal on the series using the confidence interval around the median computed by *PpdProperty*.

SkewProperty [BB] [KBC]

Input: Two PPD outlier-removed offset series
Output: *per family:* clock skew, R^2
global: difference of skews and R^2 values, angle between skews

Estimates clock skew using robust linear regression. The R^2 value is computed using ordinary linear regression, because robust linear regression does not provide that in the SciPy library that is used.

SplineProperty

Input: Two PPD outlier-removed offset series
Output: Two offset series representing the created splines

Fits a univariate cubic spline to the offset series, using twelve equidistant bins. The first and last eight data points are ignored. From the spline, values are taken in 120-second intervals.

DynamicRangeProperty

Input: Two PPD outlier-removed offset series
Output: *per family:* dynamic range
global: absolute and relative difference of dynamic ranges, average

Computes the dynamic range as defined by the difference of the 97.5th and 2.5th quantiles.

SplineDiffProperty

Input:	IPv4 and IPv6 offset splines from *SplineProperty*
Output:	absolute difference of spline means,
	— scaled by the dynamic range difference,
	85th percentile of difference curve

Moves the spline with higher mean such that the means match. For each point, the difference to the other spline is computed.

3.3.2 Clock Skew Evaluators

For clock skew, two different evaluators are provided, one using the Machine Learning model developed by Starke [Sta], and the other using only *FirstTimestampDiffProperty*, as first proposed by Scheitle et al. [SGRC].

The timestamp difference evaluator (referred to as *tcpraw*) only uses a single feature, the difference between the first packet's reception and (converted) remote timestamp in seconds. The platform supports both the original threshold as well as the modified value suggested by Starke [Sta]. An Error classification is made by this evaluator if *FirstTimestampDiffProperty* is missing, Positive if the difference is below the respective threshold, and Indecisive otherwise. The latter is due to this model being unable to provide a reliable negative classification because timestamps might have a random offset as discussed earlier and outlined by [Sta].

The Machine Learning evaluator uses the same features as proposed by Starke [Sta] and the model provided by the author in the accompanying code repository.[5] This model was created using the XGBoost Python library[6], is also used to read it. The format used in the original work has since been deprecated and replaced with a new format, which the models are converted to using scripts provided by the library.[7]

The properties used as input to the model are:

- *FrequencyProperty* – frequency difference, difference of R^2 values
- *SkewProperty* – difference of estimated skews, difference of R^2 values
- *DynamicRangeProperty* – absolute and relative difference, average dynamic range

Further, as in the original work, this model is only intended to be used if the Starke threshold for *tcpraw* does not yield a positive classification. In evaluation practice, these evaluators are used together. If observed alone, the Machine learning evaluator yields Indecisive if this precondition is not met. Error is returned if any of the three used

[5]For ease of access, we also provide this model in our `ipsiblings/assets` directory. We only use the full-runtime (*FRT*) model.

[6]https://xgboost.readthedocs.io/

[7]https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html#loading-pickled-file-from-different-version-of-xgboost

properties is missing and POSITIVE and NEGATIVE for classification values zero and one of the model, respectively.

3.4 Validation of Sibling Classification

It is further necessary to evaluate the performance of clock skew-based evaluators introduced in the previous section. Therefore, further properties and evaluators are proposed, which are also executed in the EVALUATION stage.

Primarily, the Bitcoin protocol is used for sibling verification. Four evaluators are proposed that use data collected during the HARVESTING stage. Further, methods originally shown by Starke [Sta] as supporting metrics for his algorithm are utilised. Specifically, these are a SSH *key scan* and TCP options fingerprinting.

Traditionally, a ground truth set of known siblings and non-siblings would be used, consistent with existing works. For the Bitcoin Peer-to-Peer network, it appears there is no intuitive way to produce such data set.

It was also considered to use a more general ground truth set for evaluation. However, this might not be directly applicable to the observed data set, since a very specific subset of the public Internet is considered. The author believes that there is no trivial general IP sibling ground truth available that reflects the host demographics of the Bitcoin network.

For any general ground truth data set, a sub-set would need to be carefully selected that resembles the host demographics of Bitcoin. Since this is an error-prone and tedious process, classifications are instead verified using auxiliary measurements on other properties of the target hosts. The obvious drawback of this is that results might not be as decisive, since the validating metrics may produce INDECISIVE.

3.4.1 Bitcoin Address Neighbours

The first proposed validating metric is the address neighbour evaluator. Here, the addresses provided by both peers are checked for overlaps. This intersection is then related with constraints obtained from the Bitcoin Core source code. To collect these "neighbours", the algorithm loops through all IPv4 connections and assigns the temporally closest IPv6 connections (i.e. previous and next). For each assignment, the collected address sets are intersected.

All evaluation constraints relate to the timestamp reported with an address and are based on the most recent version of Bitcoin Core as of January 2021, which is 0.20.1.[8] The `git blame` information[9] reveals that the relevant lines of code have not been semantically

[8] https://github.com/bitcoin/bitcoin/blob/v0.20.1/src/addrman.cpp#L280

[9] https://github.com/bitcoin/bitcoin/commit/5fee401fe14aa6459428a26a82f764db70a6a0b9#diff-49d1faa58beca1ee1509a247e0331bb91f8604e30a483a7b2dea813e6cea02e2R308

altered since at least 2012 and first released in version 0.6.0. As such, it is reasonable to assume that the vast majority of currently-active peers applies these constraints.

Bitcoin considers a known address as *currently online* if its associated timestamp is less than 24 hours in the past. Incoming timestamps advertised by `ADDR` packets are only applied if the existing timestamp is older than what we call the *update window*:

- existing address is *currently online*: Changes are applied if older than **one hour**.
- otherwise: Changes are applied if older than **24 hours**.

This calculation respects the two-hour penalty subtracted from incoming timestamps. That is, to check if an address is active, an accurate algorithm would need to add back these two hours to make a correct decision. The penalty is not applied to peers advertising their own address, which is however hard to detect.

Note: The algorithm used during evaluation in this work fails to apply this, so it does not recognise addresses of age between 24 and 26 hours as active (*false-inactive*). This does not lead to false-negative siblings due to the looser constraint being applied. Effectiveness of this evaluator could be improved by correcting this shortcoming, which was only discovered after analysis had already commenced. Due to low impact, analysis is not repeated with a corrected algorithm.

A correct algorithm finally decides if an address neighbour violates the constraint by evaluating if it was active at collection time:

$T_{c,later}$	Remote timestamp of the later connection, from `VERSION` message
$T_{a,later}$	Address timestamp collected at that time (incl. penalty)
$T_{A,later}$	$T_{a,later} + 2$ hours
$online:$	$T_{c,later} - T_{A,later} < 24$ hours
$\iff$	$T_{c,later} - T_{a,later} < 26$ hours

The intuitive justification for checking relative to the remote timestamp at collection time is as follows: If a new `ADDR` packet is received immediately before collection, the timestamp is overridden or retained depending on if the address was online then.

With this information, the algorithm decides on the correct update window. If the differentce between the earlier and later timestamps that are being compared is less than that, a Negative sibling decision is made. A single Bitcoin peer using the official client would not report timestamps for the same address with difference less than the update window.

We ignore the case that an address was removed from the address table and added again before the next collection. This seems unlikely because Bitcoin has complex logic in place that prevents replacement of "good" addresses. An address needs to fulfill specific criteria to be considered *terrible* and be removed, which is not expected to happen for *currently online* addresses. Additionally, this bad address would need to be added back to the table before the next measurement, which is unlikely with short intervals.

For non-standard clients to be handled correctly, their behaviour would need to be analysed, which is considered out of scope for this work due to their low fraction of total peers. Possible false-negatives produced by non-standard clients might be reduced by only applying these checks if both peers report to be using the official client.

3.4.2 Additional Bitcoin Heuristics

In addition to this consistency check, further falsifying features based on the Bitcoin protocol are suggested.

Similar to address neighbours, it is possible to evaluate the service flags stored with each address. Again, the 0.20.1 reference client introduces a useful constraint: As long as an address remains known, service flags are only added and never removed.[10] The same caveats as above related to an address being removed and re-added between measurements apply.

The resulting evaluator observes all address neighbours and reports a NEGATIVE sibling classification if any violation of the aforementioned constraint is found. Otherwise, the result is INDECISIVE.

As a supporting metric, the rDNS hostname reported by the node is checked, as reported by the `Bitnodes.io` API. If both peers report the same hostname, a POSITIVE decision is inferred. We assume that the only widespread reason to assign two IP addresses to the same hostname is them being for the same physical host. A notable exception to this is load balancing, which is however not expected to be in use for Bitcoin peers.

Finally, the most straightforward Bitcoin evaluator uses information from the `VERSION` message. A sibling candidate is rejected (NEGATIVE) if there is a mismatch of protocol version, user agent string, or service bits. For peers that change this information during measurement, INDECISIVE is inferred.

For changing information, a more complex algorithm comparing temporally related messages similar to the address neighbour assignment might be practicable. This is omitted here for simplicity.

3.4.3 Supporting Metrics [Sta]

THE SSH *key scan* and TCP options fingerprinting metrics introduced by Starke [Sta] are revived as evaluating metrics.

The original work further proposes geographic location as a falsifying feature on a country and continent level, but does not consider it for its own evaluation due to inaccuracies in

[10] `https://github.com/bitcoin/bitcoin/blob/v0.20.1/src/addrman.cpp#L284` – as above, this behaviour been unchanged since Feburary 2012, with the last significant modification being the commit referenced above.

data that is available free of charge. It is argued that due to the low data quality, this feature is not suitable for large-scale applications in practice [Sta]. Because the expected value of this feature is marginal, it is not implemented in the adapted platform.

For TCP options fingerprinting, the first set of TCP options received from a peer is considered. This evaluator considers a sibling candidate as Negative if the amount or order of TCP options differs. Further, for *window scale*, the value is required to match. Missing data yields Error, and the absence of a mismatch results in Indecisive.

This cannot be used as a verifying feature due to the fact that many hosts share TCP option signatures. Rather, it is applicable for coarse-grained operating system detection. [BB] The same matching logic is used by Starke [Sta] and Scheitle et al. [SGRC]. Beverly and Berger [BB] use a similar approach that compares values of all options except for *MSS* and *Timestamp*. It does not immediately become clear whether there is a practical difference in these approaches since we do not have data on which options are commonly used.[11]

The second protocol used to verify sibling decisions is *Secure Shell* (SSH). This protocol secures a plaintext terminal connection to a remote host and is commonly used to administer remote and headless machines. It is assumed that a great portion of Bitcoin peers will run SSH, because a considerable fraction is not hosted at home, but rather at cloud providers, as was found in measurements conducted by Neudecker between 2015 and 2018 [Neu]. In absence of contrary information, the assumption is made that this has not changed, so looking at fingerprintable information leaked by the SSH protocol seems promising for a notable portion of hosts.

In a first step required to collect SSH information, the solution developed by Starke [Sta] is adapted to support input partitioning (the original solution requests all v4/v6 addresses in a single process) and the evaluation property model. Both SSH host keys and the SSH identification string are considered for fingerprinting. This information is collected using the `ssh-keyscan` utility provided by the OpenSSH package[12].

As defined by the SSH protocol architecture RFC [YLa], SSH *host keys* are used to verify the identity of the remote party. These are assumed to either be static and manually verified by the client or cryptographically signed by a Certification Authority (CA) known to the client. In the former case, changing these keys frequently induces significant maintenance overhead, so such behaviour is not expected. This is also the default configuration of the widely-used OpenSSH, so it is reasonable to assume that the majority of internet devices has consistent host keys. In the latter case, it is possible to rotate the keys more frequently, but this is assumed to be an advanced configuration that is not in widespread use in this subset of the public internet.

[11] `https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1` (accessed 2020-11-22) – IANA maintains a list of known TCP option numbers. Most have been obsoleted or discouraged, and for the remainder it does not immediately become clear if they are commonly used, and if so, whether they are usually applied to SYN segments.

[12] `https://www.openssh.com/`

The identification string is also defined in the SSH Transport Layer Protocol RFC [YLb] and sent during the initial connection handshake. A specific format is mandated, which includes protocol and software version, as well as a comments field. The former are intended for compatibility and to indicate capabilities, while usage of the latter is not explicitly limited. In a survey of collected identification strings[13], it is found that, among the collected Bitcoin peers, versions are used as intended and the comments field is sometimes utilised by Linux distributions, notably Debian and Ubuntu, to indicate distribution-specific sub-versions. As such, hosts with differing identification strings are considered distinct.

For each evaluation batch, collection of fingerprintable data is performed as follows:

1. In the SSH key scan evaluator, pre-process all candidates of the batch and collect their IP addresses.
2. Check existing key scan results, which are shared between batches, and ignore addresses that already have a result for this evaluation run.
3. Partition the addresses by IP version, and further divide into chunks of up to 200 target addresses. For each chunk, start a target process handler.
4. Each process handler starts a new thread, which invokes a subprocess with the `ssh-keyscan` utility. Instruct it to use a timeout of two seconds per connection.
5. Upon subprocess termination, parse `stdout` for SSH host keys and `stderr` for SSH agent strings (indicating the remote SSH version).
6. After all threads have been started, join them one by one. Each process handler stores collected data, which is aggregated.
7. Store results in a file shared over all evaluation batches.
8. Provide new and existing key scan results via a special property, SshProperty.

In step 4, each subprocess is run until a pre-configured timeout. For large chunks, this timeout is increased proportionally to the amount of addresses to query.

Finally, during the evaluation process, the SSH evaluator translates collected information into a sibling decision. If either peer is missing SSH information, then the result is Indecisive. It seems intuitive to reject such a sibling pair. However, it is not uncommon for IPv4 and IPv6 firewall configurations to be inconsistent, which might cause one protocol having the SSH port open and the other having it closed [CLAB]. Thus, such a conclusion cannot be drawn.

[13]Obtained via `cat ssh-tsv | cut -d'<TAB>' -f3 | sort -u` on a data directory after evaluation.

If data is available for both protocols, a match of host keys or SSH agent constitutes a Positive sibling decision. It is sensible to assume that operators would not run separate SSH servers per protocol, since the default configuration is to listen on both versions, and there is no apparent reason to change that. As such, any mismatch results in a Negative decision.

3.5 Further Possible Verifying Metrics

In addition to these techniques, further methods were also considered. However, they were not selected for evaluation for various reasons.

The main reason is that nearly all of the proposed methods are falsifying features. That is, the collected data is sufficient to show that an address pair is very likely or definitely not a sibling pair, but the reverse is not true. This means that we cannot be as confident in verification of positive sibling decisions and expect not to be able to decide on a significant amount of pairs.

3.5.1 Address Cookies

Another basis to explore further techniques is that Bitcoin 0.21.0 is expected to change the way peers respond to address requests in a way that makes fingerprinting more difficult. At the time of our measurements, this version was a pre-release, but, especially considering future validity of our work, it is desirable to evaluate techniques that will continue to work even on the newly-released version.

This change[14], merged into Bitcoin code on August 3, 2020, caches a specific random `ADDR` response and keeps it consistent over 21 to 27 hours. This means that an adversary is only able to extract 1 000[15] peer addresses per time period and therefore realistically unable to know the full set of peer addresses at any time. Exactly this is also the named purpose of this change.

Intuitively, this would make fingerprinting easier since a consistent cache over all addresses would make it trivial to assign aliases. However, separate address sets are maintained per network (i.e. IPv4, IPv6, Tor, ...), which reverses this effect. In addition, due to the low amount of total retrievable addresses, it becomes more unlikely for multiple address responses to share peer addresses, which negatively impacts the applicability of our address fingerprinting technique. The expected impact is discussed in detail in Appendix C.

Another technique affected by this change are the address cookies suggested by Biryukov and Pustogarov [BP] in their 2015 work. Here, the idea is to advertise a unique set of IP addresses (either owned by the attacker or public addresses that do not run a

[14] `https://github.com/bitcoin/bitcoin/commit/acd6135b43941fa51d52f5fcdb2ce944280ad01e` – Pull Request: `https://github.com/bitcoin/bitcoin/pull/18991`
[15] `https://github.com/bitcoin/bitcoin/blob/v0.21.0rc2/src/net.h`

Bitcoin node) to each candidate. Subsequently, multiple `GETADDR` requests are made (the original work reports eight to be sufficient) to retrieve these "cookies" and the results compared to associate sibling IPs.

With the recent changes, this attack becomes much harder to mount since it is only possible to retrieve 1 000 addresses per 21-26 hour timeframe. Additionally, since it is hard to predict cache expiration, cookies would need to be persistent in the victim's address table for a prolonged amount of time. The original work shows that address cookies disappear over time as they are replaced or connection attempts are made. It might be possible to reduce the impact of this by analysing cache expirations and correctly predicting the five-hour timeframe in which the cache is able to expire.

However, the eight address sets suggested by the original work will not realistically be reached without long-term measurements. Hence, it would be necessary to place a substantial amount of addresses in the new table, such that it is sufficiently likely that a cookie would be chosen. The technique might be improved to require less address matches by advertising unique addresses to each peer (instead of just unique combinations). However, to measure the full Bitcoin P2P network, a substantial amount of addresses is necessary.

The effectiveness of the technique is tested by preparing a private Bitcoin node. Using a Python script, address cookies are advertised to it and the results retrieved. In this experiment, addresses from the reserved documentation block `2008:db8::/32` are used. The experiment shows not even a single address to be returned, which is due to the fact that Bitcoin has a comprehensive list of address ranges that it ignores – including all well-known reserved prefixes.[16] As such, real addresses would need to be used for the attack.

In addition, there are already security measures in place to reduce the impact that related advertising or advertised peers can have on the address table.[17] This is realised by allocating buckets depending on two hashes:

1. *secondary:* secret key, AS group of target, AS group of advertiser
2. *primary:* secret key, AS group of advertiser, primary hash modulo 64

Incorporating the secondary hash with a modulo means that a single advertising AS can only influence 64 buckets in total. In addition, a secret key is used, which makes it infeasible to predict allocation and choose advertised addresses in such a way that they influence many buckets. With this hashing, in theory, advertisers from a single AS are able to influence 64 buckets of 256 addresses each, i.e. 16 384 addresses by advertising addresses from a diverse set of ASes.

[16] `https://github.com/bitcoin/bitcoin/blob/v0.21.0rc2/src/netaddress.cpp#L466`

[17] `https://github.com/bitcoin/bitcoin/blob/v0.21.0rc2/src/addrman.cpp#L22-L31`

Further, it must be considered that other actors are also advertising (legitimate) addresses, which reduces the practical influence. An existing address is only overwritten if it is considered "terrible" or already in the table elsewhere (a single address may be in up to eight buckets). This further complicates estimation of how many addresses we need to advertise and are able to influence in practice.

Using the script in Appendix B with POS_COUNT = 16 384, the probability of not finding any, finding exactly one, or exactly two is found to be approximately zero (combined $p < 10^{-120}$). This models the unrealistic case of controlling all possible addresses, i.e. no other peers being active in the network. That is, in this case, almost always at least two addresses would be obtainable. This probability refers to an IP sibling, and other nodes would never return these addresses if the attack is performed properly.

Trying different, more realistic, values, a set of 50 controlled addresses is found to result in around 50% chance of retrieving none of the stored cookies, i.e. 50% of making a decision using the reverse probability. With 64 addresses, this decreases to 37%, yielding a decision in 63% of cases. That means, for realistic cookie sizes, more than 50 unique cookies need to be placed for each peer to get better results than a coin flip (assuming perfect accuracy, which should be achievable by placing only unrelated addresses that will not be advertised by legitimate peers).

This discussion relates to the unreleased 0.21.0rc2 version of the Bitcoin reference client. For the existing version, which is expect to mostly be encountered in our measurements, address cookies are not easy to place. The reason for this is that the 0.20.1 version does not accept address data from incoming connections[18], i.e. we cannot easily place cookies without forcing the peer to connect to us. As explained in previous work, this is difficult to do since only few peers are selected for outgoing connections [NAHa]. However, this changes in 0.21.0rc2[19], making address cookies a viable technique in the future.

As a result, for the existing 0.20.1 version that is expect to be mostly in use, it is not viable to mount address cookies, hence why application of this technique to Bitcoin is deferred to future work.

With the expected changes for 0.21.0, it will again be possible to place address cookies. Because of the 21-26 hour cache, long measurement periods will become necessary. Due to the insufficient probability of receiving back the placed cookies in a realistic setting, measurements will need to last multiple days. Further, placing a substantial amount of invalid addresses might, in extreme cases, negatively impact operation of the Bitcoin network. That is, address cookies will no longer be infeasible, but still challenging to implement once 0.21.0 has significant adoption.

[18] https://github.com/bitcoin/bitcoin/blob/v0.20.1/src/net_processing.cpp#L2166 only accepts addresses if CNode::isAddrRelayPeer() == true, which applies if m_addr_known != null – https://github.com/bitcoin/bitcoin/blob/v0.20.1/src/net.cpp#L2707 – i.e. the block_relay_only parameter to the CNode constructor must be false. For incoming connections, it is always set to true: https://github.com/bitcoin/bitcoin/blob/v0.20.1/src/net.cpp#L1035

[19] https://github.com/bitcoin/bitcoin/blob/v0.21.0rc2/src/net.h#L950

3.5.2 Adaption of InvBlock / TxProbe

A different technique that seems promising is *InvBlock*, introduced by Miller et al. [MLP+] in 2015. An extension of this technique has been proposed by Delgado-Segura et al. in 2018 [DSBPS+] (*TxProbe*) that uses orphan transactions, but the original technique is sufficient for our use-case, and the extension is more complex, which is why we disregard it. Both existing works have used this technique to reveal network topology, but not to discover IP siblings – that would be a novel application.

In *InvBlock*, a specific timeout of Bitcoin is abused: If a peer advertises knowledge of a transaction using an `INV` message and the remote has requested this transaction to be sent via `GETDATA`, it will not request this same transaction from other peers until a timeout has passed.

Up to and including version 0.20.1, transaction request logic has been as explained by a comment in `net_processing.cpp`[20]. A transaction advertised by `INV` will only be requested if it is not known already. The request is then only sent to a peer if it has not advertised too many transactions recently and has not advertised this specific transaction. It is noted that nodes in blocks-only mode never request transactions unless the advertising peer has been explicitly allowed by the node operator.

Once it has been decided to request the transaction, incoming peers receive a blanket delay of two seconds. In addition, if the transaction has been requested from another peer already, 60 seconds are added to the delay to allow it to respond. Exactly this extra delay is what is detect/exploited with *InvBlock*. For these 60 seconds, any other peer will observe a significant delay exceeding the expected two seconds. This global delay allows to identify IP siblings. The comment also notes that the two-second delay has been introduced to make *InvBlocking* attacks more difficult to mount.

For 0.21.0rc2, the transaction request logic has been overhauled by Pull Request #19988[21]. The updated logic is explained in the new file `txrequest.h`. This luckily does not affect the delay logic whose exploitation is proposed. The changes mostly relate to peer selection in case of multiple announcements.

We have seen that it is possible to use *InvBlock* to detect IP siblings, and that is viable in both versions 0.20.1 and 0.21.0rc2. However, it effectively only allows to validate a few siblings at once, making it expensive to mount for expectable data sets. It yields a POSITIVE sibling classification, which would make obtained data more meaningful. However, due to the introduced extra complexity of performing a great amount of active measurements in the EVALUATION phase, which is further infeasible in the concrete measurement setup for this work, where the node performing evaluation does not have IPv6 connectivity, implementation of this technique is left to future work. It should be noted that, to receive valid results, *InvBlock* would ideally be performed in parallel during HARVESTING, as nodes might disappear before evaluation.

[20] https://github.com/bitcoin/bitcoin/blob/v0.20.1/src/net_processing.cpp#L295

[21] https://github.com/bitcoin/bitcoin/pull/19988/files

The extended *TxProbe* method is recommended against in production setups by the original authors, because the required clearing of the orphan transaction set is a destructive action.

3.5.3 Miscellaneous Considered Metrics

In addition to these well-known methods, some further methods were considered which could possibly be interesting to explore in further work. Due to low expected success, these methods are only mentioned. For the start height in the `VERSION` message, it might be feasible to compare development using statistical methods, however a high sample rate is expected to be necessary for significant results, which is hard to achieve, especially considering that many nodes will have very similar behaviour due to quick propagation of information.

Another idea is to distinguish based on presence of `ADDR` vs. newer `ADDRv2` messages, however this is expected to be covered by the user agent string anyways. Further, headers- vs. blocks-first could be interesting, but blocks-first seems to be employed by very few nodes, and these are expected to be recognisable by the user agent string. Nodes have a mechanism to prevent self-connections, but its nonce is regenerated per connection, so cannot be exploited.

3.6 Expected Challenges

As mentioned, due to the small fraction of verifying metrics, many candidates are expected to remain Indecisive, which makes evaluation of the algorithms less meaningful. In particular, the majority of verfying metrics are TCPraw, Machine Learning, and Domain are verifying. The former two are to be validated, and the latter is only expected to yield few matches. Hence, most of the evaluation load sits on SSH Key Scan.

In addition to this, further challenges are to be expected. First, randomised timestamps, as outlined in section 2.4, have already been considered by Starke [Sta] via the additional Machine Learning model. More peers are expected to exhibit this behaviour, due to this being recommended by RFC 7323 [BBJS] and default in the Linux kernel, as explained above.

Further, for the same reasons, a higher fraction of Starke's ground truth might exhibit random timestamps by now. Re-training the model with this change would theoretically be feasible, but is considered out of scope for this work. Further work might compare such a modified model to the original one.

For the proposed attack resulting from our measurements, most of the IPv6 clients will expectedly be Dual-Stack, i.e. able to observe IPv4 clients as well. This is due to the common practice of offering both address families with servers – IPv6 addresses are cheap and easily available, and IPv4 connectivity is necessary to reach all clients until global IPv6 adaption has progressed further. If a considerable fraction fulfills this criterion, the attack will be effectively infeasible due to the large amount of targets.

Ethical implications have to be considered as well. Frequently sending TCP SYN packets without further responses is similar to a TCP SYN Denial of Service attack. However, one concurrent connection is used to each IP address and at least 30 seconds are left between attempts, we consider this to be insignificant. Still, well-monitored systems might report this as unexpected behaviour, wasting time of their operators on investigations.

We consider our Bitcoin connections harmless because a honest Bitcoin protocol is followed just collecting information. Insignificant damage is done to the community by using the resources but not providing services – The same is the case for any other non-public Bitcoin client. Our own address is not advertised, so poisoning of the network with invalid address does not happen. IP sibling detection using clock skew is already well-known, so we do not expect our contributions to significantly benefit adversarial activities. The DoS attack necessary to split the network as proposed will likely require many targets to be taken down, which will be infeasible for most criminals.

CHAPTER 4

Evaluation

In this chapter, measurements are executed using the approach described in chapter 3, *Methodology*, and the results are analysed. For the evaluation itself, the performance of Starke's algorithm is evaluated against the proposed validating metrics to verify its applicability to Bitcoin nodes. Additionally, decision rates and disagreements between the newly-proposed Bitcoin metrics and existing supporting metrics are checked. A closer examination of one of the measurements is performed to validate sanity of the results and find possible explanations for the specific values.

During evaluation, a total of 15 measurements was conducted. The first five, between 2020-09-28 and 2020-10-05, were of exploratory nature and did not include Bitcoin measurements, which were not fully implemented at the time. One measurement at 2020-10-18 did not include Bitcoin address data due to a software problem. Additionally, one measurement on 2020-10-20 failed to produce timestamps for similar reasons. This leaves us with eight measurements that have sufficient data with regard to extent and quality for analysis.

Most measurements were conducted over around 8 000 peers, while the first full execution on 2020-10-18 yielded only 4629 peers. The public charts at `Bitnodes.io` do not show such a significant decrease of peers as of 2021-01-02. Hence, this is classified as either a temporary software error or short-term node loss. The obtained data is still analysed, but the fact is considered when drawing conclusions from it.

An overview of the relevant measurements is presented in Table 4.1. These were conducted in two steps: First, a lightweight Dual-Stack machine executed the HARVESTING stage and collected SSH keys, which were then saved to disk. The files were then transferred to a more powerful (however IPv4-only) machine for computation of the EVALUATION stage.

The command lines that were used are, respectively, with the appropriate values replaced per measurement:

1.
   ```
   ipsiblings -vvv --do-harvest -hd 36000 -ti 60 -bi 4800
   -htf 180 --run-id=ms_2020-11-05_5h_ts60s_btc40m
   --eval-discard-results --eval-batch-size=50000
   --evaluator=SSH_KEYSCAN --eval-totals-in-memory
   ```

2.
   ```
   ipsiblings -vvv --targets-from=FILESYSTEM
   --run-id=ms_2020-11-05_5h_ts60s_btc40m
   --eval-batch-size=100000 --eval-totals-in-memory --skip-os
   ```

For evaluation, the metrics were divided into algorithm groups, whose decisions were combined into a single one:

1. **Definite:** Supporting metrics yielding results with high confidence – *TCP options, SSH key scan, Bitcoin address timestamps, Bitcoin version*
2. **Most Definite:** Definite metrics excluding Bitcoin, used to evaluate these proposed metrics ("supporting metrics") – *TCP options, SSH key scan*
3. **Bitcoin Definite:** Bitcoin metrics to evaluate – *Bitcoin address timestamps, Bitcoin version*
4. **Probable:** Positive metrics with lower expected confidence – *Domain*
5. **Improbable:** Negative metrics with lower expected confidence – *Bitcoin service flags*
6. **Starke:** Metrics proposed by [Sta] – *Starke TCP Timestamps, Starke Machine Learning*
7. **Scheitle:** Metrics proposed by [SGRC], not further analysed because decisions are very similar to the TCP Timestamp algorithm used by Starke – *Scheitle et al. TCP Timestamps*
8. **Keyscan**: Positives metrics – *SSH key scan*

4.1 Performance of Starke's Algorithm

One of the primary goals of this work is to verify the applicability of the algorithm proposed by Starke [Sta] for Bitcoin nodes. Evaluated of this aspect is performed by computing the fraction of correct and false decisions for each classification (positive, negative) relative to the supporting metrics. The resulting data is presented in Table 4.2.

For some portion of the data, the supporting metrics do not produce a decision, which requires special attention. These sibling candidate pairs are marked as unknown. For

		Interval				
#	Date	Duration	TCP	Bitcoin	# Peers	# Candidates
LA-1	2020-10-18	10 h	120 s	40 min	**4629**	**182 258**
LB-2	2020-12-21	10 h	120 s	**80 min**	8046	574 876
SD-3	2020-10-22	**5 h**	60 s	40 min	8034	573 920
LC-4	2020-10-26	10 h	120 s	40 min	8046	586 822
LC-5	2020-11-02	10 h	120 s	40 min	8199	632 106
LD-6	2020-11-03	10 h	60 s	40 min	8286	**2 018 448**
LD-7	2020-11-04	10 h	60 s	40 min	8265	663 327
SD-8	2020-11-05	**5 h**	60 s	40 min	8238	668 031

Table 4.1: Executed measurements and their parameters.

a fully-conclusive analysis, a low rate of unknown ratings is required, but this is not possible with the proposed method.

To prevent this fact from skewing results by signaling false confidence and suggesting wrong conclusions, well-known statistical metrics are not computed from the values. A large degree of uncertainty would result from this, and only lower and upper bounds could be given with available data, which is expected to be largely inconclusive due to the large portion of indecisive pairs.

For each of the executed measurements, it is observed that Starke's algorithm decides more than 98.7% of candidates. As expected, most of these decisions are negative. Usually, around four percent of candidates are rated positive, with notable outliers of 7.69%, 1.63% and 0.62%.

It should be noted that these numbers cannot be directly compared to the statistics presented in the original work, and in fact any of the existing works, because candidates are sanity-checked based on their Bitcoin version before evaluation. That is, the candidates presented to the evaluation algorithms already have a higher chance of being positive, and clear negatives are not counted in the statistics. This might also explain differences in observed performance to the original works proposing the algorithms.

Even with this knowledge, it is still notable that Starke's algorithm produces a consistent low false-negative rate of around 0.01%. This however might also just indicate high agreement with the supporting metrics, as around 20%—26% of negative classifications are neither confirmed nor denied by the supporting metrics, with an outlier of 34.50% in LA-1. As a result, the actual false-positive rate is below 26%, with a possible tendency for lower values if indecisive cases are distributed similarly to decisive ones.

For positive classifications, performance does not look as promising. Most measurements produce a false-positive rate of around 50%—60%, with an outlier of 70.18%. The rate of true positives is usually around 1%, with an outlier of 4.78%. Usually, around 40% of positives are not decided by the supporting metrics, which allows true values of these metrics to drift significantly in either direction. However, as, usually, only a small fraction of candidates is rated positive, these rates do not always cause high absolute errors.

For analysis of positive decisions, it is also important to note that the supporting metrics produce a relatively consistent positivity rate of around 0.05%. This might be an artefact of only having a single verifying metrics, SSH host keys, which is not able to rate a high fraction of candidates. Usually, around 75% of candidates are decided by the supporting metrics. For reported false positives, a high degree of certainty is expected as outlined in *Methodology*. True positives might also be produced by deployments where multiple hosts would share SSH host keys (e.g. badly-designed cloud images), but we do not expect this to have significant impact, as there would be very little true positives left if that were the case.

In summary, good performance is observed on negative decisions, with uncertainty of up to 25% of candidates. For positive classifications, convincing statistics are not seen. Even if all unknown positive ratings were to be true positives, a performance of 50% is not exceeded.

4.2 Performance of Bitcoin Metrics

Another key issue for this work is to evaluate the effectiveness of Bitcoin application-layer methods. This is analysed by considering the decision rate in comparison to the remaining metrics as well as the overall value. Additionally, it is analysed how decisions are spread between positive and negative, and how many of these are in conflict with other metrics. This data is presented in Table 4.3.

Over all measurements, Bitcoin decides around 50-60% of candidates, with LA-1 being an outlier with 40.24%. In contrast, the remaining metrics combined usually decide between 45 and 55%. Because the other metrics combine various aspects, a match with a tendency to exceed (except in two cases) is good performance.

Additionally, the sets of candidates that are decided by each of these metric groups do not fully overlap. That is, by considering Bitcoin-specific behaviour, overall decision rate improvements of around 20%—30% are observable in the conducted measurements.

Regarding classifications, Bitcoin does not produce positive ratings, which inhibits comparison. For the negative ratings, a rate 3%—12% higher than supporting metrics is observed, with one exception being $\approx 1.5\%$ lower. This is the measurement where the Bitcoin sample period is increased from 40 to 80 minutes, which hints at a positive effect of a higher sample rate.

Conflicts, that is, other metrics (here, this is always the SSH key scan) rating a sibling positive that is denied by Bitcoin are consistently around 0.04% of Bitcoin negative ratings, with one exception having 0.02%. Relative to positive decisions produced by the SSH key scan, this however amounts to 20-30% – due to the low amount of positives. SSH host keys might still possibly be shared over multiple hosts, while the considered

Decision	Positive Classifications				Negative Classifications			
Rate	**Rate**	**False**	**True**	**Unk.**[a]	**Rate**	**False**	**True**	**Unk.**[a]
Measurement LA-1, 10 h, TCP 120 s, BTC 40 min. (4629 peers)								
98.85%	7.69%	53.43%	0.68%	45.88%	91.16%	0.01%	65.49%	34.50%
64.37%	0.07%				64.31%			
Measurement LB-2, 10 h, TCP 120 s, BTC 80 min. (8046 peers)								
98.86%	4.21%	58.25%	1.24%	40.51%	94.65%	0.01%	73.66%	26.33%
72.96%	0.06%				72.90%			
Measurement SD-3, 5 h, TCP 60 s, BTC 40 min. (8034 peers)								
98.77%	1.63%	62.79%	1.83%	35.38%	97.14%	0.02%	74.98%	25.00%
74.77%	0.05%				74.72%			
Measurement LC-4, 10 h, TCP 120 s, BTC 40 min. (8046 peers)								
98.79%	4.95%	59.04%	0.88%	40.08%	93.75%	0.01%	74.80%	25.19%
74.05%	0.05%				74.00%			
Measurement LC-5, 10 h, TCP 120 s, BTC 40 min. (8199 peers)								
98.88%	3.80%	61.84%	1.25%	36.91%	95.08%	0.01%	79.58%	20.41%
78.99%	0.06%				78.93%			
Measurement LD-6, 10 h, TCP 60 s, BTC 40 min. (8286 peers)[b]								
98.76%	0.62%	70.18%	4.78%	25.04%	98.14%	0.02%	79.37%	20.61%
79.34%	0.05%				79.29%			
Measurement LD-7, 10 h, TCP 60 s, BTC 40 min. (8265 peers)								
98.71%	3.74%	55.46%	0.73%	43.80%	94.97%	0.00%	73.90%	26.10%
73.20%	0.03%				73.16%			
Measurement LD-8, 10 h, TCP 60 s, BTC 40 min. (8338 peers)								
98.88%	4.09%	59.64%	1.07%	39.29%	94.79%	0.01%	78.44%	21.55%
77.68%	0.05%				77.63%			

[a] Unknown, i.e. the supporting metric did not produce a decision for these cases.
[b] This measurement yields about twice as many viable candidates than comparable ones, which might explain the significantly lower positive rate.

Table 4.2: Comparison of classifications produced by Starke's algorithm (first row per section) against the set of all supporting metrics (second row per section).

Bitcoin metrics – version and address invariants – deny a sibling relationship with high confidence. Cases where the version changes are not decided by the algorithm, so these do not lower confidence here. In absolute numbers, each measurement had between 85 and 166 conflicts, with the exception of LD-6 which had significantly more candidates and 474 conflicts.

In conclusion, inclusion of Bitcoin metrics has been observed to have a significant positive effect on overall decision rate and helps increase confidence of decisions made by other algorithms.

4.3 Detailed Analysis of a Single Measurement

In this section, the measurement LD-7 is analysed in detail to find possible patterns or reasons for specific performance metrics.

4.3.1 Starke's Algorithm

The first aspect are the positive sibling decisions made by the Starke algorithm family. These are analysed using a TSV file output by the platform which lists the (IPv4, IPv6) pairs determined to belong together by the algorithm. For this run, 24 826 sibling pairs are produced, which makes a full manual analysis infeasible. These pairs involve 2731 out of 6990 IPv4 peers, which amounts to just over 39%. Of these, 825 (30%) are assigned to a single IPv6 partner. Of the 1275 IPv6 addresses, 917 (72%) have at least one IPv4 sibling assigned – 375 have a single partner. Out of these, 352 have exactly one IPv4 assigned.

Despite the significant number of suggested siblings, a general sanity check on a subset of the data is still performed. The Telnet IP-to-ASN mapping service provided by Team Cymru[1] is utilised to assist in the process. This takes a list of IP addresses and provides the Autonomous Systems numbers and names they are assigned to. This data is requested for all IPv4 and IPv6 addresses contained in any sibling pair.

These results are combined in a spreadsheet with the sibling pairs. For a qualitative analysis, the first 711 sibling pairs are analysed. In general, we expect a sibling pair to be a false positive if the AS names of the addresses differ, which is why such associations are not further verified. This assumption is validated on a small subset of this data, and no contradiction is found. During manual analysis, a check is also performed for similar AS names that are not identical, but only one such case is found (`GANDI-AS` vs. `GANDI-AS-2`).

For each of the plausible pairs, a HTTPS connection is attempted (with fallback to HTTP) and the served content as well as the common name specified in the certificate is compared. If these match, a sibling relationship is assumed. If no HTTP(S) service is

[1]`https://team-cymru.com/community-services/ip-asn-mapping/` – site accessed and data obtained 2021-01-04

	Decision Rate		Decisions		Conflicts	
Algorithm		**Overall**	**Positive**	**Negative**		**%[a]**
Measurement LA-1, 10 h, TCP 120 s, BTC 40 min. (4629 peers)						
Bitcoin	40.24%	64.37%	-	40.24%	28	0.04%
Others	43.40%		0.08%	43.44%		20.29%
Measurement LB-2, 10 h, TCP 120 s, BTC 80 min. (8046 peers)						
Bitcoin	49.49%	72.96%	-	49.49%	100	0.04%
Others	50.99%		0.08%	50.91%		21.46%
Measurement SD-3, 5 h, TCP 60 s, BTC 40 min. (8034 peers)						
Bitcoin	57.01%	74.77%	-	57.01%	134	0.04%
Others	45.69%		0.07%	45.61%		32.29%
Measurement LC-4, 10 h, TCP 120 s, BTC 40 min. (8046 peers)						
Bitcoin	55.07%	74.05%	-	55.07%	141	0.04%
Others	48.22%		0.07%	48.15%		32.19%
Measurement LC-5, 10 h, TCP 120 s, BTC 40 min. (8199 peers)						
Bitcoin	58.88%	78.99%	-	58.88%	166	0.04%
Others	53.01%		0.08%	52.03%		31.44%
Measurement LD-6, 10 h, TCP 60 s, BTC 40 min. (8286 peers)						
Bitcoin	59.55%	79.34%	-	59.55%	474	0.04%
Others	54.65%		0.08%	54.57%		30.27%
Measurement LD-7, 10 h, TCP 60 s, BTC 40 min. (8265 peers)						
Bitcoin	56.35%	73.20%	-	56.35%	85	0.02%
Others	44.82%		0.04%	44.77%		28.52%
Measurement LD-8, 10 h, TCP 60 s, BTC 40 min. (8238 peers)						
Bitcoin	57.41%	77.68%	-	57.41%	154	0.04%
Others	53.68%		0.08%	53.60%		30.02%

[a] Bitcoin: Rate of negative decisions, Others: Rate of positives.
The opposite rate is always zero.

Table 4.3: Comparison of the proposed Bitcoin-specific supporting metrics' results against the remaining metrics'.

provided, the reverse DNS entries are compared. A substantial portion of the dataset does not provide rDNS for IPv6, while all analysed IPv4s have entries.

An additional factor to be considered is that some service providers do not offer (proper) IPv6 support, which is why some administrators use special IPv4-to-IPv6 tunnelling services that provide IPv6 access to IPv4-only machines. One well-known provider is Hurricane Electric, which is excluded from automatic rating as false-positive. Surprisingly, a significant portion of analysed IPv4 addresses is associated with one or more of the same small set of tunnelled IPv6 addresses. Most of these tunnelled addressed provide HTTPS, which makes it easy to deny a sibling relationship, but this behaviour is still interesting.

Another notable occurrence is that IPv4 IPs assigned to Hetzner are commonly assigned a large amount of IPv6 addresses of the same provider. According to its published policies[2], the company assigns a /64 IPv6 subnet to each cloud server, however all IPv6 candidates are in different /64 subnets. This suggests them being false-positives. Earlier research [KBC] has already shown that, depending on the virtualisation technology used, virtual machines may or may not exhibit individual clock skews. In this case, it seems that the latter applies, causing these false-positives.

Of the 711 pairs analysed, 647 are rejected as false positives (91%). Of these, 465 ($\approx$ 72%) have mismatching AS names. 122 ($\approx$ 19%) have both sides assigned to Hetzner, with multiple Hetzner candidates for that IPv6 and no decision made by manual analysis. 40 candidates ($\approx$ 6%) are assigned to IPv4-to-IPv6 tunnels with HTTP(S) responses. 16 ($\approx$ 2%) have contradictory HTTP(S) responses. Finally, four of the candidates ($<$ 1%) use a tunnelling service when the provider is known to provide IPv6.

56 pairs ($\approx$ 8%) received an unknown rating. 22 did so because of a timeout on all considered services, and 32 because either side was no longer routable. Additionally, two consumer-grade broadband connections were found, but neither responded to HTTP(S) on IPv6. In total, eight pairs (1.1%) were rated as true siblings. Four had matching HTTP(S) responses, and another three had the same, uncommon AS. For the remaining candidate, reverse DNS entries matched.

This manual analysis suggests that the false-positive rates assigned by the verifying metrics to Starke's algorithm are realistic. In addition, we see that a high fraction of these ratings stem from assigning multiple siblings to the same node, where more than half are very implausible due to mismatch in AS and country. As originally suggested by Starke [Sta], incorporating IP address location on a country-level as a falsifying feature would improve the algorithm. In particular, this is expected to significantly reduce the false-positive rate, one major issue with the algorithm's performance in this setting.

[2] https://docs.hetzner.com/cloud/floating-ips/faq (accessed 2021-01-04)

4.3.2 Verifying Metrics

The verifying metrics assign 9 429 sibling pairs, which is close to the number of total nodes and might therefore seem unrealistic. However, the metrics tend to assign multiple siblings. The number of siblings assigned to each IPv4 and IPv6 peer is illustrated in Figure 4.1.

From these charts, around 55% of IPv4 addresses are not assigned a single sibling. 14% are assigned to a unique IPv6 sibling, which is interpreted as a high-confidence sibling assignment. Multiple siblings might legitimately be assigned when IPv4 hosts are hidden behind NAT, but this is not expected for the Bitcoin network as a large portion is hosted on public servers, which often necessarily receive dedicated IPv4 addresses. Another reason for legitimate multiple assignments is the implementation of IPv6 Privacy Extensions, which randomise the interface identifier of IPv6 addresses, causing many IPv6s to be associated with the same IPv4. For most servers, management of dynamic IP addresses is undesirable over IPv6 as well, so this is not expected to be a common setup and thus ignored during evaluation.

Another 14% of IPv4 addresses have between two and five assignments. The final 17% are assigned to more siblings, with individual values spread out similarly over the remaining buckets of five, with a bias for lover numbers. The highest assignment count for a single IPv4 is 80.

For IPv6, the distribution is different. As expected due to the low utility of IPv6-only hosts (users with IPv4-only setups cannot connect to these), only 22% of IPv6 peers do not receive a single sibling assignment. The largest group with 32% receives exactly one IPv4 sibling. Due to a high maximum of 254, assignment counts are divided into buckets of 20. The largest of these is at the lower end, 2—20, with 19%. Again, lower values are more common, with a notable exception in the 81—100 bucket. This means that a high proportion of the detected IPv4 siblings are assigned to the same few IPv6 addresses. The Hetzner anomaly described above does not explain this, since only 29 hosts are involved.

Of the 825 unique assignments for IPv4 peers and 375 IPv6 peers, 207 are bijective (one-to-one). For these pairs, a unique partner is found by the algorithm.

4.3.3 Additional Metrics

Additionally, ratings performed by metrics with lower expected accuracy are verified. The domain metric, which compares rDNS domains reported by the `Bitnodes.io` API, has made a total of 105 positive classifications. Of these, 27 were true-positives, 26 false-positives, 37 unknown and 15 conflicting or erroneous in the supporting metrics. For the siblings that were rated by supporting metrics, this means that the domain metrics adds little value, with around half of the classifications being incorrect, i.e. being equivalent to a coin toss. Depending on the actual rating of unknown results, the true-positive rate however might vary between 75% and 25%, so no confident statement can be made here.

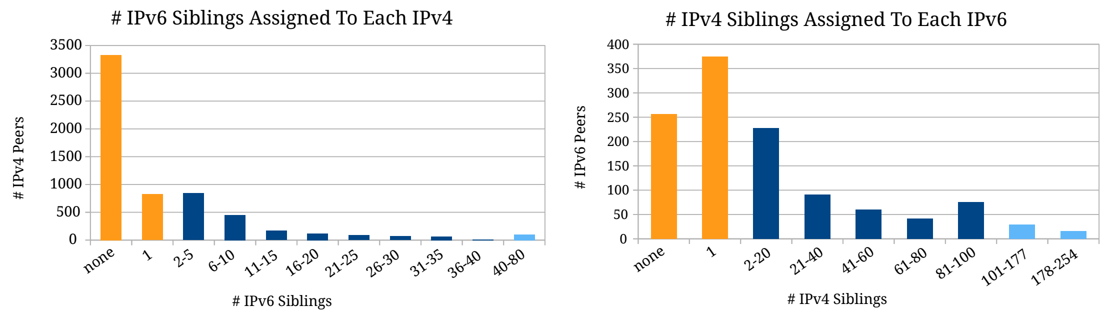

Figure 4.1: Diagrams showing the number of siblings from the opposite address family assigned to each IPv4 (left) and IPV6 (right) peer. Not all bins are equidistant, identical sizes are marked in the same colour.

For the Bitcoin address-service metric, not many decisions were expected. Still, 107 372 were made, which amounts to 16% of all candidates, and is roughly comparable to the amount of negatives produced by the keyscan metric. Two of these were rated false-negatives, which might be either due to nodes being removed and re-added to the tables or false-positives of the supporting metrics. 89 878 (83.7%) were true-negatives and 17 490 (16.2%) were unknown – this fraction shows that the metric adds further decisions. Even in the worst case of all unknowns being false-positives, the rate of true-negatives is reasonable. In the best case of only 2 false-negatives occurring, this is a highly valuable metric.

4.3.4 Bitcoin Versions

Finally, the distribution of Bitcoin user agents and versions is analysed. The version distribution for this measurement is plotted in Figure 4.2. All individual user agent strings that are not plotted each have 19 or less occurrences, with numbers quickly dropping below ten. “Others” also includes some official but very outdated versions, and more recent official versions with customised messages coded into the user agent string.

4.4 Parameter Choices

Another aspect that can be analysed with the obtained data is the influence of parameter choices on the quality of decisions produced. The data used for this analysis has already been presented in tables in this chapter.

An overview of correlation between decision rate of the metric groups and measurement parameters is gained, by plotting them against the type of measurement. The shorter Type-D measurements SD-3 and SD-8 do not produce significantly different results, so no distinction is made based on measurement duration. The resulting chart is shown in Figure 4.3.

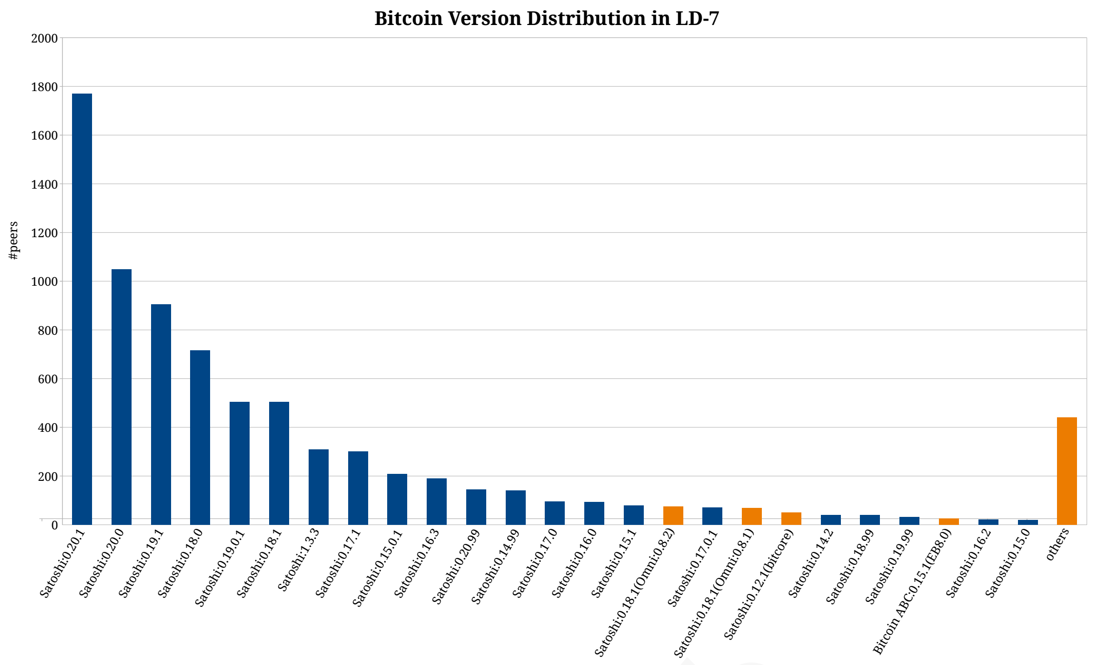

Figure 4.2: Bitcoin version distribution in measurement LD-7. Nonstandard user agents are marked in orange.

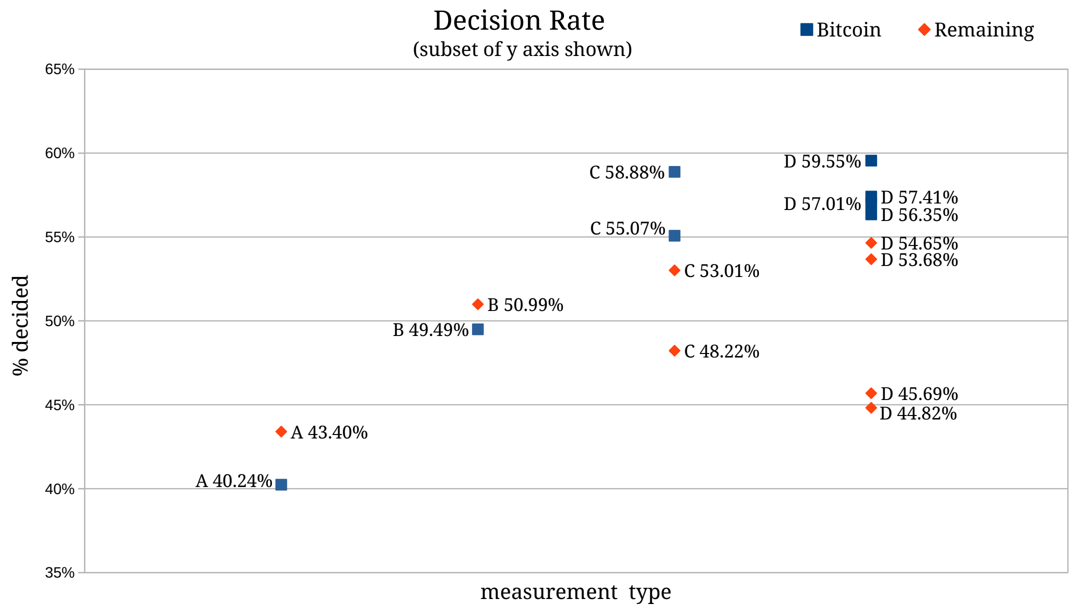

Figure 4.3: Plot of decision rate per metric group against measurement type.

From this, it is observable that, for Bitcoin, performance is relatively consistent inside a measurement group ($\pm3.5\%$), while greater variation is observed for the remaining metrics. Additionally, groups A and B show visibly lower decision rates. Group B has a doubled Bitcoin measurement interval and group A has lower peer count. As expected, group B and D do not yield significantly different results for the supporting metrics, because the varied parameter is TCP timestamp period, which should not affect them directly.

While the latter cannot be influenced by the measurer, a higher Bitcoin measurement interval may be freely chosen. For a conclusive recommendation, further measurements would need to be performed in both groups, however it seems reasonable to prefer the 40-minute interval for Bitcoin. In the conducted experiments, a single collection run among all peers was observed to take around 35 minutes, which means that significantly lower values are not realistic with the proposed algorithm and available hardware. Lower intervals might be achievable by splitting work over multiple measurement nodes or provisioning more powerful hardware.

For Starke's algorithm, measurement SD-3 with the reduced 5-hour measurement period yields only half the positive ratings compared to the remaining D measurements (in absolute numbers, and in relative numbers with the exception of LD-6 with increased candidate count). This is not true for SD-8, which shows the highest positive rate among D measurements. However, in both cases, no significant increase in false-positive or false-negative rate is observed in relation to the longer two runs of the same group. As before, this statement cannot be made with high confidence due to the significant fraction of indecisive candidates for supporting metrics.

For the proposed attack, a long measurement duration in inhibitive, so a lower measurement duration is to be preferred because no negative effect has been observed. Additionally, for the attack, false positives are preferred because even a small amount of false negatives would effectively prevent the attack. This is true in relative numbers for Starke's algorithm (again under the condition that candidates with unknown status behave similarly). In absolute numbers, this is between 14 and 109 candidate pairs for our measurements, with LD-6 producing an outlier of 495. These numbers do not directly relate to peers, since a single peer might be included in multiple candidate pairs. For purposes of the attack, a false negative is irrelevant if the peer is included in another positively-rated candidate pair.

A more decisive statement about measurement duration can be made using two additional measurements which were performed at the beginning of January 2021 with shorter duration of 60 and 30 minutes. The shorter measurement is performed with doubled timestamp acquisition rate (30 seconds), in an attempt to provide a sufficient number of timestamps. For both, similar decision rates and classification ratios to the remaining measurements were achieved for all algorithm groups except Starke, hinting at no performance degradation.

The longer measurement also achieved a similar decision rate for Starke (-2%), yet significantly fewer positives (0.17%). Classification performance was comparable, but with relatively more true positives (5.29%). In absolute numbers, there are half as many true-positives as with other measurements, so this is not significant.

For the shorter measurement, Starke's algorithm only classifies 0.03% in total, all positives. Among these, 25% true-positives and 22% false-positives are achieved, with the remainder being unknown. The reason for this is that, here, only decisions made by TCPraw are reported, the Machine-Learning algorithm produces an Error state for all candidates. This is caused by the dynamic range calculation failing, however the underlying cause is non-obvious. A conclusive decision on whether such short measurements are viable might be produced by adapting this property.

CHAPTER 5

Conclusion

In this work, a methodology to discover IP siblings in the Bitcoin network is proposed based on previous research, with special mention of [Sta], whose measurement platform was adapted to fit the peculiarities of the Bitcoin network. Additionally, three new Bitcoin-specific metrics are proposed, two of which significantly improve the decision rate relative to existing techniques. Multiple measurements were conducted to validate applicability of both new and existing methods to the Bitcoin network, and results were analysed. Due to known but necessary limitations in the methodology, many results include a high margin of error.

5.1 Interpretation

1. *Are existing techniques of IP sibling detection using clock skew measurements suitable for the Bitcoin network?*

   From the measurements conducted in this work, it is not possible to draw a final conclusion due to the large fraction of unknowns in the ground truth set. However, for rated data, good performance is observed for negative decisions, which indicates applicability. Still, a high rate of false-positives is observed for the methodology proposed by [Sta], which is based on clock skew. No evidence has been found to deny applicability of clock skew-based measurements to Bitcoin in general.

2. *Can such techniques be augmented by utilising properties specific to the Bitcoin protocol or network?*

   Yes. This work proposes three metrics based on such properties which are shown to perform well on the rated data in relation to existing metrics utilised as ground truth. The first proposed metric compares Bitcoin protocol versions and user agent strings to discard obvious non-siblings. The latter two analyse address data advertised by

Bitcoin nodes, verifying invariants imposed by the reference implementation based on timestamps and service flags, respectively.

Additionally, further metrics are sketched but not evaluated.

3. *Is it feasible to run this detection in a non-disruptive way, ideally behaving like a legitimate Bitcoin client?*

   The proposed techniques are not expected to be overly disruptive, neither to the Bitcoin network as a whole nor to individual hosts, with the parameters used in this work. As explained in the "Further Work" section, it is theoretically possible to implement measurement while behaving solely like a legitimate Bitcoin client. Actual implementation of such measurement would require significant additional work and as such is deferred to future research.

4. *What portion of the Bitcoin network is currently operated by Dual-Stack nodes, i.e. nodes that are connected via IPv4 and IPv6?*

   Due to the data quality produced by verifying metrics, it is not possible to provide exact numbers with high confidence based on the collected statistics. The following numbers do *not* take candidates into account where a decision is not produced by the verifying metrics. Peers where all candidates remain unrated could also be dual-stack nodes, so these numbers are to be seen as lower bounds.

   For IPv4, 825 peers (14%) are assigned to a single IPv6 sibling. In the other direction, 375 unique assignments are made, amounting to 32% of all IPv6 peers. 307 one-to-one assignments are made, which corresponds to a lower bound on the portion of Dual-Stack nodes.

   In the collected data, 917 IPv6 peers (78%) are assigned at least one sibling. For IPv4 peers, this number is 2731, amounting to 45%. For the algorithm results, this is an upper bound, assuming that non-unique assignments are more likely to be false positives.

5. *Is it feasible to split the Bitcoin network into an IPv4- and IPv6-only part by attacking these Dual-Stack nodes? Could such a situation be abused?*

   With the numbers from 4., in particular on the IPv6 side, the portion of Dual-Stack nodes is non-negligible. Still, this might be far less than all nodes. Only 22% of IPv6 nodes are assigned no sibling at all. As such, in the worst case (excluding peers where all assignments remain unrated), 917 nodes would need to be targeted by DDoS, which is a significant number for an attacker without excessive resources.

   Still, the natural boundary provided by the separate protocols might still be an advantage for a split that is not generally trivial to obtain in absence of other specialised attacks, some of which are presented in the Background section. As mentioned in [AZV], the Bitcoin Peer-to-Peer protocol itself is not the only means of communication between nodes, with notable exception being communication internal to mining pools and Tor. Hence, practical feasibility of such an attack is questionable.

6. *How long does it take to detect a newly-joined IP sibling with this method?*

   No negative impact was observed for the reduced measurement period of five hours, and insignificant negative impact for 60 minutes. It is therefore expected that further reductions are theoretically possible. For these shorter durations, however, the existing algorithm does not produce results, which could be improved in further work. For the proposed attack scenario, a very short measurement period in the range of seconds is necessary, otherwise a split might be resolved by a newly-joined node.

   [Sta] has already shown success with a measurement period of 80 seconds, but further reductions require additional work. Such reductions *might* be possible, but are not compatible with the Bitcoin measurements described in this work and will require fine-tuning, and are hence deferred to future work.

In conclusion, the proposed attack does *not* provide an extraordinary advantage for an adversary trying to split the Bitcoin network.

5.2 Future Work

A very obvious improvement to this work is the extension of the existing measurement platform to work as a distributed system over multiple measurement nodes, such that periods could be decreased and therefore faster decisions made. This is especially interesting for Bitcoin measurements with their current 40-minute minimum duration. In this context, it is crucial to consider the clock skew between the different measurement nodes, possibly using a similar method to [HYTC] and/or [Noo].

Another interesting idea that was not implemented as part of this work is to hide the measurements better, such that they would not show up in monitoring solutions. The proposed method of just sending TCP SYN packets that are never followed-up on is conceptually equivalent to a TCP SYN Denial of Service attack, even if at far lower frequency. Some monitoring tools might wrongly detect the measurements as such attacks, which might cause an adversary performing the measurement to be detected. Instead, Bitcoin connections might be used for measurement, since these are usually long-lived either way – similar to the concept for SSH proposed by [Sta]. Depending on concrete implementation choices, this is expected to require more resources and introduce additional complexity into the platform.

For the Bitcoin protocol itself, more advanced correlation of multiple connections to the same target might be interesting as well. This might also entail observation of further properties such as transaction announcement. In addition, further Bitcoin-based metrics might be derived, in particular a verifying metric would be beneficial.

Another improvement might be to correlate the Autonomous System information provided for each side of a pair, as manually performed during evaluation and as proposed by [Sta] on a country level. Apart from this, it might be possible to make sibling decisions based on gaps in timestamp and Bitcoin replies, which correspond to downtime periods for a node.

The current methodology does not handle Tor nodes at all, however these would in practice still connect the network. An expansion of the platform might measure these as well. What makes this difficult is that, as shown in [Mur], TCP Timestamp mechanisms cannot be directly applied over the Tor network. Instead, it might be possible to use the (low-granularity) remote timestamps provided by Bitcoin to conduct measurements.

For data analysis, improved observability for the platform could be beneficial, as applied in [BB], providing more detailed insight into the decision process and concrete reasons for acceptance or rejection of a sibling candidate.

For the SSH host key metric, an additional heuristic could be to discard key information that is reused beyond a certain threshold, e.g. five hosts. Such behaviour might suggest misconfiguration or use of default keys, which currently produces false positives. However, this would require further evaluation, since, considering the size of IPv6 subnets currently handed out with servers, a node might legitimately have a high number of IPv6 aliases. (Although it is questionable if they would all be used for Bitcoin nodes.)

As discussed in Chapter 4, address cookies are expected to become obsolete in the current form with address packet caching being introduced in Bitcoin Core 0.21.0. Further research should be conducted on how this concept might still be applied to recent Bitcoin versions. One possibility might be to predict the time of cache expiry and place a large amount of address cookies in the relevant table right before this happens, increasing the likelihood of finding a cookie in the address response over different address families. However, the expiration period of up to 27 hours entails very long measurements here, so this would not be directly applicable to the scenario discussed in this work.

Further promising techniques are InvBlocking and TxProbe, as proposed by [MLP+] and [DSBPS+], respectively. A method based on these two allows positive and negative classifications. The additional of positive decisions is highly beneficial for our method, despite these being active methods that increase measurement complexity as well as introducing Bitcoin cost. The original authors of [DSBPS+] recommend against using the original technique on the live Bitcoin network due to the data loss entailed by clearing the orphan pool.

A possible application of these methods to this work might be to announce a fake transaction to one node in a candidate pair and observe immediately on the partner if a delay is observable in the `GETDATA` request. Since the transaction would not need to be announced here, no Bitcoin cost is introduced, but this might still be intrusive. Additionally, this would need to be performed for every candidate pair in the current architecture, meaning more than 600 000 expected requests, which might induce unacceptable additional load on the Bitcoin network and is not expected to be feasible with only a single measurement node in a short time. Further theoretical and practical research needs to be conducted on the original and adapted methods to ensure applicability and deny intrusiveness.

List of Figures

3.1 Diagram modelling data and execution flow, and the evaluation process. . 18
3.2 Diagram modelling execution and data flow for the TCP Timestamp harvest provider. Notation is loosely based on that of a UML Sequence Diagram. 20
3.3 Diagram modelling execution and data flow for the Bitcoin harvest provider. Notation is loosely based on that of a UML Sequence Diagram. 21

4.1 Diagrams showing the number of siblings from the opposite address family assigned to each IPv4 (left) and IPV6 (right) peer. Not all bins are equidistant, identical sizes are marked in the same colour. 48
4.2 Bitcoin version distribution in measurement LD-7. Nonstandard user agents are marked in orange. 49
4.3 Plot of decision rate per metric group against measurement type. 49

Bibliography

[AZV] M. Apostolaki, A. Zohar, and L. Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 375–392. ISSN: 2375-1207.

[BB] Robert Beverly and Arthur Berger. Server siblings: Identifying shared IPv4/IPv6 infrastructure via active fingerprinting. In Jelena Mirkovic and Yong Liu, editors, *Passive and Active Measurement*, pages 149–161. Springer International Publishing.

[BBJS] Dave Borman, Bob Braden, Van Jacobson, and Richard Scheffenegger. RFC 7323: TCP extensions for high performance. Published: Internet Requests for Comments.

[BP] A. Biryukov and I. Pustogarov. Bitcoin over tor isn't a good idea. In *2015 IEEE Symposium on Security and Privacy*, pages 122–134. ISSN: 2375-1207.

[CLAB] J. Czyz, M. Luckie, M. Allman, and M. Bailey. Don't forget to lock the back door! a characterization of IPv6 network security policy. In *Network and Distributed Systems Security (NDSS)*.

[DSBPS$^+$] 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.* _eprint: 1812.00942.

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

[HMvdW$^+$] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph

Gohlke, and Travis E. Oliphant. Array programming with NumPy. 585:357–362.

[HYTC] D. Huang, K. Yang, W. Teng, and G. Chiu. Design of client device identification by clock skew in clouds. In *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1133–1138.

[KAC] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 906–917. Association for Computing Machinery. event-place: Raleigh, North Carolina, USA.

[KBC] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. In *2005 IEEE Symposium on Security and Privacy (S P'05)*, pages 211–225.

[MCR$^+$] Sami Ben Mariem, Pedro Casas, Matteo Romiti, Benoit Donnet, Rainer Stütz, and Bernhard Haslhofer. *All that Glitters is not Bitcoin – Unveiling the Centralized Nature of the BTC (IP) Network*. _eprint: 2001.09105.

[MLP$^+$] A. Miller, James Litton, Andrew Pachulski, Neal Gupta, D. Levin, N. Spring, and B. Bhattacharjee. Discovering bitcoin ' s public topology and influential nodes. University of Maryland.

[MST] S. B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, volume 1, pages 227–234 vol.1.

[Mur] Steven J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 27–36. ACM. event-place: Alexandria, Virginia, USA.

[NAHa] T. Neudecker, P. Andelfinger, and H. Hartenstein. A simulation model for analysis of attacks on the bitcoin peer-to-peer network. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1327–1332. ISSN: 1573-0077.

[NAHb] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. Timing analysis for inferring the topology of the bitcoin peer-to-peer network.

[Nak] Satoshi Nakomoto. Bitcoin: A peer-to-peer electronic cash system.

[Neu] Till Neudecker. Characterization of the bitcoin peer-to-peer network (2015-2018). ISSN: 2190-4782 Num Pages: 29 Series: Karlsruhe Reports in Informatics Volume: 2019.

[noa] *RFC 793: Transmission Control Protocol.* Number 793 in Request for Comments. RFC Editor. Published: RFC 793.

[Noo] Willem Noort. Gathering intelligence from the bitcoin peer-to-peer network.

[Pax] Vern Paxson. On calibrating measurements of packet transit times. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '98/PERFORMANCE '98, pages 11–21. ACM. event-place: Madison, Wisconsin, USA.

[PF] L. Polčák and B. Franková. On reliability of clock-skew-based remote computer identification. In *2014 11th International Conference on Security and Cryptography (SECRYPT)*, pages 1–8.

[PISP] S. Park, S. Im, Y. Seol, and J. Paek. Nodes in the bitcoin network: Comparative measurement study and survey. 7:57009–57022.

[Pus] Ivan Pustogarov. Deanonymisation techniques for tor and bitcoin.

[SGRC] Quirin Scheitle, Oliver Gasser, Minoo Rouhi, and Georg Carle. Large-scale classification of IPv4-IPv6 siblings with nonlinear clock skew. abs/1610.07251.

[SM] A. Soni and S. Maheshwari. A survey of attacks on the bitcoin system. In *2018 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, pages 1–5.

[Sta] Marco Manfred Starke. Identifying IP siblings on public network devices. (full text not published).

[TAM] J. Tapsell, R. Naeem Akram, and K. Markantonakis. An evaluation of the security of the bitcoin peer-to-peer network. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1057–1062.

[TCM+] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang. A stealthier partitioning attack against bitcoin peer-to-peer network. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 894–909. ISSN: 2375-1207.

[YLa] T. Ylonen and C. Lonvick. RFC 4251: The secure shell (SSH) protocol architecture. Publicshed: Internet Requests for Comments.

[YLb] T. Ylonen and C. Lonvick. RFC 4253: The secure shell (SSH) transport layer protocol. Published: Internet Request For Comments.

[ZM] Sebastian Zander and Steven J. Murdoch. An improved clock-skew measurement technique for revealing hidden services. In *Proceedings of the 17th USENIX Security Symposium*, pages 211–226.

Appendices

Appendix A. DNSSEC Status of Bitcoin DNS Seeds

As of October 25, 2020, the following DNS seeds are employed for the main chain in the Bitcoin source code (`https://github.com/bitcoin/bitcoin/blob/d67883d01e507dd22d1281f4a4860e79d6a46a47/src/chainparams.cpp#L121`). These were analysed using the DNSSEC visualiser provided by `https://dnsviz.net`, e.g. `https://dnsviz.net/d/seed.bitcoin.sipa.be/dnssec/` on that day, triggering a new analysis if existing data was older than 24 hours.

1. `seed.bitcoin.sipa.be` – Domain `sipa.be` does not have DS record, `.be` presents proof of non-existence.
2. `dnsseed.bluematt.me` – Delegations are all signed, warnings because the `.me` TLD appears to support RFC 7873 DNS cookies but does not announce that.
3. `dnsseed.bitcoin.dashjr.org` – Domain `dashjr.org` does not have DS record, `.org` provides proof of non-existence.
4. `seed.bitcoinstats.com` – analogous
5. `seed.bitcoin.jonasschnelli.ch` – analogous
6. `seed.btc.petertodd.org` – analogous
7. `seed.bitcoin.sprovoost.nl` – Domain `sprovoost.nl` itself is signed correctly, but the sub-domain is not delegated, domain provides proof of non-existence.
8. `dnsseed.emzy.de` – All delegations signed.
9. `seed.bitcoin.wiz.biz` – All delegations signed.

We see that only three of the DNS seeds ($\approx 33\%$) correctly deploy DNSSEC verification at the time of writing, with one of these producing a warning.

Appendix B. Python Script for **ADDR** mismatch

The following script computes the result of $\prod_{n=0}^{1\,000}\left(\frac{65\,535-n-1\,000}{65\,535-n}\right)$ and the respective expressions for one and two matches. Note that this ignores the possibility of addresses being in multiple buckets.

```
#!/usr/bin/python3 -u
from mpmath import mp
mp.dps = 100

POOL_SIZE=mp.mpf(65535) # size of new table
POS_COUNT=mp.mpf(1000) # addresses which are positive outcomes
ADDR_CHOICE=mp.mpf(1000)
# how many addresses we are able to obtain

def calc_prob(positives):
    result = 1
    next_pos = positives[0] if positives else None
    pos_left = POS_COUNT
    for ones_picked in range(0, ADDR_CHOICE):
        left_in_pool = POOL_SIZE - ones_picked
        is_pos = next_pos == ones_picked
        if is_pos:
            relevant_choices = pos_left
            pos_left -= 1
            positives = positives[1:]
            next_pos = positives[0] if positives else None
        else:
            relevant_choices = left_in_pool - pos_left
        result *= (relevant_choices / left_in_pool)
    return result

# zero - i.e. no positives
for_zero = calc_prob([])
print(f’no matches: {for_zero} - {for_zero:.10f}’)

# one - i.e. one positive, possible at each location
for_one = 0
for pos_idx in range(0, ADDR_CHOICE):
    for_one += calc_prob([pos_idx])
print(f’one match: {for_one} - {for_one:.10f}’)
```

The amount of positive outcomes (`POS_COUNT`) may be adjusted to compute similar expressions. Also note that these results are to be considered approximations, because floating-point arithmetic is used.

Output for POS_COUNT = 1 000:

```
no matches: 1.86288802040664e-07 - 0.0000001863
one match: 2.9320196745256213e-06 - 0.0000029320
```

Output for POS_COUNT = 16 384:

```
no matches: 8.757585111830943e-127 - 0.0000000000
one match: 2.979819622699693e-124 - 0.0000000000
```

Output for POS_COUNT = 64:

```
no matches: 0.3735954067699298 - 0.3735954068
one match: 0.3708603119691502 - 0.3708603120
```

Output for POS_COUNT = 50:

```
no matches: 0.46341989...
one match:  0.35931822...
```

Appendix C. Negative Impact on Fingerprinting of Address Caching

Further analysis is necessary to see how the address caching introduced in Section 3.5.1 impacts fingerprinting in practice. We assume a full new-address table with up to 65 535[1] peers. The 1 000 addresses returned by some peer are stored, and then 1 000 more addresses are collected from another remote. For a sibling pair, the probability that the first new address is shared is given by $\frac{1\,000}{65\,535} \approx 1.53\%$. Repeating this, the probability of not a single address matching is given by $\prod_{n=0}^{1\,000} \left(\frac{65\,535-n-1\,000}{65\,535-n} \right)$, which evaluates to $\approx 1.86 * 10^{-7}$ (see Appendix B).

This means that it is not unlikely that some addresses are still shared. For a realistic evaluation, we want to know how many addresses we can expect to be shared, i.e. the expected value of this probability distribution. The order of magnitude exhibited by the result at hand suggests that the first few values will not yield the expected value. However, with the amount of matches, the number of possible combinations increases significantly. This means that the script from Appendix B cannot be easily adapted, since it will take a long time to compute an exact result. An experiment showed around six minutes for even two matches.

However, we notice that $\frac{1\,000}{65\,535} \approx \frac{1\,000}{64\,435}$ (the match probability for the last element in the zero case), so for low values of expected matches, a reasonable approximation can be made assuming that the probability for each trial to match is constant. This yields a binomial distribution $\mathcal{B}(n = 1\,000, p = \frac{1\,000}{65\,535})$ with expected value $n * p = 15$. From the probability distribution, we also observe that after around 40, the values become negligible, i.e. these higher counts do not invalidate our approximation because their probability is so low. As such, we can still expect some addresses to match. However, looking at the proposed method, it seems much more unlikely than before that out of these, one will violate the constraint.

[1]https://github.com/bitcoin/bitcoin/blob/v0.21.0rc2/src/addrman.h#L159-L161 – 2^{10} buckets with at most 2^{6} addresses each $= 2^{16} = 65\,535$ maximum new table size