Geolocation of Bitcoin mining

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science

by

Chris Hoorenman, MSc

ing. Henk-Jan Hopman

Supervisor

dr. ir. H.P.E. Vranken

Open Universiteit

Management, Science and Technology Faculty

16th June 2019

Abstract

The vast amount of computing power needed to mine blocks on the Bitcoin blockchain leads to a large demand for energy to power the mining hardware. Recent research on this topic shows that Bitcoin mining currently has the same annual energy requirements as a small country. Only limited academic research exists on the geographical distribution of mining activity and this thesis aims to provide the Bitcoin research field with more tools to determine the location of mining activity. This information can be valuable for various academic disciplines to create a better understanding of the environmental, economic and social impact of Bitcoin mining. In this thesis we first provide a thorough analysis of current public knowledge of mining locations, both from existing surveys and from our own Internet-based research. We then develop a platform that collects information related to Bitcoin mining activity by building a crawler for the Bitcoin peer-to-peer network based on the Bitnodes open source crawler. We rework the crawler's original Bitcoin communication protocol from Python 2 to Python 3 and improve upon it. Our crawler collects IP addresses as well as other relevant data from both the reachable and unreachable nodes in the Bitcoin network whereas most other research only focuses on the reachable nodes. We subsequently use IP geolocation to map the IP addresses to regions on a world map. To improve the crawling results, we develop a mining index score for every node based on a number of criteria such as the node's services flag and its connection to fast relay networks. Furthermore we make novel use of the mark-and-recapture statistical technique to determine a node's total peer count. The mining index score is used to filter the crawling results and to increase the likelihood of finding mining activity in the remaining results. We compare the unfiltered and filtered results and show where differences occur. To add additional information to the set of crawling results outside of the Bitcoin network we integrate automated API queries into the platform to retrieve information from the two main OSINT platforms Shodan.io and Censys.io. The queries use a variety of search strings related to Bitcoin mining hardware and protocols. Finally, we collect information on mining pool gateways and perform a DNS-lookup to retrieve IP addresses of the various regional pool servers. We again plot the results on a world map and compare them to the results from the crawling process. We conclude that the combination of sources yields a realistic view on the global geographical distribution of Bitcoin mining.

Contents

Abstract **i**

1 Introduction **1**
1.1 Context . . . 1
1.2 Goal and research questions . . . 1
1.3 Methodology . . . 2
1.4 Document outline and structuring . . . 3

2 Background **4**
2.1 The Bitcoin blockchain . . . 4
2.2 Bitcoin Mining . . . 4
2.3 The Bitcoin network . . . 7
2.3.1 The Bitcoin Peer-to-Peer Network . . . 7
2.3.2 Custom networks . . . 8
2.4 Anonymity and tracing of miners . . . 10
2.5 Existing academic literature . . . 10
2.5.1 Bitcoin network and protocol . . . 10
2.5.2 Anonymity on the Bitcoin blockchain . . . 11
2.5.3 OSINT techniques to search for Bitcoin users . . . 13

3 Location of large scale mining operations **14**
3.1 Introduction . . . 14
3.2 Mining economics . . . 15
3.3 The Cryptoasset Benchmarking Report . . . 16
3.4 Review and analysis of other public sources . . . 18
3.5 Results and evaluation . . . 19

4 Building a Bitcoin geolocation application **22**
4.1 Introduction . . . 22
4.2 Setting up the platform . . . 22
4.2.1 Main objective and requirements . . . 22
4.2.2 Open University server environment . . . 23
4.2.3 Containers . . . 23
4.2.4 Bitcoin Core . . . 25
4.2.5 Access to the Internet . . . 26
4.2.6 Initial platform setup . . . 28
4.3 Building a node crawler . . . 29
4.3.1 Main objective and requirements . . . 29
4.3.2 Summary of methodology . . . 31
4.3.3 Use of existing code: Bitnodes messaging module protocol.py . . . 32
4.3.4 Update to Python 3: Changes to protocol.py . . . 33
4.3.5 Crawling algorithms . . . 35

4.4 Store, analyze and visualize the crawled data . . . 38
4.4.1 Main objective and requirements . . . 38
4.4.2 Storage and visualization of the data . . . 39
4.4.3 Geo Location . . . 41
4.5 Bringing it together: BGLF . . . 41
4.6 Testing . . . 43
4.7 Revised platform setup . . . 45
4.8 Discussion of results and validation . . . 46

5 Miner discovery by building a mining index 52
5.1 Introduction . . . 52
5.2 Mining activity estimators . . . 52
5.3 Creating a Miner Index . . . 54
5.3.1 Services Flag (SF) . . . 54
5.3.2 Fast relay networks (FR) . . . 56
5.3.3 Last seen in network (LS) . . . 58
5.3.4 Node notoriety (NN) . . . 59
5.3.5 Amount of peers in address list (PE) . . . 60
5.3.6 Mining Activity Index . . . 62
5.4 Extending the BGLF application . . . 63
5.5 Discussion of results and validation . . . 66

6 Miner Discovery outside the Bitcoin P2P network 71
6.1 Introduction . . . 71
6.2 OSINT . . . 71
6.2.1 OSINT platforms . . . 71
6.2.2 OSINT information gathering . . . 72
6.2.3 OSINT search strings and results . . . 75
6.2.4 Extending the BGLF application . . . 77
6.2.5 Bitnodes API data retrieval . . . 80
6.3 Mining Pool Gateway Extrapolation . . . 81
6.4 Discussion of results and Validation . . . 83

7 Conclusions, ethical considerations and future research 87
7.1 Overview . . . 87
7.1.1 Revisiting the research questions . . . 87
7.1.2 Combined results . . . 89
7.2 Contribution to the academic field . . . 91
7.3 Limitations to our research . . . 92
7.4 Ethical considerations . . . 94
7.5 Future research . . . 97

8 Reflections 99
8.1 Introduction . . . 99
8.2 Evaluation on changes in the thesis setup . . . 99
8.2.1 Preparatory course . . . 99
8.2.2 Reflections on final (AF) and preparatory (VAF) document . . . 101
8.3 Division of time and workload . . . 103

8.3.1 Introduction . . . 103
8.3.2 Core activities by Chris Hoorenman . . . 104
8.3.3 Core activities by Henk-Jan Hopman . . . 105
8.3.4 Estimation of time spent per activity (AF + VAF) . . . 107
8.4 Personal assessments . . . 108
8.4.1 Introduction . . . 108
8.4.2 Assessment Henk-Jan . . . 108
8.4.3 Assessment Chris . . . 109
8.5 Final words . . . 110

A Global mining activity **112**

B Development **115**
B.1 Postman . . . 115
B.2 Integrated Development Environments . . . 116

C Platform documentation **117**
C.1 Docker . . . 117
C.1.1 Docker-compose configuration file . . . 117
C.1.2 Docker-compose environment file . . . 125
C.1.3 Docker daemon configuration file . . . 125
C.2 OU Server . . . 126
C.3 Elasticsearch . . . 127
C.3.1 Mapping Node Information . . . 127
C.3.2 Mapping Node and Peer relation . . . 131
C.3.3 Mapping Bitnodes Information . . . 137
C.3.4 Mapping Shodan Information . . . 140
C.3.5 Mapping Censys Information . . . 142
C.3.6 Mapping Mining Index . . . 145

D BGLF documentation **148**
D.1 Program structure . . . 148
D.2 UML diagram . . . 149
D.2.1 Command line interface . . . 150
D.2.1.1 The *crawl* menu . . . 151
D.2.1.2 The *scrape* menu . . . 151
D.2.1.3 The *collect* menu . . . 152
D.2.1.4 The *calculate* menu . . . 152
D.2.2 The *analyz* menu . . . 153
D.2.3 The *dashboard* menu . . . 154
D.3 BGLF: the settings file . . . 155
D.4 BGLF - Debug logging example . . . 161
D.4.1 Shodan example of a search . . . 163

E OSINT **165**
E.1 Shodan . . . 165
E.2 Censys . . . 167

E.3 Bitnodes 169

F Mining Pools **170**
F.1 Mining Pools Information 170
F.2 Maps 172
F.2.1 BTC.com 172
F.2.2 AntPool 172
F.2.3 F2Pool 173
F.2.4 Poolin 173
F.2.5 SlushPool 174
F.2.6 ViaBTC 175
F.2.7 DPool 175
F.2.8 Huobi.pool 176
F.2.9 Bitcoin.com 177
F.2.10 Bixin 177

Bibliography **180**

Online Sources **184**

Sources on the location of Bitcoin mining activity **189**

1. Introduction

1.1 Context

State-backed currencies such as the dollar and euro are issued by central banks that can print new units of money whenever they want. The issuance of Bitcoin however is coded into the software that powers the Bitcoin blockchain. Bitcoins are rewarded every ten minutes to people or organisations that use computer processing power to solve complex mathematical puzzles. These mathematical puzzles are meant to validate and secure the transactions on the Bitcoin blockchain. This process is called Bitcoin mining and nowadays it is mostly performed by specialized computers that use Application-Specific Integrated Circuit (ASIC) chips that are optimised to solve the mathematical Bitcoin puzzles very efficiently.

Bitcoin mining requires cheap electricity to be profitable. Whether this means that operations are therefore more likely to be located close to a hydro dam with over-capacity or next to an old polluting coal plant will differ per region and operator. While there are specialized organizations that can give insight in the energy mix of a certain region, more information is required on the geographical location of mining Bitcoin activity.

Our research mainly covers technological aspects of the Bitcoin blockchain. However, the implications of the development and adoption of cryptocurrencies reach much further than the just the technological domain. A comprehensive overview by the International Monetary Fund (IMF) that focuses more on the social and economical aspects of cryptocurrencies and how they relate to fiat currencies can be found in [1].

1.2 Goal and research questions

The main goal of this project is to develop an application to determine the geographic location of mining activity on the Bitcoin network.

In this thesis we try to answer the following main research question:

- How can Bitcoin mining activity be identified, quantified and located?

We will divide this into several subtopics:

- What information on Bitcoin mining activity and locations is readily available from public sources?
- Can mining activity be identified on the Bitcoin network and filtered from the large amount of other information?
- What other sources are available that can help identify, locate and quantify Bitcoin mining activity?

The goal of this project has economic and social validity when looking at the concerns that have arisen in the past years on the amount of power that is being used to operate the Bitcoin network. Our results can be used by researchers in other fields, for instance to draw conclusions on the environmental impact of Bitcoin mining by relating the geographic location of miners to the energy mix in that area.

1.3 Methodology

In order to answer our first research question we try to determine how much information on the location of Bitcoin mining activity is already available in the public domain through various sources. Results are visually plotted on a world map.

Our second research question focuses on the possibility to extract information on mining activity from the Bitcoin network itself. To be able to provide answers to this question we first build a platform to start our operations from. We make sure that this platform is flexible and able to integrate additional modules to allow for research on a wider variety of topics in the future. From this platform we run a crawler tool that finds IP addresses of nodes in the Bitcoin network and retrieves information from them. The information is stored and analysed through integration with the Elasticsearch database software and the Kibana analytics and visualisation software, which are both open source. Initials results are again plotted on a world map.

To provide better answers to the second research question, we improve our results by extracting additional information from the Bitcoin network nodes and by aggregating this information into an index. By ranking the results, we try to determine which nodes have a greater likelihood to be involved in mining activity. Results are again plotted onto a world map.

Finally, we try to answer the third research question. We do this by using two distinct methodologies. First, we integrate a tool into our platform that can query the API of both Shodan.io and Censys.io, two large search engines for connected

devices. The IP addresses that we find all run web services that are used in relation to mining equipment or software. The resulting geolocations are plotted and checked against the existing information we have. Second, we retrieve IP addresses of Bitcoin mining pool gateways by identifying their host servers and performing a DNS-lookup.

1.4 Document outline and structuring

The chapters in this document are designed to allow for a clear set of milestones and proper validation of results.

The document is built up as follows:

Chapter 2 provides the necessary context for the general reader on blockchains and cryptocurrencies, the Bitcoin peer-to-peer network and the mining process. In particular we provide detailed insight into the Bitcoin communication protocol which is used extensively in our software platform. The chapter finishes with a summary of related academic work and how this applies to our project.

In Chapter 3 we document and present the results of our analysis of public sources on Bitcoin mining activity.

In Chapter 4 we document the development of our platform and the Bitcoin network crawler and we present initial findings.

Chapter 5 builds upon the results from Chapter 4. We create and document the Mining Activity Index and present final findings from the crawling process.

In Chapter 6 we detail the setup of our API querying tool and the mining pool gateway DNS-lookup and we show results.

In Chapter 7 we summarize our findings. We revisit the research questions and answer them in a comprehensive manner. We also reflect on the ethical considerations of our work. At the end of this chapter we propose various additional research topics that were outside of the scope and time frame of this thesis. We explain the rationale for this research and provide some direction on how this research could be structured and added to our research platform.

Finally, Chapter 8 holds a reflection on our work. We describe the difficulties we encountered and we give a breakdown of the time spent on the various research topics. Furthermore we discuss our experiences and observations working as a team and reflect on the individual contributions.

2. Background

2.1 The Bitcoin blockchain

The Bitcoin blockchain is a digital ledger in which transactions are recorded chronologically and publicly. Transactions are aggregated into blocks that are added to the pool of existing blocks with a link to the previous block, hence the name blockchain. A new block is validated approximately every ten minutes. Validation happens through a transparent and unfalsifiable mechanism which is referred to as 'mining'. Public blockchains are typically distributed and decentralized. This means that no single party is in control of the blockchain, that the blockchain is stored and updated in many different locations and that verification and processing of transactions is also done in different locations and by multiple parties.

Transactions are done in a digital currency ('cryptocurrency') called Bitcoin. Bitcoin is at the moment of writing the best known blockchain and the largest cryptocurrency by market cap [34]. The principle of the Bitcoin blockchain was first suggested in a 2008 paper by Satoshi Nakamoto [2], a presumed pseudonym. The Bitcoin protocol has been improved over time with the help of many developers. A great number of other cryptocurrency blockchains have been created as well since 2009, mostly following the same fundamental concepts but often different in their technical design and validation mechanisms. Notable examples are the Ethereum, Ripple and Litecoin blockchains but hundreds of others exist [34].

2.2 Bitcoin Mining

Bitcoin mining involves the bundling and verification of transactions in a block and the deployment of processing power to try to validate this block and to add it on top of the Bitcoin blockchain. Miners (i.e. individuals or companies deploying sophisticated hardware) compete to complete a complex mathematical problem that takes a very high amount of computational processing to solve, but for which the solution is easy to verify. This technique is called Proof of Work.

Proof of Work can be implemented in a variety of ways but Bitcoin uses a cryptography technique called double SHA-256 (Secure Hash Algorithm) as its mining algorithm [35]. This algorithm takes an arbitrarily long amount of data and changes it into a 256-bit number which is referred to as a 'hash'. The challenge for the miner is to find a hash that satisfies a specific difficulty condition. This condition

states that the hash has to have a minimum number of leading zeroes. A valid hash can only be found by trying random input numbers (reffered to as 'nonce') for the SHA-256 process. Therefore the more computing power a miner has, the higher the chance of finding a valid hash. The miner that generates a valid hash first, solves the block and receives a number of Bitcoins, at the moment of writing this amount is 12.5 BTC [36].

This process leads to a race between miners to find a valid hash first. The difficulty of the condition changes over time to keep pace with the total amount of network hashing power to ensure that the block interval remains approximately ten minutes. The vast amount of computing power that is needed to mine blocks has led to a large demand for energy to power the hardware. Recent research on this topic shows that cryptocurrency mining currently may have the same annual energy requirements as a small country [3].

The demand for efficient mining equipment has spurred the use of specialised hardware called ASIC (Application-Specific Integrated Circuit) which provides the highest amount of processing power ('hashing power') per kilowatt compared to other and older mining alternatives like the CPU, GPU (video card processors) and FPGAs (Field Programmable Gate Arrays that are programmed to perform mining calculations). Due to the increasing difficulty based on total network hashing power, the decrease in block rewards every 4 years and the decrease in Bitcoin prices since its hight point in December 2017, ASIC-based mining has become the only profitable alternative.

Next to dedicated hardware, miners need access to the Bitcoin network to mine Bitcoin blocks. This can be achieved by installing software called Bitcoin Core [37, 38] which will setup and run a Bitcoin network node. Bitcoin Core validates transactions and blocks and will handle communication with other nodes in the network. Using the full functionality of Bitcoin Core requires the whole blockchain to be downloaded which at the moment of writing is approximately 220 gigabytes [39]. In addition to this, Bitcoin mining software is required to instruct the ASIC mining hardware to perform the hashing of the block information. This software effectively links Bitcoin Core to the hardware. Bitcoin Core uses port 8333 [40].

Without a very large amount of hashing power and state of the art equipment, the mining of Bitcoins can be a risky and volatile endeavour resulting in erratic and low income. To counter this, individuals or companies deploying small mining operations can mine collectively through so-called mining pools. These pools combine the processing or hashing power from a large number of miners over a network. The individual miners and mining equipment can be located in many different places in the world but they all contribute to solving the same hash. Participating miners share pro rata in the block rewards and sometimes in the transaction fees. Sufficiently

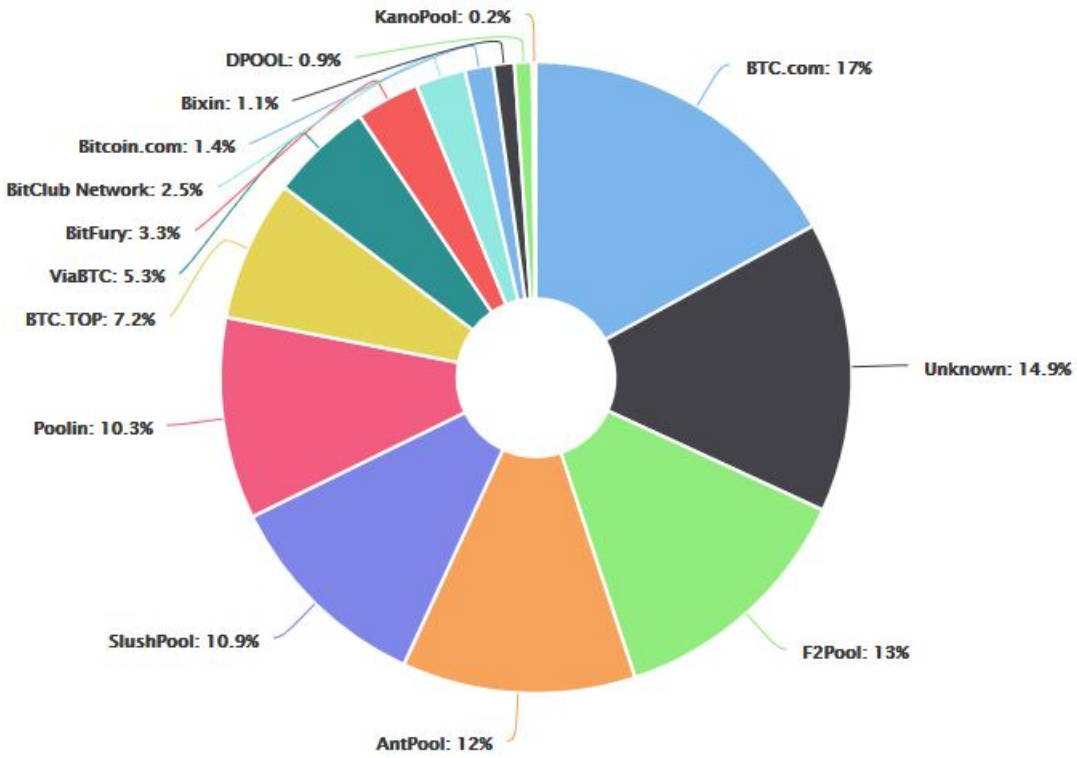


Figure 2.1: An estimation of the Bitcoin hashrate distribution.
Source: https://www.blockchain.info. 12 May 2019

large pools therefore generate a stable income over time for the miners. Miners pay a small percentage of their income in fees, typically 1-2%. The overwhelming majority of Bitcoin mining nowadays is done through pools albeit that some pools are proprietary and run by a single entity (see Figure 2.1).

There is no need for the miner to run a full Bitcoin node or install the entire blockchain when using a mining pool. Communication is handled by the pooling server through a dedicated protocol. The most widely used protocol is Stratum [4] [41] [42] which replaces the `getwork` protocol of Bitcoin core. This is a remote procedure call (RPC) that enables miners to receive jobs from mining pool servers. Individual miners solve these assigned jobs and send their results back. Because the communication occurs outside the Bitcoin blockchain and its protocol, it hard to determine who does the mining in a mining pool and from where. For a typical mining pool setup, see Figure 2.2.

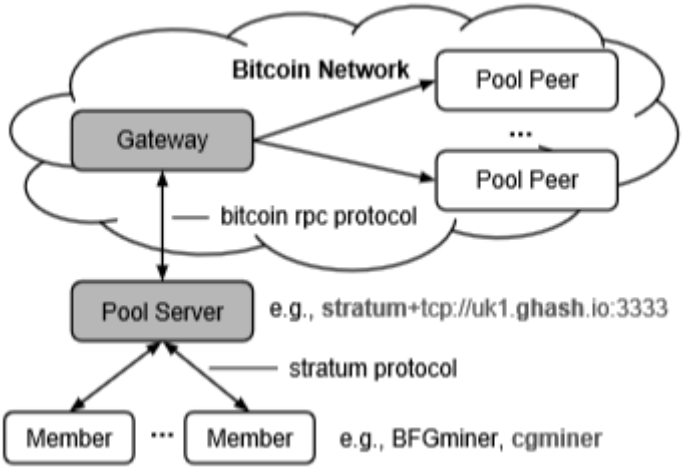


Figure 2.2: A typical mining pool architecture. Source: Miller et al. [5]

2.3 The Bitcoin network

2.3.1 The Bitcoin Peer-to-Peer Network

The Bitcoin Core protocol uses a broadcast network to send transactions and blocks [2]. Communication is done through the TCP protocol. The following main messages are available [43]:

- *version*: Information about program version and block count.
- *verack*: An acknowledgement in response to the *version* message
- *getaddr*: Requests an *addr* message.
- *addr*: Sends a list of peers with IP address and port. The list holds 23% of known addresses by the node with a maximum of 2500. However, the message field only allows for 1000 addresses to be included.
- *getblocks*: Sends the latest known block hash of the node and requests an inv message.
- *inv*: Sends a list of one or more blocks. This occurs when a new block is found or in response to a *getblocks* message
- *getdata*: Request for a single block or transaction.
- *tx*: Sends a transaction in response to a *getdata* request.
- *block*: Sends a block in response to a *getdata* request.
- *ping*: Sent to confirm that the TCP/IP connection is still valid.

- *pong*: Response to the *ping* message

To initiate a connection, a node sends a *version* message to a peer with its version of the Bitcoin Core software, number of blocks and a timestamp. The peer will send back a *verack* message and its own *version* message to which the node responds with a *verack* message in return. The node and the peer then exchange *getaddr* and *addr* messages in order to exchange and store unknown other peers.

After a connection is established, the node sends a *getblocks* message with its newest block hash. A peer may respond with an *inv* message if it sees that it has newer blocks. The node will ask for these blocks with *getdata* messages and the peer will reply with *block* messages.

To send a transaction, a node sends an *inv* message to all known peers. The peers will respond with a *getdata* message if they do not already have the transaction. If the transaction is valid, the peers will propagate the transaction to their peers with an *inv* message. Transactions are hashed into a block by miners and when a valid block is found, it is sent with an *inv* message to peers, similar to transactions.

All nodes broadcast an *addr* message every 24 hours. Addresses are stored when they are not known and a small number is relayed to other peers. This way, the entire network has a fairly good overview of all active nodes [6].

The Bitcoin Core reference implementation establishes eight outbound connections and defines a maximum of 125 connections in total (i.e. 117 inbound connections) but customized versions can handle more connections [5, 7]. Peers in the network that are discovered are stored in a local address database with their IP address, port number and a timestamp of the last connection or when the peer was last seen in the network. Upon receiving a *getaddr* request, a node will randomly select 23% of its address list, up to a maximum of 2500 addresses [8, 9, 10]. However, when sending the addresses to the peer that requests the addresses, the lists is further limited to a maximum of 1000 addresses due to buffer limits [11]. A more detailed description of the fairly complex storage and cleanup algorithms of the reference Bitcoin Core client can be found in Deshpande et al. [7]. The actual address management code can be found in the *addrman.cpp* module of the open source Bitcoin Core code on Github [44].

2.3.2 Custom networks

The standard method of communication works well but is considered relatively slow. Miners and mining pools run the risk of mining on top of an old block while a new block has not been relayed yet. That is a waste of resources. A number of faster networks exist that can alleviate this problem.

- Fast Relay Network [45]

  The Fast Relay Network is a custom network devised and managed by Matt Corallo, an active Bitcoin Core contributor. He decided to set up the network because he saw miners moving to the larger pools because smaller pools started to experience high orphaning [46], i.e. smaller pools were fast enough to mine new blocks but not fast enough in propagating them throughout the network. The relay network holds nine relay nodes that are relatively evenly distributed throughout the world. Miners or pools connect to the node that is closest to their location to send and receive blocks. The main drawbacks of the network are that it is centralized and owned by one person. The network is now largely unmaintained and replaced by the FIBRE Network.

- FIBRE [47, 48]

  FIBRE stands for Fast Internet Bitcoin Relay Engine. This network is also developed by Matt Corallo and sends packets over UDP (User Datagram Protocol) instead of TCP, using Forward Error Correction (FEC) to correct for potential data loss without additional communication. FIBRE is designed to work with Compact Blocks [49], a Bitcoin protocol improvement which is a way of minimizing relayed data by only sending compact hashes of transaction data, leaving it for the nodes to find out which transactions were included in the block. FIBRE works as an addition to Bitcoin Core, making it more decentralized than the Fast Relay Network.

- Falcon Relay Network [50]

  Blocks that are transmitted through the normal Bitcoin network are sent in different packets. Only when a node has received all packets and reconstructed the block, will it relay the block to other peers. In order to speed up this process, researchers from Cornell University have developed the Falcon Relay Network. The packets that are sent through this network are relayed immediately to peers without waiting for the remaining packets. They call this technique 'cut-through routing'. The node only checks the block header and reconstructs the block at the same time as it is relaying the individual packets. Obviously, the main drawback is that validation of the block only happens after many packets are transmitted throughout the network, leading to inefficient use of resources when packets are invalid. Cornell University itself operates as a check against a malicious sender of invalid packets, introducing centralization.

2.4 Anonymity and tracing of miners

Because (solo) miners need to run a Bitcoin full node and send validated blocks to the network, they expose their IP address provided they are not behind a Tor, firewall or NAT [12] [13] [14]. Not every Bitcoin full node is a miner but the node that first sends a validated block will be the node run by the miner that has validated that block. The first transaction in the validated block will be the transaction of 12.5 Bitcoin to the miner, which also exposes the Bitcoin address of the miner [15].

Pooled miners could be anywhere [5] [15]. Many pools accept miners from all over the world, using regional gateways to reduce latency. The pool has to run a Bitcoin full node or multiple full nodes somewhere to receive transactions, to allow for verification and to communicate a validated block to the network. This exposes the IP address of the pool node but not the IP address of the individual miners as they only communicate to the pool through client software. Therefore, the location of the node's IP address is most likely not the location where the actual hashing is done [16]. Sometimes the location of individual contributors to the pool can be found by investigating transactions made to and from the Bitcoin address of the mining pool [10].

Additional information about miners can be found by using OSINT (Open Source Intelligence) techniques i.e. crawling the internet and scanning for an open port 8333 or for common mining or mining pool related services running on other ports [5]. The use of the mining software Stratum is one example. The information and IP addresses that can be gathered in this way can be used to determine the location of miners but not the amount of hashing power that these miners have [15].

2.5 Existing academic literature

2.5.1 Bitcoin network and protocol

A number of publications discuss the Bitcoin network, its protocol and its topology in more general terms which makes for excellent introductory reading. Among these is the original paper on the Bitcoin network of developer Satoshi Nakamoto [2].

A recent paper by Delgado-Segura et al. [6] gives an in-dept description of the Bitcoin network and compares it with more standard P2P networks. The paper goes into detail on the different attacks that are possible on the network and their effects as well as the response by the network, such as a 24-hour ban for suspected malicious nodes. The authors point out that some well-known problems of P2P networks are not a concern for the Bitcoin network while other problems pose new

challenges. Therefore, they declare the Bitcoin network a new paradigm in P2P networking.

Miller et al. [5] look at the network topology of the Bitcoin blockchain. They use a custom technique called AddressProbe to discover peer-to-peer links which can map the connectable Bitcoin nodes within minutes. It can also find non-connectable nodes (behind a VPN / firewall) over time. However, the node infrastructure that is used to generate a map of the network needs to be relatively long-running for sufficient connections to establish themselves. The authors use their mapping technique to determine whether nodes exist in the network that have an unfair broadcast advantage. They do find nodes that have far more than 125 connections, these they call 'influential nodes'. They are able to map some influential nodes to known mining pools or suspected mining pools. Separate from these influential nodes, they show that the network graph is not a random graph by comparing it to a simulated graph. The reasons for this they leave to further research. Next to their original research this paper also holds an excellent introduction to the Bitcoin protocol in general and the role of miners and mining pools.

In a paper presented at the IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN, September 2018), Deshpande et al. [7] develop a framework to map the Bitcoin network topology. They call this framework BTCmap. The authors use the API from the Bitnodes project [51] to request all reachable IPv4 Bitcoin nodes, leaving out IPv6 nodes and nodes behind TOR. They then send *getaddr* messages to all the reachable nodes to which the nodes respond with *addr* messages, each containing up to 1000 peers. Because the peer database can hold up to 81920 entries, they make a *getaddr* request multiple times. The authors note that after 160 seconds, the probability to receive a response from a node is statistically negligible. Furthermore, they note that sometimes a connection is rejected due to saturation of inbound connection quota. For this reason, and because of the fact that the addresses that are received in an *addr* message are randomly selected by the peer, the authors use several iterations to request addresses. They find that after the 6th iteration, the cumulative amount of responding nodes starts to stabilize. The paper holds an in-depth analysis and description of the stochastic peer address management algorithms of the Bitcoin Core software.

2.5.2 Anonymity on the Bitcoin blockchain

Several academic papers and studies focus on the perceived anonymity of the Bitcoin blockchain and the ways in which someone could infer relationships between transactions, Bitcoin addresses and IP addresses.

A 2014 paper by Biryukov et al. [10] has a good introduction to the Bitcoin network and the details and protocol of the address and transaction propagation. The authors show how to deanonymize Bitcoin users behind Tor (and by extension other anonymity services) by exploiting the Bitcoin built-in DoS protection. This protection mechanism bans the IP of a node based on a a penalty score for false messages. The authors connect to a target node through as many Tor nodes as possible and send a message that results in an instant ban. By doing this for all Bitcoin nodes, all Tor connections are prohibited for 24 hours, potentially forcing miners to expose themselves. In a next step, the authors show how nodes that do not accept incoming connections can be identified by inferring their entry nodes from address propagation messaging. For this to work, a connection to almost every Bitcoin node must be maintained.

Powered with these tools, a deanonymization attack is conducted on the network by first mapping all peer servers by iteratively sending *getaddr* messages to known peers and storing new results. A list of IP addresses is then mapped to a set of likely entry nodes. At the same time, new transactions in the network are mapped to likely entry nodes by listening to *inv* messages in real time. In a final step transactions can be mapped to IP addresses by comparing their set of entry nodes. Just two to three similar entry nodes already give a high probability of a correct mapping. This is an inventive manner of deanonimizing Bitcoin users and miners but it is also rather intrusive. The research has therefore mainly been performed on the Bitcoin test network.

Juhász et al. [17] monitor the propagation of messages by the Bitcoin client program by installing more than a hundred Bitcoin clients to listen to as many messages as possible. They then use Bayesian statistics to determine the likely originator of the transaction message, which is assumed to be the owner of the Bitcoin address in the transaction.

Spagnuolo et al. [18] build a framework that parses the Bitcoin blockchain and then clusters Bitcoin addresses that have a high likelihood of belonging together. The authors subsequently try to attach identities to the Bitcoin addresses based on data scraped from websites. A few real world use case are explained in the paper, some of which are a combination of automated labeling and manual analysis.

2.5.3 OSINT techniques to search for Bitcoin users

Not much academic work is available on the topic of OSINT techniques in general and OSINT related to Bitcoin in specific.

Sari et al. [19] show some of the potential of OSINT by focusing on the Antminer S9, one of the most efficient ASIC-based miners for Bitcoin. A proper search requires some select keywords (or 'dorks' as they are colloquially called). In this case the authors use the search string *(antminer) AND protocols.raw:"80/http" AND 80.http.get.title:"401"* to collect IP addresses of Antminer users through OSINT search engines such as censys.io or shodan.io. The paper then continues to explain how attackers could use this information to their benefit to break into these systems once they have found the IP address and ports.

3. Location of large scale mining operations

3.1 Introduction

The chart in paragraph 2.2 shows that the majority of Bitcoin mining is done through pools. Some of these mining pools may consist of aggregated small mining activity throughout the world run by private individuals deploying one or more ASIC miners or lower quality hardware to augment their income or just as a hobby. Other mining pools are used by corporations that run large-scale mining operations, deploying thousands of ASIC miners in industrial complexes. Some pools are a combination of the two.

Before we start with an in-depth analysis in the next chapters on the Bitcoin Network and the technical extraction of information, it is useful to research how much information on the location of Bitcoin mining is already in the public domain and how much of the network hashing power this represents. To do this, we will look at the largest known mining organisations and use public sources on the internet.

The development of ASIC miners is continuous and more efficient machines become available frequently. In the analysis in this chapter we have assumed that a single Bitcoin miner has an energy consumption of roughly $1350\,\mathrm{W}$, based on the power usage of arguably the best ASIC-based mining hardware in 2018, the Antminer S9 [52]. As an example, the Bitmain facility in Ordos, Mongolia that is reported to house 25.000 ASIC miners, is therefore expected to have a total energy requirement of 25000*1350 which is roughly $35\,\mathrm{MW}$.

Where gigawatt per hour is reported (Gwh), we have assumed this is for a full year unless otherwise indicated. Again as an example, the Bitfury facility in Mo i Rana in Norway is reported to have bought 375 Gwh from energy company Vattenfall. With 8760 hours in a year of 365 days, this equates to $43\,\mathrm{MW}$.

The Antminer S9 is reported to generate around 14 terahash (TH) per second which gives it an efficiency of roughly 10 gigahash per watt. Most other efficient ASIC miners are around this number. We will use this conversion when the capacity of a mining operation is reported in hashrate.

Latest developments and ASIC mining statistics are hard to find in one overview but a decent effort can be found at [53].

3.2 Mining economics

Profitability of mining stems from a number of factors that determine income and costs:

- Income:
  - Bitcoin reward per mined block
  - Price of Bitcoin
  - Hardware hash rate
  - Network hash rate and mining difficulty
- Costs:
  - Fixed hardware costs
  - Cost of housing and people
  - Energy costs
  - Energy consumption
  - Pool fees

Furthermore mining organizations have a preference for regions that have a friendly regulatory environment, a stable political situation, good internet connectivity and a cool climate [20].

There is not much that mining corporations can do to influence income except deploying more hardware. The reward for a mined block and the network hash rate change over time but cannot be influenced by a single party. The price of Bitcoin is subject to supply and demand. That means the only way to influence profitability is to look at the cost of operations. On the cost side, there are a number of factors that are dependent on the choice of location. These are the cost of housing and people, electricity costs and energy consumption.

The cost of housing and people is lowest in less developed countries. However, operations do not have to be in cities or economic hot-spots as long as there is a reliable supply of electricity. This means that the cost of housing may also not be very large in more developed countries. The benefit of more developed countries generally is the higher level of education and the clear rule of law.

Energy costs depend on the cost of generating power in the specific area. Regions with a high level of renewable energy that have more supply than demand will be able to provide cheap electricity as their marginal cost of generation is very low. However, the same may hold for power plants operating on fossil fuel that are

not running at full capacity. A large amount of studies are performed every year comparing the costs of electricity by source [54]. As renewable energy becomes cheaper, or fossil more expensive, more and more Bitcoin mining corporations may move to regions with renewable energy in high supply.

The use of ASIC mining hardware at full of near-full capacity generates a lot of heat which requires cooling. Regions with natural cooling (cold climate, cooling water) will lead to lower energy consumption and lower costs.

3.3 The Cryptoasset Benchmarking Report

The Cambridge Centre for Alternative Finance (CCAF) released a report in December 2018 [21], which is one of the few academic studies that tracks mining activity. This report was the second edition and expanded on the first issue early 2017 [20]. This study obtained data on the geographical locations of cryptocurrency mining by means of two surveys of mining pools between May and July 2018. The authors furthermore reviewed company websites, press releases, news articles, public forums, social media, and insights from industry experts. The study did not focus on Bitcoin but targeted cryptocurrencies in general.

The 2017 issue at the time reported worldwide mining activity as aggregated in Figure 3.1.

The study presents mining activity in megawatt (MW) which is also the metric that most operators and news organizations use in their reporting. The authors found a total of 232 MW of known mining activity. They estimated the required power for the total Bitcoin network at the time at over 460 MW and indicated that their findings represent roughly half of the Bitcoin network hashrate. This is questionable however as the mining activity may not be solely used for the Bitcoin network. Furthermore, the authors assume that every miner uses the most efficient equipment. This is rather unlikely although Bitcoin mining does become unprofitable quite quickly when using less efficient hardware.

More than one and a half year later in the 2018 study, the landscape has drastically changed. The study identifies a total of 1.7 gigawatt (GW) for 93 mining facilities, again for all cryptocurrencies. The data is partly based on 56 identified pools for both Ethereum and Bitcoin, 43 pools for Zcash, 38 pools for Monero and a number of others. For Bitcoin the report shows the geographical division for mining pools as shown in Figure 3.2. The share of renewable sources is also estimated as shown in Figure 3.3.

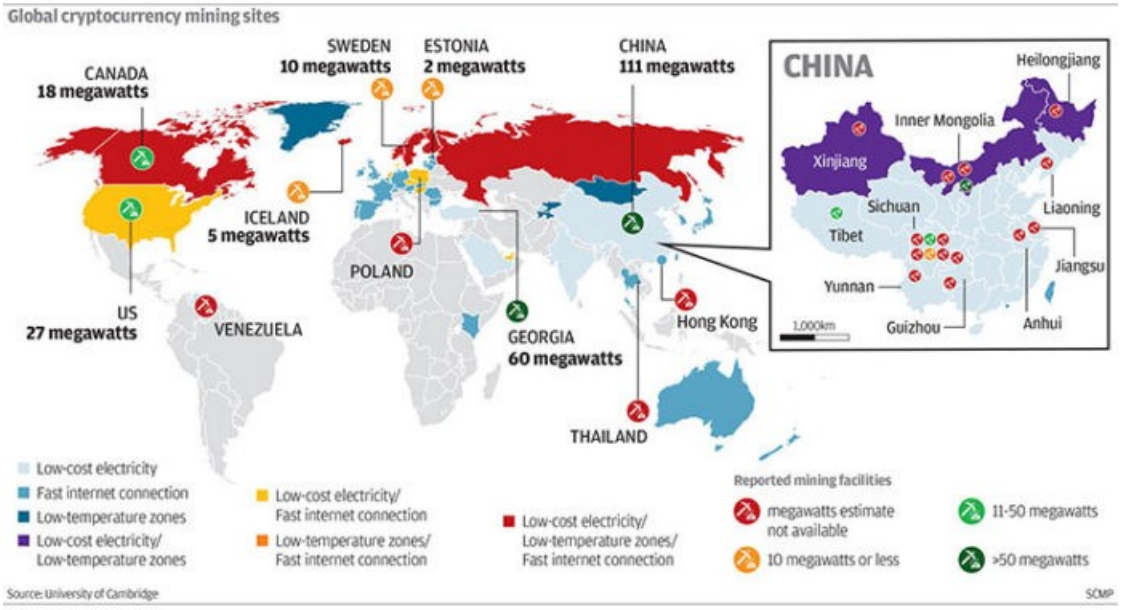


Figure 3.1: An estimation of the geographical distribution of large cryptocurrency mining operations in 2017.
Source: University of Cambridge

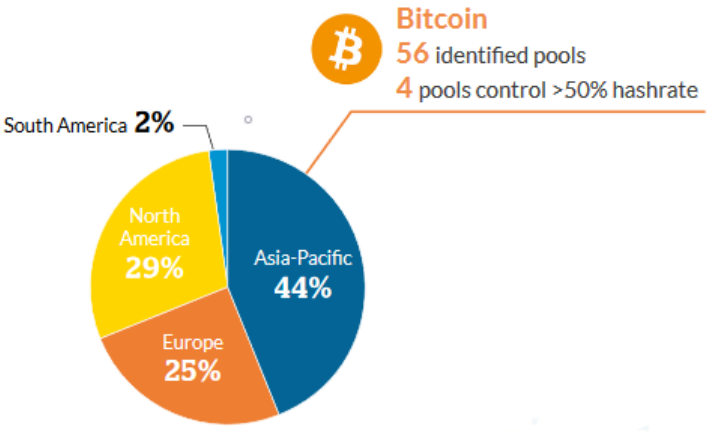


Figure 3.2: An estimation of the geographical distribution of 56 Bitcoin mining pools in 2018.
Source: University of Cambridge

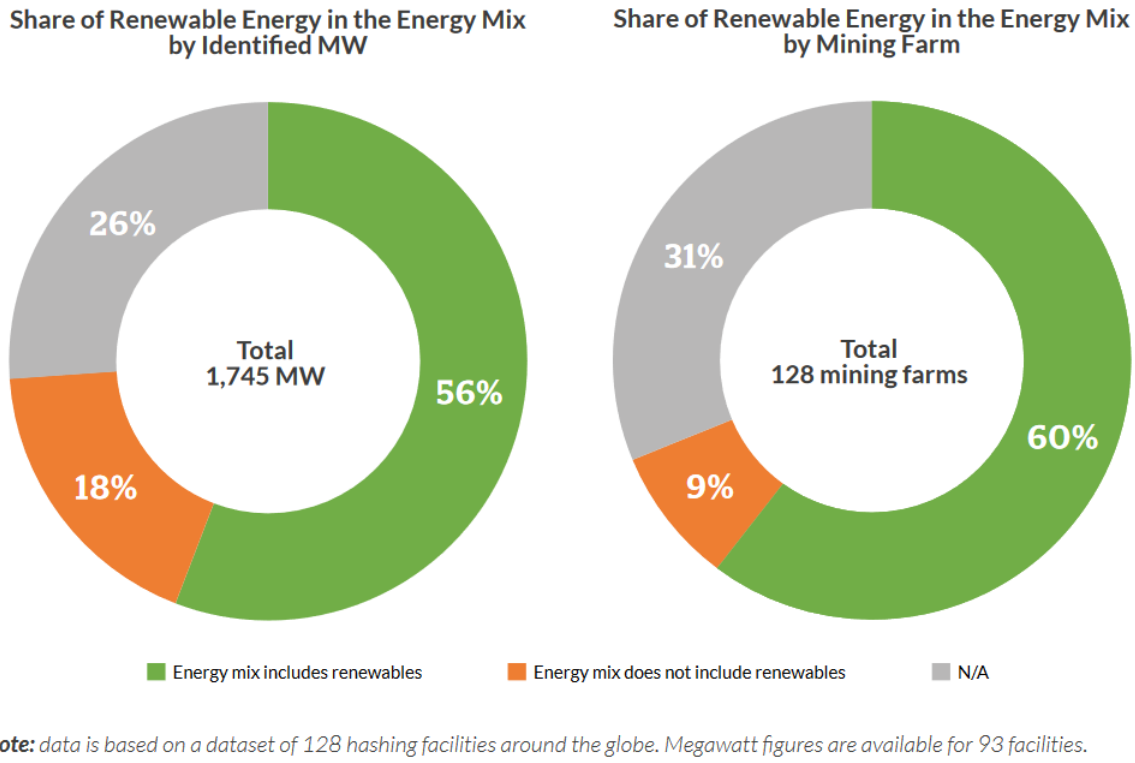


Figure 3.3: An estimation of the geographical distribution of 56 Bitcoin mining pools in 2018.
Source: University of Cambridge

3.4 Review and analysis of other public sources

To a certain extent it is logical to have a focus on the aggregate of all all cryptocurrencies because mining facilities do not always mine one single coin. It is therefore difficult to determine what part of a 20 MW facility is used for mining Bitcoin, Ether or other currencies. As our research focuses solely on Bitcoin, we have performed our own research into locations that are likely Bitcoin-only or have made assumptions regarding the share of Bitcoin mining in one facility. There is no inherent benefit of mining corporations to disclose their location and any analysis therefore runs the risk of relying on information that is difficult to prove (short of visiting the locations themselves which is not within the scope of this thesis). We therefore try to find multiple sources that report the same location. Furthermore, sources need to be recent as mining activity is not fixed to a specific location and operators can move to areas where mining economics are better. Incidentally operations will also just go bankrupt.

Next to searching the internet, a second reliable source for mining operations can be found through IPO (Initial Public Offering) documentation. A number of the larger mining operators are manufacturing companies of ASIC-based mining hardware that deploy some of their own hardware. Among these are well-known companies

such as Bitmain (Antminer), Canaan Creative (Avalon), Halong (Dragonmint) and Ebang (E-line). Some of these manufacturers have filed for an IPO to be listed on a stock exchange. This process requires a (draft) prospectus in which the company is explained in full detail, including the location of their main operations. This is a legal document that can be considered accurate and true. Some of the listing requests were pulled recently but we were able to retrieve the prospectus for Bitmain Technologies [99], Canaan Creative [100] and Ebang [101]. Only the prospectus for Bitmain held information on proprietary mining locations. This either means that Canaan and Ebang only produce mining hardware and do not mine themselves or that they have separated their mining activity into a different entity that is not in scope for the IPO.

The Bitmain prospectus was short on details per location but it disclosed that Bitmain operated 11 mining farms, mainly located in the Sichuan Province, Xinjiang and Inner Mongolia. The largest farm is in Inner Mongolia. As of June 30, 2018, 125.000 mining sets were hosted, with total potential capacity for about 200.000 mining sets. At the same date, the firm also had sites under construction in Wasjington State, Texas and Tennessee which were expected to become operational in the first quarter of 2019. We have not been able to verify the Bitmain development locations and have not included them in this overview. The firm was also planning to expand mining farm operations to Quebec, Canada where cheap hydropower is available. 125.000 mining rigs equates to about 170 MW. We have appointed 65 MW to Inner Mongolia in the knowledge that the largest facility out of the existing 11 is located there and there is also at least one other facility. To the Sichuan Province we have appointed the same number as most of the facilities seem to be located there. The remainder (25.000 mining sets) we have attributed to Xinjiang.

3.5 Results and evaluation

Appendix A holds a non-exhaustive list of all large scale Bitcoin mining locations that we have found through our research. It is based on publicly available data from at least three sources published no earlier than 2018. Many of the operators in these locations are linked to some of the larger mining pools. This analysis of main regions covers a total of more than 750 MW of Bitcoin mining. The global distribution can be seen in in the map in figure Figure 3.4

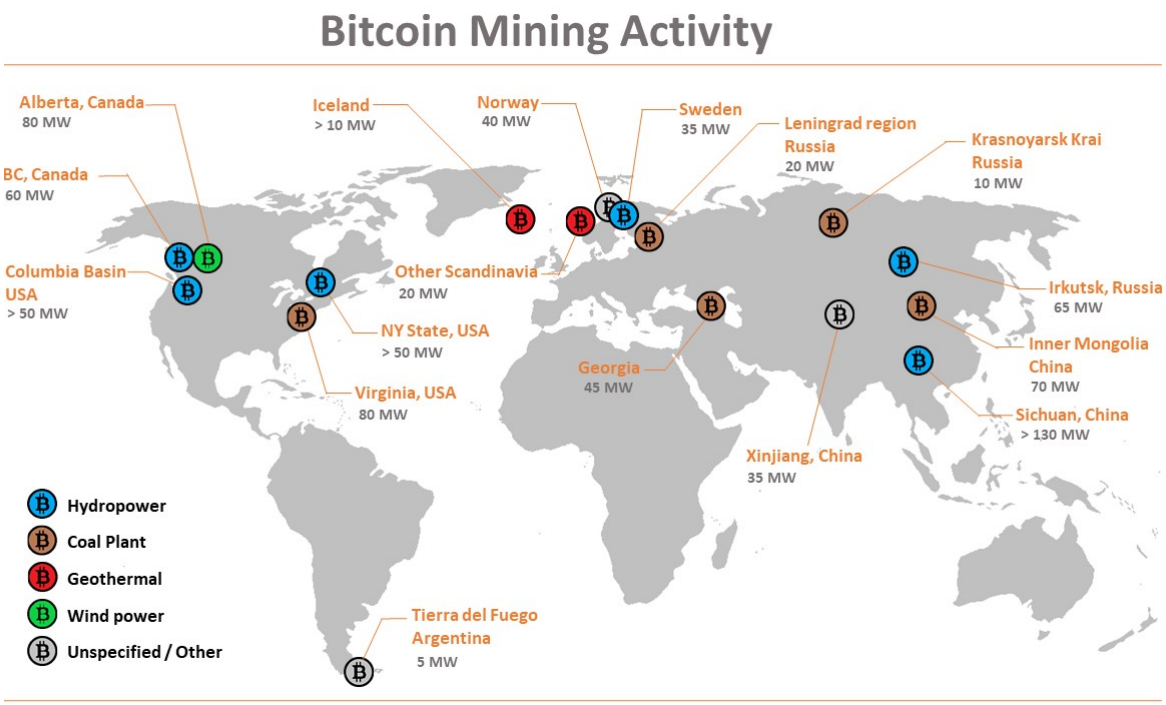


Figure 3.4: Global Bitcoin Mining Activity Map

In a back-of-the-envelope calculation, we estimate a lower bound on the total electricity consumed by Bitcoin mining. This can be derived from $P = R/E$, where R is the cumulative hash rate of all mining nodes in the Bitcoin network (in hashes computed per second), and E is the energy efficiency of Bitcoin mining hardware (in hashes computed per Joule) [22]. Considering that $R = 4.35 \cdot 10^{19}$ hashes/second (average hash rate of the Bitcoin network from January up to April 2019 [55]), and $E = 1.02 \cdot 10^{10}$ hashes/Joule (assuming that all mining is done using the most efficient mining hardware), this results in a total electricity consumption of 4.3 GW. This means that our own analysis covers only 16% of total Bitcoin mining activity.

Our map shows a great deal of overlap with the information from the Cryptoasset Benchmarking Report (both 2017 and 2018). In both reports, the mining facilities are primarily located in China (particularly in provinces in the north such as Inner Mongolia, Xinjiang, Hebei, and Heilongjiang, and in the southwest, such as Sichuan, Yunnan, and Guizhou), USA, Canada, Russia, and Georgia. The study indicates that mining has become more geographically distributed since 2017. Mining has moved partly from China to certain regions in North America (particularly Washington and New York in the US, and Québec, British Columbia, and Alberta in Canada) and Scandinavia (particularly Iceland, Norway, and Sweden). However, China still remains the major mining region and the restrictive measures on mining by the Chinese government seem not to have led to a massive exodus of Chinese miners yet. The new mining regions all have cheap electricity in ample volumes and seasonal cold climate, and are among the most developed in the world. In addition,

the report finds that mining activity is increasing in some South American countries (such as Argentina, Colombia, and Venezuela) and Western European countries (such as France, UK, and Switzerland).

We have not been able to verify Bitcoin mining locations in a number of Chinese and South American locations that are mentioned in the report. We assume this is partly due to the increased need for secrecy in many of those locations where Bitcoin mining is illegal or semi-legal at best and authorities active close down operations.

4. Building a Bitcoin geolocation application

4.1 Introduction

In this chapter we develop an application to crawl nodes in the Bitcoin network and retrieve and store information about them. By trying to connect to all the nodes in the network through our own crawler application, we are able to gather much more information about the network than is available in public sources that focus on the reachable nodes only. The application visually graphs this information with the help of open source geolocation software. The information that is collected gives a general overview of the geographic distribution of the Bitcoin network.

The crawler application is a hybrid of custom code and existing software and libraries. In particular it builds upon the Python Bitcoin protocol library of Bitnodes [51], which we have reworked to function with Python 3.

4.2 Setting up the platform

4.2.1 Main objective and requirements

The main objective of this section is to set up a platform that we can use to crawl the Bitcoin peer-to-peer network, to store results and to analyse data. We have set the following requirements:

- Compatibility with OU server and systems
- Easy to set up and maintain
- Stable entry point to the Bitcoin P2P Network
- Platform must allow for non-intrusive testing
- Network access to the IPv4 and IPv6 part of the internet
- A Bitcoin network for testing and development

4.2.2 Open University server environment

The Open University provided us with a virtual server to be able to build our application. This virtual server is part of SURFnet[1]. The server operating system is based on *SUSE Linux Enterprise Server 12 SP3*. At the start of our project the server was dimensioned at to 2 CPUs, $2\,\mathrm{GB}$ of memory and total disk space of $65\,\mathrm{GB}$, partitioned into smaller parts. During the development of our application we noticed that the virtual machine was too limited. Our setup was extended with two disks of $300\,\mathrm{GB}$, the CPU capacity was increased to 3 CPU's with 4 cores and the memory was increased to $10\,\mathrm{GB}$.

4.2.3 Containers

Applications, either custom-developed or adapted from open source projects, will be at the heart of our platform. In the past most applications ran on a dedicated physical machine. The introduction of virtual machines changed this and applications can now run inside a virtual machine. Multiple virtual machines can run side-by-side on a physical machine. Each virtual machine reserves an amount of system resources, such as CPU processing power, memory and disk space. The downside of this approach is that every virtual machine still needs one operating system and the resource of the physical server cannot be easily shared between the virtual machines.

The next evolution in this development is containerized applications. A container is an isolated environment on a single operating system. Containers are lightweight and do not need pre-allocated resources. This reduces the overhead compared to virtual machines and has led to better resource usage and higher performance. A container is based on an image. This image contains the application and everything else needed to correctly run the application. The most common technology used to run containers is Docker [56]. The Docker company website describes their product in the following manner: "A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, run-time, system tools, system libraries and settings." This essentially means that with Docker, developers can easily develop and ship applications (along with libraries and other dependencies) and ensure that the applications run the same everywhere, regardless of infrastructure.

The two approaches are depicted in Figure 4.1 [57].

[1] Surfnet(`https://www.surf.nl`) is a high-quality network for scientific research and higher education.

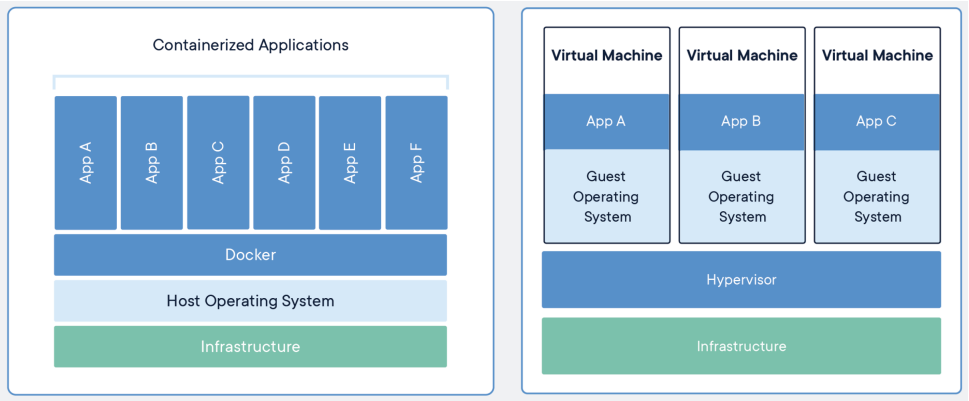


Figure 4.1: Docker containers vs. virtual machines.

To satisfy our requirements for easy setup and maintenance, the ground layer of our platform is based on containers with technology from Docker. We have chosen to use containers to ensure compatibility and portability for future users of our platform. Furthermore we expect that results are better reproducible due to the stable environment. Containers are also relatively easy to set up, likely saving us a lot of time compared to installing software straight on the OU server and getting everything to work according to the server specifications. We have not tested any alternative container solution because Docker is the leading provider of container services and the preferred option of many companies. This combined with the fact that it is open source software means it works well with many other open source tools. The Rightscale 2019 State of the Cloud Survey [58] among 700+ global IT professionals shows container usage on the rise with Docker as the preferred option for companies, as can be seen in Figure 4.2

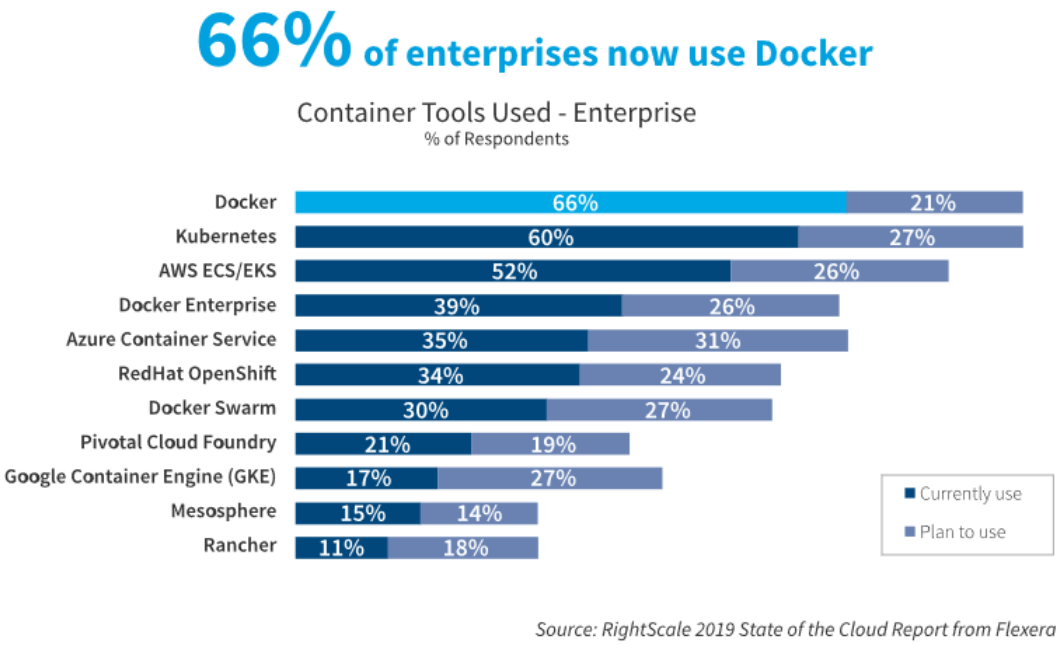


Figure 4.2: Docker usage

4.2.4 Bitcoin Core

To achieve the goals of our thesis, it is not strictly necessary for our platform to run a Bitcoin node. Information can be collected from the Bitcoin peer-to-peer network by engaging with it using the correct commands. This will be shown in section 4.3 where we do exactly that. However, there are a number of practical and ethical reasons why we would want the platform to host a Bitcoin node.

- **Testing**
  The Bitcoin Core software allows for three different networks modes, one active main network called *mainnet* and two test networks called *testnet* and *regtest* where developers can test applications.

  The Bitcoin Core regression test network (*regtest* mode) is a local network for situations where interaction with random peers and blocks is unnecessary or unwanted. Developers have control over block generation and can freely spend the rewards for creating those blocks.

  The Bitcoin test network (*testnet* mode) is a global network that acts like the main network but where currency has no real world value.

  It is important that we do not use valuable Bitcoin *mainnet* resources if we can avoid it. Testing of the basic functions of our program against a local node or test network is therefore the preferred course of action.

- **Creating a stable point of entry**
  It is important to start our research at an active node that can provide a starting list of addresses. For the main network a list with a number of IP addresses of seed nodes exists to bootstrap the operation of new nodes. The test network has a similar setup although less well documented. We could use these lists or nodes as starting point for our crawler. However, we have no control over these nodes and there is no guarantee that they will stay active over time. Therefore, it is better to have our own active node that we can query for starting addresses.

- **No harm to the main network**
  By engaging with the network, we disclose our IP address. The nodes that accept communication from our platform will see our platform as an active node because we have successfully responded to their handshake requests. They will store our address and propagate it further into the network. We see the active propagation of non-responsive IP addresses as harmful to the network. We solve this problem by running an active and reachable Bitcoin node on our platform. This way, we actually contribute to the peer-to-peer network by adding a stable peer.

The Bitcoin Core software is run inside a Docker container for which we found pre-existing images [59]. We set up a cluster of 3 local *regtest* nodes with no connection to the rest of the network which is ideal for testing and software development. We also set up a separate container that runs a *testnet* node for further testing and a container that runs a *mainnet* node to generate a local entry point and minimize network pollution.

4.2.5 Access to the Internet

The Bitcoin P2P network makes use of the entire internet to establish communication between the Bitcoin Nodes. As described in a paper by Delgado-Segura et al. [6] there are three subsets of the Bitcoin P2P network. The first is the *reachable* Bitcoin network, this network is composed of nodes that can communicate through the Bitcoin protocol and listen to other nodes. The second subset of the Bitcoin network are the *nonreachable* Bitcoin nodes. These nodes communicate through the Bitcoin protocol but do not accept incoming requests from other Bitcoin nodes. The latest subset is the *extended* network, which compromises the nodes from the mining pools and the fast relay networks that we have discussed in more detail in paragraph 2.3.2. The *reachable* and the *nonreachable* nodes can either use the Internet Protocol version 4 (IPv4) or version 6 (IPv6) to connect to the internet.

On top of the IPv4 and IPv6 protocols, there is the TOR network. TOR is an overlay network specially designed for anonymous use of the internet by obfuscating the IP address of the user. As was also described in paragraph 2.4, Bitcoin nodes sometimes operate from behind the TOR network. Obviously, by not disclosing their true IP address, they are nonreachab To gather as much information as possible we will need to connect to the internet by using as many protocols and techniques as possible. The types of access are listed below:

IPv4

IPv4 is the 'internet' protocol to connect systems. It is the oldest part and still the biggest part of the internet. Access to this part of the internet is provided through the Open University by using a virtual server which is part of SURFnet. The various Docker containers access the IPv4 network via the Network Address Translation (NAT) function which is built into Docker.

IPv6

The network infrastructure of SURFnet also has an IPv6 network. That gives our platform the possibility to access this part of the internet. Our OU server works with a single IPv6 address. The default behavior of Docker is that it expects an IPv6 network address block. Containers then make use of an IP address out of this block. Because the server has a single address, our platform is unable to operate in the manner required by Docker. The only other way to access the IPv6 network from the Docker internal network environment is to somehow enable IPv6 NAT capabilities within Docker, next to the default IPv4 NAT. This is technically possible but not advisable, because the philosophy of IPv6 is dynamically selecting an address and not assigning one. Therefore, we have decided to run our crawling application directly on the server and not in a container.

TOR

Because nodes behind TOR are not reachable, we have not implemented TOR usage on our platform. Biryukov et al. [10] show that these nodes can be deanonimized by eliciting a ban from the Bitcoin network for all TOR IP addresses, forcing the nodes to connect through other means. This is a very invasive and aggressive method and would need to be repeated every 24 hour. We see no valid reason to pursue this further.

FIBRE relay network

The FIBRE network is the successor of the Fast Relay Network, the latter is not maintained anymore. These networks were discussed in paragraph 2.3.2. Because the network is targeted at those who need low-latency block relay (i.e. miners), its nodes do not accept connections from anyone by default. We have added our IP address to the organisation's whitelist using the form on the FIBRE website and

by doing so were able to gain access to the network [2]. The nodes of the FIBRE relay network are in the settings file of the crawl application which is included in Appendix D.3.

Falcon relay network

The Falcon Relay Network was not accessible to us because the network has to initiate a connection. In the interest of time we have not pursued further actions to include a direct connection to the Falcon Relay Network in our platform.

4.2.6 Initial platform setup

Figure 4.9 shows a setup of our initial research platform. It includes the Docker containers for the different Bitcoin network nodes and the network access for IPv4 and IPv6. Details of the setup and configuration can be found in Appendix D. This setup meets all the requirements of paragraph 4.2.1.

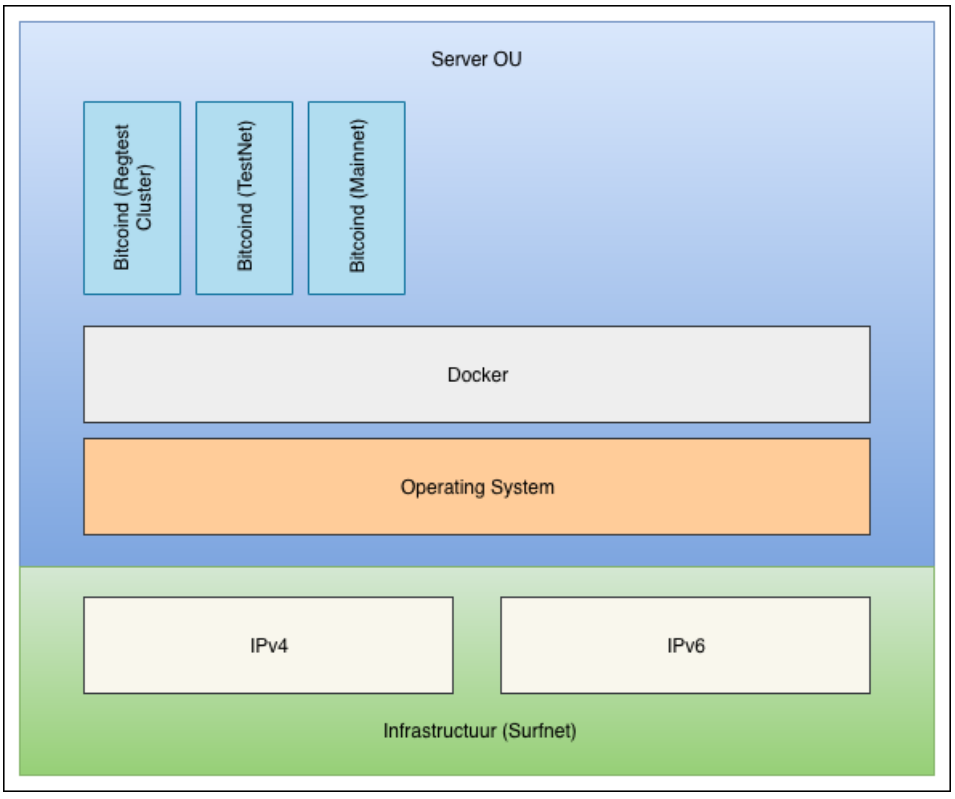


Figure 4.3: The virtual server.

[2] http://bitcoinfibre.org/public-network.html

4.3 Building a node crawler

By exploring the Bitcoin P2P network we can find out more about the characteristics of nodes in the network and their level of interaction. The topology of the network itself may not accurately reflect the geographic distribution of mining activity since most mining activity is done through pools. However, it will give us a decent base level of information to start with. Furthermore, nodes hold more information than just IP addresses of their peers which can be used to further enhance results.

A good method to explore the network is through a crawler program. In its most basic form, this is a program that starts in one node that is known to be reachable and operational, queries this node for other node addresses and then jumps from node to node until there are no more nodes to add. In this paragraph we build such a basic crawler and then add smarter and more sophisticated search and selection algorithms to improve the speed and efficiency of the crawling process.

4.3.1 Main objective and requirements

The main objective of this section is to create a working crawler that can map the entire Bitcoin network. The crawler program needs to iteratively request IP addresses continuously to traverse the network but it should also collect other useful information about the nodes it connects to. The base set of information to collect consists of:

- IP address
- Port
- Status of the node (e.g. reachable, unreachable)
  This can be evaluated when opening the connection to the node, i.e. when sending a *version* message.
- The Bitcoin Core version that the node runs
  An active and reachable node will send a *version* message in response
- The services that the node has available
  This is sent in the *version* message and indicates the features that are available, such as whether a node can be asked for full blocks instead of just headers.
- List of IP addresses known by this node
  These can be requested by sending *addr* messages

- Delay of the network the node resides in
  This can be evaluated by measuring the response time of the node from our vantage point.

It is not necessary to crawl the entire Bitcoin network to create a list of all nodes in the network. Because addresses are propagated throughout the network, the address databases of nodes will have a lot of overlap. To create a list of all addresses in the network, a crawler could be set up to iteratively add new addresses to the list until no new addresses can be found during a certain interval. However, to establish whether a node is active or not, there has to be a connection attempt. This means that to gather a full overview of active and inactive nodes, all nodes will have to be visited. It is important to make this distinction and to realize this is a requirement because traversing the entire network is a resource intensive process with implications for the required crawling algorithm. We have developed a number of methods and algorithms to traverse nodes which we will touch upon in the next paragraphs.

To collect the required information from the Bitcoin network, the crawler will have to send messages that adhere to the Bitcoin protocol. One solution to do this is to customize a reference Bitcoin Core client to connect to additional nodes and to store additional information. While this is possible (the code is open source and fairly well documented [38]) we have discarded this option as unnecessarily complex and inflexible for the following reasons:

- The Bitcoin reference client is doing a lot more than just connecting to nodes and requesting address lists. This complicates the customization process.
- The most likely way of deployment for the Bitcoin node is from the OU server which limits individual local development and testing
- The Bitcoin node would in essence be a rogue node in the network which might be frowned upon by the wider community

A different way to achieve our goals is to write code to send messages that behave according to the Bitcoin protocol but that are not generated by a Bitcoin Core client. This methodology is more practical and flexible. However it requires in-depth and exact knowledge of the protocol as the nodes will actively drop the connection if they receive a message that cannot be interpreted and they may even flag our perceived 'node' and IP address as malicious. Developing messaging code from scratch would be very time-consuming and could be a thesis subject on its own. However, some Python open source code already exists to handle protocol messaging, most notable is the Bitnodes project (see section 4.3.3). We can adjust this code and further build

upon its basic functionality. The existence of this code is one of the main reasons to choose this second approach.

4.3.2 Summary of methodology

Chronologically, a basic crawler will need to perform the following steps:

1. Initialize settings to connect to the relevant Bitcoin network
2. Select a starting node that is guaranteed to deliver addresses
3. Connect to the node
4. Send a *version* message to the node:
   (a) Create the building blocks of the *version* message
   (b) Convert the building blocks into bytes
   (c) Serialize the byte strings according to protocol specification
   (d) Send the information to the node
5. If the node does not respond, closes connection or an error occurs:
   → Select the next node and go back to step 3
6. Listen to messages from the node:
   (a) Collect incoming message packets
   (b) Deserialize the message
   (c) Convert bytes into relevant building blocks
   (d) Respond to handshake messaging when necessary
      i. When message is *version*, respond with a *verack* message
      ii. When message is *ping*, respond with a *pong* message
   (e) If no *version* or *verack* message has been received: → Select the next node and go back to step 3
7. Send a *getaddr* message to the node (process similar to 4)
8. Listen to *addr* message from the node (process similar to 6)
9. Make an array of the received address list
10. Filter the new addresses against known addresses
11. Add the new addresses to the crawling list
    → Select the next node and go back to step 3

In paragraph 2.5.1 we summarize findings from the article by Deshpande et al [7] that shows that not all nodes in the network respond to the first *getaddr* request. The authors find that after 6 requests, 90% of reachable nodes respond. We implement this as a feature by repeating step 7 to step 10 six times. This has the added benefit of retrieving additional addresses from most of the nodes. Because the address list of nodes is typically populated by tens of thousands of addresses and a *getaddr* request will only return a maximum of 1000 semi-randomised addresses, repeatedly querying a node will give us a much larger number of addresses per node.

4.3.3 Use of existing code: Bitnodes messaging module protocol.py

Bitnodes [51] is a project that has been developed to estimate the size of the Bitcoin network by finding all the reachable nodes. The software sends *getaddr* recursively starting from a set of seed nodes. It uses the Redis caching software for in-memory data structure storage. The crawling program is written in Python 2.

The most important component of the Bitnodes program is its Bitcoin protocol module. This module holds a class Serializer to serialize and deserialize the Bitcoin messaging according to the protocol definitions. It also holds a class Connection that builds up the connection to the node. The basic structure of relevant methods can be seen in figure 4.4:

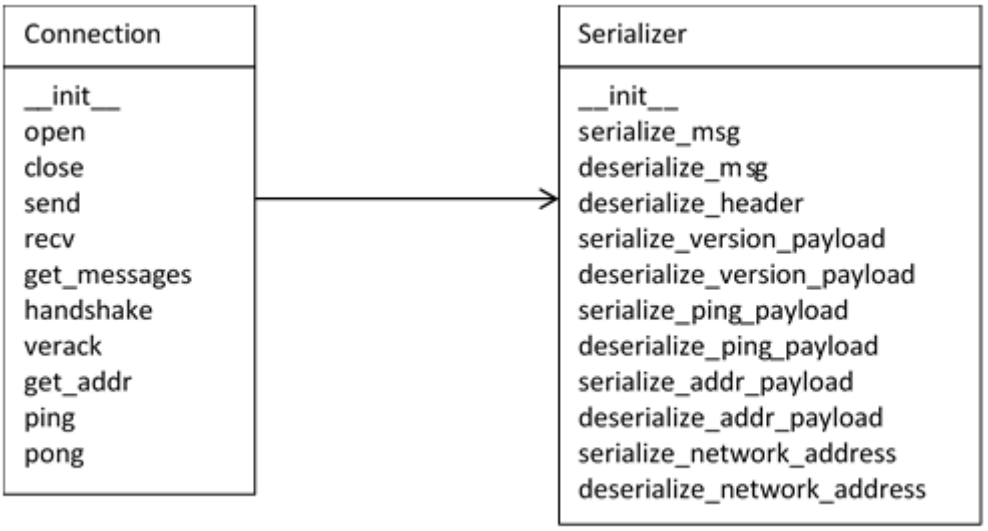


Figure 4.4: High level diagram of protocol.py

A number of other methods exist in the protocol module but for this chapter they are not relevant. With the help of these two classes and methods we are able to create the crawling algorithm defined in the previous subsection.

4.3.4 Update to Python 3: Changes to protocol.py

The Bitnodes protocol messaging code is written in Python 2 under its latest version 2.7. In 2008 Python 3 was released which was not backward compatible with Python 2. New updates of Python 2.7 with support and bug fixes will end in 2020 and users have been actively encouraged to move to the more powerful and versatile Python 3 for years which is being actively developed further with new features. The latest release of Python 3 at the moment is version 3.7.

One of the major changes between Python 2 and 3 is the way strings and bytes are handled. In Python 2 a string literal is called a *str* object and by default it is stored in bytes. In Python 3, a string literal is also called *str* but by default it is stored in Unicode code points. There are a number of ways to make the conversion between Unicode strings and bytes.

- Using the prefix 'b' in front of a string literal will create a bytes object in Python 3 and do nothing in Python 2. However, this will only work on ASCII characters.
- The method .encode() will encode Unicode strings into bytes. A parameter indicating the codec can be passed but by default UTF-8 is used.
- The method bytes('string') achieves the same result as 'string'.encode() in Python 3 but is not backwards compatible with Python 2.
- A Python 2 string can be turned into Unicode with the prefix 'u'.
- The Struct class from the module *struct* has a method pack(format, v1, v2) that converts values into byte strings according to a specified format. This works in both Python 2 and Python 3.

The Bitnodes protocol.py module holds about 1000 lines of code and most of the code makes extensive use of bytes and string conversion in its serialization and deserialization methods because it needs to transform the building blocks into byte strings. Some of this is done by concatenating string literals (such as 'version') with byte strings. It is not surprising therefore that the module does not work under Python 3 as the two are different objects that cannot be concatenated.

To make this project future-proof it is therefore necessary to convert the Bitnodes messaging protocol into Python 3. In addition to this, the operating system of our virtual server is based on *SUSE Linux Enterprise Server 12 SP3* which supports Python 3.4. This means that we cannot run the original Python 2 code but we also cannot use some of the latest functionality from Python 3.5 and higher.

A number of one-off adjustments had to be made but many of the changes are generic. Where possible we have aimed for a solution that works for both Python 2 and 3. Some examples:

We used the line `struct.pack("12s", command.encode('utf-8'))` to create the protocol command (*version*, *getaddr*, etc.) as opposed to the original concatenation of the command literal with Unicode representations of zero in `command + "\x00" * (12 - len(command))`. Both methods add padding of zeros up to 12 positions left of the command literal but only the restated command guarantees a byte string in both Python versions.

Serialization methods return their building blocks (often in bytes) in one string through the command `return "".join(iterable)`. The join method returns a string concatenated with the elements of an iterable, which can be a list, tuple, string, etc., as long as the iterable only holds string values. This obviously will not work in Python 3 and a TypeError will be raised for trying to concatenate bytes with strings. However, the join method will also work on bytes and therefore this code can easily be amended by prefixing `"".join` to `b"".join` as long as we also make sure that all individual elements of the iterable are bytes-like objects.

Protocol.py uses class StringIO from the buffer module cStringIO to handle the incoming data from nodes. StringIO provides a convenient means of working with text in memory using the file API methods (read, write. etc.). In Python 3, the cStringIO module is no longer available but instead it is replaced by the module io which has backward compatibility to Python 2.7. However, in Python 3 we can only handle Unicode strings with io.StringIO. Byte strings will have to be handled by the new method io.BytesIO. Given the fact that incoming data that has to be deserialized starts out as a byte string, our updated protocol will have to use BytesIO to keep the same functionality. However, because BytesIO has no equivalent in Python 2, this is not backwards compatible. Therefore the following code is used throughout the protocol module for every instance of StringIO:

```
try:
    data = StringIO(data)
except TypeError:
    data = BytesIO(data)
```

These are only a few examples to highlight the type of transformations required to achieve compatibility of the module for Python 3.

4.3.5 Crawling algorithms

The crawling method can be implemented in a number of ways with increasing levels of complexity. To create an upper boundary for the time it takes to traverse the full network, we will start with a naive algorithm that takes the IP address of the startnode that we hardcoded, then stores the collected IP addresses in an array (after filtering for known addresses) and then continuously selects the next IP address in the array. The basic structure of relevant methods can be seen in figure 4.5:

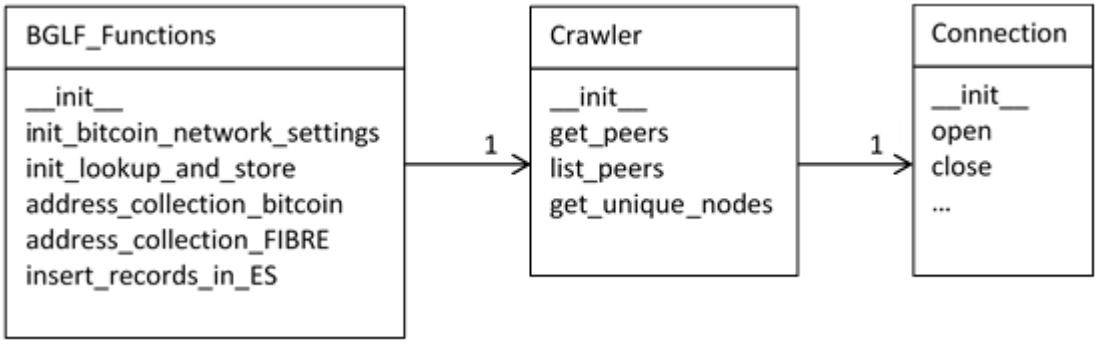


Figure 4.5: High level diagram of crawler classes functions

The program flow is directed in a handler class called `BGLF_Functions`. In the initialization step, all the relevant configuration data is loaded from a config file through the method `init_bitcoin_network_settings`. Different settings are used based on parameters set at execution of the program. See Appendix D.3 for more information on configuration and parameters. The method `get_peers` builds up a connection with the Bitcoin network through the class `Connection` in the protocol.py module. After building the connection the handshaking (*version, verack*) is executed and when successful a *getaddr* message is sent and a *addr* message received. The crawler then breaks down the field of received addresses into separate records using the `list_peers` method. Both the list of peers and the additional information for the node that was just crawled are returned to the program handler which then calls the method `get_unique_nodes` of the crawler object to filter out the addresses that are already known. This algorithm traverses on average 3 nodes per minute. While this is sufficiently fast to create a list of all nodes in the network in a period of a few hours, it is too slow to traverse all nodes in the network in a reasonable time frame. This means that we need to deploy a form of concurrency.

To do this we make use of the *gevent* asynchronous programming module [60, 61]. This is a module that can spawn concurrent threads called *greenlets*. A greenlet will run until it chooses to give up control which it will do when handling IO operations. Because those operations are the main component or our crawler program, the use of the gevent module should significantly speed up the total process. To make greenlets function in this manner, the gevent module has a method called monkey.patch_all() that has to be called at the start of the program. This command makes all other modules asynchronous, i.e. it 'patches' potentially blocking code such as IO operations to automatically yield control back to the system when required.

With the help of *gevent*, we are able to significantly speed up the crawler program. Instead of 3 nodes per minute, the crawler can now process hundreds of nodes per minute. The next algorithm we now use is a threaded version of the first algorithm. We start again with the hardcoded startnode. This node supplies us with a set of new peer IP addresses. We start greenlets to crawl each of these addresses. We then call `gevent.joinall(greenlets)`, telling the main process to wait until all the greenlets have finished. Once this happens, we have a (much larger) set of new addresses. Again we deploy greenlets to crawl each address and wait for all of them to finish. By following this method, within minutes we have traversed the bulk of the network nodes. However, the algorithm then becomes much slower because very few new addresses are found during each iteration and therefore only a limited number of greenlets will be deployed concurrently to connect to these addresses. The algorithm has to wait each time until the batch is finished to start a new batch which, for only a few final nodes, reverts its behaviour and speed back to that of the first algorithm without threading.

Ideally, we would want a constant number of greenlets to crawl addresses that can start their work as soon as a new address is found and that do not have to wait until previously activated greenlets are all finished. This is possible but requires a slightly more complex concept. From the module gevent.pool we import the class Pool. This is a structure that can maintain a fixed group of running greenlets which are initiated by the call `pool = Pool(500)`, if we choose to have 500 concurrent greenlets. Using this, we now form our third algorithm. Again we start with the hardcoded startnode. We start greenlets in the pool to crawl the addresses that are returned. Each greenlet may return new addresses and will do so at different moments. When new addresses are returned, new greenlets are immediately added to the pool to crawl these addresses, with a maximum of 500 (in this example). Once the maximum of 500 is reached, new IP addresses are still processed but will be put on hold in a waiting list and only added to the pool when one of the active greenlets dies. We instruct the main process to wait until all the greenlets in the pool have finished. Normally this is done by the command `pool.join()`.

However, we cannot put this command in the `for` loop that continuously adds the new addresses to the pool because then every loop would again just act as a batch. As an alternative we perform a check ourselves whether all greenlets have finished through the expression `is_over = all(th.ready() for th in pool.greenlets)`. If this is the case, we break the loop that keeps the process going. If not, we instruct the main process to wait 10 seconds and let the loop run again.

This algorithm is again a lot faster than the previous one because it does not suffer from the batching delay problem towards the final stage of the crawling. Any new address is added to the pool or waiting list as soon as it is found and activated when the number of active greenlets drops below 500. There are no drawbacks to this algorithm compared to the previous two, it is easy to scale by increasing or decreasing the greenlet pool limit and we do not see a lot of room for further optimization which is why we selected it as the final algorithm.

Table 4.1 shows the size of the greenlet pool versus the time it takes to run the platform and the total nodes found. The rum time covers more activities than just crawling, which is reasonable fast when data is kept in memory. In particular the storage of results takes a large amount of time.

No. Greenlets	***Run ID***	***Crawl Time***	***Total Nodes Found***
100	419	7:39:49	37662
200	420	8:07:35	41007
300	421	6:14:10	40119
400	422	7:26:25	49993
500	423	4:23:42	38621
550	429	3:41:34	40682
600	424	4:59:54	41477
650	430	4:33:43	42481
700	425	4:09:13	42249
750	431	4:55:19	42249
800	426	5:20:45	44076
850	432	4:01:26	42302
900	427	4:14:57	40885
1000	428	4:44:20	40778

Table 4.1: Number of gevent greenlets versus crawl time and total nodes found, mainnet

We found that when crawling with more then 700 greenlets, the process starts to slow down again. Through testing and inspection we found that the bottleneck is Elasticsearch, our database for storage which we will touch upon in paragraph 4.4.2. Elasticsearch apparently can not keep up with a large number of greenlets that are all trying to insert and index data. Based on these results we set the amount of greenlets in the pool to 700.

From March 24 to May 15, 2019 we ran the crawler daily. Figure 4.6 shows the number of nodes found during this period. The crawler found on average 41,500 nodes per daily run, ranging from 32,000 to 48,000 nodes. These are large differences that we have not researched further. However, a previous study during May to July 2018 based on data from Bitnodes reports a high amount of daily churn in the network (i.e., the rate at which nodes independently enter and leave the network) [23]. This could be a partial explanation for the daily differences. Furthermore, our crawler does not request 100% of the peer database because we only crawl every node six times. We suspect that if we significantly increased the amount of *getaddr* requests per node it could decrease the daily volatility. This would come at the cost of speed however. We leave this analysis for future research and improvements.

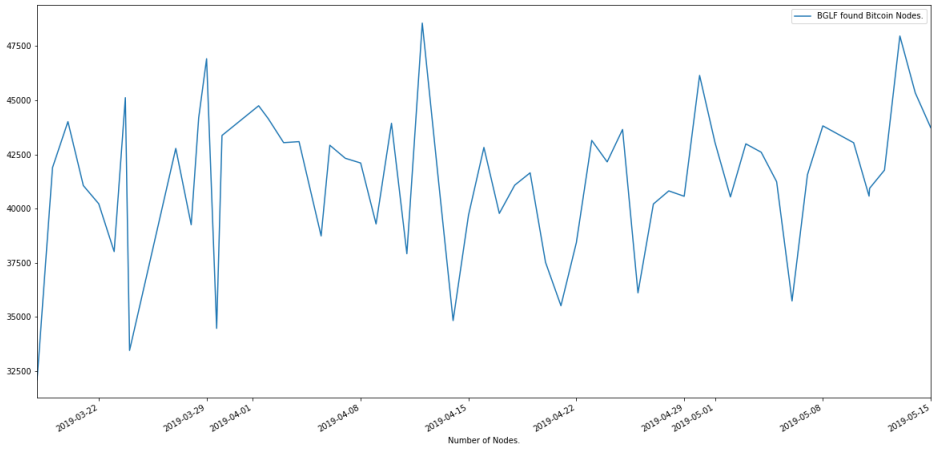


Figure 4.6: Total found nodes per run_id in the period of March 24 to May 15 2019.

4.4 Store, analyze and visualize the crawled data

Crawling the data of the Bitcoin P2P network is only the first stage of our approach. The data from the crawler now needs to be stored in a database, checked for inconsistencies and analysed for relevance.

4.4.1 Main objective and requirements

The main objective of this section is to create a platform where the crawled data can be stored and enriched with geographical information. The analytical functions and the presentation of results should be available through an attractive visual dashboard. Because we aim for a modular platform that can support future research, it

should have the possibility to easily add new data without the need to fully rewrite the code.

We have defined the requirements as follows:

- **Ability to handle big data**
  The crawling results should be stored in a database that can handle a large amount of data. The required operations should happen almost real-time.
- **High level of integration with the crawling application**
  The crawling results should be fed into the database automatically without additional actions from the user. Therefore the database must also be accessible through endpoints.
- **Enrichment of crawling results with geographical data**
  When feeding the crawling results into the database, they need to be enriched with geographical information.
- **Attractive direct mapping and visualization of the collected data**
  The enriched data needs to be presented on a world map. The visualization of this map should be fast and should not require additional actions from the user.
- **Analysis of the collected data should be straightforward and flexible**
  It should be possible to analyze the data and do some statistical analysis without the need to write code. On the other hand it must also be possible to extract the data directly using scripts.
- **Functional but not state of the art**
  The development of a database systems with analytical tools is time consuming. The aim of this project is the collection and geographical analysis of data from the Bitcoin P2P network, not the software development of a database and analytics system.
- **Easy handling of the platform**
  The total platform must be easy to clone, setup and maintain so that it can be used in future research.

4.4.2 Storage and visualization of the data

The data that is collected by the crawler is stored in an Elasticsearch [62] database. Elasticsearch is an open-source, RESTful[3] full-text search and analytics engine for

[3]Web services that conform to the REST architectural style (Representational State Transfer) follow a uniform approach to operations, typically using HTTP requests to GET, PUT, POST and DELETE data.

storing and analyzing big volumes of data in near real time. Elasticsearch stores data in JSON document form and can subsequently be queried using JSON format in order to retrieve data. The software allows for a distributed architecture to increase operational speed.

Data in Elasticsearch is organized into indices, which are collections of documents with similar characteristics. A document is a basic unit of information, expressed in JSON, that can be indexed. Within an index many documents can be stored. Each index is made up of one or more shards. A shard can be seen as a self-contained search engine that indexes and handles queries for a subset of the data in an Elasticsearch cluster. A shard is an unbreakable entity in Elasticsearch, in the sense that a shard can only stay on one node. An index which is a group of shards can spread across multiple nodes but shards can not.

Data in documents is defined by so-called mappings which are user-made definitions of how a document and its fields are stored and indexed. For instance, mappings can be defined to determine which string fields have to be treated as full text fields or which fields contain numbers, dates, or geolocation data.

On our platform we have set up a cluster of three nodes (a node is an Elasticsearch instance, for this instances we use a docker image [63]). Indexes, shards and mappings are monitored and controlled using the Head Plugin [64] and ElasticHQ [65].

Kibana [66] is an open-source analytics and visualization tool that is used to interact with the data stored in Elasticsearch. With Kibana (is also a docker image [67]) the collected data can easily be understood, analyzed and visualized. Kibana can be seen as a dashboard add-on for Elasticsearch with which you can easily and automatically create visualizations such as pie charts, line charts, and many others. For our research it is particularly helpful because it allows us to plot geolocation data directly into a world map.

To summarize, by using and combining Elasticsearch and Kibana we create the following benefits:

- Direct off-the-shelf open source solution
- Easy exploration and analysis of the collected data
- Dashboard function with multitude of visualization options

The chosen setup more than fulfills our requirements and can even be considered a state of the art solution, even though this is not what we set out to achieve. The technical details, parameters and configurations necessary to achieve a functional cluster are too extensive to list in this chapter but they can be found in Appendix C.

As a final side note, Elasticsearch and Kibana are often integrated in a structure that is called an ELK-stack [68]. The letters represent Elasticsearch, Logstash and Kibana. Logstash is an agent that runs on a server, extracts data from server logs and pushes this data into Elasticsearch. We do not require Logstash for our research because we use our Python-based platform to directly feed data into Elasticsearch.

4.4.3 Geo Location

IP addresses by themselves do not hold geolocation data. Rather, ranges of IP addresses are known to originate from certain places in the world. Databases exist that can provide these mappings. The database that we use is Maxmind GeoLite [69], a free version and comparable to, but less accurate than, MaxMind's commercial GeoIP databases. The commercial databases have accuracy up to street level, which is not necessary for our research. Other web-based sources exist that provide an API to retrieve geolocation data. Most of these have a limit of 10.000 requests per month for the free version. Since a single crawl of the *mainnet* already requires more than that, these websites are not useful to us. The only alternative therefore is using a local database which is exactly what MaxMind GeoLite provides.

Before each crawl, our platform downloads the latest version of the database. Next to latitude and longitude, the database provides the country and city (not always). Furthermore the database holds information on the network that the IP address belongs to and the organisation that owns or operates the network.

For this thesis we have chosen to focus on a single source of data that we can integrate in our platform. To limit the scope of our research we have refrained from exploring further optimisations. The accuracy of the MaxMind database is reviewed by the organisation itself [70] and by more independent sources [24]. The general conclusion is that the data is useful and accurate, both for IPv4 as for IPv6.

In addition to retrieving and mapping geolocation data using MaxMind, we also execute a reverse DNS lookup because the data that is returned often holds geolocation information. At the moment this information is not used for anything but this might be useful for future improvements. Extensive academic and practical research exists on geolocation and the many ways to improve and optimise results [25, 26, 27, 28].

4.5 Bringing it together: BGLF

In this paragraph we describe how the results of the previous paragraphs are combined into a single tool. We call this tool the Bitcoin GeoLocation Finder (BGLF).

Our tool is written in the Python 3 language. It is composed of several modules that each have their own functionality. This allows for loose coupling of classes and therefore the ability for future users or developers to relatively easily add, delete or replace specific modules. The tool is set up to function from the command line interface (CLI) and allows for a number of user-defined options. As an example, the user could give the following command:

```
python.exe BGLF.py --verbose --network_type=testnet
    --no_store_data crawl bitcoin-network.
```

This means that the user is requesting to crawl the Bitcoin *testnet* (`--network-type=testnet`) without storing the data (`--no_store_data`) but with detailed on-screen output from the logging system (`--verbose`). The logging is set up using the standard and flexible Python `logging` module [71]. Detailed information on command-line options can be found in Appendix D.2.1

The BGLF tool will proceed to implement the chosen command line parameters throughout the code with the help of the imported module `click` [72] and it will read the configuration file that is included in Appendix D.3 and parse the parameter values using the `Config` Class in the `config_parser` module.

The next step in the process is the initialization of components:

- Elasticsearch
  - Check if the indexes are already created. If not, insert static mapped indexes [4]
  - Connect to the Elasticsearch environment defined in the settings file.
- GeoIP
  - Load the downloaded Maxmind GeoIP lite databases.
- Bglf_DB
  - Connect to a SQLite configuration database. This database holds the `run_id`, which is an unique identifier to identify what data is collected during a particular execution of the program.
  - the program takes the latest known `run_id` and increases it with one. If no `run_id` is found then the value is 500 (merely a 'magic' start value).

[4] Elasticsearch has dynamic and static mapped indexes. Dynamic mapped means that Elasticsearch tries to find to most suitable type for a value. By static mapped indexes the type of a value is already defined. This makes it easier to benefit from the power of Elasticsearch's database search aggregation functionality (which is comparably with *GroupBy* of SQL).

Besides the initialization of the different components, also the crawling environment is initialized. In this phase, input parameters are used to determine which network (*mainnet*, *testnet*, *regtest*, *FIBRE*) is crawled. The tool then starts its crawling activities, which is started from the helper class `BGLFFunctions.py`. As explained in paragraph 4.2, to bootstrap the crawling process we use the address of our own Bitcoin node. The inner workings of the crawler are explained in paragraph 4.3.

During the crawl phase the collected information is stored in two different indexes. The information in the first index is the data of the crawled Bitcoin node. This is information about the *date* of crawling, the crawled *network_type*, the *run_id* as well as which *services* are provided to the Bitcoin network by the node. For a full overview of the stored information see Appendix C.3.1. Before the information is stored, additional geolocation information is requested by a reverse DNS-lookup. The second index contains the relation between the crawled node and the peers in its address table. Every relation is stored separately. Because the address table of a node can hold thousands of nodes, it takes time to insert every relation into Elasticsearch. Therefore we have chosen to perform a 'bulk' insert. This means that the relation of one node with all its peers and all other related data and the requested geolocation data is inserted in one go. The overview of this index can be found in Appendix C.3.2.

A basic UML diagram for the BGLF application is show in Figure 4.7 and a detailed overview of the structure of the code, packages and files, can be see in Appendix D.1 Program structure and D.2 UML diagram.

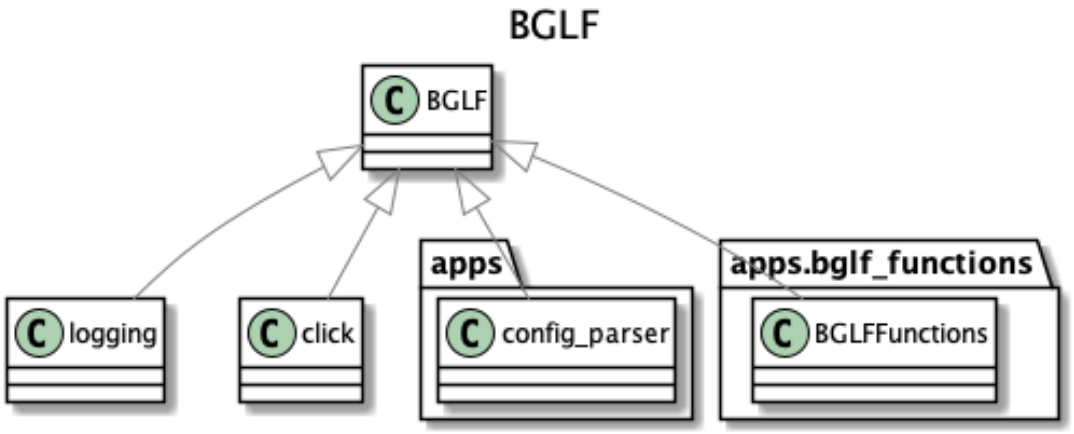


Figure 4.7: Basis UML class diagram for BGLF.

4.6 Testing

Development and testing of the application posed its own challenges. Ideally we would have developed an application that is self-contained and for which the code

can be broken up in individual elements that lend itself to unit testing. However, due to the large amount of existing open-source software, the customization of this software and need for integration of all the different elements, we have not used unit testing or built classes to perform unit testing. Rather, we have consistently and repeatedly checked the individual parts of the platform for logical and expected behaviour and output.

For the base platform with server hardware settings, containers, Bitcoin Core cluster and access to the internet we have checked behaviour by testing whether all installed containers where operational with `docker-compose` (see Figure 4.8) and by pinging different hosts on the internet. The connections and network traffic were tested with network protocol analyzer software Wireshark [71].

```
$ sudo /usr/local/bin/docker-compose ps
          Name                        Command               State                                                          Ports
-------------------
```

on the *testnet* network and we refrained from solving the issue. The *regtest* cluster is still part of the platform and can be used in potential future work.

The messaging between nodes is bound to a strict protocol format. Communication only works when all input parameters are correctly used and serialized into bytes in the right order. Inversely, communication can only be understood when an incoming message is deserialized in the right order and reworked into the right python lists and variables. The best source for the protocol details in our opinion is its Wikipedia site [43] in which message structure, fields and data types are explained. We have assumed proper functionality when correct responses were returned by nodes to our request and vice versa. For instance, address collection will not work if handshaking (*version*, *verack*, *ping*, *pong*) is not executed correctly. The response to a *getaddr* has to be a *addr* message holding a (maximum) 1000 IP addresses. By checking that no node delivers more than 1000 addresses, we assume correct functionality.

The functionality of Elasticsearch, Kibana and other add-on tools has been tested by manually feeding data into the applications and checking the database for right representation and storage as well as the visual dashboard for proper presentation of the test data.

Finally, the combined BGLF tool is developed and tested in iterative steps. During the development all the different modules were added one by one into `bglf_functions`. Every code module has an option to log information about its execution. This is called `debug logging` and an example can be found in Appendix D.4. The testing of our code was mainly done by looking very carefully at the different log messages. During the integration testing between the crawler and the module for storage in Elasticsearch, we crawled only one Bitcoin node at a time. By comparing the debug logging of the crawled Bitcoin node and the stored data in Elasticsearch and by making use of the *discovery* function of Kibana, we were able to validate that all the data was accurately stored. The static mapped index configuration for Elasticsearch was of real value during the development because it prevented mistakes in the ingestion of data.

4.7 Revised platform setup

We implemented the BGLF tool on the OU server to make it work together with the different container structures on our platform. The BGLF tool is installed alongside the Docker environment but not inside a Docker container because of the required IPv6 network access. With its full array of functionality, the platform consists of 6 Docker containers:

- Elasticsearch, for storing the crawled data.

- Kibana, for analytics and visualization.
- Head Plugin and ElasticHQ, the graphical interface for monitoring and maintenance of Elasticsearch
- Bitcoin Regtest Cluster
- Bitcoin Testnet Node
- Bitcoin Mainnet Node

A visualization of the integration of the Docker containers and BGLF together with the OU server is depicted in Figure 4.9.

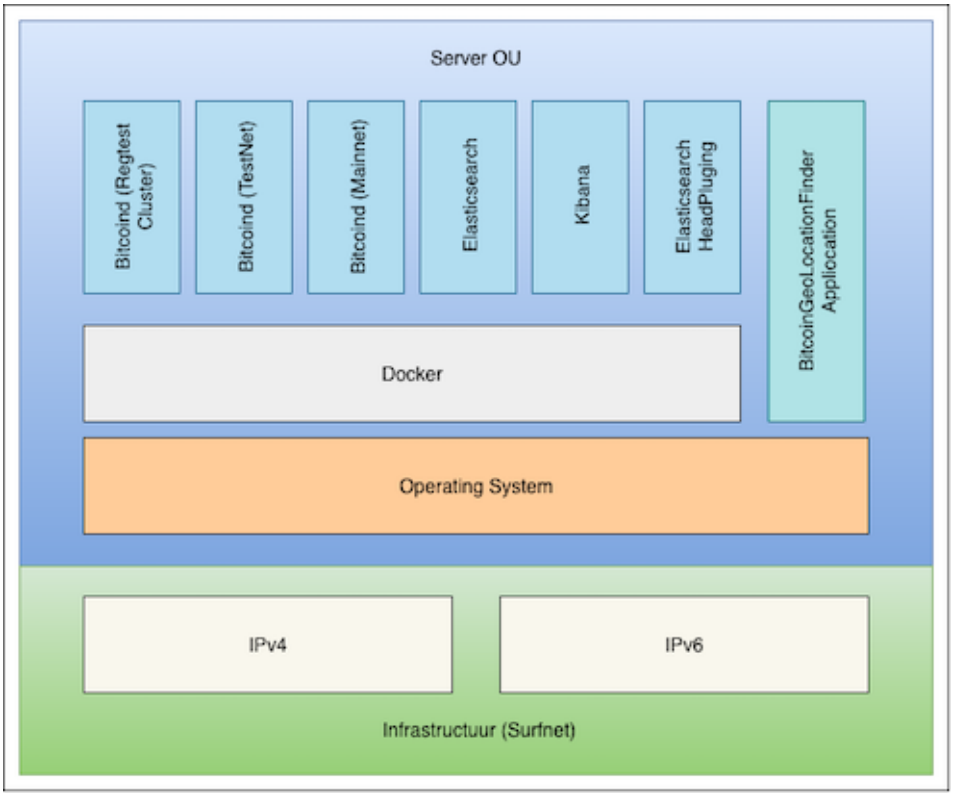


Figure 4.9: Docker configuration overview.

4.8 Discussion of results and validation

The final and tested tool was deployed on the *mainnet* to gather a live overview of the size and geographical distribution of the Bitcoin network. A full run of the tool including storage of all data takes between the 4 and about 6 hours. A single run is identified by a run ID in the database so data can easily be extracted and compared.

The visual representation in Kibana of a run is shown in Figure 4.10. This figure shows the total of all crawled data for the *mainnet* between April 1 and April 20 2019.

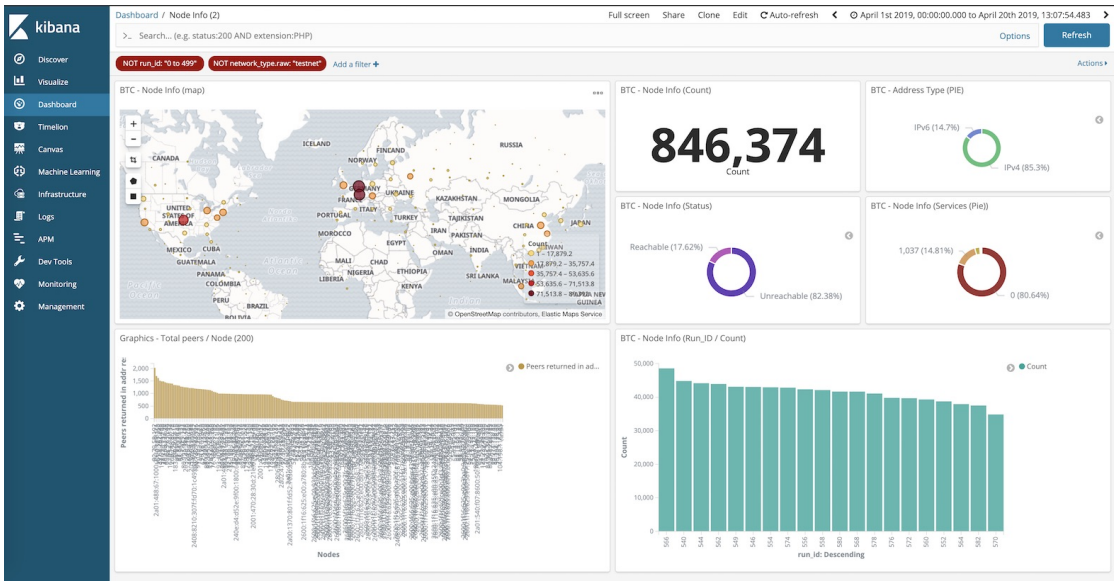


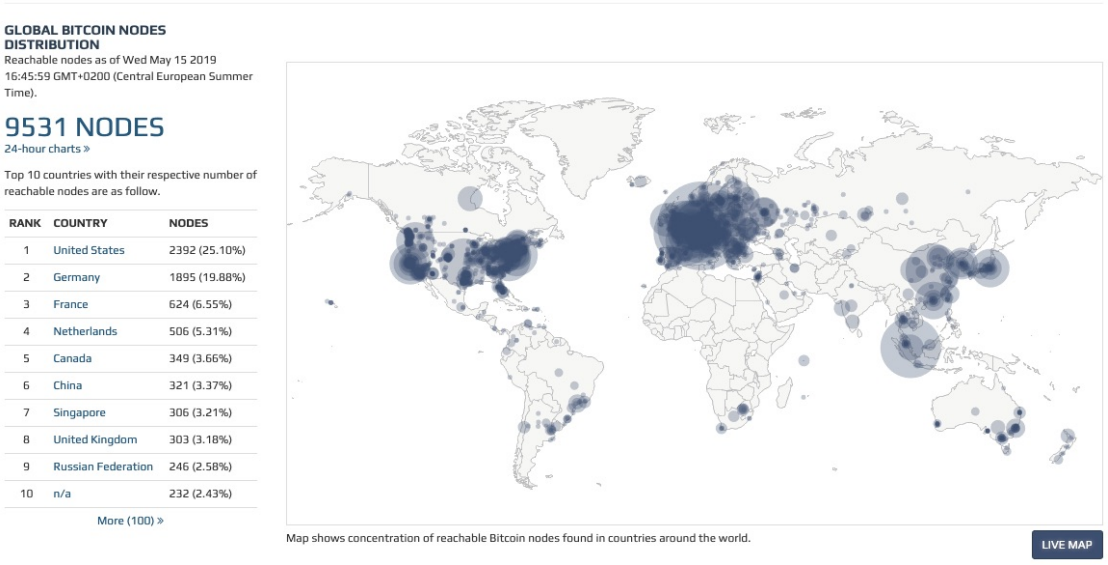
Figure 4.10: Kibana dashboard showing the crawling results of our BGLF platform

There is a clear concentration of nodes in the US, Western Europe an and Eastern China. While this partly coincides with the results from Chapter 3, there are also very notable differences. For instance, Nordic countries are not among the major hubs and the same holds for some regions in the US and China for which we know there is a lot of mining activity. This could partially be explained by the fact that IP Geolocation is not perfect and will return just one set of coordinates when the location cannot be made more precise than country level, creating one large single point of gravity in that country. This seems to have happened for the US and Germany in particular.

However, when looking at the map closely, the most likely explanation for the distribution of nodes is that they just reflect large urbanized areas and are a cross section of people using Bitcoins instead of partaking in Bitcoin mining. This is not surprising given the fact that we are just locating Bitcoin nodes and nothing else (albeit not only the reachable nodes but also the unreachable nodes). If we assume that the results from Chapter 3 are at least partly accurate, then simply crawling all Bitcoin nodes is a very imprecise estimation of Bitcoin mining activity. We will work in Chapter 5 towards refining our results to increase the likelihood that the nodes that we plot represent mining activity.

It is interesting to see how our results compare to the Bitnodes project [74]. Note that the Bitnodes organization only stores the reachable nodes. The results can be retrieved through an API call to their website but the Bitnodes organization also

provides a detailed world map of results on their site. A screen shot of this map, generated from results of May 15, 2019, is shown in Figure 4.11.



GLOBAL BITCOIN NODES DISTRIBUTION

Reachable nodes as of Wed May 15 2019 16:45:59 GMT+0200 (Central European Summer Time).

9531 NODES

24-hour charts »

Top 10 countries with their respective number of reachable nodes are as follow.

RANK	COUNTRY	NODES
1	United States	2392 (25.10%)
2	Germany	1895 (19.88%)
3	France	624 (6.55%)
4	Netherlands	506 (5.31%)
5	Canada	349 (3.66%)
6	China	321 (3.37%)
7	Singapore	306 (3.21%)
8	United Kingdom	303 (3.18%)
9	Russian Federation	246 (2.58%)
10	n/a	232 (2.43%)

More (100) »

Map shows concentration of reachable Bitcoin nodes found in countries around the world.

LIVE MAP

Figure 4.11: Bitnodes crawling results (May 15 2019)

To be able to compare our results with the results from the Bitnodes project and plot them in a similar manner, we have added functionality to our platform to query the Bitnodes API. For practical reasons, we explain the technical details of this query in paragraph 6.2.5 and only show results in this chapter. Both sets of data are shown in Figure 4.12 and Figure 4.13. The data is collected on May 14, 2019. Because we use the information to compare and validate our crawling results, only the reachable nodes found by our own crawler are shown in this picture. The maps look fairly different from the original Bitnodes map in Figure 4.11 due to the way Kibana clusters results in certain regions and makes them stand out more than other regions (e.g. South America looks less pronounced in the Kibana maps). However when comparing both sets of results in Kibana, we can see they have a high degree of similarity.

For another check we compare our crawler results with the results from a search for devices running port 8333 (the default Bitcoin Core port) from search engine Shodan. Shodan is explained in more detail in paragraph 6.2.1. These results are also mapped in Figure 4.12 and shown as a pie chart with the top 20 countries in Figure 4.13. The data set from Shodan shows the same countries in roughly the same order but percentages differ. These differences are not surprising as the two methods do not exactly measure the same thing and.

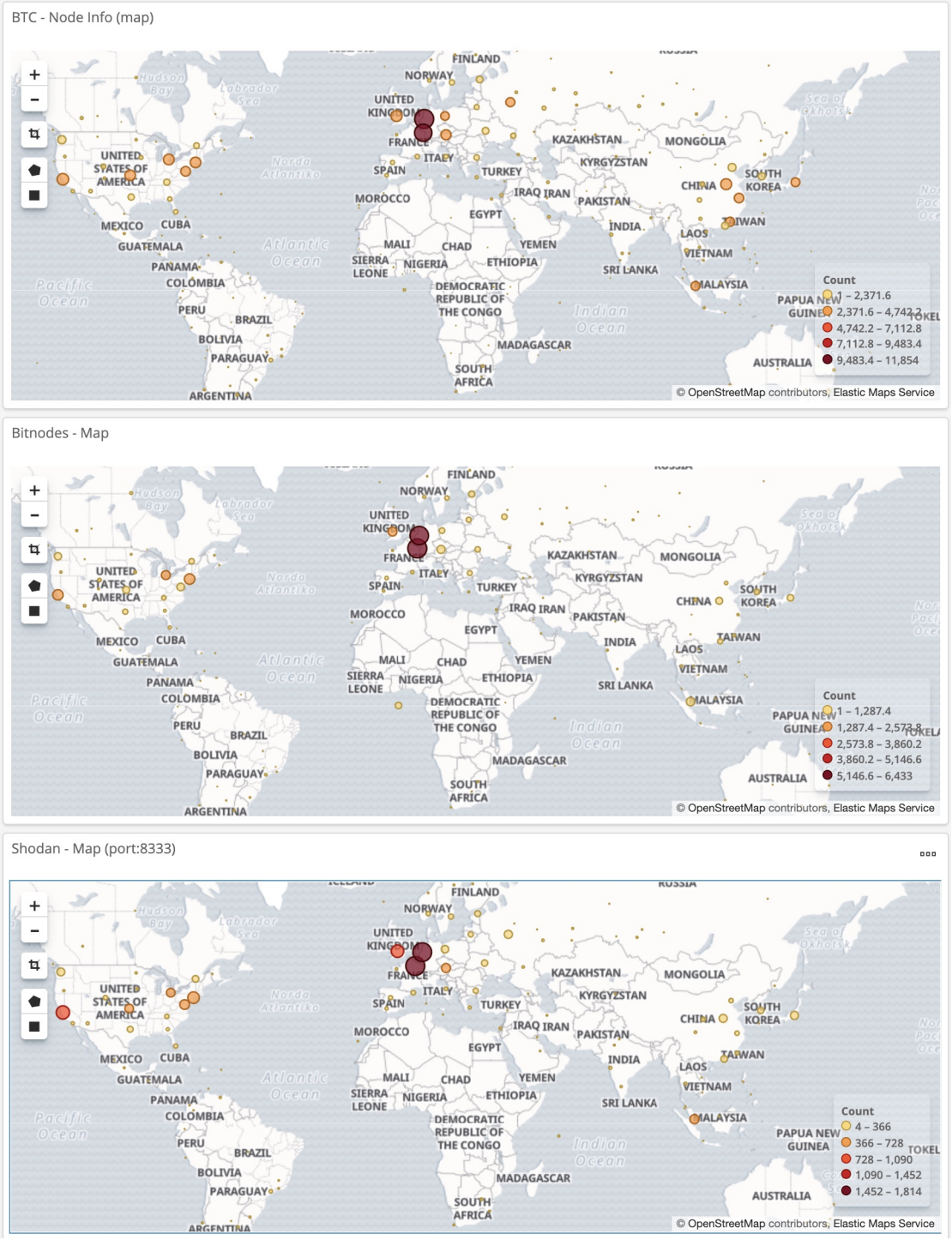


Figure 4.12: Comparison of crawling results of BGLF and Bitnodes and Shodan (port:8333) (top is BGLF, middle is Bitnodes, bottom is Shodan) show on as maps of May 14, 2019.

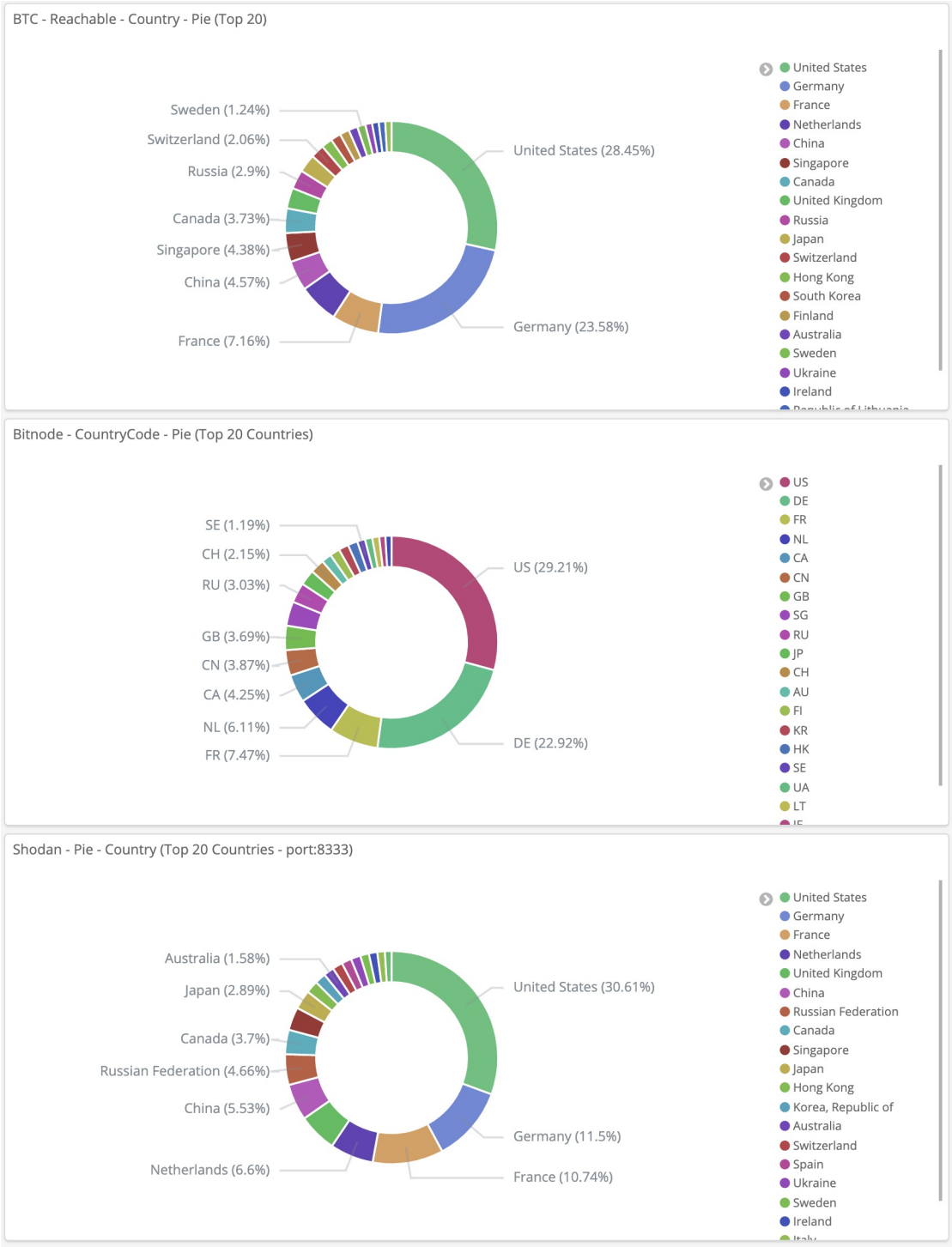


Figure 4.13: Comparison of crawling results of BGLF and Bitnodes and Shodan (port:8333) (top is BGLF, middle is Bitnodes, bottom is Shodan) shown as a list with the top 20 countries for May 14, 2019.

Table 4.2 displays the three data sets for a period of one week in May 2019. To better compare our results with Bitnodes, we only select the reachable nodes from the results of our BGLF crawler. When we look at this table, we see that Bitnodes on average finds a thousand more reachable nodes than our crawler. Results from

Bitnodes are also more stable over time. The results are however trending in exactly the same manner. We have not looked into the differences in detail as it would require an in-depth analysis of the intricacies of the Bitnodes crawling algorithm. We theorize that it may be due to the fact that Bitnodes does not perform a full crawl of all nodes for every update which may smooth results. Furthermore our crawling approach might be seen as too aggressive by the network and might lead to periodical bans from a number of nodes. It is useful to explore this further but we leave this for future research. Due to a technical defect, the data for Shodan is not collected on May 16 2019, hence the value is zero.

Date	***Run ID***	***BGLF***	***Bitnodes***	***Shodan***
2019-05-12	632	8524	9493	3172
2019-05-13	634	8457	9489	3427
2019-05-14	636	8397	9482	3417
2019-05-15	638	8430	9497	3414
2019-05-16	640	8355	9454	0
2019-05-17	642	8276	9431	3511
2019-05-18	644	8211	9397	3752

Table 4.2: Collected data for BGLF (only reachable nodes), Bitnodes and Shodan.

When we look at the collected data for the same sample and differentiate between the reachable and unreachable nodes (Figure 4.14), then we see more dots occurring in China for the unreachable nodes than the reachable nodes. We theorize that this is a result of the Great Firewall of China. It is also clear that the majority of the Bitcoin network consists of unreachable nodes.

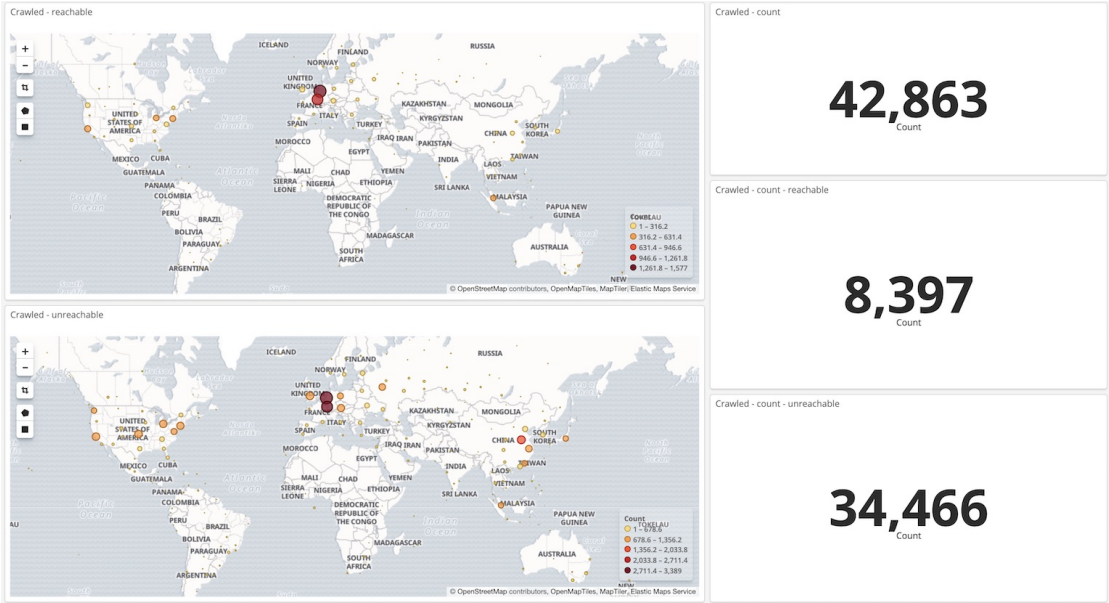


Figure 4.14: Difference between found reachable (upper map) and unreachable (bottom map) nodes (May 14, 2019).

5. Miner discovery by building a mining index

5.1 Introduction

In this chapter we use the information that is collected from nodes by the geo-location application in Chapter 4 and we try to filter and refine this information by adding additional functionality to the application. We then set up an indexing mechanism to score nodes in the network on the likelihood that they are involved in mining activity. Using this score, we improve the first rough technical estimate in Chapter 4 on the geographic location of mining activity.

5.2 Mining activity estimators

Information from nodes is retrieved both by directly requesting it (sending *getaddr* messages) or by analysing communication for residual information (e.g. how fast does the node respond, what does its refusal message imply). We can infer from this set of information whether a node is more or less likely to engage in mining behavior. A number of variables or characteristics seem relevant:

Services Flag
During the handshake process with a node, information is exchanged about the features that that node has enabled. This is a field in the *version* message called 'services'. If we look at the different services that a Bitcoin node can perform in the network, there are quite some variations and we can see that not every Bitcoin node will offer all possible services. When a Bitcoin node has enabled all services then the node is called a full node. Full nodes provide the following services:

- Validating transactions and blocks against Bitcoin's consensus rules
- Filtering transactions on behalf of lightweight nodes
- Transmitting new transactions from users
- Broadcasting new blocks from miners to other nodes
- Relaying transactions and blocks
- Optional: Sending historical blocks to nodes that have been offline for some time

Miners have the need to broadcast their block solution as fast as possible to receive the payment for solving a block. It is therefore unlikely or impossible for a (solo) miner to run a lightweight node. However, as we showed earlier, most mining is done through pools nowadays and different protocols exist for this. It is possible to mine through a mining pool without running a full node. Nevertheless, we assume that a full node has a higher likelihood of being involved in mining activity than a lightweight node.

Fast relay networks

In paragraph 2.3.2 we discussed the existence of fast relay networks. These networks can propagate information faster than the ordinary Bitcoin P2P network. For that reason we make the hypothesis that Bitcoin miners are connected to one or both of the two functioning fast relay networks, the FIBRE and Falcon Relay network. Gencer et al. [29] state in their 2018 study on the level of decentralization in the Bitcoin and Ethereum networks that the Falcon Relay Network is directly connected to 36.4% of the entire hashing power in the Bitcoin network.

Last seen in network

Nodes or hardware involved in mining activity need to have near continuous uptime to be able to mine efficiently. When a Bitcoin node receives a new block it updates the timestamp of the sending node in his peer database. Nodes which have a recent timestamp are active nodes by definition but we hypothesize that they also have a higher likelihood to be up continuously and by extension potentially involved in mining activity. Timestamps are received during the handshaking process as one of the fields in the *version* message.

Node notoriety

Miners benefit when many nodes connect to their node because they need to receive information about newly mined blocks as soon as possible to prevent pointless work and forks. We assume that mining nodes are more often included in the active address list buckets of other nodes. Not only because they are stable and potentially long-running nodes but also because they actively broadcast blocks.

Amount of nodes in the address list

Miners benefit from broadcasting their block solution to as many active nodes as possible to ensure their block is chosen as the new block. The amount of addresses that a node stores cannot be requested directly. However, it can be statistically inferred by querying the node a number of times with *getaddr* messages and by tracking how many of the same IP addresses the node returns every time.

5.3 Creating a Miner Index

The mining activity estimators in the previous chapter can be measured on a scale that is specific for each estimator. These can then be combined into one single mining index that gives a score that denotes the likelihood of mining activity at a certain node. To be able to do this, we need to define the scoring methodology for every estimator.

5.3.1 Services Flag (SF)

The `protocol.h` module of the Bitcoin Core source code [75] holds the the definitions of each possible service, stored in an enum called `ServiceFlags`.

```
enum ServiceFlags : uint64_t {
    // Nothing
    NODE_NONE = 0,
    // NODE_NETWORK means that the node is capable of serving the complete block chain. It is currently
    // set by all Bitcoin Core non pruned nodes, and is unset by SPV clients or other light clients.
    NODE_NETWORK = (1 << 0),
    // NODE_GETUTXO means the node is capable of responding to the getutxo protocol request.
    // Bitcoin Core does not support this but a patch set called Bitcoin XT does.
    // See BIP 64 for details on how this is implemented.
    NODE_GETUTXO = (1 << 1),
    // NODE_BLOOM means the node is capable and willing to handle bloom-filtered connections.
    // Bitcoin Core nodes used to support this by default, without advertising this bit,
    // but no longer do as of protocol version 70011 (= NO_BLOOM_VERSION)
    NODE_BLOOM = (1 << 2),
    // NODE_WITNESS indicates that a node can be asked for blocks and transactions including
    // witness data.
    NODE_WITNESS = (1 << 3),
    // NODE_XTHIN means the node supports Xtreme Thinblocks
    // If this is turned off then the node will not service nor make xthin requests
    NODE_XTHIN = (1 << 4),
    // NODE_NETWORK_LIMITED means the same as NODE_NETWORK with the limitation of only
    // serving the last 288 (2 day) blocks
```

```
    // See BIP159 for details on how this is implemented.
    NODE_NETWORK_LIMITED = (1 << 10),
```

The value of the service flags are binary and obtained by bitwise shifting to the left. The flag that describes a full node is NODE_NETWORK, with value 1. All other relevant flags hold values that result from left bitwise shifting by two or more (except Nothing, which is 0), which means all other flag values are even. The values of all the services that are provided by the node are added to create the services field value that is communicated to other nodes. This means that when the services field holds an odd value, the node is serving the full blockchain and therefore a full node. In figure 5.1 we see the results obtained by storing the services fields when crawling the network.

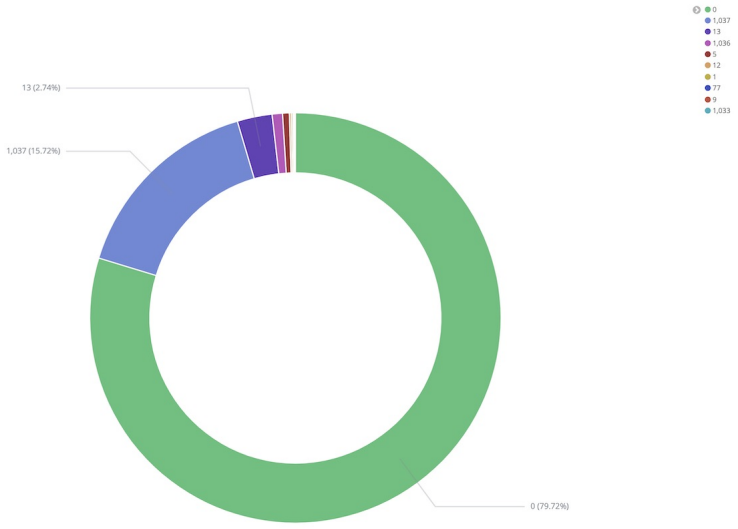


Figure 5.1: Example of found services flags of a crawl.

The vast majority of the pie is taken by the services flag with the value 1037. These nodes are offering the services NODE_NETWORK (binary 1), NODE_BLOOM (binary 10 -> decimal 4), NODE_WITNESS (binary 100 -> decimal 8), NETWORK_NODE_LIMITED (binary 1000000000 -> decimal 1024). The existence of both NODE_NETWORK and NETWORK_NODE_LIMITED may be confusing but what the node is signalling is that it can serve nodes that hold the complete blockchain as well as nodes that only store the last 288 blocks.

Based on this observation we only have to check if the services field has an odd value to determine if the node is a full node. The scoring for this is either 1 (full node) or 0 (not a full node). The procedure is described in Algorithm 1.

Algorithm 1 Calculate Service Index

Input: $NodeSet$
Output: Updated $NodeSet$ with Service Index score added for every node

```
for all nodes ∈ NodeSet do
    if services % 2 == 0 then
        score ← 1
    else
        score ← 0
```

5.3.2 Fast relay networks (FR)

In paragraph 4.2.5 we explained how the platform gains access to the FIBRE network. With this access we can use our crawler to request IP addresses from the address database of the FIBRE network nodes. While there are six FIBRE nodes, we have only been able to connect to four of them. A graph analysis, shown in Figure 5.2 shows that a number of nodes make their appearance in the address database of several FIBRE nodes, which of course is not surprising given the fact that all nodes communicate together.

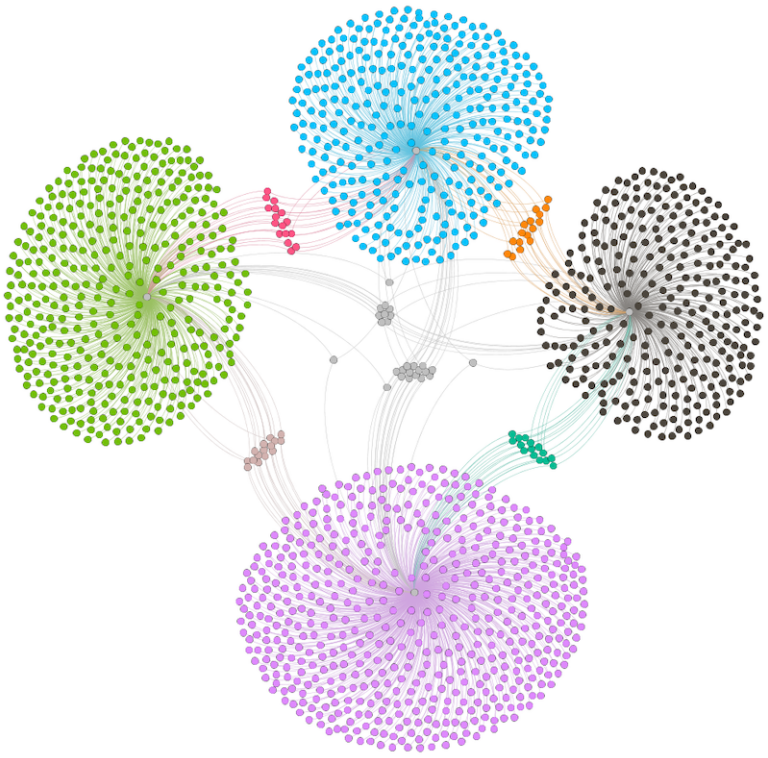


Figure 5.2: Graph visualization of nodes related to the Fibre relay network.

Instead of using a binary score based on whether a node has access to the FIBRE network or not, this allows us to rank nodes based on the number of times we find them in address lists of FIBRE nodes. For this to make sense we need to be reasonably sure that we extract the majority of addresses from the address database, otherwise the results would vary too greatly for each crawling run. For most or all FIBRE nodes this will mean that querying them 6 times is not enough. Therefore we have set the number of *getaddr* requests to 15.

The procedure is shown in Algorithm 2.

Algorithm 2 Calculate FIBRE Index

Input: $NodeSet$
Output: Updated $NodeSet$ with Fibre Index score added for every node

```
1: for all n ∈ NodeSet do
2:     score ← 0
3:     for all FN ∈ FibreNetwork do
4:         if n ∈ addresslist(FN) then
5:             score ← score + 1
6:     divide score by the number of responding FN
```

In paragraph 4.2.5 we also explained how we were not able to access the Falcon network because the network has to initiate a connection. However, by browsing through the `user_agent` strings in the Bitnodes search result, we found 9 nodes that communicated using the `/Cornell-Falcon-Network:0.1.0/` user agent (see mid-left section of the wordcloud of results in Figure 5.3. We collected the IP addresses of these nodes and compared them with the crawling results in our database. Each of the IP addresses had a hit with a crawled node in our database. We therefore applied the same methodology to those Falcon network nodes as we did to the FIBRE nodes.

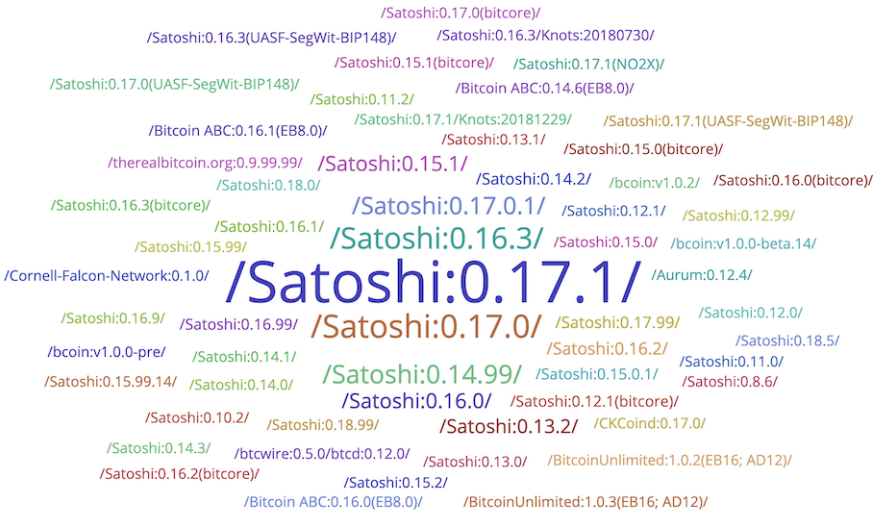


Figure 5.3: Bitnodes user agent wordcloud

We assume these are actual nodes of the Falcon network. However, there might be other reasons why the `user_agent` strings show up in our database as any node owner can change this field. Case in point is our BGLF crawler that runs with its own custom user_agent `/bglf.search.com:0.2/`. We have therefore decided to give a lower weight to the Falcon results. The resulting overall Relay score will be determined for 75% by the FIBRE score and 25% by the Falcon score.

5.3.3 Last seen in network (LS)

The 'Last Seen' score calculates per node what the smallest time difference is between the moment of crawling the peer database of a Bitcoin node and the timestamp of the peer in the database. For easy calculation the result is transformed to a value between 0 and 1. We hypothesize that the closer to 1 the score is, the more active a node will be.

$$LS = \frac{(last_seen - max_age)}{(crawl_date - max_age)} \tag{5.1}$$

Where:

last_seen = The latest timestamp when a node is last seen in peer database
crawl_date = The timestamp at the moment of crawling
max_age = The latest timestamp for which nodes are accepted as eligible by our BGLF program (e.g. 24 hours or a week ago)

The procedure is shown in Algorithm 3.

Algorithm 3 Calculate Last Seen Index

Input: $NodeSet$
Output: Updated $NodeSet$ with Last Seen Index score added for every node

```
1: for all n ∈ NodeSet do
2:     retrieve the set of nodes KnowsN for which holds: n ∈ addresslist(KnowsN)
3:     lastSeen ← 0
4:     for all kN ∈ KnowsN do
5:         retrieve timestamp belonging to peer n
6:         if timestamp > lastSeen then
7:             lastSeen ← timestamp
8:     score ← result from equation (5.1) using lastSeen
```

5.3.4 Node notoriety (NN)

After already having calculated the Last Seen score, it is straightforward to calculate what we call the Node Notoriety Index. This index holds a score that reflects how 'famous' the node is among other nodes, i.e. how well distributed it is among the address databases of other nodes. Mining nodes likely benefit from fame because they are more likely to receive information directly and therefore faster or more often. Miners would want to optimise this effect and the hypothesis therefore is that the more frequently we find a node's IP address in an address database, the more likely it is to involved in mining activity.

To calculate notoriety, a count of address lists that hold a particular node suffices. We scale this outcome between 0 and 1 by using the maximum and minimum count of the set of all address lists. The count of a set of address lists S is denoted by its cardinality $|S|$.

In formal terms, the score is computed by

$$N_i = S_i/(S_{max} - S_{min})$$

where $S_i = |\{A_n|\ i \in A_n, n \in nodeSet\}|$ (in which A_n is the set of all addresses obtained from node n, and $nodeSet$ is the set of all nodes), $S_{max} = max|\{S_n|n \in nodeSet\}|$, and $S_{min} = min|\{S_n|n \in nodeSet\}|$.

The procedure can be seen in Algorithm 4 and is just the first part of Algorithm 3 after which a count is executed instead of a search for the timestamp.

Algorithm 4 Calculate Node Notoriety

Input: $NodeSet$
Output: Updated $NodeSet$ with Last Seen Index score added for every node

1: **for all** $n \in NodeSet$ **do**
2: retrieve the set of nodes $KnowsN$ for which holds: $n \in addresslist(KnowsN)$
3: $score \leftarrow$ count($KnowsN$)

5.3.5 Amount of peers in address list (PE)

When we get two or more more responses from a node when querying it for a subset of its address list, we are likely to receive some addresses multiple times. We assume that different nodes have address lists of different sizes, depending on their age and activity. A node with a very large address database will send out more diverse subsets of 1000 addresses each time. A node with a small address database will send subsets of 1000 addresses that hold more duplicates. We can use the fraction of duplicates in the received sets of addresses from a node to make a statistical estimation of the total size of the node's address list.

For this we use a technique that is called Mark-and-Recapture [76] (also: Capture-Recapture). This technique is typically used in the field of Ecology to estimate the total population of animals. A certain number of animals in the population is captured, marked, and released (marked animals M). Some time later, a new set of animals is captured (second captured group C) and the number of marked animals within that group is counted (recaptured animals R). Under the right conditions, the fraction of marked recaptured animals in the second group should be proportional to the fraction of marked animals in the total population (all animals N):

$$\frac{M}{N} = \frac{R}{C} \qquad \text{or} \qquad N = \frac{CM}{R} \tag{5.2}$$

This formula is called the Petersen estimate of population size. It holds under a number of conditions [30]:

1. The population is closed, N is constant
2. All animals have the same chance of getting caught
3. Marking an animal does not affect the chance to get captured
4. Animals do not lose marks in the period between capture and recapture
5. All marks are reported upon discovery during recapture

Since we do not have to deal with human or animal behaviour in the Bitcoin network, only conditions 1 and 2 seem relevant. They can be loosely translated to:

1. The total number of IP addresses in an address database is fixed
2. All addresses have equal chance to be distributed after a *getaddr* request

In a sufficiently short period, condition 1 will hold. We deem it unlikely that nodes will add new addresses to their database during our series of *getaddr* requests. Condition 2 does not hold completely because the address management modules divides the addresses in different buckets based on a number of criteria but in particular based in whether the nodes are proven to be active or not. The effect of heterogeneity in individual capture probabilities is well-researched, among others in Grim et al. [31], and can lead to over or under estimation of total population size. This can be addressed by using adjusted estimators, often through specialised software [77] that can account for different sources of variation in capture probabilities of animals such as a generic response after having been trapped once (experience) or just completely unique capture probabilities per animal. The latter seems most applicable to nodes in the Bitcoin network. We assume undercounting occurs because the Bitcoin Core address management module prioritizes active peers when populating the list of addresses to send in an *addr* message, leading to more recaptures of those peers than in a purely random selection mechanism.

We are mainly interested in the ranking of nodes and not the exact number of addresses in the database and therefore we can accept undercounting or overcounting as long as it is consistent over all nodes. For now we assume this is the case for the majority of nodes since they all operate in the same manner although very new nodes may show deviations. It would be highly interesting to test and validate mark and recapture estimators using Monte Carlo simulation modelling based on exact knowledge of Bitcoin node address management but this would would take a large amount of time to do well. Therefore we have left this for future research.

Schnabel extended the Peterson estimate to work with more than two samples by determining for every sample:

C_t = number of animals caught in sample t
R_t = number of recaptured (marked) animals in sample t
U_t = number of animals marked for the first time in sample t

whereby without deaths $C_t = R_t + U_t$

The accumulated number of marked animals M is defined as the sum of marked animals over previous samples. Let M_t denote the number of marked animals *just before* sample t:

$M_t = \sum_{i=1}^{t-1} U_i$

The Schnabel estimate for the total population can now be defined by:

$$N = \frac{\sum_t (C_t M_t)}{\sum_t R_t} \tag{5.3}$$

In our research, IP addresses are the equivalent of animals and marking is done by storing all retrieved IP addresses in computer memory against which the next sample is compared. For every node, the required variables are calculated during the successive *getaddr* requests and the Schnabel estimate for the total size of the address database of the node is stored. We then determine the maximum and minimum estimate over all crawled (active) nodes and use this to scale our results between 0 and 1.

For readers interested in this topic we would point to an article by Accettura et al.[32], published in 2015, in which the use of Mark-and-Recapture techniques is explored for computer networks in general. The authors do not focus on crypto-currency networks but they do describe efforts to estimate the size of peer-to-peer networks. Overall it is a good introductory reading into the application of the methodology in the computer science domain. We have not used this article for our thesis but the authors discuss the same methods.

5.3.6 Mining Activity Index

The mining index M_i represents the best indication we can give on whether a Bitcoin node is involved in mining activity. The index score for a node is created by taking the sum of the five index scores in the previous paragraphs, each of which has a scaling between 0 and 1. In our framework, the closer the index is to 5, the higher we estimate the chance that a Bitcoin node is involved in mining activity.

The formula for the calculation of the mining index is:

$$Score_M = Score_{SF} + Score_{FR} + Score_{LS} + Score_{NN} + Score_{PE} \tag{5.4}$$

A example of the index can be seen in Figure 5.4.

fibre_index	falcon_index	node_index	peer_last_seen_index	services_index	total_peer_index	mining_index
0	0	0.5	0.83	1	1	3.33
0	0	0.91	0.83	1	0.36	3.1
0	0	0.93	0.82	1	0.31	3.06
0.2	0	0.95	0.79	1	0.09	2.98
0.2	0	0.92	0.83	1	0.07	2.97
0.2	0	0.92	0.83	1	0.06	2.96
0.2	0	0.94	0.78	1	0.09	2.96
0.2	0	0.92	0.81	1	0.08	2.96
0.2	0	0.89	0.83	1	0.08	2.95
0	0	0.89	0.83	1	0.22	2.94
0	0	0.87	0.83	1	0.23	2.93
0.2	0	0.87	0.82	1	0.09	2.93
0	0	0.89	0.83	1	0.21	2.93
0.2	0	0.88	0.83	1	0.06	2.92
0	0	0.99	0.83	1	0.09	2.91
0	0	1	0.83	1	0.08	2.91
0	0	1	0.83	1	0.08	2.91

Figure 5.4: The Mining Index (Example of May 14, 2019).

5.4 Extending the BGLF application

With the methodology in place we can extend our application with code that captures and stores the necessary information and that creates a mining index score for every node.

The calculation of the mining index is based on the results of a crawl. Therefore the BGLF application as described in paragraph 4.5 is extended with a class CalculateMiningIndex. This class delegates the calculation of the various index scores and the final mining activity index score to dedicated calculator classes for each index. These child classes inherit from the base parent class calculate_index. This base class contains all common methods, such as methods for interaction with Elasticsearch and a memory database. The UML diagram of all classes is depicted in Figure 5.5

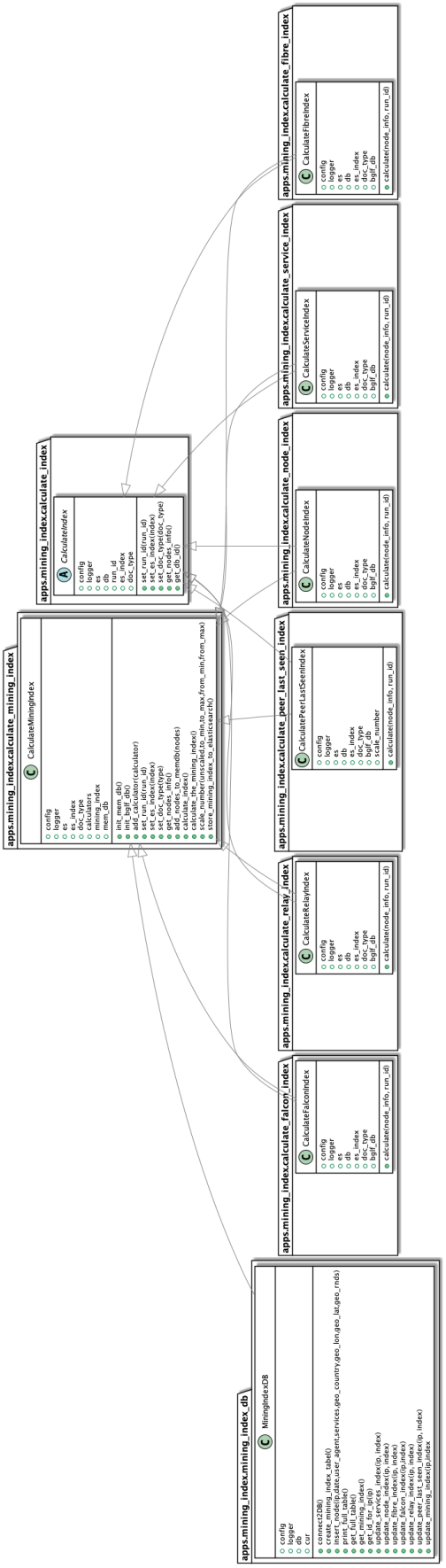


Figure 5.5: UML diagram of classes involved in the calculation of the Mining Index.

All the child classes inherit one calculation method with the same parameter definitions, this is the method calculate(nodes_info, self.run_id). The elegance in this approach is the fact that this allows the different classes to become members of a list that we can iterate and this provides for flexible inclusion or removal of calculators. See code listing 5.1 for a code snippet of the implementation.

```
1     self.calculators = []
2     # Add the different calculator for calculating the index
3     self.add_calculators(CalculateServicesIndex(....))
4     self.add_calculators(CalculateFibreIndex(...))
5     self.add_calculators(CalculateFalconIndex(...))
6     self.add_calculators(CalculatePeerLastSeenIndex(...))
7     self.add_calculators(CalculateIndex(...))
8     self.add_calculators(CalculateTotalPeerIndex(...))
9     ...
10    # Calculate every index separately
11    for calculator in self.calculators:
12        calculator.calculate(nodes_info, self.run_id)
13    ...
14    # Calculate the index
15        self.calculate_the_mining_index()
```

Code Listing 5.1: Calculate the mining index

The interim results are saved in a in-memory database. This database is created when the calculation is started and removed after calculation is completed and results have been stored in Elasticsearch. The mapping for the index in Elasticsearch can be found in Appendix C.3.6.

Analysis and visualization of the mining index can easily be done with Kibana. An impression of the type of analysis that can be done is shown in Figure 5.6. To create this dashboard the data is filtered to show only the nodes from one specific run ID. The results are projected on a world map. Next to the map, the top 10 countries are shown in a pie chart.

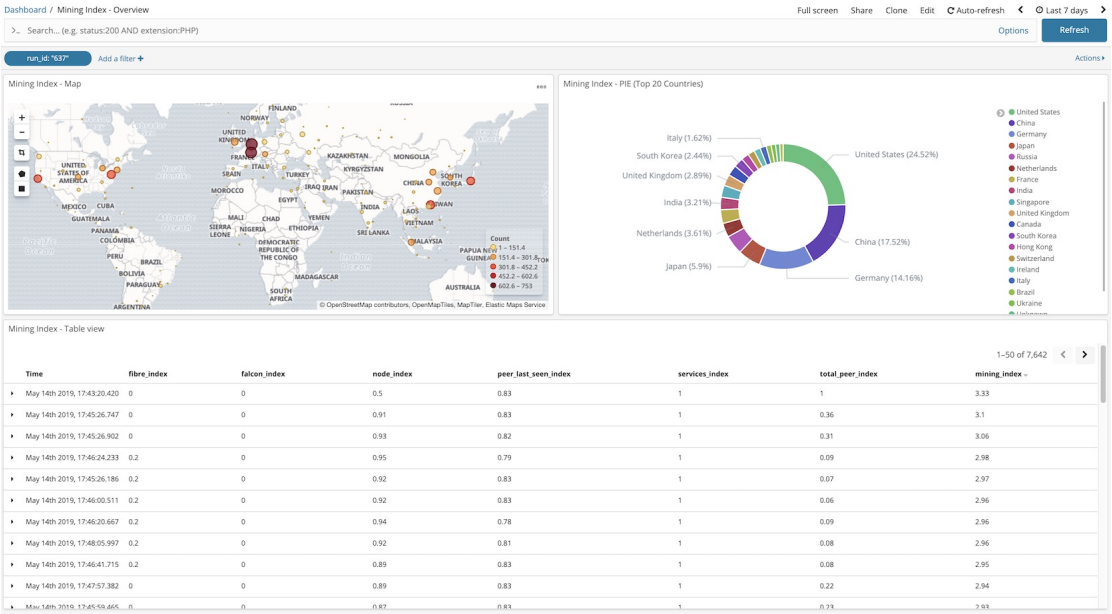


Figure 5.6: Dashboard for Mining Index results (May 14, 2019).

5.5 Discussion of results and validation

Figure 5.7 shows the mining index MI_i for the nodes identified by our crawler (sorted by mining index). The figure shows a clear distinction between four groups of nodes: a first group of about 7,500 nodes that have a mining index above 2.0; a second group of about 2,500 nodes that have a mining index between 1.0 and 2.0; a third group of about 20,000 nodes that have a mining index between 0.5 and 1.0; and, a fourth group of about 10,000 nodes that have a mining index below 0.5.

The figures in 5.8 show the scores of the five indexes *services flag* SF_i, *fast relay networks* FR_i, *last seen* LS_i, *notoriety* NN_i, and *total peers* PE_i. Also a sixth score is added, the *node reachable score*. This score is not part of the mining index calculation, but is added to check for the possibility that high mining index scores are just a reflection of nodes being reachable.

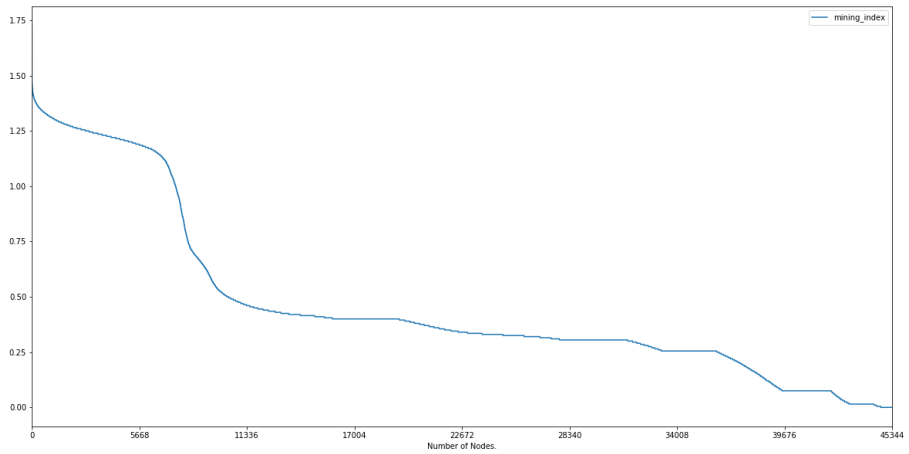
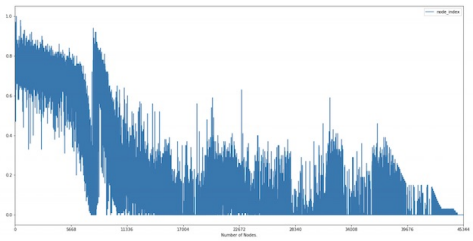
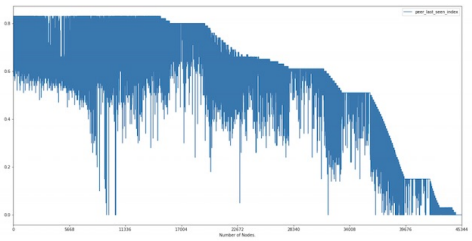


Figure 5.7: The Mining Index (May 14, 2019).

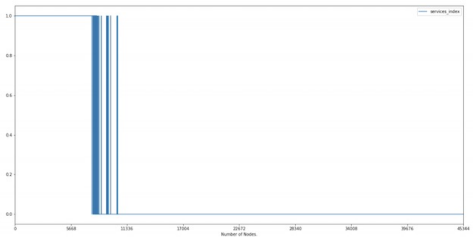
The *services* index is most distinguishing. It clearly has a binary nature since a node is either a full node or not. The *fast relay networks* index is less distinguishing. It is very likely that mining nodes are connected to fast relay networks. The figure confirms the results shown in Figure 5.2 and indicates that most mining nodes are connected to a single relay node. The *last seen* index is least significant, but confirms that the full nodes connected to fast relay networks also have a high score for this index. The *notoriety* index also is clearly distinguishing and follows the trend of the *services* and *fast relay networks* indexes. The *total peers* index is to a large extent similar to the *fast relay networks* index. It is reasonable to assume that nodes with a mining index above 2.0 are a lot more likely to be involved in mining activity than nodes with a mining index below 0.5. The situation is less conclusive for nodes with a mining index between 0.5 and 2.0. To make this somewhat visible, all mining index results are plotted in Figure 5.9 and the nodes with a score higher than 2 are plotted in Figure 5.10.



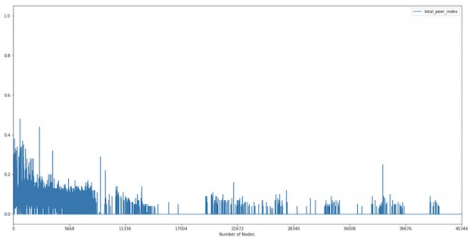
(a) Node Notoriety Index



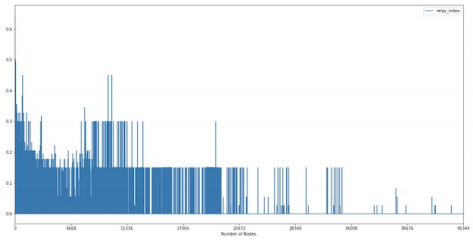
(b) Peer Last Seen Index



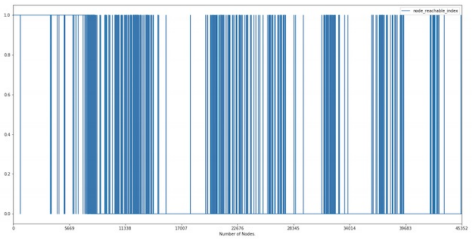
(c) Services Index



(d) Total Peer Index



(e) Relay Index



(f) Node Reachable Index

Figure 5.8: Overview of the different indexes.

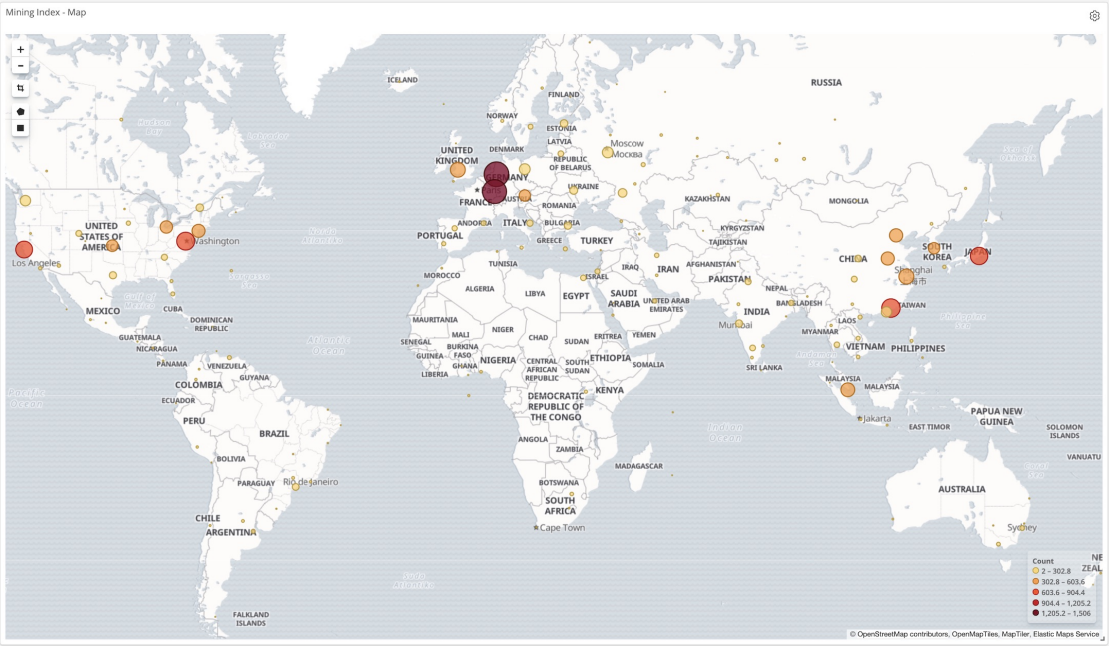


Figure 5.9: All Mining Index results (May 14, 2019).

Figure 5.10: Mining Index results for values of 2 and higher only (May 14, 2019).

As a final note, in Western Europe we often see see two large dots on the map. This is a consequence of the automatic aggregation of results in Kibana when maps are zoomed out rather than proof of high Bitcoin mining activity in the middle of Germany and The Netherlands. We present a more detailed map of the area in Figure 5.11 to show this.

Figure 5.11: Detailed view of the Mining Index in Europe (May 14, 2019).

6. Miner Discovery outside the Bitcoin P2P network

6.1 Introduction

In this chapter we answer our third research question and identify and analyse sources outside the Bitcoin network that provide further clarity on the location of Bitcoin mining activity. We focus on two distinct means of data gathering, namely the retrieval of information through OSINT tools and the extrapolation of geodata from mining pool gateways.

6.2 OSINT

OSINT is an acronym for Open Source Intelligence. While this can mean almost any open information gathering, in this context it is limited to the Internet. In today's world many devices are connected to the Internet and these devices can be traced by search engines that use a variety of filters. A number of organisations exist that crawl the Internet, scan open ports (e.g. 8333) and gather information about web services and connected devices. As Bitcoin mining is inherently a connected process, we extend our application platform with the ability to search the internet for information on IP addresses that can be linked to Bitcoin mining related services or hardware. We do this by querying the API of Shodan.io and Censys.io, two well-known OSiNT platforms.

6.2.1 OSINT platforms

Shodan.io

Shodan.io [78] is a search platform for internet-connected devices (webcams, routers, servers, etc.) The systems related to the platform crawl the internet looking for directly connected devices and query these devices for publicly available information. This is different from a typical search website like Google which only crawls the World Wide Web.

Most of the information is retrieved from banners. Banners are metadata about the services that are running on a device. Those services are what Shodan collects information about. For example, websites are hosted on devices that run a web

service. Shodan will gather information by trying to communicate with that web service. The information for each service is stored in an object called a 'banner' which is retrieved when communicating or listening.

To quote from a short introductory blog by Kapersky Lab [79]:

"In a certain sense Shodan is like a guy who walks throughout the city and knocks on every door he sees. But instead of doors Shodan 'knocks' on every IPv4 address, and instead of some city there is the whole world."

Censys.io

Censys.io [80] has a similar method of operation as Shodan. It scans the IPv4 space, looks for services and gathers the provided banners. Censys started as an academic research project at the University of Michigan but has spun out from the University to become an independent commercial company. Censys typically shows fewer results than Shodan, this is a deliberate difference in presentation. Censys only shows the current state of the Internet and prunes out hosts that were not seen in the last week whereas Shodan shows all results from its database. According to their website FAQ, the company is aggressive in pruning out old data because this allows users to accurately understand how the Internet is changing over time and it prevents overcounting. Censys is more limited compared to Shodan, because the company only scans a select number of ports. Detailed information on the workings of Censys and the differences in results with Shodan can be found in their 2015 research paper [33].

6.2.2 OSINT information gathering

By using the right filters in the two search engines, it is possible find very specific services. These search filters are colloquially called 'dorks'. As an example, a search for 'Antminer' leads to the discovery of 1596 IP addresses (March 2019). These addresses can be identified because the Antminers are operated through a web server. The communication protocol for authentication and configuration uses the words "antMiner Configuration". We therefore find only Antminer activity that is operated through a web server. However, if we make the assumption that there is no geographic bias in the manner of operating mining hardware, this should give a good estimation of Antminer activity throughout the world. Figure 6.1 shows the results of this query on 17 March 2019.

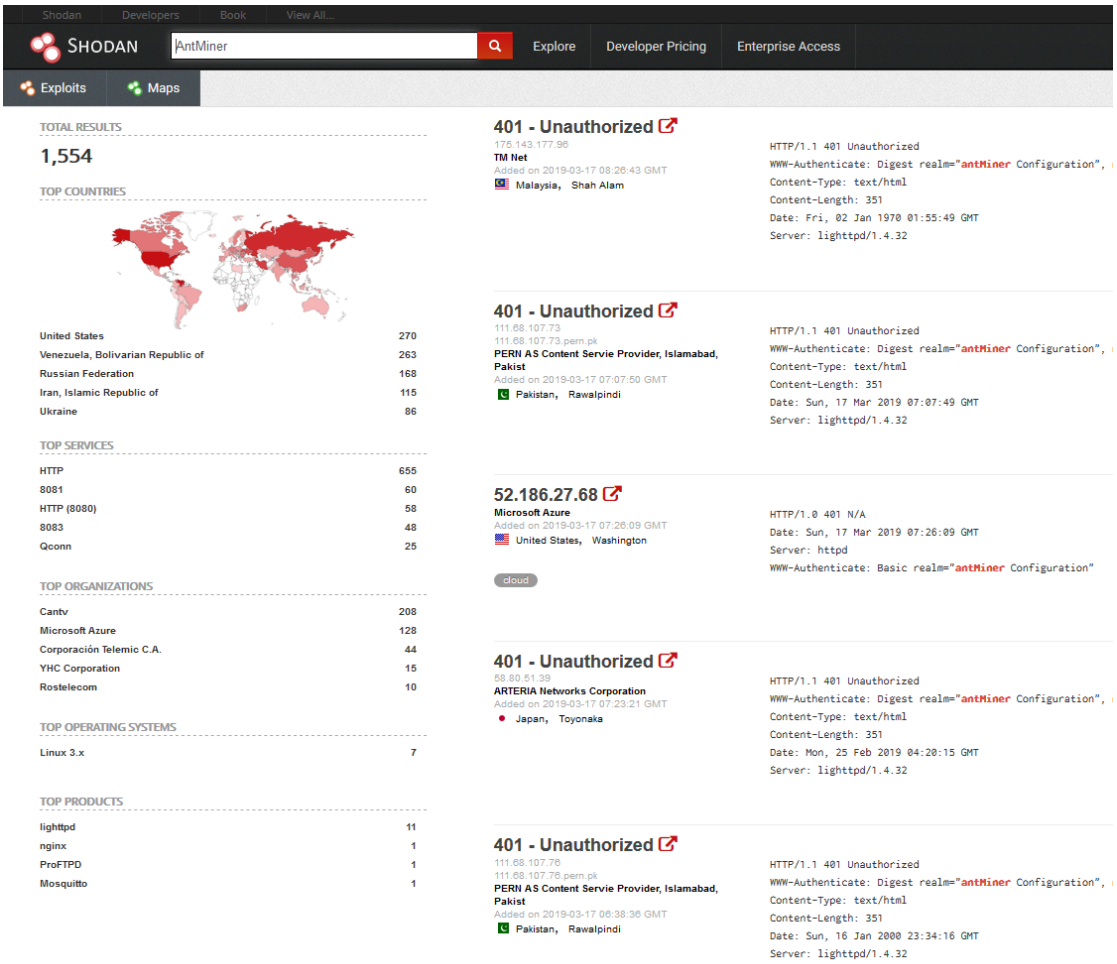


Figure 6.1: Shodan default search interface using the 'AntMiner' keyword

Running the same query with Censys (April 16 2019) gives us the results shown in Figure 6.2

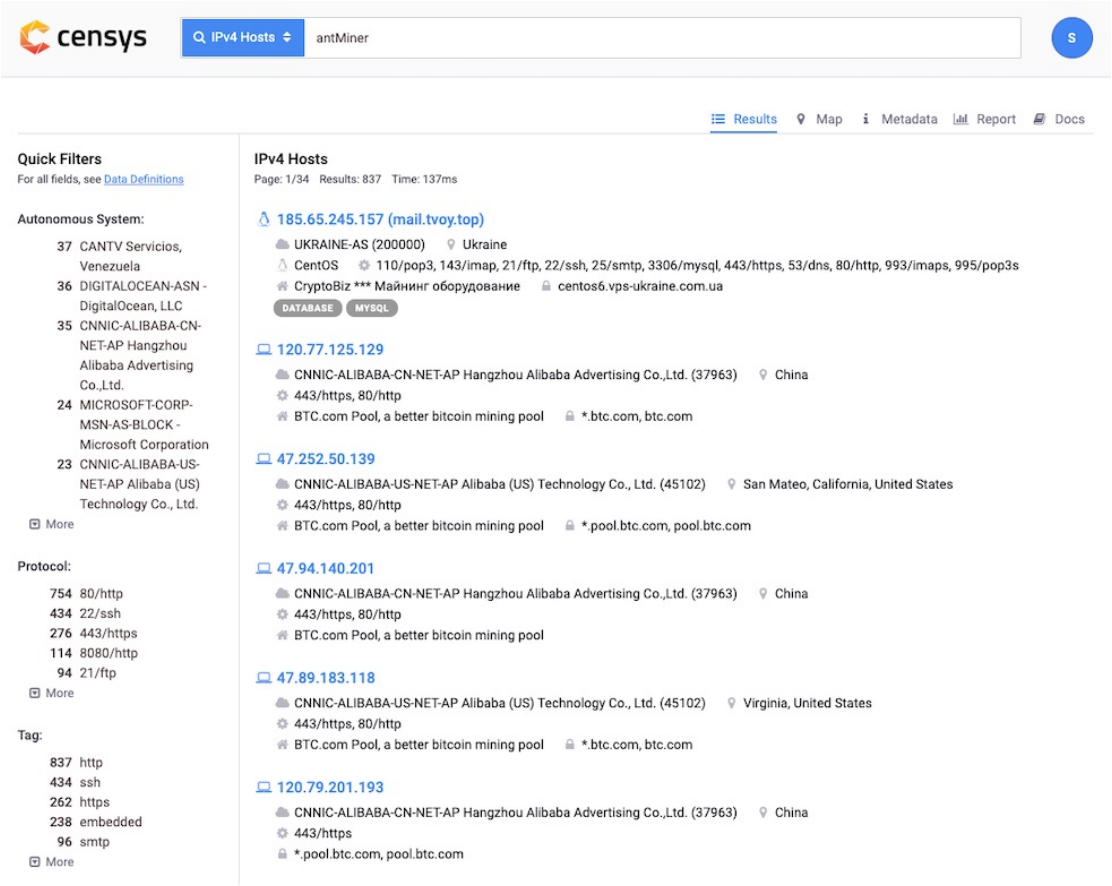


Figure 6.2: Censys default search interface using the 'AntMiner' keyword

The trick to finding relevant information is composing the right search strings or dorks, This requires in-dept understanding of communication protocols and services. In the Antminer example, we were able to retrieve information from Shodan because the word 'Antminer' is in the main response by the server itself which is stored under the **data** property, the only property that is searched by default. Other properties can be searched by specifying their filtername. For instance, a search for all routers in The Netherlands with username and password combination of 'admin' and '1234' respectively can be performed by the search string "admin+1234 port:8080 country:NL".

Shodan restricts queries using filters to paying users and academic researchers. Censys limits the number of queries that can be executed per month and the results returned per query. Both organizations have generously provided us with research access upon request.

6.2.3 OSINT search strings and results

The selection of the right dorks or search strings is a mix of trial and error and careful consideration. We can filter the IP addresses that are returned from the various searches which means there is no risk of double counting and therefore we can just try to retrieve as much information as possible to validate and augment the results from the previous chapters.

Given our initial tests with Shodan and Censys, the names and processes tied to the different ASIC miners are the most likely candidates to return a large number of hits. From both Shodan and Censys we already requested information on the use of the Bitmain 'Antmimer' ASIC mining hardware, leading to a large number of hits.

We also checked for the hardware series of other manufacturers like Halong's 'DragonMint' and Canaan's 'AvalonMiner' but we got no or only very few hits. This probably means that the series name is not used in the web service messaging.

We tried a large number of additional search strings related to Bitcoin mining activity, both for pools and for solo mining, but the results were disappointing. This may mean that the large number of results for 'Antminer' is just a fluke but we cannot be certain about this. Webservices for mining pool operators and contributors as well as for mining hardware in general may use words, phrases or a combination of characters that are so specific that they can lead to identification of mining activity. Much deeper research into the communication protocols might uncover this but unfortunately this is beyond the scope of this thesis.

The following list states the four most relevant search strings for which we did retrieve IP addresses. The first was found using both Shodan and Censys. For the last three we were only able to use shodan because the results are related to specific ports. Censys mainly scans port 80 and 8080 as well some other frequently used ports whereas Shodan tries to collect information for every port.

- *"Antminer"*
  This search string returns results from a search in the HTTP response, especially in the *WWW-Authentication* message. A response example is: "WWW−Authenticate: Digest realm="antMiner Configuration", nonce="f34e090316397f78d8b3c03ee6685c5e", qop="auth""

- *"bitmain-inc"*

Bitmain is the manufacturer of Antminer ASIC harware. This search string reveals *NetBIOS*[1] and *LADP*[2] responses. An example of a *NetBIOS* response is:

```
NetBIOS Response
Servername: CNBJAPP03

Names:
    CNBJAPP03 <0x0>
    BITMAIN-INC <0x0>
    CNBJAPP03 <0x20>
```

Both these protocols are used inside computer networks. The return of hits on this search string confirms that a Bitmain computer network is operated at the scanned IP address.

- *"stratum+tcp"*
  Stratum is the protocol that handles communication between a miner and a mining pool [42], see also Figure 2.2. The hits for this search string may reveal mining pool servers or gateways or mining pool contributors.

- *"port:3333"*
  The default TCP port of the Stratum protocol is *3333* [42]. Hits for this search string may again reveal mining pool servers or gateways or mining pool contributors.

- *"port:8333"*
  The default TCP port of the Bitcoin protocol is *8333*. Hits for this search string may reveal Bitcoin nodes.

Not all results are equally valued by us. The results of the search for 'Antminer' hardware are most useful to use as these are unequivocally related to mining. However, a small caveat here is that Bitmain has started to produce Antminer ASIC hardware for other hashing methods also (i.e. not only for SHA-256) which means that some results may be related to the mining of cryptocurrencies other than Bitcoin.

The results for "Bitmain-inc' are interesting and could very well point to mining farms. However, Bitmain is a large organisation and the results could also be related to network servers in the company's office.

[1] NetBIOS is an acronym for Network Basic Input/Output System it provides communication services on local networks.

[2] Lightweight Directory Access Proctol (LDAP) is an application protocol for accessing and maintaining distributed directory information systems. LDAP is commonly used to provide a central place to store usernames and passwords. Which allows different applications and services to connect to a LDAP serverto validate users

The results for Stratum and port 3333 are very interesting and we would love to dig into the protocol a bit deeper, if we had more time, to try to filter these results or retrieve additional results. It is important to note that Stratum is not only used for Bitcoin mining pools but also for other cryptocurrency pools. By extension, the same holds for port 3333. In addition to this, port 3333 is also related to some other relatively obscure services that have nothing to do with cryptocurrencies [81].

6.2.4 Extending the BGLF application

To be able to automatically query both OSINT platforms and store the results, the BGLF platform is extended with a class ShodanScraper and a class CensysScraper. These classes query the API's of Shodan and Censys. A generic class Scraper is defined, from which both classes inherit. This class holds the common methods for connecting to the API's. The UML structure for the parent and child classes is shown in Figure 6.3.

Next to the classes Shodan and Censys also the class BitnodesScraper shows up in the UML diagram because querying the API of Bitnodes follows the same methodology. The Bitnodes data is used for validation purposes in Chapter 4 and will be addressed in paragraph 6.2.5.

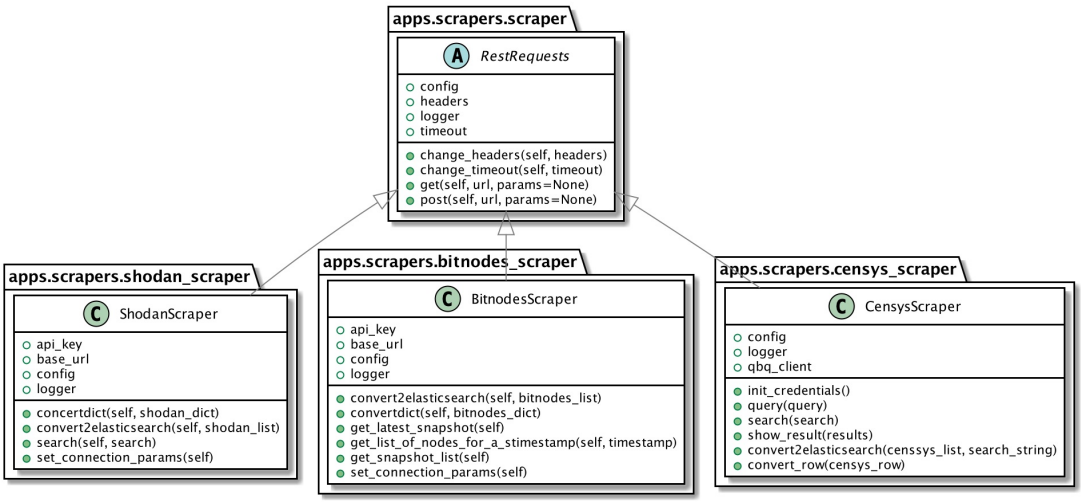


Figure 6.3: Interface design for data extractors.

The parent class RestRequests contains the necessary HTTP code that is necessary to communicate with a REST API and to retrieve the results. This is done in the form of a GET request [82]. By implementing a generic class for the GET request we have more control over the request and the error handling.

Shodan.io

The ShodanDataExtractor contains a specific *search* method with a parameter for the search string. This method queries the Shodan API endpoint. The specification is the following:

```
https://api.shodan.io/shodan/host/search?key={YOUR_API_KEY}&query={query}&
    page=(page_number)
```

This request contains three parameters. The first parameter is the API key. Linked to this key is the academic (educational) access that was granted to us. With this access we are allowed to not only query the API, but also retrieve more than 1000 results, something that cannot be done without a key. Shodan uses a credit point system to keep track of the total number of allowed queries for an account. In the 'development' dashboard the consumption of these credits can be followed, see Figure 6.4 for a overview of the consumption until 16 April 2019.

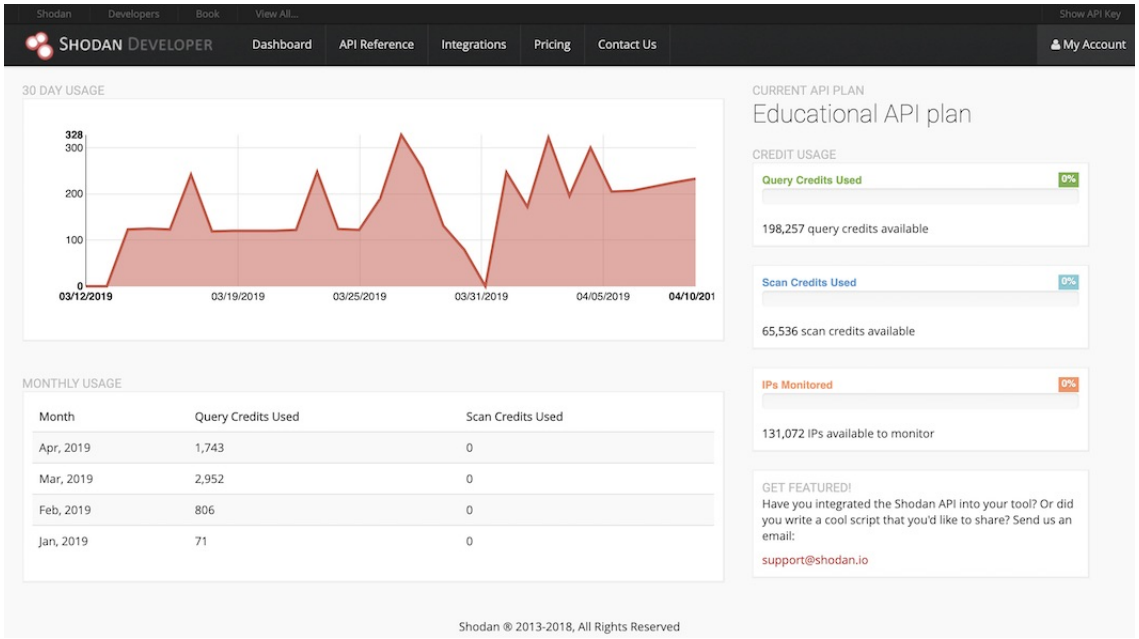


Figure 6.4: Shodan developers (academic access) dashboard.

The second parameter is the search string which we will elaborate on later. The function of the third parameter is to add pagination because the API only returns the first 100 results. The JSON file that is returned when we execute an API request holds a field with the total number of results which we can use to specify the extent of pagination. See Appendix D.4.1 for an example. The results are stored in a separate index in Elasticsearch which can be found in Appendix C.3.4. Before storing the results, the data is converted to the static index mapping of Elasticsearch.

An impression of the collected information from Shodan is shown in Figure 6.5.

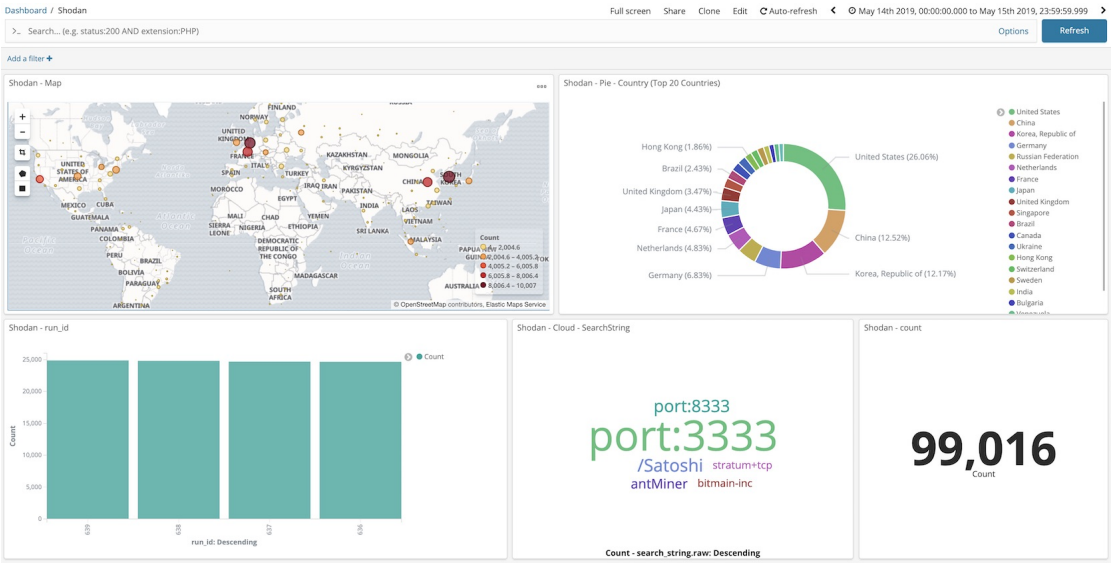


Figure 6.5: Kibana dashboard for the Shodan results (May 14, 2019).

Censys.io

Censys makes its data available through Google BigQuery [83] [84]. BigQuery is a data warehouse for storing and querying massive data sets. This can normally take hours but BigQuery solves this problem by enabling very fast SQL queries using the power of Google's infrastructure. Censys makes 4 data sets accessible via BigQuery:

- *Certificates.* A certificates table that contains all unique X.509 certificates found during any of the scans.
- *IPv4 hosts.* Information about current state of known IPv4 hosts and how public protocols are configured
- *IPv4 Banners.* Results from Censys' daily light-weight scanning
- *Popular Websites.* A daily snapshot of the current state of the Alexa Top Million Domains collection.

The *IPv4* data set (Censys does not have an IPv6 data set) is the data set we query when we are sending search strings related to Bitcoin mining activity. The SQL query to search for "antMiner" is :

```
#standardSQL
SELECT
  ip, ports, autonomous_system.asn, autonomous_system.description, location.
      country_code, location.latitude , location.longitude, location.city
FROM
  `censys-io.ipv4_public.current`
WHERE
  p80.http.get.headers.www_authenticate LIKE '%antMiner%'or p8080.http.get.
      headers.www_authenticate LIKE '%antMiner%';
```

Code Listing 6.1: Censys Google BiqQuey SQL query.

The Google BigQuery platform has a development environment for testing and writing queries. The environment is shown in Figure 6.6.

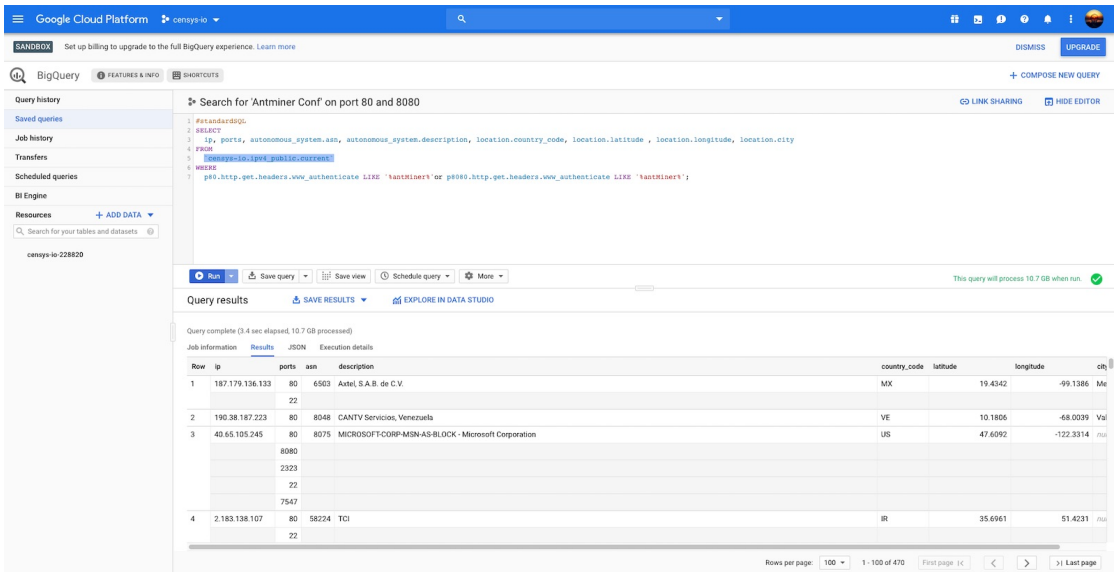


Figure 6.6: Censys google bigquery platform (April 16 2019).

Google BigQuery has two ways to interact with the data sets. The first one is via a REST API and the second one is via a client library (google.cloud python library). Although at first sight it seems obvious to implement an API request tool because it mimics what we do with our ShodanScraper, we decided to work with the client library. The reason for this is the simplicity of authentication when making use of the library. Furthermore, the SQL queries we developed for Shodan can be used in the exact same manner. See Appendix E.2 for the Censys Python code that we developed to query Google BigQuery.

One of the goals for the BGLF tool is to run it without a lot of added complexity and as such we prefer to implement as little user interaction as possible. To achieve this level of automation, we required a Censys Service Account. After reaching out to Censys, the organisation once more accepted our research status and added the service account to our verified Google account.

6.2.5 Bitnodes API data retrieval

A side step in this chapter is the collection of information via the Bitnodes REST API [85]. Similar to the classes ShodanScraper and CensysScraper, we have made a class BitnodeScrapper that inherits the methods for retrieving data data through a HTTP GET request from the parent Scraper class.

The Bitnodes API has different endpoints. The most import one for our project is:

```
https://bitnodes.earn.com/api/v1/snapshots/latest
```

Via this endpoint we retrieve the latest collection of reachable Bitcoin nodes which are stored in our Elasticsearch database. The mapping of this data in can be seen in Appendix C.3.3. An example response for calling this API can be seen in Appendix E.3.

6.3 Mining Pool Gateway Extrapolation

Figure 2.1 in paragraph 2.2 shows that the majority of Bitcoin mining is done through pools. Mining pools often have pool servers deployed all around the world to reduce latency and provide redundancy. Miners are automatically routed to the nearest located server or they can choose to connect to a specific regional gateway. It is likely that miners will always connect to the nearest gateway to optimize their results. We use these insights to create an additional source of information that focuses solely on mining pools as opposed to the Bitcoin P2P network.

From the websites of the 14 largest mining pools (see Table 6.1), we retrieved the host names of the servers that clients should use to connect to the pool. We next retrieved the corresponding IP addresses by DNS-lookup, and for some host names multiple IP addresses were resolved. The results are shown in Figure 6.7 and Figure 6.8. Note that the locations of some servers nearly coincide, which is not visible in the graph.

No	***Pool***	***Note***
1	BTC.com [86]	
2	AntPool [87]	Mining Nodes AntPool has deployed many nodes all over around the world to improve performance and provide redundancy. Miners are automatically routed their closest location, including United States, Russia, Germany, Hong Kong, Dubai, England,Beijing, Shenzhen, Qingdao, Hangzhou and more.
3	F2Pool [88]	
4	Poolin [89]	
5	SlushPool [90]	
6	Unknown	No info, this could be solo miners and anonymous pools
7	ViaBTC [91]	
8	BTC.TOP	Needs registration to get Stratum server info
9	BitFury	No public clients
10	DPOOL [92]	
11	Bitclub	Needs registration to get Stratum server info
12	Huobi.pool [93]	
13	Bitcoin.com [94]	
14	WAYI.CN	
15	Bixin [95]	

Table 6.1: Top 15 Mining Pools of May 2019

Figure 6.7: Mining Pools based on hostname (May 7 2019).

Figure 6.8: Mining Pools based on IP address (May 7 2019).

In Appendix F we present a table and maps with the results for each pool. By not only looking at the host name of the servers but by also requesting IP addresses from the host, we are able to discover more locations. In particular Slushpool generates a fair amount of additional IP addresses this way.

Our results coincide with results from the 2018 Cryptoasset Benchmarking Study [21] which reports on 56 mining pools, and indicates that as of mid-November 2018, these pools are located in Asia-Pacific (44%), North America (29%), and Europe (25%), and a small part in South America (2%).

6.4 Discussion of results and Validation

In this chapter we have answered our third research question “What other sources are available that can help identity and quantify Bitcoin mining activity/” by looking at OSINT platforms and Mining Gateways.

With the data extraction from Shodan and Censys we have found a way to find IP addresses that are linked to the use of Bitcoin mining equipment and services. This information is distinct from the information that was gathered in Chapter 3, 4 and 5 because the methodologies focus on different aspects of the Bitcoin network. The differences are shown visually in a Kibana dashboard in Figure 6.9.

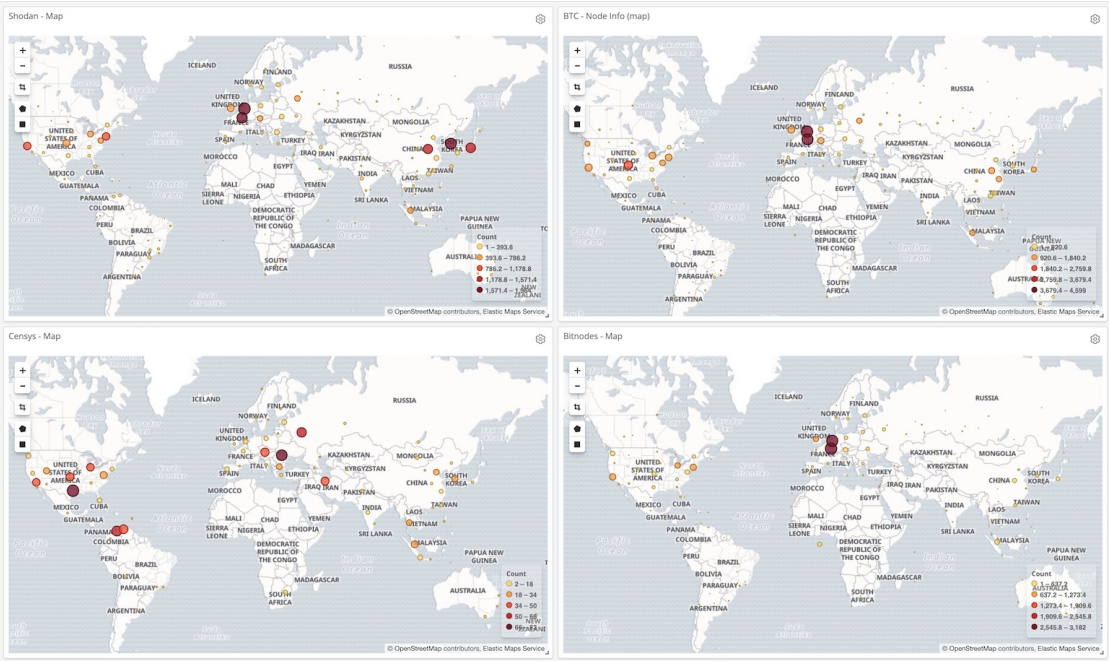


Figure 6.9: Kibana dashboard with maps of Shodan (upper left) BGLF (upper right), Censys (bottom left) and Bitnodes (bottom right) data (April 28 2019).

The four maps show clear differences which means there is significant value in looking at sources outside the Bitcoin P2P network itself. Censys finds more hardware related to mining activity in Venezuela, Iran, Romania than Shodan. On the other hand, the combined results of Shodan shows more devices in China, South Korea and Japan. When the maps of Shodan and Censys are compared with the maps of the results of BGLF, we can see that completely new regions with Bitcoin mining activity have been identified.

The results of the search strings "*stratum+tcp*" and "*port:3333*" likely identify mining pool activity. It is unfortunately beyond the scope of this project to investigate this in further detail and we leave this for future research. Maps displaying the results for both queries can be found in Appendix E. Figure E.3 holds the results for port 3333 and the majority of results is found in South Korea, Japan, China and the United States. Figure E.4 displays the results for "*stratum+tcp*" and paints a slightly different picture. Here the majority of IP addresses is found in the United States, Germany, Canada, China and the UK.

The search results from Shodan and Censys for the search string 'Antminer' are shown in Figure 6.10 and Figure 6.11. These maps give a detailed overview of the location of dedicated mining hardware. Both OSINT platforms confirm mining activity in the United States, Venezuela, Iran, China, Russia, Romania and Singapore.

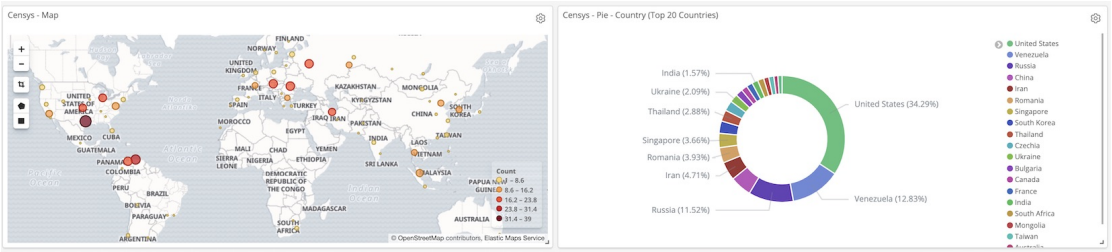


Figure 6.10: Censys search results for 'antminer' (May 11, 2019).

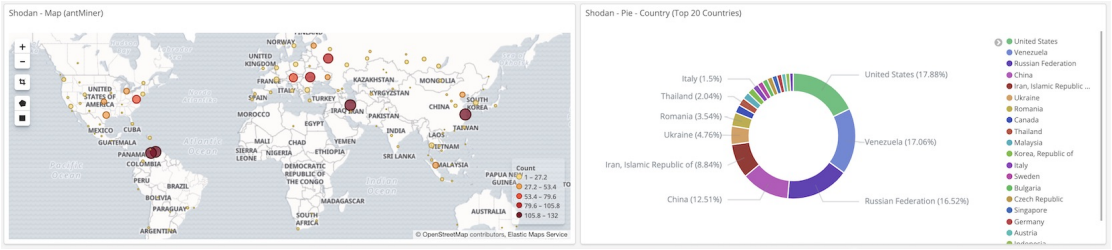
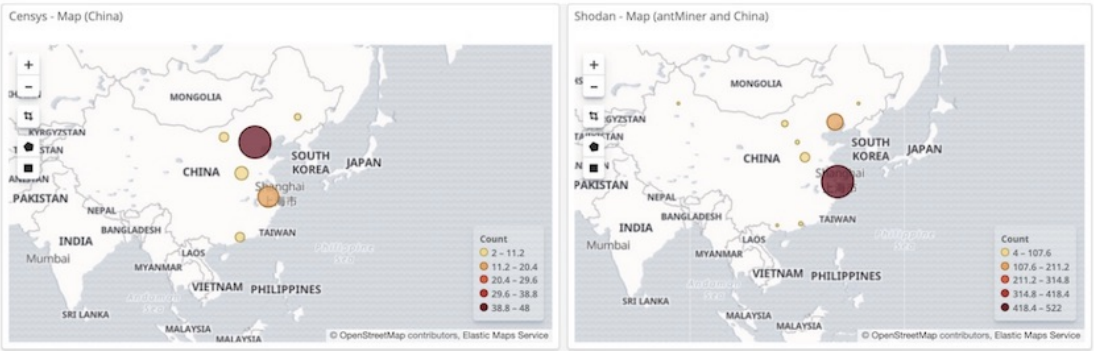
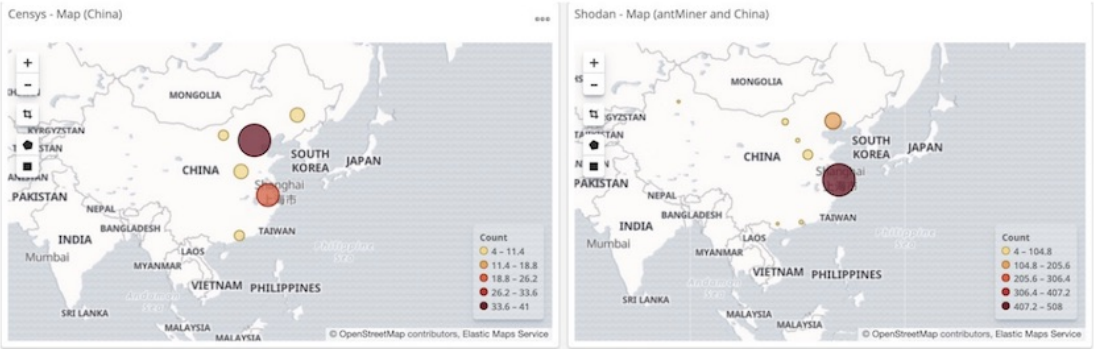


Figure 6.11: Shodan search results for 'antminer' (May 11, 2019).

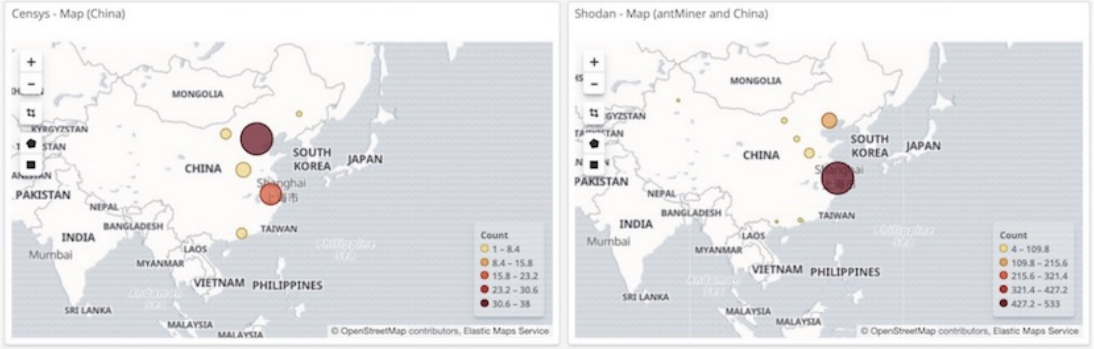
To test whether our results held over a longer period of time we made weekly Shodan and Censs requests for "Antminer' over a period of 4 weeks. The results are shown in Figure 6.12. As can be seen, there is almost no difference in results between weeks.



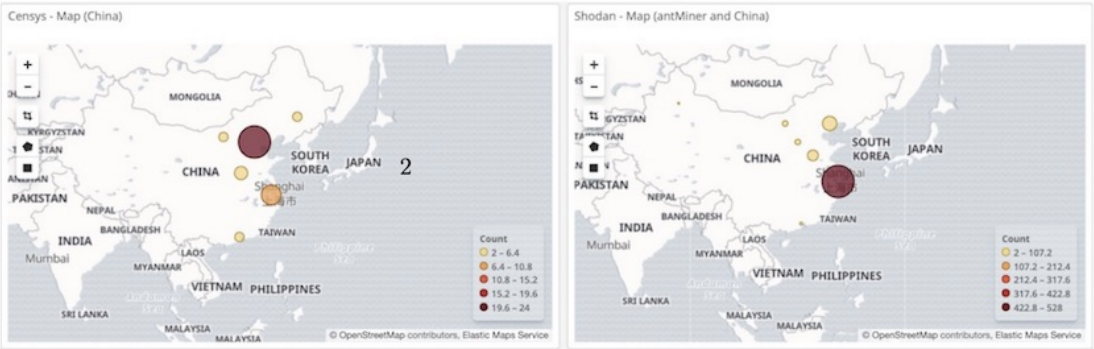
(a) April 23, 2019



(b) April 30, 2019



(c) May 7, 2019



(d) May 15, 2019

Figure 6.12: Comparison OSINT for 'antminer' in China.

7. Conclusions, ethical considerations and future research

7.1 Overview

In the academic field limited research exists on the the geographical distribution of Bitcoin mining. Our goal for this thesis was to contribute to the research field by analysing existing data sources and finding new ones in a format that can be replicated. We have achieved this goal through the development of the application Bitcoin GeoLocation Finder (BGLF) and the associated platform. The BGLF tool provides the functionality to crawl the Bitcoin peer-to-peer network and to collect a variety of information. On top of this, it refines this information by constructing a Mining Index based on several parameters that acts as a filter to increase the likelihood that results are indeed linked to mining activity.

We have also implemented functionality to extract data from two OSINT platforms, Shodan and Censys, from the Bitnodes project and from Mining pool gateway servers. Each of these data sources provides additional results and added value to the total platform and research goal.

The overall platform that facilitates all our research is built on the server environment of the Open University. The platform makes efficient use of Docker container software, an Elasticsearch database and Kibana analytical tools and dashboards. These tools are all built with modern open source software, which makes our platform future-proof and very maintainable.

7.1.1 Revisiting the research questions

At the start of this thesis we defined our main research question in 1.2: “How can Bitcoin mining activity be identified, quantified and located?”

This was subsequently broken down into three separate questions. The first of these stated:

“What information on Bitcoin mining activity is readily available from public sources?”.

We provided the necessary information to answer this question by looking at a number of different sources in Chapter 3. The first is an in-depth search on the internet about known locations where Bitcoin mining activities take place. We try to reliably identify Bitcoin mining locations by requiring that information about the locations is found in at least three different sources and has a date no earlier then 2018. This information is presented in a handmade map with details on power consumption and energy source. The second source of information is the collection of IPO (Initial Public Offering) documentation of large Bitcoin mining hardware manufacturers that can also act as miners themselves. We indeed retrieved useful information from these documents but established that only Bitmain Technologies had disclosed it was involved in mining on a large scale. This does not mean the other manufacturers are not involved, and in fact we suspect they are, but it does mean that the legal entities to which the IPO documentation referred were not involved. A third source was the Cryptoasset Benchmark Report from the Cambridge Centre for Alternative Finance (CCAF) which held a wealth of information on cryptocurrency mining but to a much lesser extent held information specific to Bitcoin mining.

The second research question stated:

"Can mining activity be identified on the Bitcoin network and filtered from the large amount of other information?"

We answered this question in Chapter 4 and 5. First we built a software application to crawl the Bitcoin peer-to-peer network, based on the existing Bitcoin protocol module of the Bitnodes project. We developed the platform from which to run this crawler including data storage and visualisation of its results, culminating in the combined tool that we named Bitcoin Geolocation Finder (BGLF). What sets this tool apart from all other endeavours in this field is that it can connect to both the IPv4 and IPv6 part of the internet. Furthermore the tool initiates a connection attempt to both reachable and unreachable nodes and retrieves and infers information from both.

Having extracted as much information from the nodes as we could with the crawler, we theorised and built a mining index to filter this information. By scoring nodes on the mining index and by using only the nodes above a certain threshold value, we increase the probability that a node is involved to mining activity. The mining index is built up solely from data that was collected using our crawler tool. Through smart use of database queries and novel use of the mark-and-recapture statistical technique from the Ecology field, we were able to expand this initial dataset. The results from the mining index do not provide conclusive evidence that we are predominantly dealing with nodes involved in Bitcoin mining. However, we provide clear reasoning why we increase at least the likelihood of finding mining activity.

Our third and last research question stated:

"What other sources are available that can help identify and quantify Bitcoin mining activity?"

The answers to this question are detailed in Chapter 6. We describe how we expand the BGLF tool with functionality to retrieve data from the OSINT platforms Shodan.io and Censys.io. We query their API to find internet-connected devices that are related to Bitcoin mining activities with the use of special search strings. We find that it is complex to construct useful search strings but also conclude that a select number of search strings indeed provide additional information on the location of Bitcoin mining activity. This information does however not allow us to draw conclusions on the amount of mining activity, i.e. the mining hashrate that is available at each location.

In addition to this we look at mining pool gateways. These gateways are setup in different regions to reduce latency in the communication between miners and the mining pool. We identify the IP address of the server hosts of the largest 14 mining pools and perform a DNS-lookup to find all related IP addresses. This leads to a set of locations that is again distinct from the data retrieved by our crawler tool. Again, this information cannot be used to draw conclusions on the hashrate of miners.

Both methodologies provide a useful and creative answer to our last research question. A number of other sources may exist that can provide additional information to come to an even more comprehensive set of data. We leave this to future research.

7.1.2 Combined results

Figure 7.1 combines all results, showing results from our BGLF crawler (top left, including all reachable and non-reachable nodes in the Bitcoin network), the Bitnodes crawler (mid left, including all reachable nodes in the Bitcoin network), and results from Censys (top right) and Shodan (mid right) for the search string *Antminer*. The figure also shows the results from our BGLF crawler for nodes with mining index $MI_i > 2$ (bottom). The results from our BGLF crawler show more nodes than the Bitnodes results, since we also include non-reachable nodes. When we restrict the BGLF crawler results to only those nodes with mining index above 2.0, the results look very similar to the Bitnodes results. One may tend to conclude that these nodes are in fact the reachable nodes, which seems plausible since mining nodes are in general well connected and also willing to accept new connections. On a more detailed level however, we can observe differences, as shown in Table 7.1. This table shows the results for the top 20 countries with the largest number of nodes. It is clear that our BGLF crawler with mining index identifies more nodes than Bitnodes, and furthermore 5 of the 20 countries are different, and there are some differences in the ranking of the countries.

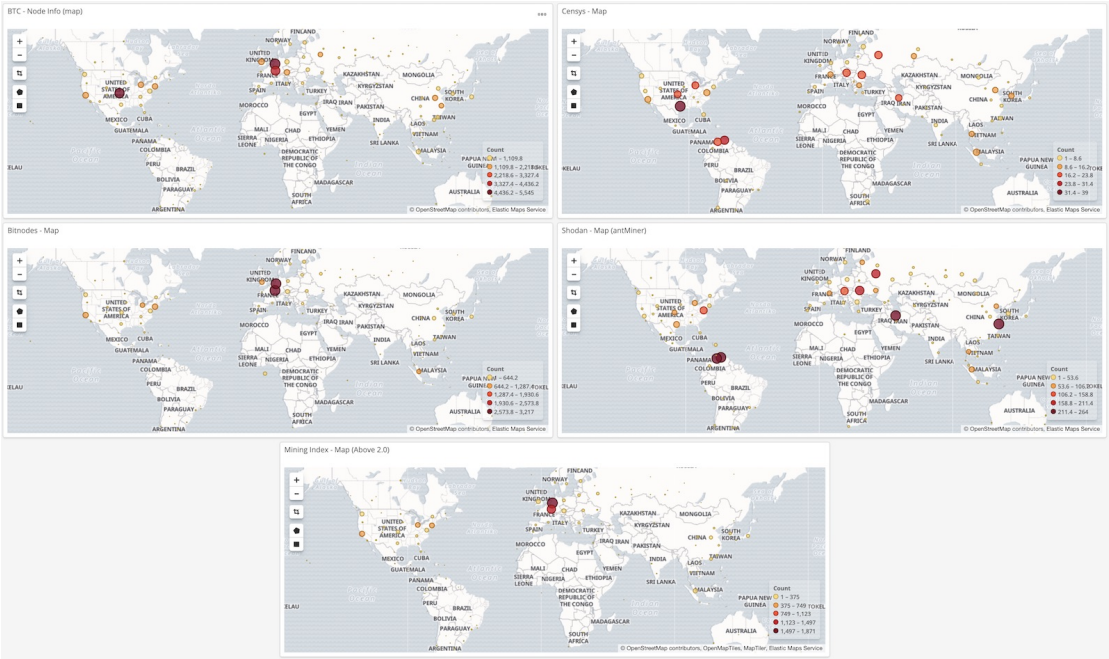


Figure 7.1: Geographical distribution of Bitcoin nodes (May 11, 2019)

The general conclusion is that Bitcoin mining nodes are located primarily in the USA, Europe, and Asia-Pacific. This is in line with the findings from our research in Chapter 3 as well as with findings from other studies (among others the Cryptocurrency Benchmark Report). It is also in line with the locations we found from the research on gateways of mining pools. The results from Censys and Shodan show in addition that Bitcoin mining activity can be found in South America.

No.	BGLF		Bitnodes		Censys (antMiner)		Shodan (antMiner)		Mining Index	
1	United States	14,574	United States	4,800	United States	131	United States	263	United States	14,208
2	Germany	6,657	Germany	3,772	Venezuela	49	Venezuela	251	Germany	6,412
3	China	5,002	France	1,228	Russia	44	Russian Federation	243	China	4,814
4	Russia	3,036	Netherlands	1,024	China	21	China	184	Russia	2,997
5	Netherlands	1,669	Canada	698	Iran	18	Iran, Islamic Republic of	130	Netherlands	1,580
6	United Kingdom	1,606	Comoros	632	Romania	15	Ukraine	70	United Kingdom	1,575
7	France	1,561	United Kingdom	619	Singapore	14	Romania	52	France	1,490
8	Canada	1,459	Senegal	617	South Korea	12	Canada	31	Canada	1,413
9	Japan	979	Russia	504	Thailand	11	Thailand	30	Singapore	903
10	Singapore	974	Japan	397	Czechia	10	Malaysia	27	Japan	868
11	Ukraine	809	China	357	Ukraine	8	Korea, Republic of	25	Ukraine	799
12	Australia	746	Australia	248	Bulgaria	7	Italy	22	Australia	733
13	Hong Kong	687	Finland	247	Canada	6	Sweden	21	Hong Kong	647
14	South Korea	671	Hong Kong	242	France	6	Bulgaria	20	Switzerland	626
15	Switzerland	643	Kiribati	239	India	6	Czech Republic	20	South Korea	625
16	Italy	507	Sweden	201	South Africa	6	Singapore	19	Italy	484
17	Sweden	486	Ukraine	183	Mongolia	5	Germany	18	Sweden	471
18	Spain	440	Lithuania	182	Taiwan	5	Austria	16	Spain	428
19	Brazil	394	Spain	161	Australia	4	Indonesia	15	Brazil	383
20	Ireland	373	Italy	158	Chile	4	Poland	14	Ireland	339

Table 7.1: Number of mining nodes in top 20 countries (May 11, 2019).

7.2 Contribution to the academic field

The main contribution of our project to the academic field is the large dataset that we unlock through our tools. We do this through a number of novel approaches that leads to data that is different from existing sources:

- *We uncovered a significant amount of locations from public sources*
  The information that we extracted from our search through public sources yielded a very comprehensive overview of mining locations. We present this in an attractive handmade map that holds an abundance of information in a condensed manner. Furthermore we focus solely on Bitcoin which sets our results apart from the Cryptoasset Benchmarking Report which is arguably the landmark publication for qualitative research on cryptocurrency mining locations.

- *We retrieve and infer information from nonreachable nodes in new ways*
  The crawler we built traverses both reachable and unreachable nodes and stores information on both. This is a contribution to the field in itself as the public datasite Bitnodes only shows reachable nodes. However, information that is retrieved from unreachable nodes through a connection attempt is obviously limited. The main contribution therefore is that we not only look at the data directly originating from nonreachable nodes. These nodes still have to connect to the network which means they disclose information through their outgoing connections. By indexing and searching the results of the reachable nodes in our our database for information on IP addresses of nonreachable nodes, we are able to uncover a lot of information on these nodes.

- *Our crawler connects to nodes using both IPv4 and IPv6*
  In all academic literature on the topic that we have gathered, IPv6 was excluded. Bitnodes however does include IPv6 in their crawling results.

- *Our crawler explicitly requests addresses from FIBRE nodes*
  While some of the academic articles we gathered mention the existence and use of the fast relay networks, they are not specifically targeted to retrieve additional information.

- *We define an overlay with our mining index to filter data*
  We define a number of node characteristics that should theoretically improve mining success. Assuming that miners will try to optimise their financial results, we calculate a mining index using these parameters and filter nodes in the network by their score on the index. This has not been done before.

- *We developed a modular and flexible research platform*
  Our platform with modular setup has consistently been developed with the

future use by others in mind. The platform has simultaneous access to the three Bitcoin networks (*mainnet*, *testnet*, *regtest*) to allow for extensive testing and research. It runs independently on the OU server which means that, with the right access credentials, it can be used by anyone. This allows other researchers to build upon our work or use the platform for completely different research on the Bitcoin network. We list some research alternatives that were beyond the scope of our research in paragraph 7.5.

- *Use of mark and recapture technique*
  While we found some academic literature on the use of the mark-and-recapture statistical techniques in the computer science field, we seem to be the first to use this for analysis on cryptocurrency networks. The lack of interest in these techniques is surprising to us as the first application in the field of Ecology stems from more than a century ago and the techniques seem perfectly suited for analysis on peer-to-peer networks.

- *We show that OSINT platforms provide a very useful and efficient way to unlock additional and different data* We show that OSINT platforms are able to provide valuable information when exploring connected devices. Other publications that address information gathering through OSINT platforms are found primarily outside the academic domain. Those that are concerned with cryptocurrency focus mainly on the preparation for or execution of various hacks, or on the improvement of security of connected devices. Our research does neither and focuses purely on the wealth of data these platforms can unlock which we use to augment and validate our results.

7.3 Limitations to our research

While we believe that we have answered our research questions adequately, we are not blind to the limitations to our work. In this paragraph we address some of the most notable issues.

- *Mining is primarily done through pools, why do we focus on nodes*
  Throughout our research we have indicated that most mining nowadays is done through mining pools. Miners may not use a (full) node and may have their income from the pool directly credited to an exchange, making them virtually untraceable. We cannot be sure if nodes represent miners and as such we cannot be sure our crawler actually finds miners. We can try to filter and increase the likelihood that we find nodes that represent miners, as we have done in Chapter 5 but there is no guarantee that this actually works. Therefore the results of the crawler are not taken at face value but they are

compared with other data which we know must represent miners, such as a Shodan search for 'Antminer'. In the end no single source will provide a definitive answer to the question where Bitcoin mining takes place but the collection of sources we have researched and uncovered should go a long way.

- *OSINT platforms are a black box*
  We have put a lot of work into automatically retrieving select data from Shodan ans Censys. In its most basic form, one could see this as taking data from a black box and drawing all sorts of conclusions from it. We have not verified that the Shodan and Censys engines actually work the way that their respective websites say they do, which would be proper academic practice. However, this was not the topic of our thesis and not possible within our time frame. We have briefly contemplated trying to build a simple connected device crawler ourselves but the resources required to do this go well beyond ours means and there is no blueprint available like there was for our Bitcoin node crawler in the form of the Bitnodes project. We have therefore relied on the accuracy of Shodan and Censys as we would with Google, coupled with the fact that Censys has academic credentials and is explained in detail in the academic article that that the organisation requires users with research access to cite [33].

- *Why focus on Bitcoin only*
  Our research is focused on Bitcoin only. This is a deliberate choice as it was the first blockchain and cryptocurrency and is still the largest by market cap. However, over the past ten year, hundreds of other cryptocurrencies have been launched. The global use of energy to power all cryptocurrencies is much higher than for only the Bitcoin blockchain. Conclusions would be much more impressive if we included all of those or even the largest ten currencies. The problem here is that most blockchains use different protocols and concepts and we would have very limited synergy effects from including additional cryptocurrencies. To expand our work to the largest five currencies would probably increase the workload more than five-fold because other protocols are not as well-documented as Bitcoin and we also would not have such a great starting tool as Bitnodes' `protocol.py` module. Furthermore we would get diminishing results from every additional currency included as fewer people are mining it. However, our research may serve as a helpful starting point to set up research for other cryptocurrencies. In addition, the Cryptocurrency Benchmark Report that we discussed in Chapter 3 does cover a number of the top currencies in detail in their qualitative research. They are able to achieve synergies in their survey to large players in the field as mining pool operators often run pools for multiple currencies and mining farms may mine various currencies in one location.

- *Some results may represent mining activity but not Bitcoin*
  For some of the data we retrieve it is impossible to state with 100% certainty that it is related to Bitcoin only. For instance, a Shodan search for port 3333 will point to the use of Stratum mining pool software but the software is not exclusively used for Bitcoin mining pools (in fact, port 3333 is also used for a few obscure non-crypto related services but we have chosen to ignore this). To a much lesser extent this also holds for a search for 'Antminer'. An increasing amount of ASIC-based mining rigs are manufactured to perform different hashing functions as they differ per blockchain. Even in our literature and other sources research in Chapter 3, for some of the locations that we have found we may have overstated or understated the mining activity related to Bitcoin as different currencies are mined in the same building. We have tried as much as possible to exclude non-Bitcoin related mining activity and certainly our crawler-based results and the search related to port 8333 are exclusively Bitcoin.

- *Proof of Work will be replaced by Proof of Stake or other methods*
  One could argue that the Proof of Work (PoW) algorithm that underpins the safety and success of the Bitcoin blockchain is also its biggest detractor. In recent years, various alternatives have sprung up, in particular Proof of Stake (PoS) [96]. Under Proof of stake, there are no mathematical puzzles to solve but the creator of a new block is chosen in a deterministic way, depending on its wealth, or stake. This solves the problem of energy consumption while also aligning holders of the currency with the safety and direction (forking) of their blockchain. The second largest blockchain, Ethereum, is actively working towards a Proof of Stake algorithm after having chosen in favour of their Ethash Proof of Work algorithm at launch. So why is PoS not implemented yet? The problem with Proof of Stake is that an attack on the network becomes a lot cheaper and therefore the method needs to be bulletproof. An attack on a Proof of Work network is very expensive due to the high amount of required processing power and may require more money than the attackers are able to steal. Ethereum may switch to Proof of Stake in the coming year following the development of the Casper protocol [97] but for Bitcoin there are no such developments yet.

7.4 Ethical considerations

The research activities that are documented in this thesis could be seen as invasive or wasteful by other users of the Bitcoin network or the general (informed) public and this warrants some further discussion. Our project aims to contribute to meaningful research on various aspects of the Bitcoin network, not only for this thesis

but also by building a platform for future researchers. This research could not exist without communicating with the Bitcoin network. Using some of the network's resources to improve understanding of the network is acceptable in our view but there could be a point where the means do not justify the end.

We have identified and listed some of the main considerations:

1. The use of Bitcoin network resources

A large amount of people and organizations deploy Bitcoin nodes and run Bitcoin Core software to contribute to the Bitcoin network, its decentralization and distribution. By using crawlers to query nodes for information we take valuable time and processing power from these nodes that is not spent on running the network and propagating block and transactions. We see this effect as small and, with our legitimate goal in mind, as acceptable. Our crawler limits communication to handshaking and to a request for addresses six times in short succession. The node is then left alone until a new crawling process is started which for our research is not required more often than once per week. The effect on the node's operations should be negligible.

2. The propagation of IP addresses

When crawling the network and communicating with nodes, the IP address that the communication originates from is included into nodes' address lists and it is seen as an active node. The address is also propagated throughout the network from time to time. It is detrimental to the network to pollute it with false addresses. To mitigate this, we have set up an active and reachable Bitcoin Core full node that performs all function that are expected of a node. This way, no pollution to the network takes place. The crawling of the main network is performed from the address that runs the Bitcoin Core software (the OU server). Testing takes place on the test network, sometimes from personal computers that are not running Bitcoin Core. This pollutes the test network which is far from ideal. However, since it is meant for testing and the currency on the test network has no real world value, we find this acceptable for our research.

3. Retrieving IP addresses of Bitcoin nodes

The Bitcoin network is seen by many users as a relatively anonymous way of doing transactions. Whether this is truly the case is debatable. A user could stay anonymous with the right precautions such as running connections over VPN or through the TOR network. This can be done relatively easily but it seems that a lot of users do not see the need to do this. Running a Bitcoin node requires a decent amount of knowledge and sophistication that should also be sufficient to set up a VPN or Tor.

We assume that the users that we can find in the network are not worried about exposing their IP address. Furthermore, we see limited controversy around just collecting IP addresses for the aim of this project. At the moment addresses are stored in a location that is not readily accessible to third parties. It is also useful to consider that each Bitcoin node is gathering IP addresses in order to function as efficiently as possible and that the addresses we collect are already in the public domain.

4. OSINT techniques

Websites such as shodan.io and censys.io scan systems connected to internet and look for open ports or services that listen on a port. They can be used to conduct initial research on system vulnerabilities to prepare for an attack. They can also be used by professionals in the cybersecurity field to look for vulnerabilities that should be patched. Their existence may (rightfully) scare some people but they are a logical result of the Internet of Things and the hyperconnectivity trend.

Our project uses these search engines to collect IP addresses of Bitcoin miners. The sole aim is to determine the regional distribution of miners. It is likely that search engines like shodan.io make it easier for malicious actors to conduct illegitimate activities. However, they also provide greater transparency on the internet and the opportunity for research. Their main aim is to supply information and in that sense they are not really different from the basic Google search engine that most people use. We therefore see no reason to ban or discourage the use of these sites but we are responsible for the data that we retrieve from them and take care in storing it safely. As a side note, both shodan.io and censys.io have programs for researchers that allow free access to their systems and both have generously allowed us to make use of them at request.

To summarize, we have taken great care in adhering to the following rules:

- We continuously run a full node from the OU server, contributing to a stable Bitcoin network
- We run all research activity on the mainnet from the OU server so as not to pollute the network with useless IP addresses.
- All development and testing of the crawler has been done using the Bitcoin testnet network
- Information that is required for our research will be safely stored on the OU server only.
- We only show aggregated results in our dashboards, maps and tables. No individual IP addresses are disclosed.

7.5 Future research

During the course of our research we have identified the potential for additional research that unfortunately was outside of the scope and time frame of this thesis. In this chapter We explain the rationale for this research and how it adds to our existing work. We give some practical tips how this research could be structured and added to the application platform.

Quick wins for the current platform

Some additional research that should be easily available through the current platform has not been explored. For instance, the platform does not try to deanonymize nodes behind TOR and does not connect to the Falcon Network because we refrained from looking into these possibilities further. Furthermore, we were able to get the *regtest* network operational but failed to solve the communication issues we ran into. Finally, more search strings may be devised for OSINT platforms by those with more intimate knowledge on mining pool protocols and mining hardware services. We also believe the speed of the crawler can be improved by using more processor cores.

Mapping Bitcoin addresses to IP addresses with listening nodes

One of the most promising but also challenging approaches to improve upon our work is to try to map Bitcoin addresses to IP addresses. This can be done by listening to all nodes simultaneously and then identifying those nodes that first broadcast new blocks. Since the first broadcasting node is the node that solved the block hash, this must be a miner. And because the first transaction in a block is a payment of 12.5 BTC to the miner, we can also determine its Bitcoin address. This information opens a whole range of additional research opportunities such the possibility to analyse transactions from mining pools to individual miners to determine pool contributors.

The problem is that this requires a large amount of listening nodes to be deployed because the communication should ideally be intercepted directly from the originating node. The deployment of a large amount of listening nodes to capture every transaction could be seen as invasive and the nodes do not contribute to the aim of the network. In fact, they work against it when regular nodes connect to the listening nodes and send transactions that are not relayed by the listening nodes. This issue could be alleviated by making the listening nodes also propagate transactions through the network so time and resources of other nodes are not spent in vain. This way, the nodes would contribute to the distribution of the network. However, this analysis requires a significant amount of effort and resources and it goes against some of the fundamental ideas of the Bitcoin network.

Mapping Bitcoin addresses to IP addresses with website crawlers

There are are a number of situations in which Bitcoin addresses can already be linked to personal data including IP addresses and it seems the wider community has accepted this. Cryptocurrency exchanges ask for personal information to conduct their anti money laundering obligations. People leave their Bitcoin addresses on forums or websites for tips together with other personal information or they acquire goods with Bitcoin that are sent to their home address, again linking their cryptocurrency address to personal information. Smart website crawlers can be used to collect combinations of IP addresses and Bitcoin addresses from different sources.

Bitcoin addresses were not meant to be reused and Bitcoin.org advises against reuse of addresses. The best way to ensure anonymity and safety is to use a different address for every transaction. A lot of people do not do this and use a small number of addresses for all their transactions, albeit sometimes because they have no choice because their mining pool or exchange requires this. If people stopped reusing, the analysis presented here would be impossible or pointless.

Topology research

A new research avenue opens up by using the crawler tool to determine the topology or shape of the network. With some minor adjustments, the tool can be used to graph the entire network and potentially look for inconsistencies or threats and also for analysis of the change in network topology over time. This also may recreate some of the findings that were found by Miller et al. [5] such as further proof of the existence of the so-called 'Influential nodes'.

Mark and Recapture techniques for cryptocurrency networks

We were very excited about the applicability of the Mark and Recapture technique in our research as the technique came from a completely different field. While some research exists on its use for computer networks, we feel there is potential for additional experimental academic research specifically focused on the various statistical estimation techniques and their application to cryptocurrency networks or Peer-to-Peer networks in general. More specifically, as discussed in paragraph 5.3.5, size estimators for address lists of nodes could be validated against results from simulation models that use parameters based on the technical details of Bitcoin node address management.

Time series analysis

Because we continuously collect data from all the sources that we discuss in this thesis, eventually we will have a dataset that is long enough to analyse it for structural changes or statistical patterns.

8. Reflections

8.1 Introduction

This thesis was initiated in May 2018 as the final piece towards fulfillment of the requirements for the degree of Bachelor of Science in Computer Science at the Open University. This final chapter holds a reflection on our work and experiences over the past year. It has only limited bearing on the content of our research but will focus on individual contributions and learning points, time spent on the various topics as well as observations on the process and our way of working as a team. Because we have built and documented a platform for use by others and because this thesis document may need to be used by those that will build upon it or that want to replicate our findings, we have chosen to write our reflections in a separate chapter that can easily be taken out.

8.2 Evaluation on changes in the thesis setup

8.2.1 Preparatory course

When we joined the first meeting for the preparatory course for the thesis project on 26 May 2018 we did not know each other. However we both shared an interest in cryptocurrencies. Henk-Jan Hopman has been working in the field of computer networks for a long time and the emergence of the Bitcoin network and its intricacies had piqued his interest. Chris Hoorenman had been mining ZCash coins using a number of the latest GPU chips until the collapse of cryptocurrency prices in the first quarter of 2018 and he was experimenting with some of the first non-fungible tokens (NFT) games on the Ethereum blockchain.

We both expected cryptocurrencies to be a very popular topic among the group of students that were present at that first meeting. Surprisingly, this was not the case, leaving the only topic that was available in the field, Geolocation of Bitcoin Mining, up for grabs. Since we both work full time and had the same amount of weekly hours in mind that we could spend on this project, teaming up felt like a good plan.

Harald Vranken was appointed as our supervisor and was also the main researcher for which we performed our project. During our first discussions he gave us a fair amount of freedom in approaching the topic. We devised a large number of approaches and tools that were relevant in the field and that could be of use to

Harald. We soon figured out we had to limit these activities if we wanted to finish within the allotted time.

This culminated over a period of five months into a preparatory document in which we presented a number of questions that we wanted to answer:

1. How can miners be identified and filtered from other actors on the blockchain
   (a) How can we identify all Bitcoin nodes?
   (b) How can we determine whether a Bitcoin node is a miner?
   (c) How can we identify Bitcoin mining pools?
   (d) Can we link a Bitcoin node to a Bitcoin address (user)?
2. How can miners be located based on available data
   (a) How do we geolocate a Bitcoin node based on its IP address?
   (b) Is it possible to find the IP address of miners and members of a mining pool by using OSINT?
   (c) Is it possible to find the IP address of members of a mining-pool by studying (small) repeating transactions (i.e. the payments from a pool to a miner)?

These questions were split up into three project phases with clear requirements:

First phase: Topology discovery of the Bitcoin network

Goal: Determine the IP address of Bitcoin nodes **Main activities:** Network crawling, data collection, interface scripting *Additional activities:* Adjust the Bitcoin Core open source software and deploy a customized client **Required base result:** A tool showing the number and percentage of nodes per country *Additional results:* Geolocation tool interface to show data as a world heat-map

Second phase (Optional): Discover miners and members of mining-pools with OSINT

Goal: Add more information of the Bitcoin nodes IP addresses to the database and refine mining pool information into a collection of miner locations by using OSINT techniques
Main activities: API scraping, data collection, interface scripting.
Required base result: A tool that retrieves information of IP addresses from OSINT websites
Additional results: Integration with the tool from the first phase

Third phase (Optional): Discover members of mining-pools by mapping Bitcoin addresses to IP addresses

Goal: Refine mining pool information into weighted miner locations
Main activities: Network listening, data collection, interface scripting, statistical analysis
Required base result: A tool that retrieves pairs of Bitcoin addresses and IP addresses
Additional results: A tool that maps geographical distribution of miners in mining pools. Integration with the base tool from the first phase.

This was very much a theoretical exercise based on our knowledge of the Bitcoin network at that time and before we had done any coding. To paraphrase a famous quote from the Prussian military commander Von Moltke: "No plan survives first contact with the network".

In the next paragraph we will discuss to what extent we kept and met our original goals and where we changed these, added to them or dropped them completely.

8.2.2 Reflections on final (AF) and preparatory (VAF) document

Evaluation of the proposal for the first phase against final results
The goals of the first phase in the preparatory document remain largely the same as the activities we deployed to answer our first research question in the main document, covered in Chapter 4. We crawl the Bitcoin network and collect almost all of its data through the use of our Python code. The coding entailed a lot more work than we anticipated under 'interface scripting' because the Bitnodes Bitcoin protocol was not ready to be deployed due to its reliance on Python 2. Furthermore we had to build the crawling algorithm ourselves because Bitnodes makes use of a very different in-memory data storage system and because we wanted to retrieve more and different information than Bitnodes. The optional additional activity that we proposed was the deployment of a customized Bitcoin Core client. We decided

not to pursue this because it would add a lot of complexity and because it would not lead to better results.

The required base result, a tool showing the number and percentage of nodes per country was completely achieved. Our tool does not report percentages because they can be derived from the country numbers. Also the additional results were achieved although our platform presents the nodes as dots of different sizes in a world map with, which technically is not a heat map but provides similar information.

Evaluation of the proposal for the second phase against final results
The goal for the second phase, which we listed as optional in our document, was to add information to the data set from the first phase by using OSINT search engines. We envisaged we would do this by making requests to APIs and scraping the websites. Due to our research access to Shodan and Censys and the useful APIs these organisations have deployed, there was no need to build a website scraper. As we explained in Chapter 6 and 7, we were able to meet our goal but the amount of useful search strings proved disappointing. However, we deployed a number of other initiatives to come to our required base result. Most notably we built the Mining Index in Chapter 5, work that we had not anticipated in the preparatory document. Furthermore we explored addresses from mining pool gateways which we also had not foreseen initially. Together, these activities gave us more information about Bitcoin nodes and allowed us to filter nodes. The information on mining pools was more limited than we expected and if we had not built the mining pool gateway DNS lookup we would have had virtually no data on mining pools. The additional results were achieved as we integrated all our querying tools into the BGLF platform and built visualisation dashboards for them in Kibana.

Evaluation of the proposal for the third phase against final results
The entirety of Phase 3, which we also listed as optional in our document, was not achieved. This has a variety of reasons. First and foremost, after we started working on our crawler and platform it soon became evident that the time required for the third phase was was going to take us well beyond the time limit we had set for this thesis. In addition to this, we realised more clearly that the resources required for this phase were going to be significant. Furthermore, while we think it is possible to run a large number of listening nodes through our platform by using many containers, this would be very invasive and potentially detrimental to the network. We have proposed the work required for this phase in our paragraph on future research in Chapter 7. However we suggest to those that wish to take this up to spend a significant amount of time on devising a strategy that minimises the impact on the network.

Evaluation of additional work

Next to the work we described in our preparatory document, we have done research that was not foreseen initially. Two major achievements were already mentioned in this paragraph, the mining index and the mining pool gateway DNS lookup. We also added the literature and sources overview on mining farm activity that is discussed in Chapter 3. We did this because we decided we needed to have a least a good rudimentary overview of publicly available information against which we could evaluate or validate our own results.

IMC 2019 conference paper

During the first two weeks of May 2019, after we had the first set of full results from our platform, we wrote a paper together with Harald in response to the call for papers for the ACM Internet Measurement Conference 2019 (IMC 2019). In the paper we focused on our crawler methodology, the details and results of our mining index node scoring system and the validation of our results against other sources. By using most of our available time in those two weeks on the paper, we were able to submit the document before the deadline of May 13. The paper will be peer-reviewed and feedback is expected in the second half of July. The IMC conference is an A-ranked conference according to the CORE conference ranking system [98].

8.3 Division of time and workload

8.3.1 Introduction

During the first stages of the final project we had lengthy discussions about the division of work and the requirement to do a personal substudy and write a personal chapter about this. While we included personal substudies in our preparatory document, the final project did not lend itself very well for this setup, especially when it became clear that we had to leave the mapping of Bitcoin addresses to IP addresses to future research. The largest amount of our time was spent developing the platform with integrated crawler and this required our combined effort. Therefore we notified Harald relatively early in the process that the requirement to divide the project into personal components proved problematic for us. During one of the progress sessions we were allowed to drop this requirement and we agreed instead to write a paragraph on what the core of our personal contributions had been. In the following paragraph we will describe this in more detail but we would like to stress that the division is only a relative indication and that on most topics both of us shared the work. In particular the more generic development and Python coding of the platform was done by both of us such as devising the process flow, the debugging, refactoring and the handling or errors.

Given the enormous workload we did try to be as efficient as possible and make the best use of our personal backgrounds with the existing and detailed knowledge of Henk-Jan on computer networks, platforms and systems and the existing knowledge of Chris in the field of statistics and optimization due to his background in econometrics.

We had weekly Skype sessions that lasted about one hour. These were centered around progress and new developments and the division of work. There was also a clear need to continuously discuss closely how the ever increasing amount of results from the crawling process would be stored in Elasticsearch and subsequently how the Python code would retrieve it again from the database. Occasionally we had more frequent discussions when we encountered errors, or during the frantic efforts early May 2019 to get our first results ready for the submission deadline for the IMC 2019 conference in Amsterdam. During the year that we worked on this project, there were obviously a number of holidays, travels for work or just busy periods in which one of us was less active but this proved to be very balanced between us.

We had sessions with our supervisor every three weeks. These also lasted roughly one hour and were mainly focused on our progress in the preceding weeks and the plans for the coming weeks. We like to think we were pretty self-sufficient in our approach, planning and creativity and showed good work ethics. The input from Harald was invaluable on the need to stay focused on the main research questions and not get distracted by side issues. He also provided us with relevant literature and at times was critical when we made matter-of-fact statements not backed by academic literature. This led us to pursue every topic with additional care and references and made us review old paragraphs regularly when we gained new knowledge. At the same time Haraldwas understanding of the fact that the cryptocurrency field is not dominated by academic research and the latest developments and insights are to a large extent still somewhat experimental. Overall, his periodical assent that we were doing the right things and were on the right track was most helpful. We experienced a very intense but highly enjoyable week early May 2019 when the three of us were working together to meet the deadline of the IMC conference call for papers with a number of Skype sessions during that week.

8.3.2 Core activities by Chris Hoorenman

Latex (now called Overleaf)

Chris set up the Latex modular structure and a large part of the Latex class (.cls) file that allowed us to draft and write the project efficiently. While Latex itself is not a part of our project, we do think that our setup may be valuable to other students so they do not have to reinvent the wheel like we did in the start-up phase of our project. In particular we very much like the setup where each chapter has its own

latex (.tex) file which allowed us to search and restructure the document easily. We also like our paragraph heading style which was not trivial, we like the way we have made separate bibliographies and we like our algorithm layout.

Literature review
Chris provided the largest part of the literature review, in particular the research for mining locations across the world that was written in Chapter 3.

Reworking of Protocol.py to Python 3
The Bitcoin protocol module Protocol.py from the Bitnodes project open source code was reworked to function under Python 3 as explained in paragraph 4.3.4. This was mostly done by Chris but we both needed functional knowledge to build upon it further.

Development of crawling algorithms (from basic to concurrent)
Chris developed and coded the crawling logic and the improvements towards concurrency in a iterative approach where we tested each iteration together on our platform.

Research on OSINT search strings
Chris did most of the research on appropriate search strings for the OSINT API calls, looking into mining hardware web services and mining pool protocols. However, the result was disappointing as most of the search results we tried did not yield IP addresses.

Mark and Recapture methodology
Chris did research on the appropriate way to estimate the total number of peers in the address list of a node and came up with the Mark and Recapture methodology and the Schnabel estimator. This was subsequently coded into the crawler.

8.3.3 Core activities by Henk-Jan Hopman

Server setup and base layer of platform
Henk-Jan was instrumental in operating the server, setting up the Docker containers and basically building the backbone of the platform with access to the different networks including IPv6 and the Bitcoin Core networks.

Elasticsearch and Kibana implementation, integration with the platform
Henk-Jan set up the Elasticsearch database and Kibana dashboard, provided the indices and coded the storage logic into the BGLF Python code.

Command line interface and configuration of settings
Henk-Jan built the command line interface to enable flexible use of the BGLF tool

and built the logic that enables us to work with a single settings file through the use of the python Click module.

IP Geo lookup

Henk-Jan developed and coded the lookup functionality for the geolocation of IP addresses through the use of the Maxmind databases as explained in paragraph 4.4.3.

OSINT and Bitnodes API request functionality

Henk-Jan developed and coded the functionality required to make API calls to the various external platforms we use for validation and extension of our data. This is an automated process integrated into the BGLF tool including the storage into Elasticsearch and visualization of result.

Mining index basics and implementation

Henk-Jan came up with the idea to build a mining index to filter the crawling results and did the initial research for the relevant variables as well as the largest part of the implementation into the BGLF tool. The mining index code makes extensive use of Elasticsearch indices, in particular when searching through all nodes for known addresses.

8.3.4 Estimation of time spent per activity (AF + VAF)

Activities	**Hours**
Weekly discussions (x2)	100
Discussions with supervisor (x2)	50
Live meetings (x2)	10
VAR presentation	10
Meetings and communication total	***200***
Latex setup and research	20
Setup of IDE and Bitbucket Git solution (x2)	5
Server setup and maintenance	15
Setup and maintenance of support systems total	***40***
Research on containers and implementation	20
Implementation of Elasticsearch incl. indices	30
Implementation of Kibana incl. dashboards	30
Implementation of Bitcoin Core networks	15
Setup of IPv6	5
Base platform construction & maintenance total	***100***
Review and summary of academic literature	30
Search for mining locations in public sources incl. world map	45
Academic literature and other sources total	***75***
Reworking protocol.py from Python 2 to 3 (incl. research)	30
Developing the crawler incl. coding	35
Implementing storage logic and communication between Python and ES	40
CLI and configuration file implementation	30
IP Geo lookup coding	5
Implementation of crawler and storage total	***140***
Mark and Recapture research and coding	10
Fast Relay networks research and coding	10
Research and coding for rest of mining index	25
Mining index total	***45***
Research on OSINT platform search string keywords	10
OSINT and Bitnodes API queries incl. Elasticsearch handling	30
Mining pool gateway research	5
OSINT and Mining gateway total	***45***
Integration of components	25
Final testing and debugging	25
Coding overhead and testing total	***50***
Writing total	***250***
Additional work on IMC paper	***30***
Total	***975***

Table 8.1: Main activities and time spent

8.4 Personal assessments

8.4.1 Introduction

In this paragraph Chris and Henk-Jan each write a separate evaluation of their experience throughout this project. What did each of us learn, where were the synergies, what did we consider difficult, what could we have done better and what did we like best.

8.4.2 Assessment Henk-Jan

Looking back on the past year I am very positive about the cooperation with Chris and the pleasant way of working together. During our discussions there was sufficient room for differences of opinion and we were always able to come to a compromise. Chris has a relaxed personality which makes for easy discussions and he always met the deadlines we had set for ourselves. It just makes life a bit easier when you can count on each other. Even though we both have demanding jobs, there was always time for questions or feedback.

I may have been a bit overly optimistic myself on the amount of time required to develop and finalize certain parts of our project. In particular we were way off in our estimation on the hours required for the software development. I did not initially take into account that we had to rework the protocol module into Python 3 but Chris quickly convinced me, given the impending obsolescence of Python 2, that we really needed to do this to make our platform functional and future-proof.

This project was a long series of learning points for me, as it should be. Clearly, I learnt a lot about Bitcoin and cryptocurrencies and the way the peer-to-peer network operates. Furthermore I learnt a lot from working with 'Big Data', in particular in combination with the use of Elasticsearch and Kibana. The use of Docker and the flexibility this offers was an eye-opener for me. It is not only the speed in development of the platform that we could achieve by using containers but also the installation of complex software packages that were completely hassle-free. Even though I have worked in the field of computer systems and data communication for more than 20 years by now, this project was the first where I had to gain access to the IPv6 part of the internet and this was highly interesting. Finally, even though I was familiar with Python, all the coding further improved my programming skills and insight into the language.

8.4.3 Assessment Chris

During our time working on this project I have gained a lot of appreciation for the skills Henk-Jan has in setting up servers and computer systems. This is a field that is not addressed much during the Bachelor program and it would have been very time-consuming for me to learn that from scratch. Henk-Jan was fun to work with and often already started setting up the required configurations and systems for an idea we discussed before we had finished talking. This was a very pleasant way of working as it meant we could test ideas quickly and also drop ideas quickly when results were poor or implementation too difficult.

I have a tendency to think long about a problem before implementing it and research it from all angles first. Henk-Jan's different approach forced me to work more agile which was a very good combination. Conversely, I'd like to think I also saved Henk-Jan from doing some activities that were never going to fly. In any case it made for very good dynamics between the two of us and this was exacerbated by the fact that we had very different but relevant backgrounds, in systems engineering and statistics.

In the first months of our final project I convinced Henk-Jan about the need for literature research on the location of Bitcoin mining farms, to have some base result to compare our more technical results against and to give our work an additional edge. While I am satisfied with the end result in Chapter 3, this took a lot more time than I imagined, especially because I forced myself to find multiple sources for each location. The work was also slightly boring after a while. Looking back, I would have set this up differently and maybe focus on a specific area of interest such as China or Russia. At the same time, I've learnt a lot about the the player dynamics and economics from reading all of these sources and I additionally learnt a lot about popular opinion towards Bitcoin mining operations.

I have gained a lot of knowledge from the more technical side of our project. When it became clear that we were going to write the code for our crawler and platform in Python, I decided to build a small program for the Ethereum blockchain in Python that I needed for a personal project to get working knowledge of the language. This also taught me some of the main differences between Python 2 and 3 which proved very valuable for our project. I enjoyed coming up with ways to give our project a specific edge that added academic value, for instance the Mark and Recapture methodology and the various attempts to optimise the speed and output of the crawling algorithm. These things kept it very exciting. The work with OSINT platforms, which I had not used before, was very interesting and also served as a reminder both how useful and dangerous the internet is. I absolutely loved diving into the Bitcoin protocol and researching all its intricacies to be able to rework the Bitnodes Python protocol and connect to and crawl the nodes and to understand

their response. I believe that blockchain has a bright future and a long way to go and the knowledge we have gained from this project will be very valuable to me and Henk-Jan for years to come.

8.5 Final words

In this final paragraph we would like to say a few words of thanks to the people that were instrumental in finishing this project. First of all, we thank our supervisor Harald Vranken for his time, support, his critical review of our work and for the pleasant and relaxed meetings and way of working throughout the year. We very much hope that it has been a very enjoyable period for you also. Furthermore we thank the Open University for the use of their server and additional facilities. Finally we would like to thank Shodan and Censys for providing us with research access to their search engines.

We would both like to thank a few people individually:

First of all I would would like to thank my parents for their support and interest throughout the Bachelor's program and their insistence to never stop learning and doing what you love. I also would like to thank my brothers Eduard and Jaap and sister Hanna for the many (so many!) nerd-jokes over the past years when the coding on Saturday and Sunday prevented me from pursuing more earthly activities. I started the Bachelor program together with my friend Maarten and would like to thank him for the companionship and support in the years when we were following the same program and competing for the highest exam grades. Finally I thank Nargiza for her loving support during the work on this thesis, especially the long and sometimes boring winter weeks, and also for her valuable help in searching for mining farm locations in the Russian language.

-Chris

With all my heart I would like to thank my wife Andrea for the time and space she has so lovingly sacrificed over the past eight years to let me pursue and finish this bachelor's program and thesis. After a difficult period in which we have learnt not to take our health for granted I am so very grateful that you are able to experience this moment together with me.

I would also like to thank my children Mariska and Yannick for their understanding over the past years. There have been numerous occasions where I needed to focus on my studies. You have always remained interested and supportive and regularly

told me we could also do the fun activities next week. I look forward to doing those with you more often now.

There are so many other people I need to thank, from my parents and parent-in-law to my brothers, sisters, brothers and sisters-in-law, friends and co-workers. You have all helped me to keep going during the sometimes long and lonely evenings with supportive remarks and with the interest you took in my studies.

But above all, Andrea, it is your loving support and understanding that made me succeed.

-Henk-Jan

A. Global mining activity

Region: Ordos, Inner Mongolia, China
Company: Bitmain [102]
Rationale: Cheap coal plant electricity, production of ASICs in China - limited downtime, low-cost region
Estimated size: 70 MW (50.000 ASICs) [103, 99]
Sources: [104, 103, 105]

Region: Sichuan, China
Company: Bitmain [102], various others
Rationale: Cheap hydropower in wet season
Estimated size: Unclear, assumption 135 MW+ (100.000 ASICs)
Sources: [106, 99]

Region: Xinjing, China
Company: Bitmain [102], various others
Rationale: Unclear
Estimated size: Unclear, assumption 35 MW (25.000 ASICs)
Sources: [106, 99]

Region: Iceland
Company: Genesis Mining [107]
Rationale: Geothermal energy, cold climate
Estimated size: Unclear, assumption 10 MW
Sources: [108, 109, 110]

Region: Tbilisi, Georgia
Company: Bitfury [111]
Rationale: State-subsidized electricity, cold climate, low cost region, no taxes
Estimated size: 45 MW [112]
Sources: [113, 103, 112]

Region: Alberta, Canada
Company: Hut 8 Mining Corp [114], Bitfury [111]
Rationale: Low electricity costs due to abundant wind generation and natural gas
Estimated size: 80 MW [114]
Sources: [103, 115, 116]

Region: British Columbia, Canada
Company: DMG [117]
Rationale: Cheap hydropower
Estimated size: 60 MW [114]
Sources: [103, 115, 116]

Region: Columbia Basin, East Washington State, United States
Company: Giga Watt [118], Salcido Enterprises [119], Bitmain [102]
Rationale: Abundance of hydro-electric power, cool winters, dry air, high-speed internet
Estimated size: 50 MW+ [120, 121]
Sources: [103, 120, 122, 121]

Region: New York State, United States
Company: Coinmint [123]
Rationale: cheap hydropower and windpower
Estimated size: 50 MW (35.000 ASICs Massena, 10 MW Plattsburgh)
Sources: [124, 125, 122, 126]

Region: Scandinavia
Company: GMO [127]
Rationale: Cheap geothermal and hydropower, cold climate **Estimated size:** 20 MW (15.000 ASIC mining rigs)
Sources: [128, 129, 130]

Region: Mo i Rana, Norway
Company: Bitfury [111]
Rationale: Favorable tax code, renewable energy sources
Estimated size: 40 MW (375 GWh)
Sources: [131, 132]

Region: Boden, Sweden
Company: Canaan [133], Blockbase [134], Hydro66 [135]
Rationale: Low energy tax, cold climate, hydropower
Estimated size: 35 MW
Sources: [132, 136, 137, 138]

Region: Virginia, United States
Company: Bcause LLC (Chapter 11) [133]
Rationale: Low taxes for data centres
Estimated size: 80 MW
Sources: [103, 139, 140]

Region: Tierra del Fuego, Argentina
Company: Bit Patagonia [141]
Rationale: Cold climate, tax reductions
Estimated size: 5 MW
Sources: [142, 143, 144]

Region: Irkutsk Region, Russia
Company: Minery (MNRY) [145], Bitriver [146]
Rationale: Cold climate, hydropower
Estimated size: 65 MW
Sources: [147, 148, 149]

Region: Leningrad region, Russia
Company: Cryptouniverse [150]
Rationale: Cold climate, cheap electricity
Estimated size: 20 MW
Sources: [151, 152, 153, 154]

Region: Krasnoyarsk Krai, Russia
Company: BitBaza [155]
Rationale: Cold climate, cheap electricity
Estimated size: 10 MW
Sources: [156, 157, 158]

B. Development

B.1 Postman

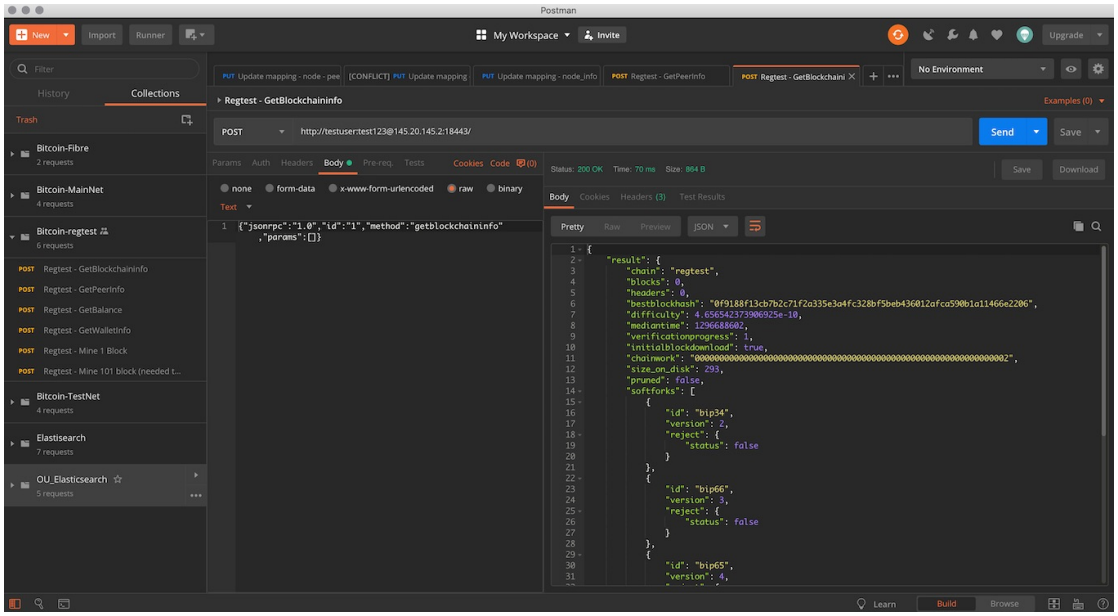


Figure B.1: RPC-call to get blockchain info from *regtest* cluster.

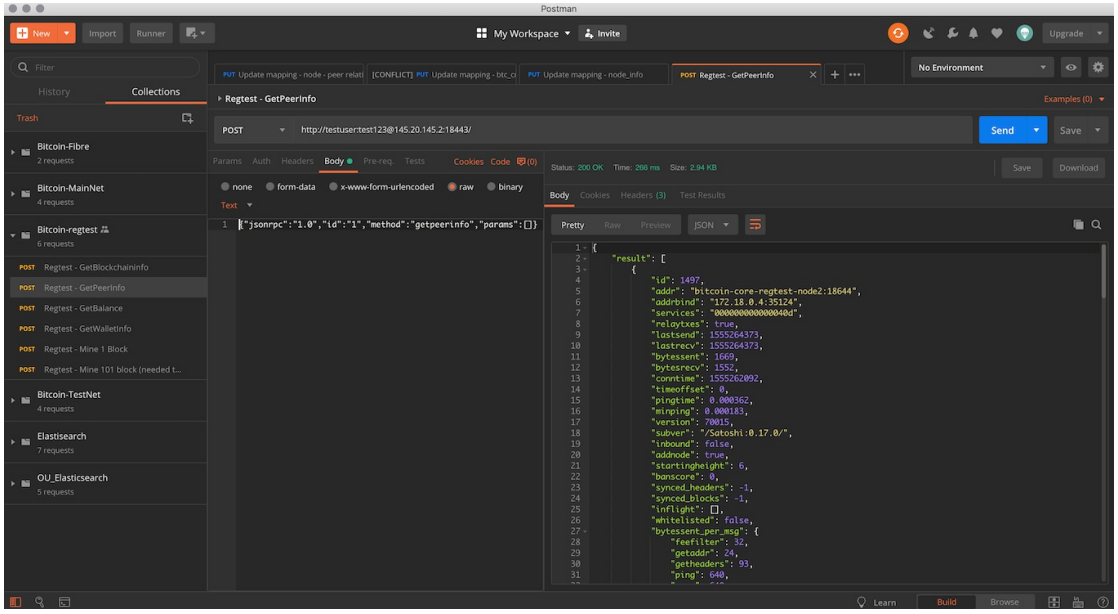


Figure B.2: RPC-call to get peer information info from *regtest* cluster.

B.2 Integrated Development Environments

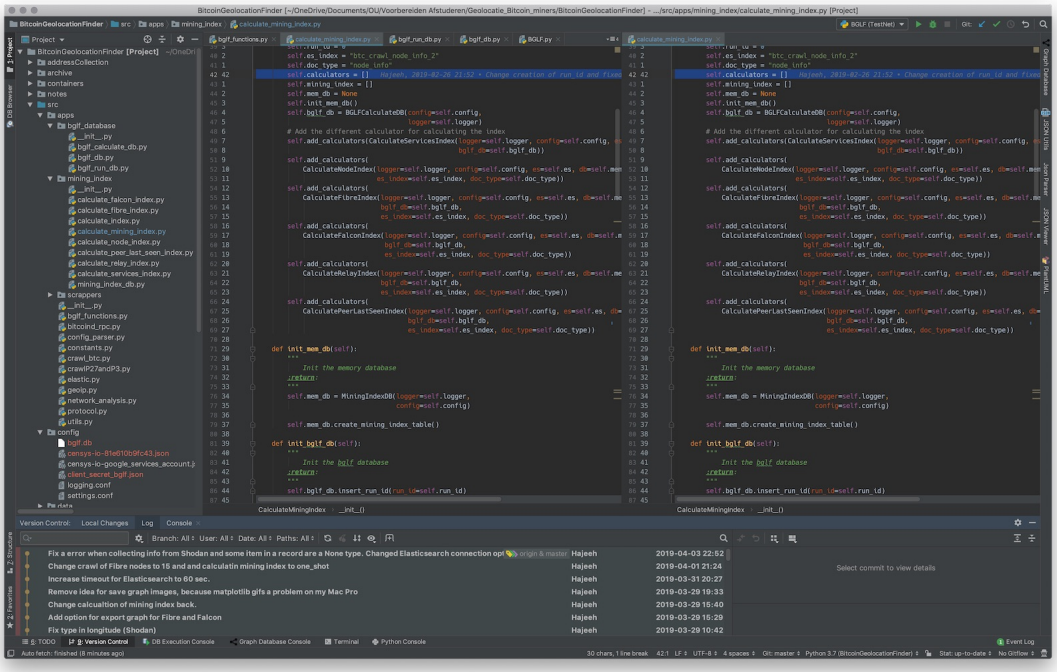


Figure B.3: Pycharm Community 2019, Python IDE

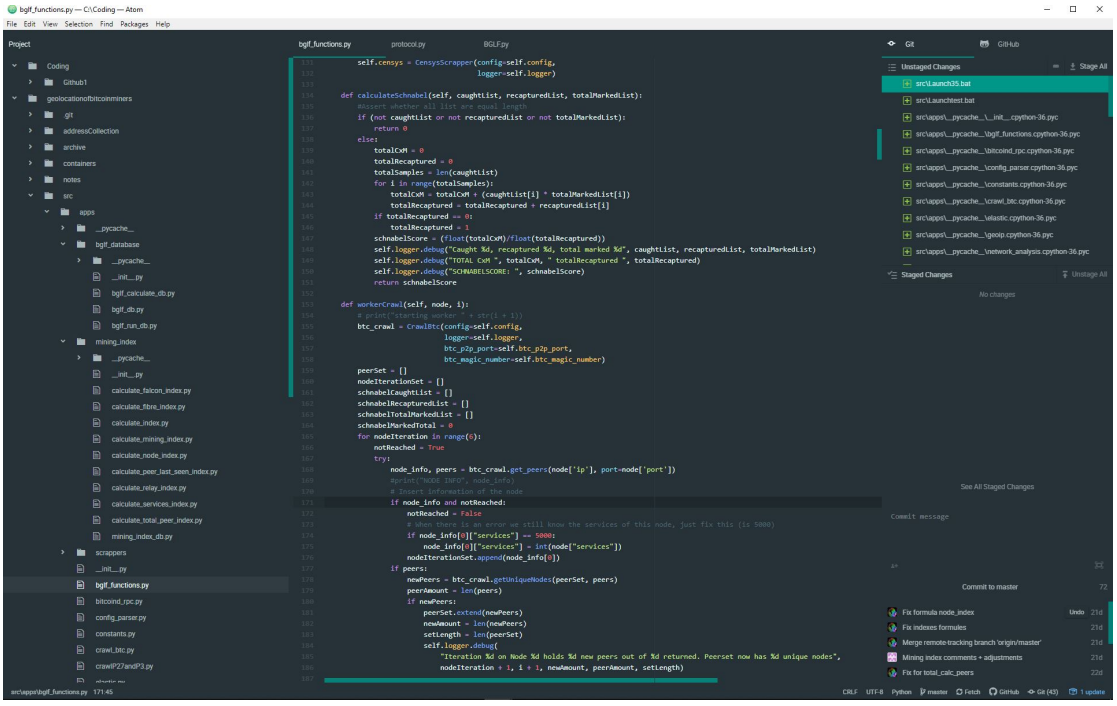


Figure B.4: Atom generic IDE customised for Python

C. Platform documentation

C.1 Docker

C.1.1 Docker-compose configuration file

```
# Microservices needed for to the project Bitcoin Geo Location Finder
#   Authors : H.J. Hopman & C. Hoorenman
#   Date    : 27-12-2018

version: '3.3'

services:
  # Elasticsearch-Cluster and Kibana config are based on work form : https://
      github.com/blacktop/docker-elasticsearch-alpine
  ###############
  ## Master Node
  ###############
  elasticsearch:
    #image: docker.elastic.co/elasticsearch/elasticsearch:6.4.2
    image: blacktop/elasticsearch:x-pack
    restart: unless-stopped
    container_name: elasticsearch
    environment:
      - node.name=es01
      - cluster.name=docker-cluster
      - bootstrap.memory_lock=true
      - http.cors.enabled=true
      - http.cors.allow-origin=*
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
      - ELASTIC_PASSWORD=pleasechangeme
      - xpack.security.audit.enabled=true
      - node.master=true
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    cap_add:
      - ALL
    volumes:
```

```
37        - esdata1:/usr/share/elasticsearch/data
38      ports:
39        - 9200:9200
40        - 9300:9300
41      networks:
42        - es_network
43      logging:
44       driver: "json-file"
45       options:
46        # limit logs retained on host to 25MB
47        max-size: "500k"
48        max-file: "5"
49
50    elasticsearch2:
51      #image: docker.elastic.co/elasticsearch/elasticsearch:6.4.2
52      image: blacktop/elasticsearch:x-pack
53      restart: unless-stopped
54      container_name: elasticsearch2
55      environment:
56        - node.name=es02
57        - cluster.name=docker-cluster
58        - bootstrap.memory_lock=true
59        - http.cors.enabled=true
60        - http.cors.allow-origin=*
61        - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
62        - ELASTIC_PASSWORD=pleasechangeme
63        - xpack.security.audit.enabled=true
64        - node.master=true
65        - "discovery.zen.ping.unicast.hosts=elasticsearch"
66      depends_on:
67          - elasticsearch
68      restart: on-failure
69      ulimits:
70        memlock:
71          soft: -1
72          hard: -1
73        nofile:
74          soft: 65536
75          hard: 65536
76      volumes:
77        - esdata2:/usr/share/elasticsearch/data
78      networks:
79        - es_network
80      logging:
81       driver: "json-file"
82       options:
83        # limit logs retained on host to 25MB
```

```
84          max-size: "500k"
85          max-file: "5"
86
87      elasticsearch3:
88        #image: docker.elastic.co/elasticsearch/elasticsearch:6.4.2
89        image: blacktop/elasticsearch:x-pack
90        restart: unless-stopped
91        container_name: elasticsearch3
92        environment:
93          - node.name=es03
94          - cluster.name=docker-cluster
95          - bootstrap.memory_lock=true
96          - http.cors.enabled=true
97          - http.cors.allow-origin=*
98          - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
99          - ELASTIC_PASSWORD=pleasechangeme
100         - xpack.security.audit.enabled=true
101         - node.master=true
102         - "discovery.zen.ping.unicast.hosts=elasticsearch"
103       depends_on:
104           - elasticsearch
105       restart: on-failure
106       ulimits:
107         memlock:
108           soft: -1
109           hard: -1
110         nofile:
111           soft: 65536
112           hard: 65536
113       volumes:
114         - esdata3:/usr/share/elasticsearch/data
115       networks:
116         - es_network
117       logging:
118        driver: "json-file"
119        options:
120         # limit logs retained on host to 25MB
121         max-size: "500k"
122         max-file: "5"
123
124     kibana:
125       #image: docker.elastic.co/kibana/kibana:6.4.2
126       image: blacktop/kibana:x-pack
127       container_name: kibana
128       environment:
129         - elasticsearch.url=http://elasticsearch:9200
130         - elasticsearch.username=elastic
```

```
131         - elasticsearch.password=pleasechangeme
132         - xpack.monitoring.ui.container.elasticsearch.enabled=true
133       depends_on:
134           - elasticsearch
135       restart: on-failure
136       ports:
137         - 5601:5601
138       networks:
139         - es_network
140       ulimits:
141         memlock:
142           soft: -1
143           hard: -1
144       logging:
145        driver: "json-file"
146        options:
147         # limit logs retained on host to 25MB
148         max-size: "500k"
149         max-file: "5"
150
151     # Is base on : https://github.com/mobz/elasticsearch-head
152     ESHeadPlugin:
153       image: 'mobz/elasticsearch-head:5'
154       container_name: eshead
155       ports:
156        - '9100:9100'
157       networks:
158        - es_network
159       logging:
160        driver: "json-file"
161        options:
162         # limit logs retained on host to 25MB
163         max-size: "500k"
164         max-file: "5"
165
166     # Is based on : https://github.com/ElasticHQ/elasticsearch-HQ
167     ElasticHQ:
168       image: 'elastichq/elasticsearch-hq'
169       container_name: eslastichq
170       ports:
171         - '5000:5000'
172       networks:
173         - es_network
174       logging:
175         driver: "json-file"
176         options:
177           # limit logs retained on host to 25MB
```

```
178          max-size: "500k"
179
180    # Bitcoin containers are based on : https://github.com/ruimarinho/docker-
         bitcoin-core
181    bitcoin-core-regtest:
182      image: ruimarinho/bitcoin-core:0.17
183      container_name: bitcoin-core-regtest
184      environment:
185          BITCOIN_DATA: ${BTC_DATA}/regtest
186      ports:
187       - 18443:18443
188       - 18444:18444
189      volumes:
190       - btcdatareg_node:/home/bitcoin/.bitcoin
191      networks:
192       - es_network
193      command: [ "-dnsseed=0", "-listen", "-regtest=1", "-rpcallowip=0.0.0.0/0"
           , "-rpcuser=testuser" , "-rpcpassword=test123" , "-addnode=bitcoin-
           core-regtest-node1:18544", "-addnode=bitcoin-core-regtest-node2:18644
           " , "-shrinkdebugfile" ]
194      logging:
195       driver: "json-file"
196       options:
197        # limit logs retained on host to 25MB
198        max-size: "500k"
199        max-file: "5"
200
201    bitcoin-core-regtest-node1:
202      image: ruimarinho/bitcoin-core:0.17
203      container_name: bitcoin-core-regtest-node1
204      ports:
205       - 18543:18543
206       - 18544:18544
207      volumes:
208       - btcdatareg_node1:/home/bitcoin/.bitcoin
209      networks:
210       - es_network
211      command: [ "-dnsseed=0", "-listen", "-regtest=1", "-port=18544", "-
           rpcport=18543", "-rpcallowip=0.0.0.0/0", "-rpcuser=testuser" , "-
           rpcpassword=test123" , "-addnode=bitcoin-core-regtest:18444", "-
           addnode=bitcoin-core-regtest-node2:18644", "-shrinkdebugfile" ]
212      logging:
213       driver: "json-file"
214       options:
215        # limit logs retained on host to 25MB
216        max-size: "500k"
217        max-file: "5"
```

```
218     depends_on:
219      - bitcoin-core-regtest
220     restart: on-failure
221
222   bitcoin-core-regtest-node2:
223     image: ruimarinho/bitcoin-core:0.17
224     container_name: bitcoin-core-regtest-node2
225     ports:
226      - 18643:18643
227      - 18644:18644
228     volumes:
229      - btcdatareg_node2:/home/bitcoin/.bitcoin
230     networks:
231      - es_network
232     command: [ "-dnsseed=0", "-listen", "-regtest=1", "-port=18644", "-
          rpcport=18643", "-rpcallowip=0.0.0.0/0", "-rpcuser=testuser" , "-
          rpcpassword=test123" , "-addnode=bitcoin-core-regtest:18444", "-
          addnode=bitcoin-core-regtest-node1:18544" , "-shrinkdebugfile" ]
233     logging:
234      driver: "json-file"
235      options:
236       # limit logs retained on host to 25MB
237       max-size: "500k"
238       max-file: "5"
239     depends_on:
240      - bitcoin-core-regtest
241     restart: on-failure
242
243   bitcoin-core-testnet:
244     image: ruimarinho/bitcoin-core:0.17
245     container_name: bitcoin-core-testnet
246     restart: always
247     ports:
248      - 18332:18332
249      - 18333:18333
250     volumes:
251      - btcdatatest:/home/bitcoin/.bitcoin
252     networks:
253      - es_network
254     command: [ "-testnet", "-rpcallowip=0.0.0.0/0", "-rpcuser=testuser" , "-
          rpcpassword=test123" , "-prune=550" , "-shrinkdebugfile", "-
          debuglogfile=0" ]
255     logging:
256      driver: "json-file"
257      options:
258       # limit logs retained on host to 25MB
259       max-size: "500k"
```

```
260         max-file: "5"
261
262     bitcoin-core-mainnet:
263       image: ruimarinho/bitcoin-core:0.17
264       container_name: bitcoin-core-mainnet
265       restart: always
266       ports:
267        - 8332:8332
268        - 8333:8333
269       volumes:
270        - btcdatamain:/home/bitcoin/.bitcoin
271       networks:
272        - es_network
273       command: [ "-rpcallowip=0.0.0.0/0", "-rpcuser=testuser" , "-rpcpassword=
            test123", "-shrinkdebugfile", "-debuglogfile=0" ]
274       logging:
275        driver: "json-file"
276        options:
277         # limit logs retained on host to 25MB
278         max-size: "500k"
279         max-file: "5"
280
281     bitcoin-core-fibre:
282       image: ruimarinho/bitcoin-core:0.17
283       container_name: bitcoin-core-fibre
284       restart: always
285       ports:
286        - 8352:8332
287        - 8353:8333
288       volumes:
289        - btcdatafibre:/home/bitcoin/.bitcoin
290       networks:
291        - es_network
292       command: [ "--dnsseed=0", "--dns", "--addnode=v4.eu.fibre.
            bitcoinrelaynetwork.org", "--addnode=v4.us-west.fibre.
            bitcoinrelaynetwork,org", "--addnode=v4.us-east.fibre.
            bitcoinrelaynetwork.org", "--addnode=v4.hk.fibre.bitcoinrelaynetwork.
            org", "--addnode=jpy.fibre.bitcoinrelaynetwork.org","--addnode=v4.cn.
            fibre.bitcoinrelaynetwork", "-rpcallowip=0.0.0.0/0", "-rpcuser=
            testuser" , "--prune=550", "-rpcpassword=test123", "-shrinkdebugfile"
            , "-debuglogfile=0" ]
293       logging:
294        driver: "json-file"
295        options:
296         # limit logs retained on host to 25MB
297         max-size: "500k"
298         max-file: "5"
```

```
299
300 volumes:
301   esdata1:
302       driver: local
303       driver_opts:
304           type: none
305           device: ${ES_DATA}/es1
306           o: bind
307   esdata2:
308       driver: local
309       driver_opts:
310           type: none
311           device: ${ES_DATA}/es2
312           o: bind
313   esdata3:
314       driver: local
315       driver_opts:
316           type: none
317           device: ${ES_DATA}/es3
318           o: bind
319   btcdatafalcon:
320       driver: local
321       driver_opts:
322           type: none
323           device: ${BTC_DATA}/falcon
324           o: bind
325   btcdatafibre:
326       driver: local
327       driver_opts:
328           type: none
329           device: ${BTC_DATA}/fibre
330           o: bind
331   btcdatamain:
332       driver: local
333       driver_opts:
334           type: none
335           device: ${BTC_DATA}/mainnet
336           o: bind
337   btcdatatest:
338       driver: local
339       driver_opts:
340           type: none
341           device: ${BTC_DATA}/testnet
342           o: bind
343   btcdatareg_node:
344       driver: local
345       driver_opts:
```

```
346             type: none
347             device: ${BTC_DATA}/regtest/node1
348             o: bind
349     btcdatareg_node1:
350         driver: local
351         driver_opts:
352             type: none
353             device: ${BTC_DATA}/regtest/node2
354             o: bind
355     btcdatareg_node2:
356         driver: local
357         driver_opts:
358             type: none
359             device: ${BTC_DATA}/regtest/node3
360             o: bind
361
362 networks:
363     es_network:
```

Code Listing C.1: Docker-compose configuration file

C.1.2 Docker-compose environment file

```
1 # Elasticsearch specific mounts and settings
2 # Elasticsearch base data directory
3 ES_DATA=/ou/bitcoinond/store
4 # Bitcoin base directory
5 BTC_DATA=/ou/bitcoinond/collection
```

Code Listing C.2: Docker-compose environment file

C.1.3 Docker daemon configuration file

```
1 #/etc/docker/daemon.json
2 {
3   "ipv6": true,
4   "fixed-cidr-v6": "fd00:dead:beef::/48"
5 }
```

Code Listing C.3: Docker daemon configuration.

C.2 OU Server

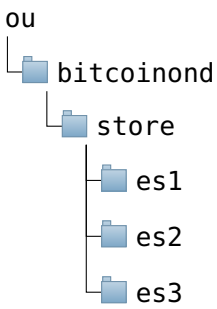
Our platform makes use of a server of the OU. The server has 12 CPU's and a memory capacity of $9.76\,\mathrm{GiB}$.

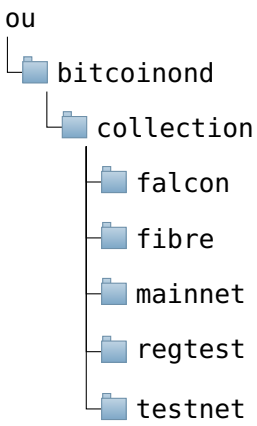
There are two extra disks with a capacity of $300\,\mathrm{GiB}$ mount to the server to expand the storage capacity.

```
#/etc/fstab
/dev/Datavg/Bitcoinlv   /ou/bitcoinond/collection  xfs  defaults   1  2
/dev/Datavg/Bitcoin2lv  /ou/bitcoinond/store       xfs  defaults   1  2
```

Code Listing C.4: Disk mount configuration

The directorie structure is as follows:





```
$ crontab -l
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.Pm1vMU installed on Sun Mar  3 15:39:11 2019)
# (Cronie version 4.2)
# Every day run the collection script for the mainnet
01 00 * * * /home/hho/Project/geolocationofbitcoinminers/src/scripts/
    bglf_collect_all_data_2.sh > /home/hho/Project/cron.log 2>&1
# Every day run the collection script for the testnet
40 15 * * * /home/hho/Project/geolocationofbitcoinminers/src/scripts/
    bglf_collect_all_data_for_testnet_2.sh > /home/hho/Project/cron_test.log
    2>&1
```

Code Listing C.5: Crontab configutation

C.3 Elasticsearch

C.3.1 Mapping Node Information

```
        {
            "settings": {
                "number_of_shards": 3,
                "number_of_replicas": 0
            },
            "mappings": {
                "node_info": {
                    "dynamic": "strict",
                    "properties": {
                        "date": {
                            "type": "date",
                        },
                        "network_type": {
                            "type": "text",
                            "fields": {
                                "raw": {
                                    "type": "keyword"
                                }
                            }
                        },
                        "run_id": {
                            "type": "integer"
                        },
                        "services": {
                            "type": "integer"
                        },
```

```
27                          "relay": {
28                              "type": "text",
29                              "fields": {
30                                  "raw": {
31                                      "type": "keyword"
32                                  }
33                              }
34                          },
35                          "address_type": {
36                              "type": "text",
37                              "fields": {
38                                  "raw": {
39                                      "type": "keyword"
40                                  }
41                              }
42                          },
43                          "tor_address": {
44                              "type": "text",
45                              "fields": {
46                                  "raw": {
47                                      "type": "keyword"
48                                  }
49                              }
50                          },
51                          "ip": {
52                              "type": "ip"
53                          },
54                          "port": {
55                              "type": "integer"
56                          },
57                          "total_of_peers": {
58                              "type": "integer"
59                          },
60                          "version": {
61                              "type": "text",
62                              "fields": {
63                                  "raw": {
64                                      "type": "keyword"
65                                  }
66                              }
67                          },
68                          "user_agent": {
69                              "type": "text",
70                              "fields": {
71                                  "raw": {
72                                      "type": "keyword"
73                                  }
```

```
74                                  }
75                              },
76                              "status": {
77                                  "type": "text",
78                                  "fields": {
79                                      "raw": {
80                                          "type": "keyword"
81                                      }
82                                  }
83                              },
84                              "network_error": {
85                                  "type": "text",
86                                  "fields": {
87                                      "raw": {
88                                          "type": "keyword"
89                                      }
90                                  }
91                              },
92                              "connection_rtt_time": {
93                                  "type": "float"
94                              },
95                              "geoip": {
96                                  "properties": {
97                                      "country_name": {
98                                          "type": "text",
99                                          "fields": {
100                                             "raw": {
101                                                 "type": "keyword"
102                                             }
103                                         }
104                                     },
105                                     "city_name": {
106                                         "type": "text",
107                                         "fields": {
108                                             "raw": {
109                                                 "type": "keyword"
110                                             }
111                                         }
112                                     },
113                                     "country_iso_code": {
114                                         "type": "text",
115                                         "fields": {
116                                             "raw": {
117                                                 "type": "keyword"
118                                             }
119                                         }
120                                     },
```

```
121                                     "location": {
122                                         "type": "geo_point"
123                                     },
124                                     "asn": {
125                                         "type": "text",
126                                         "fields": {
127                                             "raw": {
128                                                 "type": "keyword"
129                                             }
130                                         }
131                                     },
132                                     "org": {
133                                         "type": "text",
134                                         "fields": {
135                                             "raw": {
136                                                 "type": "keyword"
137                                             }
138                                         }
139                                     },
140                                     "timezone": {
141                                         "type": "text",
142                                         "fields": {
143                                             "raw": {
144                                                 "type": "keyword"
145                                             }
146                                         }
147                                     },
148                                     "longitude": {
149                                         "type": "float",
150                                     },
151                                     "latitude": {
152                                         "type": "float",
153                                     },
154                                     "rdns": {
155                                         "type": "text",
156                                         "fields": {
157                                             "raw": {
158                                                 "type": "keyword"
159                                             }
160                                         }
161                                     }
162                                 }
163                             }
164                         }
165                     }
166                 }
167             }
```

Code Listing C.6: Mapping Node Information

C.3.2 Mapping Node and Peer relation

```
{
            "settings": {
                "number_of_shards": 5,
                "number_of_replicas": 0
            },
            "mappings": {
                "node_peer": {
                    "dynamic": "strict",
                    "properties": {
                        "date": {
                            "type": "date",
                        },
                        "network_type": {
                            "type": "text",
                            "fields": {
                                "raw": {
                                    "type": "keyword"
                                }
                            }
                        },
                        "run_id": {
                            "type": "integer"
                        },
                        "node_address_type": {
                            "type": "text",
                            "fields": {
                                "raw": {
                                    "type": "keyword"
                                }
                            }
                        },
                        "node_tor_address": {
                            "type": "text",
                            "fields": {
                                "raw": {
                                    "type": "keyword"
                                }
                            }
                        },
                        "node_ip": {
```

```
41                                  "type": "ip"
42                              },
43                              "node_port": {
44                                  "type": "integer"
45                              },
46                              "node_services": {
47                                  "type": "integer"
48                              },
49                              "node_relay": {
50                                  "type": "text",
51                                  "fields": {
52                                      "raw": {
53                                          "type": "keyword"
54                                      }
55                                  }
56                              },
57                              "node_version": {
58                                  "type": "text",
59                                  "fields": {
60                                      "raw": {
61                                          "type": "keyword"
62                                      }
63                                  }
64                              },
65                              "node_user_agent": {
66                                  "type": "text",
67                                  "fields": {
68                                      "raw": {
69                                          "type": "keyword"
70                                      }
71                                  }
72                              },
73                              "node_status": {
74                                  "type": "text",
75                                  "fields": {
76                                      "raw": {
77                                          "type": "keyword"
78                                      }
79                                  }
80                              },
81                              "node_network_error": {
82                                  "type": "text",
83                                  "fields": {
84                                      "raw": {
85                                          "type": "keyword"
86                                      }
87                                  }
```

```
88                          },
89                          "node_total_of_peers": {
90                              "type": "integer"
91                          },
92                          "node_connection_rtt_time": {
93                              "type": "integer"
94                          },
95                          "node_geoip": {
96                              "properties": {
97                                  "country_name": {
98                                      "type": "text",
99                                      "fields": {
100                                         "raw": {
101                                             "type": "keyword"
102                                         }
103                                     }
104                                 },
105                                 "city_name": {
106                                     "type": "text",
107                                     "fields": {
108                                         "raw": {
109                                             "type": "keyword"
110                                         }
111                                     }
112                                 },
113                                 "country_iso_code": {
114                                     "type": "text",
115                                     "fields": {
116                                         "raw": {
117                                             "type": "keyword"
118                                         }
119                                     }
120                                 },
121                                 "location": {
122                                     "type": "geo_point"
123                                 },
124                                 "asn": {
125                                     "type": "text",
126                                     "fields": {
127                                         "raw": {
128                                             "type": "keyword"
129                                         }
130                                     }
131                                 },
132                                 "org": {
133                                     "type": "text",
134                                     "fields": {
```

```
135                                         "raw": {
136                                             "type": "keyword"
137                                         }
138                                     }
139                                 },
140                                 "timezone": {
141                                     "type": "text",
142                                     "fields": {
143                                         "raw": {
144                                             "type": "keyword"
145                                         }
146                                     }
147                                 },
148                                 "longitude": {
149                                     "type": "float",
150                                 },
151                                 "latitude": {
152                                     "type": "float",
153                                 },
154                                 "rdns": {
155                                     "type": "text",
156                                     "fields": {
157                                         "raw": {
158                                             "type": "keyword"
159                                         }
160                                     }
161                                 }
162                             }
163                         },
164                         "peer_address_type": {
165                             "type": "text",
166                             "fields": {
167                                 "raw": {
168                                     "type": "keyword"
169                                 }
170                             }
171                         },
172                         "peer_tor_address": {
173                             "type": "text",
174                             "fields": {
175                                 "raw": {
176                                     "type": "keyword"
177                                 }
178                             }
179                         },
180                         "peer_ip": {
181                             "type": "ip"
```

```
182                         },
183                         "peer_port": {
184                             "type": "integer"
185                         },
186                         "peer_services": {
187                             "type": "integer"
188                         },
189                         "peer_timestamp": {
190                             "type": "float"
191                         },
192                         "peer_timestamp_str": {
193                             "type": "text",
194                             "fields": {
195                                 "raw": {
196                                     "type": "keyword"
197                                 }
198                             }
199                         },
200                         "peer_geoip": {
201                             "properties": {
202                                 "country_name": {
203                                     "type": "text",
204                                     "fields": {
205                                         "raw": {
206                                             "type": "keyword"
207                                         }
208                                     }
209                                 },
210                                 "city_name": {
211                                     "type": "text",
212                                     "fields": {
213                                         "raw": {
214                                             "type": "keyword"
215                                         }
216                                     }
217                                 },
218                                 "country_iso_code": {
219                                     "type": "text",
220                                     "fields": {
221                                         "raw": {
222                                             "type": "keyword"
223                                         }
224                                     }
225                                 },
226                                 "location": {
227                                     "type": "geo_point"
228                                 },
```

```
229                                     "asn": {
230                                         "type": "text",
231                                         "fields": {
232                                             "raw": {
233                                                 "type": "keyword"
234                                             }
235                                         }
236                                     },
237                                     "org": {
238                                         "type": "text",
239                                         "fields": {
240                                             "raw": {
241                                                 "type": "keyword"
242                                             }
243                                         }
244                                     },
245                                     "timezone": {
246                                         "type": "text",
247                                         "fields": {
248                                             "raw": {
249                                                 "type": "keyword"
250                                             }
251                                         }
252                                     },
253                                     "longitude": {
254                                         "type": "float",
255                                     },
256                                     "latitude": {
257                                         "type": "float",
258                                     },
259                                     "rdns": {
260                                         "type": "text",
261                                         "fields": {
262                                             "raw": {
263                                                 "type": "keyword"
264                                             }
265                                         }
266                                     }
267                                 }
268                             }
269                         }
270                     }
271                 }
272             }
```

Code Listing C.7: Mapping Node and Peer relation

C.3.3 Mapping Bitnodes Information

```

            "settings": {
                "number_of_shards": 3,
                "number_of_replicas": 0
            },
            "mappings": {
                "nodes": {
                    "dynamic": "strict",
                    "properties": {
                        "date": {
                            "type": "date",
                        },
                        "run_id": {
                            "type": "integer"
                        },
                        "connected_since": {
                            "type": "date",
                        },
                        "services": {
                            "type": "integer"
                        },
                        "address_type": {
                            "type": "text",
                            "fields": {
                                "raw": {
                                    "type": "keyword"
                                }
                            }
                        },
                        "tor_address": {
                            "type": "text",
                            "fields": {
                                "raw": {
                                    "type": "keyword"
                                }
                            }
                        },
                        "ip": {
                            "type": "ip"
                        },
                        "port": {
                            "type": "integer"
                        },
                        "height": {
                            "type": "integer"
```

```
46                          },
47                          "version": {
48                              "type": "text",
49                              "fields": {
50                                  "raw": {
51                                      "type": "keyword"
52                                  }
53                              }
54                          },
55                          "user_agent": {
56                              "type": "text",
57                              "fields": {
58                                  "raw": {
59                                      "type": "keyword"
60                                  }
61                              }
62                          },
63                          "geoip": {
64                              "properties": {
65                                  "country_name": {
66                                      "type": "text",
67                                      "fields": {
68                                          "raw": {
69                                              "type": "keyword"
70                                          }
71                                      }
72                                  },
73                                  "city_name": {
74                                      "type": "text",
75                                      "fields": {
76                                          "raw": {
77                                              "type": "keyword"
78                                          }
79                                      }
80                                  },
81                                  "country_iso_code": {
82                                      "type": "text",
83                                      "fields": {
84                                          "raw": {
85                                              "type": "keyword"
86                                          }
87                                      }
88                                  },
89                                  "location": {
90                                      "type": "geo_point"
91                                  },
92                                  "asn": {
```

```
93                                      "type": "text",
94                                      "fields": {
95                                          "raw": {
96                                              "type": "keyword"
97                                          }
98                                      }
99                                  },
100                                 "org": {
101                                     "type": "text",
102                                     "fields": {
103                                         "raw": {
104                                             "type": "keyword"
105                                         }
106                                     }
107                                 },
108                                 "timezone": {
109                                     "type": "text",
110                                     "fields": {
111                                         "raw": {
112                                             "type": "keyword"
113                                         }
114                                     }
115                                 },
116                                 "longitude": {
117                                     "type": "float",
118                                 },
119                                 "latitude": {
120                                     "type": "float",
121                                 },
122                                 "rdns": {
123                                     "type": "text",
124                                     "fields": {
125                                         "raw": {
126                                             "type": "keyword"
127                                         }
128                                     }
129                                 }
130                             }
131                         }
132                     }
133                 }
134             }
135         }
```

Code Listing C.8: Mapping Bitnodes Information

C.3.4 Mapping Shodan Information

```
{
            "settings": {
                "number_of_shards": 3,
                "number_of_replicas": 0
            },
            "mappings": {
                "data": {
                    "dynamic": "strict",
                    "properties": {
                        "date": {
                            "type": "date",
                        },
                        "updated_at": {
                            "type": "date",
                        },
                        "run_id": {
                            "type": "integer"
                        },
                        "address_type": {
                            "type": "text",
                            "fields": {
                                "raw": {
                                    "type": "keyword"
                                }
                            }
                        },
                        "tor_address": {
                            "type": "text",
                            "fields": {
                                "raw": {
                                    "type": "keyword"
                                }
                            }
                        },
                        "ip": {
                            "type": "ip"
                        },
                        "port": {
                            "type": "integer"
                        },
                        "search_string": {
                            "type": "text",
                            "fields": {
                                "raw": {
                                    "type": "keyword"
```

```
46                                  }
47                              }
48                          },
49                          "data": {
50                              "type": "text",
51                              "fields": {
52                                  "raw": {
53                                      "type": "keyword"
54                                  }
55                              }
56                          },
57                          "geoip": {
58                              "properties": {
59                                  "country_name": {
60                                      "type": "text",
61                                      "fields": {
62                                          "raw": {
63                                              "type": "keyword"
64                                          }
65                                      }
66                                  },
67                                  "city_name": {
68                                      "type": "text",
69                                      "fields": {
70                                          "raw": {
71                                              "type": "keyword"
72                                          }
73                                      }
74                                  },
75                                  "country_iso_code": {
76                                      "type": "text",
77                                      "fields": {
78                                          "raw": {
79                                              "type": "keyword"
80                                          }
81                                      }
82                                  },
83                                  "location": {
84                                      "type": "geo_point"
85                                  },
86                                  "asn": {
87                                      "type": "text",
88                                      "fields": {
89                                          "raw": {
90                                              "type": "keyword"
91                                          }
92                                      }
```

```
                                    },
                                    "org": {
                                        "type": "text",
                                        "fields": {
                                            "raw": {
                                                "type": "keyword"
                                            }
                                        }
                                    },
                                    "timezone": {
                                        "type": "text",
                                        "fields": {
                                            "raw": {
                                                "type": "keyword"
                                            }
                                        }
                                    },
                                    "longitude": {
                                        "type": "float",
                                    },
                                    "latitude": {
                                        "type": "float",
                                    },
                                    "rdns": {
                                        "type": "text",
                                        "fields": {
                                            "raw": {
                                                "type": "keyword"
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
```

Code Listing C.9: Mapping Shodan Information

C.3.5 Mapping Censys Information

```
{
            "settings": {
                "number_of_shards": 3,
                "number_of_replicas": 0
```

```
5           },
6           "mappings": {
7               "data": {
8                   "dynamic": "strict",
9                   "properties": {
10                      "date": {
11                          "type": "date",
12                      },
13                      "updated_at": {
14                          "type": "date",
15                      },
16                      "run_id": {
17                          "type": "integer"
18                      },
19                      "services": {
20                          "type": "integer"
21                      },
22                      "address_type": {
23                          "type": "text",
24                          "fields": {
25                              "raw": {
26                                  "type": "keyword"
27                              }
28                          }
29                      },
30                      "tor_address": {
31                          "type": "text",
32                          "fields": {
33                              "raw": {
34                                  "type": "keyword"
35                              }
36                          }
37                      },
38                      "ip": {
39                          "type": "ip"
40                      },
41                      "search_string": {
42                          "type": "text",
43                          "fields": {
44                              "raw": {
45                                  "type": "keyword"
46                              }
47                          }
48                      },
49                      "port_80_www_authenticate": {
50                          "type": "text",
51                          "fields": {
```

```
52                                  "raw": {
53                                      "type": "keyword"
54                                  }
55                              }
56                          },
57                          "port_8080_www_authenticate": {
58                              "type": "text",
59                              "fields": {
60                                  "raw": {
61                                      "type": "keyword"
62                                  }
63                              }
64                          },
65                          "geoip": {
66                              "properties": {
67                                  "country_name": {
68                                      "type": "text",
69                                      "fields": {
70                                          "raw": {
71                                              "type": "keyword"
72                                          }
73                                      }
74                                  },
75                                  "city_name": {
76                                      "type": "text",
77                                      "fields": {
78                                          "raw": {
79                                              "type": "keyword"
80                                          }
81                                      }
82                                  },
83                                  "country_iso_code": {
84                                      "type": "text",
85                                      "fields": {
86                                          "raw": {
87                                              "type": "keyword"
88                                          }
89                                      }
90                                  },
91                                  "location": {
92                                      "type": "geo_point"
93                                  },
94                                  "asn": {
95                                      "type": "text",
96                                      "fields": {
97                                          "raw": {
98                                              "type": "keyword"
```

```
99                                          }
100                                     }
101                                 },
102                                 "org": {
103                                     "type": "text",
104                                     "fields": {
105                                         "raw": {
106                                             "type": "keyword"
107                                         }
108                                     }
109                                 },
110                                 "timezone": {
111                                     "type": "text",
112                                     "fields": {
113                                         "raw": {
114                                             "type": "keyword"
115                                         }
116                                     }
117                                 },
118                                 "longitude": {
119                                     "type": "float",
120                                 },
121                                 "latitude": {
122                                     "type": "float",
123                                 },
124                                 "rdns": {
125                                     "type": "text",
126                                     "fields": {
127                                         "raw": {
128                                             "type": "keyword"
129                                         }
130                                     }
131                                 }
132                             }
133                         }
134                     }
135                 }
136             }
137         }
```

Code Listing C.10: Mapping Censys Information

C.3.6 Mapping Mining Index

```
1 {
2             "settings": {
```

```
3                   "number_of_shards": 3,
4                   "number_of_replicas": 0
5               },
6               "mappings": {
7                   "bitcoin": {
8                       "dynamic": "strict",
9                       "properties": {
10                          "date": {
11                              "type": "date",
12                          },
13                          "run_id": {
14                              "type": "integer"
15                          },
16                          "services": {
17                              "type": "integer"
18                          },
19                          "ip": {
20                              "type": "ip"
21                          },
22                          "user_agent": {
23                              "type": "text",
24                              "fields": {
25                                  "raw": {
26                                      "type": "keyword"
27                                  }
28                              }
29                          },
30                          "node_index": {
31                              "type": "float"
32                          },
33                          "services_index": {
34                              "type": "float"
35                          },
36                          "fibre_index": {
37                              "type": "float"
38                          },
39                          "falcon_index": {
40                              "type": "float"
41                          },
42                          "relay_index": {
43                              "type": "float"
44                          },
45                          "peer_last_seen_index": {
46                              "type": "float"
47                          },
48                          "mining_index": {
49                              "type": "float"
```

```
50                          },
51                          "geoip": {
52                              "properties": {
53                                  "country_name": {
54                                      "type": "text",
55                                      "fields": {
56                                          "raw": {
57                                              "type": "keyword"
58                                          }
59                                      }
60                                  },
61                                  "location": {
62                                      "type": "geo_point"
63                                  },
64                                  "rdns": {
65                                      "type": "text",
66                                      "fields": {
67                                          "raw": {
68                                              "type": "keyword"
69                                          }
70                                      }
71                                  }
72                              }
73                          }
74                      }
75                  }
76              }
77          }
```

Code Listing C.11: Mapping Mining Index

D. BGLF documentation

D.1 Program structure

```
.
├── BGLF.py
├── README.md
├── __init__.py
├── apps
│   ├── __init__.py
│   ├── bglf_database
│   │   ├── __init__.py
│   │   ├── bglf_calculate_db.py
│   │   ├── bglf_db.py
│   │   └── bglf_run_db.py
│   ├── bglf_functions.py
│   ├── config_parser.py
│   ├── constants.py
│   ├── crawl_btc.py
│   ├── elastic.py
│   ├── geoip.py
│   ├── mining_index
│   │   ├── __init__.py
│   │   ├── calculate_falcon_index.py
│   │   ├── calculate_fibre_index.py
│   │   ├── calculate_index.py
│   │   ├── calculate_mining_index.py
│   │   ├── calculate_node_index.py
│   │   ├── calculate_peer_last_seen_index.py
│   │   ├── calculate_relay_index.py
│   │   ├── calculate_services_index.py
│   │   └── mining_index_db.py
│   ├── network_analysis.py
│   ├── protocol.py
│   ├── scrappers
│   │   ├── __init__.py
│   │   ├── bitnodes_scrapper.py
│   │   ├── censys_scrapper.py
│   │   ├── scrapper.py
│   │   └── shodan_scrapper.py
│   └── utils.py
├── config
│   ├── bglf.db
│   ├── client_secret_bglf.json
│   ├── logging.conf
│   └── settings.conf
├── data
├── geoip
│   ├── COPYRIGHT.txt
│   ├── GeoLite2-ASN.mmdb
│   ├── GeoLite2-City.mmdb
│   ├── GeoLite2-Country.mmdb
│   ├── LICENSE.txt
│   ├── README.txt
│   └── update.sh
├── log
├── requirements.txt
└── scripts
    ├── bglf_collect_all_data.sh
    └── bglf_collect_all_data_for_testnet.sh
```

Figure D.1: BGLF program structure

D.2 UML diagram

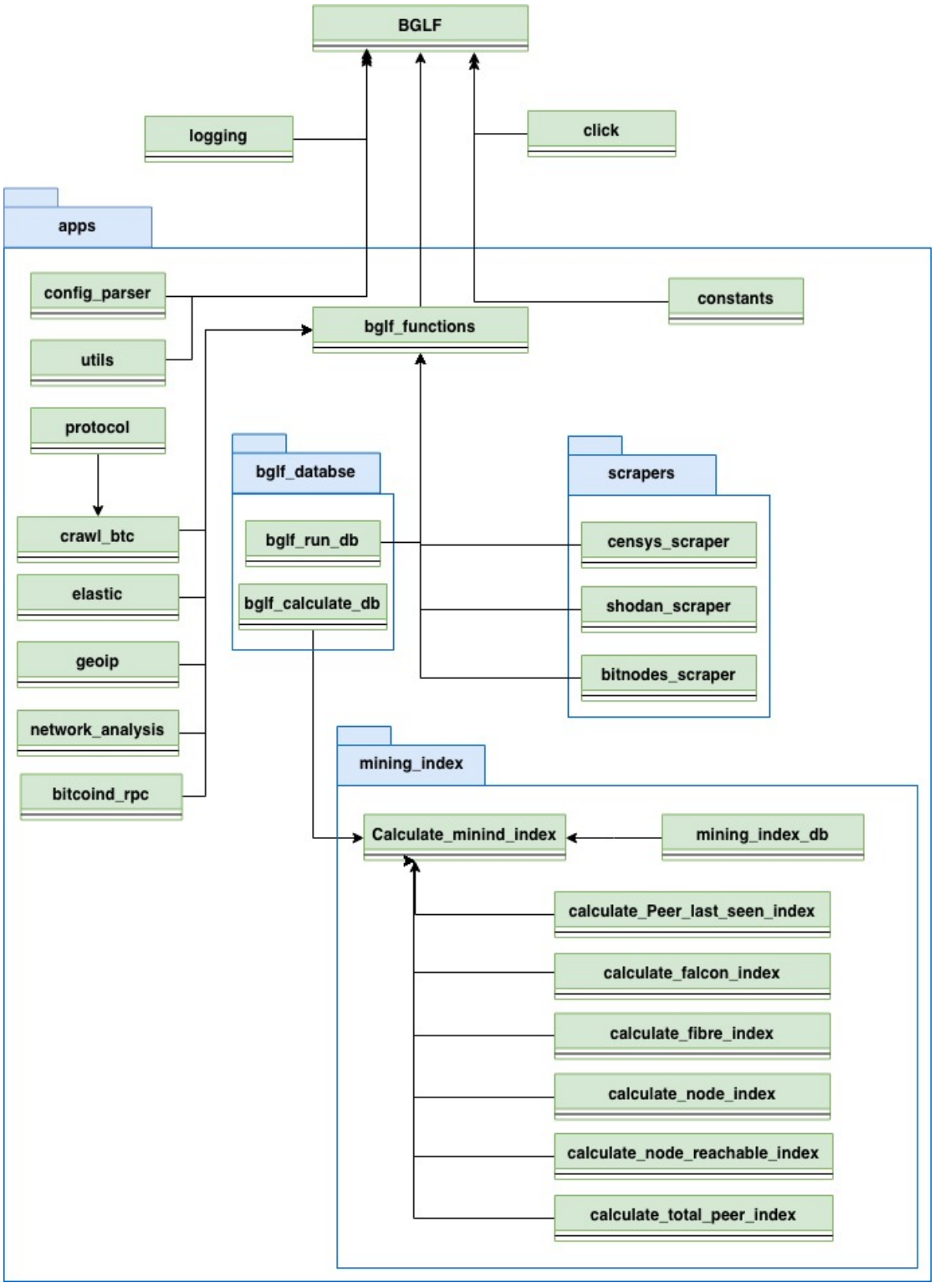


Figure D.2: BGLF UML class diagram.

D.2.1 Command line interface

The Bitcoin GeoLocation Finder (BGLF) is a command line tool. The different options of BGLF are structured in a menu. These menu and the different menu options are depicted here below:

```
$ python3 ./BGLF.py --help
Usage: BGLF.py [OPTIONS] COMMAND [ARGS]...

  Bitcoin GeoLocation Finder is a tool and a set of utilities for
  Geolocation of the miners of the Bitcoin Network. The software is part of
  a Bachelor graduation at the Open University of the Netherlands.

  The software is developed by, ing H.J. Hopman & C. Hoorenman MSc.

Options:
  --config-file PATH              Specify a configuration file
                                  (Default =
                                  config/settings.conf)
  --verbose / --no-verbose        Enable debug output.
  --network_type [mainnet|testnet|regtest|fibre|falcon]
                                  Choice Bitcion Network type
                                  (Default=mainnet)
  --store_data / --no_store_data  For testing purpose disable storing of
                                  collected data.
  --help                          Show this message and exit.

Commands:
  analyze    Analyse the collected data.
  calculate  Calculate the mining index or parts of the index.
  collect    Collect the data for geolocation of bitcion miners.
  crawl      Collect the geolocation of bitcion miners.
  dashboard  Show information of the different runs (Crawls/Scrappes).
  scrape     Try to collect as much as possible about the geolocation of
             bitcion miners with OSINT.
```

Code Listing D.1: BGLF argument options

D.2.1.1 The *crawl* menu

```
$ python3 ./BGLF.py crawl
Usage: BGLF.py crawl [OPTIONS] COMMAND [ARGS]...

  Collect the geolocation of bitcion miners.

Options:
  --help  Show this message and exit.

Commands:
  bitcoin-network  Crawl the bitcoin network (mainnet, testnet, regtest) to
      ...
  falcon           Crawl the FALCON (Cornell) fastrelaynetwork to collect...
  fibre            Crawl the FIBRE fastrelaynetwork to collect bitcoin...
```

Code Listing D.2: BGLF crawl options

D.2.1.2 The *scrape* menu

```
$ python3 ./BGLF.py scrape
Usage: BGLF.py scrape [OPTIONS] COMMAND [ARGS]...

  Collect the geolocation of bitcion miners via scraping different
  websites/search engines.

Options:
  --help  Show this message and exit.

Commands:
  bitnodes_scraper  Scrape Bitnodes.
  censys_scraper    Scrape Censys.
  scrapers          Run all scrapers.
  shodan_scraper    Scrape Shodan.
```

Code Listing D.3: BGLF scrape options

D.2.1.3 The *collect* menu

```
$ python3 ./BGLF.py collect
Usage: BGLF.py collect [OPTIONS] COMMAND [ARGS]...

  Collect all the date.

Options:
  --help  Show this message and exit.

Commands:
  one_shot  Collect the data.
```

Code Listing D.4: BGLF collect options

D.2.1.4 The *calculate* menu

```
$ python3 ./BGLF.py calculate
Usage: BGLF.py calculate [OPTIONS] COMMAND [ARGS]...

  Calculate the mining index or parts of the index.

Options:
  --help  Show this message and exit.

Commands:
  mining_index  Calculate the Mining Index.
```

Code Listing D.5: BGLF calculate options

For calculating the mining index based on a run ID, BGLF has the following option:

```
$ python3 ./BGLF.py calculate mining_index --help
Usage: BGLF.py calculate mining_index [OPTIONS]

  Calculate the Mining Index.

Options:
  --run_id INTEGER  Give a run_id for the data to exported. If no id is given
,
                    then the latest data exported.
  --help            Show this message and exit.
```

Code Listing D.6: BGLF calculate options

D.2.2 The *analyz* menu

```
$ python3 ./BGLF.py analyze
Usage: BGLF.py analyze [OPTIONS] COMMAND [ARGS]...

  Analyze the collected data

Options:
  --help  Show this message and exit.

Commands:
  Calculate_diff        Analyse the data
  convert2Ggeojson      Convert a CSV file with info about Mining Pools to a
      ...
  convertIP2Ggeojson    Convert a CSV file with info (IP address) about...
  export2Gephi          Export all records which belong to a run_id to Gephi
      ...
  export2Gephi4Falcon   Export all records which belong to a run_id and...
  export2Gephi4Fibre    Export all records which belong to a run_id and
                        Fibre...
  exportAllData2CSV     Export all collected data for a run ID to CSV files.
  exportNodeStatus      Export for a run id the node and his status...
```

Code Listing D.7: BGLF analyze options

D.2.3 The *dashboard* menu

The *dashboard* option has a suboption for displaying information about the last ten runs.

```
$ python3 ./BGLF.py dashboard
Usage: BGLF.py dashboard [OPTIONS] COMMAND [ARGS]...

  Show data for the different runs

Options:
  --help  Show this message and exit.

Commands:
  ten_latest_runs  Show information about the 10 latest runs.
```

Code Listing D.8: BGLF analyze options

D.3 BGLF: the settings file

```
[application]
rpcdebug = False
#rpcdebug = True

rpc_user = testuser
rpc_passwd = test123


# Print debug output
debug = False

config_dir = config/
# Database for storing info
config_db = bglf_config.db
# Database for configuration and Archive
bglf_db = bglf.db

[Elasticsearch]
# Information about the data-store
#elcs_host = 127.0.0.1
elcs_host = 145.20.145.2
elcs_port = 9200

[Bitcoind]
# Information about the different Bitcion networks
# Mainnet
btc_mainnet_ip = 145.20.145.2
btc_mainnet_rpc_port = 8332
btc_mainnet_p2p_port = 8333
btc_mainnet_magic_number = f9beb4d9

#TestNet
btc_testnet_ip = 145.20.145.2
btc_testnet_rpc_port = 18332
btc_testnet_p2p_port = 18333
btc_testnet_magic_number = 0b110907

#Regtest
btc_regtest_ip = 145.20.145.2
btc_regtest_rpc_port = 18443
btc_regtest_p2p_port = 18444
btc_regtest_magic_number = fabfb5da

# FIBRE Relay network (Part of mainnet)
btc_fibre_ip = 145.20.145.2
```

```
46 btc_fibre_rpc_port = 8332
47 btc_fibre_p2p_port = 8333
48 btc_fibre_magic_number = f9beb4d9
49
50 # FALCON Relay network (Part of mainnet)
51 btc_falcon_ip = 145.20.145.2
52 btc_falcon_rpc_port = 8332
53 btc_falcon_p2p_port = 8333
54 btc_falcon_magic_number = f9beb4d9
55
56 [Scrappers]
57 # Config information for scrapping the results of different websites
58
59 # Shodan.io
60 shodan_api_key = 8iVIFbpZ5LMtlsZZVuIgeSZSOl1Wbz6t&query
61
62 # Censys.io
63 censys_project_id = censys-io-228820
64 censys_google_json_key_file = config/censys-io-google_services_account.json
65
66 # Search strings for scrapping Shodan & Censys
67 search_strings =
68     antMiner
69     bitmain-inc
70     /Satoshi
71     stratum+tcp
72     port:3333
73
74 [crawl]
75 # Config information for the crawl process
76
77 # Logdir
78 logdir = log/
79
80 # Network magic number
81 # For mainnet
82 #magic_number = f9beb4d9
83 # For testnet
84 magic_number = 0b110907
85 # For regtest
86 #magic_number = fabfb5da
87
88 # List of FIBRE fastrelaynetwork nodes
89 fibre_nodes =
90     v4.us-west.fibre.bitcoinrelaynetwork.org
91     v4.us-east.fibre.bitcoinrelaynetwork.org
92     v4.hk.fibre.bitcoinrelaynetwork.org
```

```
    v4.eu.fibre.bitcoinrelaynetwork.org
    v4.jpy.fibre.bitcoinrelaynetwork.org
    v4.cn.fibre.bitcoinrelaynetwork.org


# Based on /Cornell-Falcon-Network:0.1.0/ from https://bitnodes.earn.com/nodes/leaderboard/?q=Cornell-Falcon
falcon_nodes =
    ec2-34-192-109-181.compute-1.amazonaws.com
    ec2-46-51-216-153.ap-southeast-1.compute.amazonaws.com
    ec2-52-67-166-61.sa-east-1.compute.amazonaws.com
    ec2-35-162-94-196.us-west-2.compute.amazonaws.com
    ec2-52-213-83-100.eu-west-1.compute.amazonaws.com
    ec2-52-199-163-228.ap-northeast-1.compute.amazonaws.com
    ec2-52-9-163-208.us-west-1.compute.amazonaws.com
    ec2-52-78-200-40.ap-northeast-2.compute.amazonaws.com
    ec2-35-156-94-60.eu-central-1.compute.amazonaws.com

# List of DNS seeders to get a subset of reachable nodes
seeders =
    dnsseed.bitcoin.dashjr.org
    dnsseed.bluematt.me
    seed.bitcoin.jonasschnelli.ch
    seed.bitcoin.sipa.be
    seed.bitcoinstats.com
    seed.btc.petertodd.org
    seed.bitnodes.io

# Number of concurrent workers (greenlets)
workers = 750

# Protocol version to use for outgoing version message
protocol_version = 70015

# User agent (BIP 0014) to use for outgoing version message
# ------------------------------------------------------------------------------
# NOTE TO USERS
# Please consider changing the user agent before running an instance of this
# crawler. This is so that users will not confuse your crawler with another
# instance that is already running and generating data for the project.
# ------------------------------------------------------------------------------
user_agent = /bglf.search.com:0.2/

```

```
135 # Blockchain Height
136 height = 0
137
138 # Services to use for outgoing network address message
139 services = 1037
140
141 # Set to 1 to receive all txs
142 relay = 0
143
144 # Socket timeout
145 socket_timeout = 60
146
147 # Max. age for peering node to be included in crawl set
148 # 12 hours
149 max_age = 43200
150 # 24 hours
151 #max_age = 86400
152 # 1 week
153 #max_age = 604800
154
155 # Attempt to establish connection with IPv6 nodes
156 ipv6 = True
157
158 # Limit max. nodes per IPv6 network prefix
159 ipv6_prefix = 64
160 nodes_per_ipv6_prefix = 1
161
162 # List of excluded ASNs
163 exclude_asns =
164
165 # List of excluded IPv4 networks
166 exclude_ipv4_networks =
167     0.0.0.0/8
168     10.0.0.0/8
169     100.64.0.0/10
170     127.0.0.0/8
171     169.254.0.0/16
172     172.16.0.0/12
173     192.0.0.0/24
174     192.0.0.0/29
175     192.0.0.170/32
176     192.0.0.171/32
177     192.0.0.8/32
178     192.0.2.0/24
179     192.168.0.0/16
180     192.175.48.0/24
181     192.31.196.0/24
```

```
182         192.52.193.0/24
183         192.88.99.0/24
184         198.18.0.0/15
185         198.51.100.0/24
186         203.0.113.0/24
187         240.0.0.0/4
188         255.255.255.255/32
189
190     # List of excluded IPv6 networks
191     exclude_ipv6_networks =
192
193     # Exclude IPv4 bogons
194     exclude_ipv4_bogons = True
195
196     # Attempt to establish connection with .onion nodes
197     onion = True
198
199     # Tor proxy is required to connect to .onion address
200     tor_proxy_ip = 145.20.145.2
201     tor_proxy_port = 9050
202
203     # List of initial .onion nodes
204     onion_nodes =
205         4crhf372poejlc44.onion
206         5ghqw4wj6hpgfvdg.onion
207         5k4vwyy5stro33fb.onion
208         bitcoinostk4e4re.onion
209         bk5ejfe56xakvtkk.onion
210         btcnet3utgzyz2bf.onion
211         czsbwh4pq4mh3izl.onion
212         e3tn727fywnioxrc.onion
213         evolynhit7shzeet.onion
214         i2r5tbaizb75h26f.onion
215         jxrvw5iqizppbgml.onion
216         kjy2eqzk4zwi5zd3.onion
217         pqosrh6wfaucet32.onion
218         pt2awtcs2ulm75ig.onion
219         szsm43ou7otwwyfv.onion
220         td7tgof3imei3fm6.onion
221         tfu4kqfhsw5slqp2.onion
222         vso3r6cmjoomhhgg.onion
223         xdnigz4qn5dbbw2t.onion
224         zy3kdqowmrb7xm7h.onion
225
226     # Relative path to directory containing timestamp-prefixed JSON crawl files
227     crawl_dir = data/crawl/f9beb4d9
228
```

```
229 # Relative path to directory containing data for analysis
230 data_dir = data/analysis/
231
232 [Geo]
233 # Location of the Geo information from Maxmind.
234 # Use update.sh in geoip dir. to download all databases
235 geoip_dir = geoip/
236
237 # The city information of an IP prefix
238 geoip_city     = GeoLite2-City.mmdb
239 # The country information of an IP prefix
240 geoip_country = GeoLite2-Country.mmdb
241 # Information about the Autonomous System and Organisation of an IP prefix
242 geoip_asn      = GeoLite2-ASN.mmdb
```

Code Listing D.9: Mapping Node Information

D.4 BGLF - Debug logging example

```
2019-04-14 21:21:41,897,897.9 : INFO    BGLF.bglf - Changing loglevel to DEBUG.
2019-04-14 21:21:41,897,898.0 : DEBUG   BGLF.bglf - Configfile : config/settings.conf
2019-04-14 21:21:41,898,898.0 : DEBUG   config_parser.parse_configuration_file - Parse configuration file : [config/settings.
    conf]
2019-04-14 21:21:41,898,898.9 : DEBUG   config_parser.init_config_paramters - Set Verbose to [False]
2019-04-14 21:21:41,898,898.9 : DEBUG   config_parser.init_config_paramters - Set rpcdebug to [False]
2019-04-14 21:21:41,898,899.0 : DEBUG   config_parser.init_config_paramters - Set RPC user : [testuser]
2019-04-14 21:21:41,899,899.0 : DEBUG   config_parser.init_config_paramters - Set RPC password : [test123]
2019-04-14 21:21:41,899,899.1 : DEBUG   config_parser.init_config_paramters - Set debug : [None]
2019-04-14 21:21:41,899,899.2 : DEBUG   config_parser.init_config_paramters - Set Elasticsearch host to ip [145.20.145.2]
2019-04-14 21:21:41,899,899.2 : DEBUG   config_parser.init_config_paramters - Set Elasticsearch port to [9200]
2019-04-14 21:21:41,899,899.3 : DEBUG   config_parser.init_config_paramters - Set MAINNET entry host ip [145.20.145.2]
2019-04-14 21:21:41,899,899.3 : DEBUG   config_parser.init_config_paramters - Set MAINNET RPC port [8332]
2019-04-14 21:21:41,899,899.4 : DEBUG   config_parser.init_config_paramters - Set MAINNET P2P port [8333]
2019-04-14 21:21:41,899,899.4 : DEBUG   config_parser.init_config_paramters - Set MAINNET magic number [b'f9beb4d9']
2019-04-14 21:21:41,899,899.5 : DEBUG   config_parser.init_config_paramters - Set TESTNET entry host ip [145.20.145.2]
2019-04-14 21:21:41,899,899.5 : DEBUG   config_parser.init_config_paramters - Set TESTNET RPC port [18332]
2019-04-14 21:21:41,899,899.6 : DEBUG   config_parser.init_config_paramters - Set TESTNET P2P port [18333]
2019-04-14 21:21:41,899,899.7 : DEBUG   config_parser.init_config_paramters - Set TESTNET magic number [b'0b110907']
2019-04-14 21:21:41,899,899.8 : DEBUG   config_parser.init_config_paramters - Set REGTEST entry host ip [145.20.145.2]
2019-04-14 21:21:41,899,899.8 : DEBUG   config_parser.init_config_paramters - Set REGTEST RPC port [18443]
2019-04-14 21:21:41,899,899.9 : DEBUG   config_parser.init_config_paramters - Set REGTEST P2P port [18444]
2019-04-14 21:21:41,899,899.9 : DEBUG   config_parser.init_config_paramters - Set REGTEST magic number [b'fabfb5da']
2019-04-14 21:21:41,899,900.0 : DEBUG   config_parser.init_config_paramters - Set FIBRE entry host ip [145.20.145.2]
2019-04-14 21:21:41,900,900.0 : DEBUG   config_parser.init_config_paramters - Set FIBRE RPC port [8332]
2019-04-14 21:21:41,900,900.1 : DEBUG   config_parser.init_config_paramters - Set FIBRE P2P port [8333]
2019-04-14 21:21:41,900,900.1 : DEBUG   config_parser.init_config_paramters - Set FIBRE magic number [b'f9beb4d9']
2019-04-14 21:21:41,900,900.2 : DEBUG   config_parser.init_config_paramters - Set FALCON entry host ip [145.20.145.2]
2019-04-14 21:21:41,900,900.2 : DEBUG   config_parser.init_config_paramters - Set FALCON RPC port [8332]
2019-04-14 21:21:41,900,900.3 : DEBUG   config_parser.init_config_paramters - Set FALCON P2P port [8333]
2019-04-14 21:21:41,900,900.3 : DEBUG   config_parser.init_config_paramters - Set FALCON magic number [b'f9beb4d9']
2019-04-14 21:21:41,900,900.4 : DEBUG   config_parser.init_config_paramters - Set logdir: [log/]
2019-04-14 21:21:41,900,900.4 : DEBUG   config_parser.init_config_paramters - Set FIBRE nodes : [['v4.us-west.fibre.
    bitcoinrelaynetwork.org', 'v4.us-east.fibre.bitcoinrelaynetwork.org', 'v4.hk.fibre.bitcoinrelaynetwork.org', 'v4.eu.fibre.
    bitcoinrelaynetwork.org', 'v4.jpy.fibre.bitcoinrelaynetwork.org', 'v4.cn.fibre.bitcoinrelaynetwork.org']]
2019-04-14 21:21:41,900,900.5 : DEBUG   config_parser.init_config_paramters - Set FALCON nodes : [['ec2-34-192-109-181.compute
    -1.amazonaws.com', 'ec2-46-51-216-153.ap-southeast-1.compute.amazonaws.com', 'ec2-52-67-166-61.sa-east-1.compute.amazonaws
    .com', 'ec2-35-162-94-196.us-west-2.compute.amazonaws.com', 'ec2-52-213-83-100.eu-west-1.compute.amazonaws.com', 'ec2
    -52-199-163-228.ap-northeast-1.compute.amazonaws.com', 'ec2-52-9-163-208.us-west-1.compute.amazonaws.com', 'ec2
    -52-78-200-40.ap-northeast-2.compute.amazonaws.com', 'ec2-35-156-94-60.eu-central-1.compute.amazonaws.com']]
2019-04-14 21:21:41,900,900.5 : DEBUG   config_parser.init_config_paramters - Set seeders : [['dnsseed.bitcoin.dashjr.org', '
    dnsseed.bluematt.me', 'seed.bitcoin.jonasschnelli.ch', 'seed.bitcoin.sipa.be', 'seed.bitcoinstats.com', 'seed.btc.
    petertodd.org', 'seed.bitnodes.io']]
2019-04-14 21:21:41,900,900.6 : DEBUG   config_parser.init_config_paramters - Set workers : [750]
2019-04-14 21:21:41,900,900.6 : DEBUG   config_parser.init_config_paramters - Set source address : [0.0.0.0]
2019-04-14 21:21:41,900,900.6 : DEBUG   config_parser.init_config_paramters - Set protocol version : [70015]
2019-04-14 21:21:41,900,900.7 : DEBUG   config_parser.init_config_paramters - Set user agent : [/bglf.search.com:0.2/]
2019-04-14 21:21:41,900,900.7 : DEBUG   config_parser.init_config_paramters - Set height : [0]
2019-04-14 21:21:41,900,900.8 : DEBUG   config_parser.init_config_paramters - Set services : [1037]
2019-04-14 21:21:41,900,900.8 : DEBUG   config_parser.init_config_paramters - Set relay : [0]
2019-04-14 21:21:41,900,900.9 : DEBUG   config_parser.init_config_paramters - Set socket timeout : [60]
2019-04-14 21:21:41,900,901.0 : DEBUG   config_parser.init_config_paramters - Set max age : [43200]
2019-04-14 21:21:41,901,901.0 : DEBUG   config_parser.init_config_paramters - Set ipv6 : [True]
2019-04-14 21:21:41,901,901.1 : DEBUG   config_parser.init_config_paramters - Set ipv6 prefix : [64]
2019-04-14 21:21:41,901,901.1 : DEBUG   config_parser.init_config_paramters - Set nodes per ipv6 prefix : [1]
2019-04-14 21:21:41,901,901.2 : DEBUG   config_parser.init_config_paramters - Set exclude asns : [['']]
2019-04-14 21:21:41,901,901.4 : DEBUG   config_parser.init_config_paramters - Set exclude ipv4 networks : [[(3221225642,
    4294967295), (3227017984, 4294967040), (3323068416, 4294836224), (3232235520, 4294901760), (4026531840, 4026531840),
    (3224682752, 4294967040), (3405803776, 4294967040), (3325256704, 4294967040), (2130706432, 4278190080), (1681915904,
    4290772992), (2851995648, 4294901760), (3221225472, 4294967288), (3223307264, 4294967040), (3221225480, 4294967295),
    (3221225984, 4294967040), (0, 4278190080), (167772160, 4278190080), (2886729728, 4293918720), (3221225472, 4294967040),
    (3232706560, 4294967040), (3221225643, 4294967295), (4294967295, 4294967295)]]
2019-04-14 21:21:41,901,901.5 : DEBUG   config_parser.init_config_paramters - Set initial exclude ipv4 networks : [{(3221225642,
     4294967295), (3227017984, 4294967040), (3323068416, 4294836224), (3232235520, 4294901760), (4026531840, 4026531840),
    (3224682752, 4294967040), (3405803776, 4294967040), (3325256704, 4294967040), (2130706432, 4278190080), (1681915904,
    4290772992), (2851995648, 4294901760), (3221225472, 4294967288), (3223307264, 4294967040), (3221225480, 4294967295),
    (3221225984, 4294967040), (0, 4278190080), (167772160, 4278190080), (2886729728, 4293918720), (3221225472, 4294967040),
    (3232706560, 4294967040), (3221225643, 4294967295), (4294967295, 4294967295)}]
2019-04-14 21:21:41,901,901.5 : DEBUG   config_parser.init_config_paramters - Set exclude ipv6 networks : [[]]
2019-04-14 21:21:41,901,901.6 : DEBUG   config_parser.init_config_paramters - Set exclude ipv4 bogons : [True]
2019-04-14 21:21:41,901,901.6 : DEBUG   config_parser.init_config_paramters - Set onion : [True]
2019-04-14 21:21:41,901,901.7 : DEBUG   config_parser.init_config_paramters - Set tor proxy : [('145.20.145.2', 9050)]
```

```
2019-04-14 21:21:41,901,901.7 : DEBUG   config_parser.init_config_paramters - Set onion nodes : [['4crhf372poejlc44.onion', '5ghqw4wj6hpgfvdg.onion', '5k4vwyy5stro33fb.onion', 'bitcoinostk4e4re.onion', 'bk5ejfe56xakvtkk.onion', 'btcnet3utgzyz2bf.onion', 'czsbwh4pq4mh3izl.onion', 'e3tn727fywnioxrc.onion', 'evolynhit7shzeet.onion', 'i2r5tbaizb75h26f.onion', 'jxrvw5iqizppbgml.onion', 'kjy2eqzk4zwi5zd3.onion', 'pqosrh6wfaucet32.onion', 'pt2awtcs2ulm75ig.onion', 'szsm43ou7otwwyfv.onion', 'td7tgof3imei3fm6.onion', 'tfu4kqfhsw5slqp2.onion', 'vso3r6cmjoomhhgg.onion', 'xdnigz4qn5dbbw2t.onion', 'zy3kdqowmrb7xm7h.onion']]
2019-04-14 21:21:41,901,901.8 : DEBUG   config_parser.init_config_paramters - Set include checked : [False]
2019-04-14 21:21:41,901,901.8 : DEBUG   config_parser.init_config_paramters - Set crawl dir. : [data/crawl/f9beb4d9]
2019-04-14 21:21:41,901,901.9 : DEBUG   config_parser.init_config_paramters - Set GeoIp dir. : [geoip/]
2019-04-14 21:21:41,901,902.0 : DEBUG   config_parser.init_config_paramters - Set GeoIP City DB : [GeoLite2-City.mmdb]
2019-04-14 21:21:41,902,902.0 : DEBUG   config_parser.init_config_paramters - Set GeoIP Country DB : [GeoLite2-Country.mmdb]
2019-04-14 21:21:41,902,902.0 : DEBUG   config_parser.init_config_paramters - Set GeoIP ASN DB : [GeoLite2-ASN.mmdb]
2019-04-14 21:21:41,902,902.1 : DEBUG   config_parser.init_config_paramters - Set config DB : [bglf_config.db]
2019-04-14 21:21:41,902,902.1 : DEBUG   config_parser.init_config_paramters - Ser BGLF Db : [bglf.db]
2019-04-14 21:21:41,902,902.2 : DEBUG   config_parser.init_config_paramters - Set config dir : [config/]
2019-04-14 21:21:41,902,902.2 : DEBUG   config_parser.init_config_paramters - Set data dir : [data/analysis/]
2019-04-14 21:21:41,902,902.3 : DEBUG   config_parser.init_config_paramters - Set Shodan API Key : [8iVIFbpZ5LMtlsZZVuIgeSZSOl1Wbz6t&query]
2019-04-14 21:21:41,902,902.3 : DEBUG   config_parser.init_config_paramters - Set Censys project ID : [censys-io-228820]
2019-04-14 21:21:41,902,902.4 : DEBUG   config_parser.init_config_paramters - Set Censys JSON key file : [config/censys-io-google_services_account.json]
2019-04-14 21:21:41,902,902.4 : DEBUG   config_parser.init_config_paramters - Set OSINT search strings : [['antMiner', 'bitmain-inc', '/Satoshi', 'stratum+tcp', 'port:3333']]
2019-04-14 21:21:41,902,902.6 : DEBUG   config_parser.enable_verbose - Enable verbose
2019-04-14 21:21:41,902,902.6 : DEBUG   BGLF.bglf - Logfile : [log/bgfl_crawl_testnet.log]
2019-04-14 21:21:41,902,902.7 : DEBUG   config_parser.set_network_type - Set Bitcoin network type to [testnet]
2019-04-14 21:21:41,902,902.8 : DEBUG   config_parser.set_store_data - Set store data to [True]
Crawl the Bitcoin network to collect ip address of bitcions P2P network nodes.
2019-04-14 21:21:41,903,903.2 : DEBUG   bglf_functions.init_bitcion_network_settings - Initialize the settings for accessing the Bitcoin Network
2019-04-14 21:21:41,903,903.3 : DEBUG   bglf_functions.init_bitcion_network_settings -  Bitcoin network type = testnet
2019-04-14 21:21:41,903,903.4 : DEBUG   bglf_functions.init_bitcion_network_settings -  Host ip = 145.20.145.2
2019-04-14 21:21:41,903,903.4 : DEBUG   bglf_functions.init_bitcion_network_settings -  Host rpc port = 18332
2019-04-14 21:21:41,903,903.5 : DEBUG   bglf_functions.init_bitcion_network_settings -  Host p2p port = 18333
2019-04-14 21:21:41,903,903.6 : DEBUG   bglf_functions.init_bitcion_network_settings -  Magic number = b'0b110907'
2019-04-14 21:21:41,968,968.5 : DEBUG   elastic.connect_elasticsearch - Elasticsearch Connected to 145.20.145.2/9200
2019-04-14 21:21:42,100,100.5 : DEBUG   geoip.read_city_db - Read GeoIP city database
2019-04-14 21:21:42,106,106.8 : DEBUG   geoip.read_country_db - Read GeoIP country database
2019-04-14 21:21:42,112,112.5 : DEBUG   geoip.read_asn_db - Read GeoIP ASN database
2019-04-14 21:21:42,113,113.9 : DEBUG   bglf_db.connect2db - Create and connect to SQlite Database : config/bglf.db
2019-04-14 21:21:42,114,114.4 : DEBUG   bglf_run_db.create_bglf_run_table - Create Table for BGLF index.
2019-04-14 21:21:42,114,115.0 : DEBUG   bitcoind_rpc.create_url - Connect URL : [http://testuser:test123@145.20.145.2:18332]
2019-04-14 21:21:42,115,115.1 : DEBUG   shodan_scrapper.set_connection_params - Base URL ; [https://api.shodan.io/shodan/host/search?key=8iVIFbpZ5LMtlsZZVuIgeSZSOl1Wbz6t&query]
2019-04-14 21:21:42,115,115.3 : DEBUG   bitnodes_scrapper.set_connection_params - Base URL ; [https://bitnodes.earn.com/api/]
2019-04-14 21:21:42,115,115.4 : DEBUG   censys_scrapper.init_credentials - Connect to Google Cloud Platform and Authenticate....
Start collecting the ip addresses of Bitcoin nodes.
 Network type : testnet

2019-04-14 21:21:42,117,117.8 : DEBUG   bglf_db.get_latest_used_run_id - Latest used run_id = 13
2019-04-14 21:21:42,117,118.0 : DEBUG   bglf_functions.StartAddressCollectionOfBitcoinNodesThreaded - Increase latest run ID with 1 for this run (run id = [14]
2019-04-14 21:21:42,118,118.1 : DEBUG   bglf_run_db.insert_run_id - Insert to table [BGLF] : run_id = [14] - network_type = [testnet]
2019-04-14 21:21:42,119,119.4 : DEBUG   crawl_btc.get_peers - Connecting to ('145.20.145.2', 18333)
2019-04-14 21:21:42,235,236.0 : DEBUG   crawl_btc.get_peers - Handshake message : [[{'magic_number': b'\x0b\x11\t\x07', 'command': 'version', 'length': 102, 'checksum': b'\x11d^\xcc', 'version': 70015, 'services': 1036, 'timestamp': 1555269702, 'to_addr': {'timestamp': None, 'services': 0, 'ipv4': '145.20.142.175', 'ipv6': '', 'onion': '', 'port': 57201}, 'from_addr': {'timestamp': None, 'services': 1036, 'ipv4': '', 'ipv6': '::', 'onion': '', 'port': 0}, 'nonce': 16580641769907636880, 'user_agent': b'/Satoshi:0.17.0/', 'height': 1489071, 'relay': True}, {'magic_number': b'\x0b\x11\t\x07', 'command': 'verack', 'length': 0, 'checksum': b']\xf6\xe0\xe2'}]]
2019-04-14 21:21:53,261,261.5 : DEBUG   crawl_btc.get_peers - ('145.20.145.2', 18333) Expected 1037, got 1036 for services
2019-04-14 21:21:53,261,261.7 : DEBUG   crawl_btc.enumerate_peers - Peer address 54.236.58.221:30000 service = 1069
2019-04-14 21:21:53,261,262.0 : DEBUG   crawl_btc.enumerate_peers - Peer address 51.143.97.218:18333 service = 1037
2019-04-14 21:21:53,262,262.2 : DEBUG   crawl_btc.enumerate_peers - Peer address 54.241.159.189:18333 service = 1037
2019-04-14 21:21:53,262,262.5 : DEBUG   crawl_btc.enumerate_peers - Peer address 185.11.238.218:18333 service = 1037
2019-04-14 21:21:53,262,262.9 : DEBUG   crawl_btc.get_peers - ('145.20.145.2', 18333) Peers found : 4
2019-04-14 21:21:53,265,265.4 : DEBUG   crawl_btc.get_peers - Connecting to ('54.236.58.221', 30000)
2019-04-14 21:21:53,265,265.8 : DEBUG   crawl_btc.get_peers - Connecting to ('51.143.97.218', 18333)
2019-04-14 21:21:53,266,266.2 : DEBUG   crawl_btc.get_peers - Connecting to ('54.241.159.189', 18333)
2019-04-14 21:21:53,266,266.5 : DEBUG   crawl_btc.get_peers - Connecting to ('185.11.238.218', 18333)
2019-04-14 21:21:53,316,316.8 : DEBUG   crawl_btc.get_peers - Connection Error: ('185.11.238.218', 18333): [Errno 61] Connection refused
2019-04-14 21:21:53,317,317.3 : DEBUG   bglf_functions.workerCrawl - No new peers found for node 4 in iteration 1. PeerSet has 0 unique nodes
2019-04-14 21:21:53,317,317.5 : DEBUG   bglf_functions.workerCrawl - Number of unreachable node 1 >= 1 of 1 iterations. Save node info with unreachable info.
```

```
2019-04-14 21:21:53,455,455.5 : DEBUG    geoip.resolve_geoip - Geo for (185.11.238.218) : [{'city_name': 'Porto Cristo', '
    country_iso_code': 'ES', 'country_name': 'Spain', 'location': {'lat': '39.5447', 'lon': '3.338'}, 'latitude': 39.5447, '
    longitude': 3.338, 'timezone': 'Europe/Madrid', 'asn': 'AS57910', 'org': 'Soluciones Corporativas IP, SL', 'rdns':
    '185.11.238.218.user.conectabalear.com'}]
2019-04-14 21:21:53,456,456.0 : DEBUG    bglf_functions.insert_records_in_es - Elastic. Insert to index [btc_crawl_node_info_2] :
     {'user_agent': 'Unknown', 'version': 'Unknown', 'relay': 'Unknown', 'services': 0, 'ip': '185.11.238.218', 'address_type
    ': 'IPv4', 'port': 18333, 'network_type': 'testnet', 'date': '2019-04-14T21:21:53.317205', 'total_of_peers': 0, '
    connection_rtt_time': 0, 'status': 'Unreachable', 'network_error': 'Connection refused', 'run_id': 14, 'geoip': {'
    city_name': 'Porto Cristo', 'country_iso_code': 'ES', 'country_name': 'Spain', 'location': {'lat': '39.5447', 'lon':
    '3.338'}, 'latitude': 39.5447, 'longitude': 3.338, 'timezone': 'Europe/Madrid', 'asn': 'AS57910', 'org': 'Soluciones
    Corporativas IP, SL', 'rdns': '185.11.238.218.user.conectabalear.com'}}
2019-04-14 21:21:53,473,473.7 : DEBUG    elastic.store_record - Storing object : {'_index': 'btc_crawl_node_info_2', '_type': '
    node_info', '_id': 'dl9MHWoBoEYuzZjdfgww', '_version': 1, 'result': 'created', '_shards': {'total': 1, 'successful': 1, '
    failed': 0}, '_seq_no': 1417006, '_primary_term': 8}
2019-04-14 21:21:53,473,473.9 : DEBUG    bglf_functions.workerCrawl - Node 4 has no unique nodes. Nodeset has 4 unique nodes
2019-04-14 21:21:53,600,600.1 : DEBUG    crawl_btc.get_peers - Handshake message : [[{'magic_number': b'\x0b\x11\t\x07', 'command
    ': 'version', 'length': 121, 'checksum': b'\xf7\xeag\xe3', 'version': 70015, 'services': 1037, 'timestamp': 1555269713, '
    to_addr': {'timestamp': None, 'services': 0, 'ipv4': '86.92.252.236', 'ipv6': '', 'onion': '', 'port': 57207}, 'from_addr
    ': {'timestamp': None, 'services': 1037, 'ipv4': '', 'ipv6': '::', 'onion': '', 'port': 0}, 'nonce': 14089995825170680690,
     'user_agent': b'/Satoshi:0.17.0.1(bitcoin-testnet)/', 'height': 1489071, 'relay': True}, {'magic_number': b'\x0b\x11\t\
    x07', 'command': 'verack', 'length': 0, 'checksum': b']\xf6\xe0\xe2'}]]
-------- end of snippet ------
```

Code Listing D.10: BGLF, debug logging example.

D.4.1 Shodan example of a search

```
{
matches: [
    {
        hash: -341191863,
        ip: 1318682682,
        org: "LLC Company Interlan Communications",
        isp: "LLC Company Interlan Communications",
        transport: "tcp",
        data: "HTTP/1.1 401 Unauthorized WWW-Authenticate: Digest realm="
            antMiner Configuration", nonce="bdbfc7e6339b4de97abe4964c94400ef"
            , qop="auth" Content-Type: text/html Content-Length: 351 Date:
            Tue, 06 Jan 1970 13:33:11 GMT Server: lighttpd/1.4.32 ",
        asn: "AS28738",
        port: 80,
        hostnames: [
            "58-128-153-78.interlan.ru"
        ],
        location: {
            city: "Moscow",
            region_code: "48",
            area_code: null,
            longitude: 37.61500000000001,
            country_code3: "RUS",
            latitude: 55.75200000000001,
            postal_code: "115432",
            dma_code: null,
            country_code: "RU",
            country_name: "Russian Federation"
```

```
33          },
34          timestamp: "2019-04-24T03:45:55.251115",
35          domains: [
36              "interlan.ru"
37          ],
38          http: {
39              robots_hash: null,
40              redirects: [ ],
41              securitytxt: null,
42              title: "401 - Unauthorized",
43              sitemap_hash: null,
44              robots: null,
45              favicon: null,
46              host: "78.153.128.58",
47              html: "<?xml version="1.0" encoding="iso-8859-1"?> <!DOCTYPE html
                    PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
                    w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html xmlns="
                    http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"> <head>
                    <title>401 - Unauthorized</title> </head> <body> <h1>401 -
                    Unauthorized</h1> </body> </html> ",
48              location: "/",
49              components: { },
50              securitytxt_hash: null,
51              server: "lighttpd/1.4.32",
52              sitemap: null,
53              html_hash: -8589007
54          },
55          os: null,
56          _shodan: {
57              crawler: "f7946cbe2dc20c40fcbcb81ad90aa01731b690ab",
58              ptr: true,
59              id: "f01c2e09-f043-41cd-97cc-8cdb1be4b46f",
60              module: "http",
61              options: { }
62              },
63              ip_str: "78.153.128.58"
64          },
65      ...
66      ],
67      total: 1683
68  }
```


Code Listing D.11: Shodan example of a search.

E. OSINT

E.1 Shodan

The results below are collected in the period between 3 and 10 April 2019.

Results for 'antMiner'

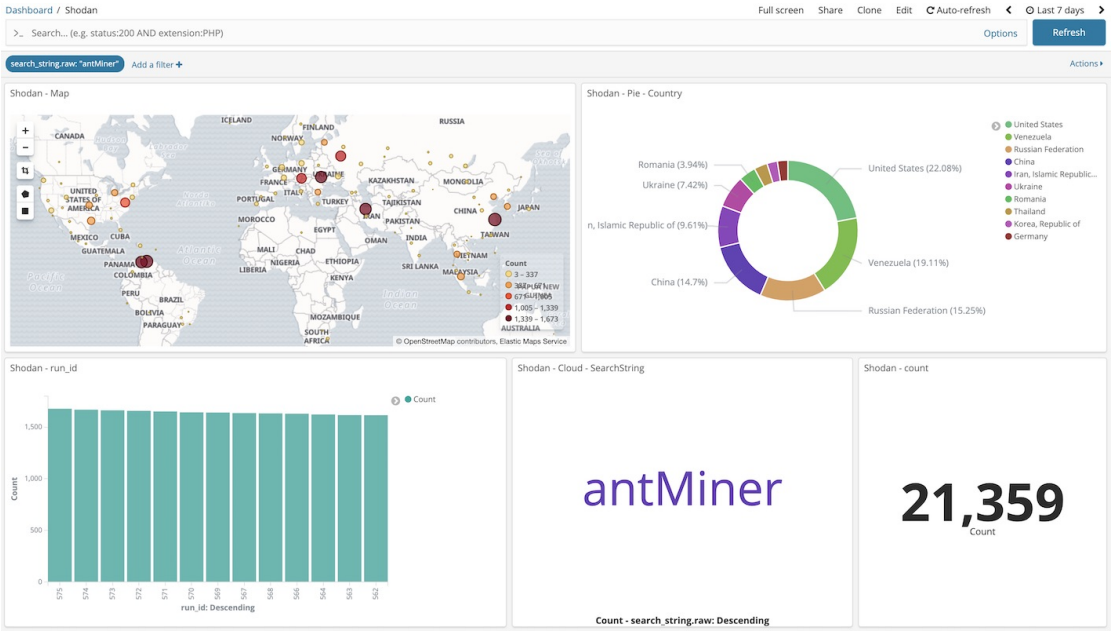


Figure E.1: Shodan results for 'antMiner'.

Results for 'bitmain-inc'

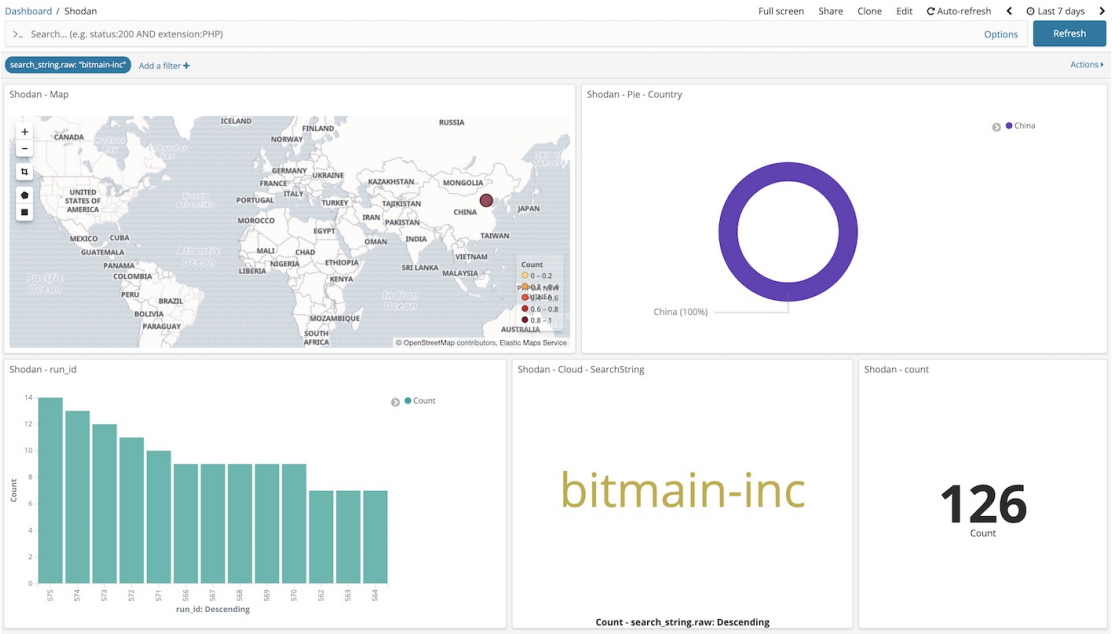


Figure E.2: Shodan results for 'bitmain-inc'.

Results for 'port:3333'

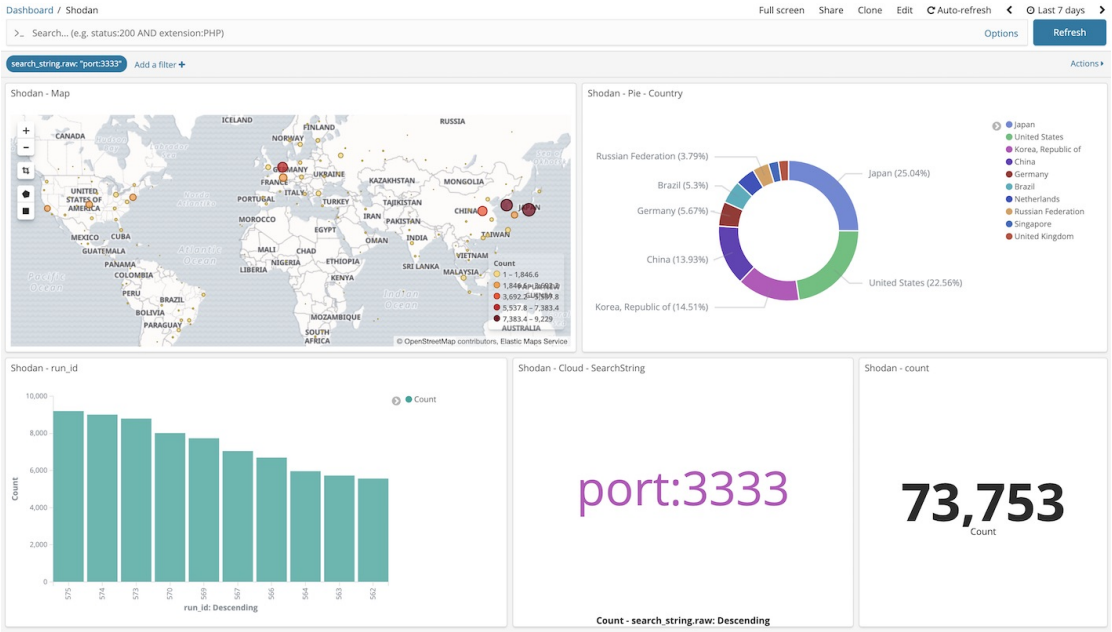


Figure E.3: Shodan results for 'port:3333'.

Results for 'stratum+tcp'

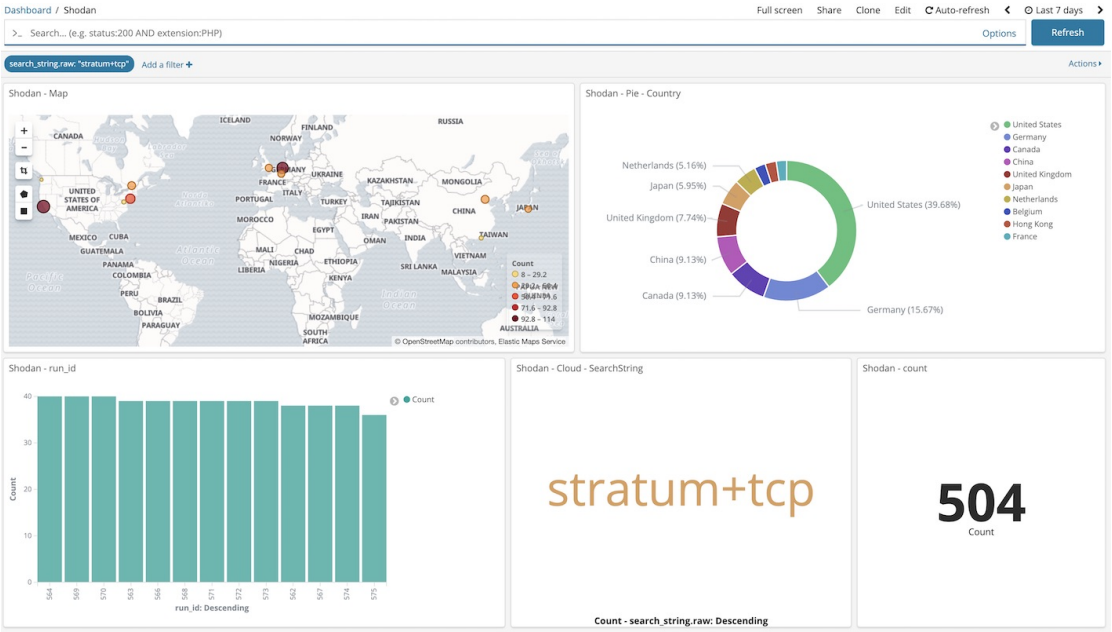


Figure E.4: Shodan results for 'stratum+tcp'.

E.2 Censys

```
class CensysScrapper(RestRequests):
    """
        Class for collecting results from Censys data on Google BigQuery
            Platform
    """

    def __init__(self, config, logger):
        """
            Init the class
        :param config:
        :param logger:
        """
        self.config = config
        self.logger = logger
        self.gbq_client = None
        super().__init__(config=config,
                         logger=logger)
        self.init_credentials()

    def init_credentials(self):
        """
            Setup the credentials needed for access to the Google Cloud
                Platform (BigQuery)
```

```
90          :return:
91          """
92          self.logger.debug("Connect to Google Cloud Platform and Authenticate
                ....")
93          #  Make use of a service-account. Granted access by Censys.io
94          self.gbq_client = bigquery.Client.from_service_account_json(self.
                config.censys_google_json_key_file)
95
96      def query(self, query):
97          """
98              Query the Censys dataset
99          :return:
100         """
101         result = self.gbq_client.query(query)
102         return result
103
104     def search(self, search):
105         """
106             Search for info in the Censys.io Datatset on Google Cloud
                    Platform
107
108         :return:
109         """
110         self.logger.debug(
111             "Search in Censys.io dataset (port 80 & 8080 header authenticate)
                    for info with the following search string: [{}]".format(
112                 search))
113         query = """
114                 SELECT
115                   ip,
116                   ports,
117                   autonomous_system.asn,
118                   autonomous_system.organization,
119                   location.country,
120                   location.country_code,
121                   location.latitude,
122                   location.longitude,
123                   location.city,
124                   location.timezone,
125                   p80.http.get.headers.www_authenticate as
                          port_80_www_authenticate,
126                   p8080.http.get.headers.www_authenticate as
                          port_8080_www_authenticate,
127                   updated_at
128                 FROM
129                   `censys-io.ipv4_public.current`
130                 WHERE
```

```
                    p80.http.get.headers.www_authenticate LIKE '%{item}%'OR
                        p8080.http.get.headers.www_authenticate LIKE '%{item
                        }%';
            """.format(item=search)
        result = self.query(query=query)
        return result, search
```

Code Listing E.1: Python class for querying Google Bitquery

E.3 Bitnodes

```
{
    timestamp: 1556314621,
    total_nodes: 9479,
    latest_height: 573377,
    nodes: {
        [2001:41d0:2:334e::]:8333: [
        70015,
        "/Satoshi:0.17.1/",
        1554242159,
        1037,
        573377,
        "2001:41d0:2:334e::",
        null,
        "FR",
        46,
        2,
        "Europe/Paris",
        "AS16276",
        "OVH SAS"
    ],
    ...
    }
}
```

Code Listing E.2: Example JSON response for Bitnodes API

F. Mining Pools

F.1 Mining Pools Information

No	*Pool*	*Region*	*Poolserver*	*Color*
1	BTC.com	China Beijing	stratum+tcp://cn.ss.btc.com:443	RED
1	BTC.com	Europa	stratum+tcp://eu.ss.btc.com:443	RED
1	BTC.com	USA	stratum+tcp://us.ss.btc.com:443	RED
2	AntPool	China	stratum+tcp://stratum.antpool.com:3333	MAROON
3	F2Pool	China	stratum+tcp://stratum.f2pool.com:3333	YELLOW
4	Poolin	Unknown	stratum+tcp://btc.ss.poolin.com:443	OLIVE
4	Poolin	China Beijing	stratum+tcp://btc-bj.ss.poolin.com:443	OLIVE
4	Poolin	USA	stratum+tcp://btc-va.ss.poolin.com:443	OLIVE
5	SlushPool	General	stratum+tcp://stratum.slushpool.com:3333	GREEN
5	SlushPool	Europe	stratum+tcp://eu.stratum.slushpool.com:3333	GREEN
5	SlushPool	USA East Coast	stratum+tcp://us-east.stratum.slushpool.com:3333	GREEN
5	SlushPool	Canada	stratum+tcp://ca.stratum.slushpool.com:3333	GREEN
5	SlushPool	Singapore South Asia	stratum+tcp://sg.stratum.slushpool.com:3333	GREEN
5	SlushPool	Japan Pacific	stratum+tcp://jp.stratum.slushpool.com:3333	GREEN
5	SlushPool	China	stratum+tcp://cn.stratum.slushpool.com:3333	GREEN
5	SlushPool	China	stratum+tcp://cn02.stratum.slushpool.com:3333	GREEN
5	SlushPool	China	stratum+tcp://cn03.stratum.slushpool.com:3333	GREEN
7	ViaBTC	Unknown	stratum+tcp://btc.viabtc.com:3333	TEAL
7	ViaBTC	Unknown	stratum+tcp://bitcoin.viabtc.com:3333	TEAL
10	dpool	North China	Stratum+tcp://btc-bj.s.dpool.top:443	BLUE
10	dpool	South China	Stratum+tcp://btc-hn.s.dpool.top:443	BLUE
10	dpool	North American	Stratum+tcp://btc-va.s.dpool.top:443	BLUE
10	dpool	European	Stratum+tcp://btc-eu.s.dpool.top:443	BLUE
12	Huobi.pool	Unknown	Stratum+tcp://stratum.poolhb.com:8888	NAVY
12	Huobi.pool	Unknown	Stratum+tcp://bak.poolhb.com:8888	NAVY
12	Huobi.pool	Unknown	Stratum+tcp://stratum.btc.top:8888	NAVY
13	Bitcoin.com	North America	Stratum+tcp://stratum.bcc.pool.bitcoin.com:3333	PURPLE
15	Bixin	Unknown	Stratum+tcp://stratum.bixin.com:3333	LIME

Table F.1: Mining Pools gateway overview.

No	Pool	Poolserver	Color
1	BTC.com	116.211.174.223	RED
1	BTC.com	119.23.117.231	RED
1	BTC.com	120.78.27.201	RED
1	BTC.com	47.91.65.135	RED
1	BTC.com	47.254.143.43	RED
1	BTC.com	47.89.133.88	RED
1	BTC.com	47.252.50.130	RED
2	AntPool	35.177.197.177	MAROON
3	F2Pool	203.107.32.162	YELLOW
4	Poolin	35.186.233.235	OLIVE
4	Poolin	47.95.70.99	OLIVE
4	Poolin	35.186.233.235	OLIVE
2	AntPool	35.177.197.177	MAROON
3	F2Pool	203.107.32.162	YELLOW
4	Poolin	35.186.233.235	OLIVE
4	Poolin	47.95.70.99	OLIVE
4	Poolin	35.186.233.235	OLIVE
5	SlushPool	34.240.133.179	GREEN
5	SlushPool	34.245.14.121	GREEN
5	SlushPool	35.182.18.215	GREEN
5	SlushPool	35.182.229.99	GREEN
5	SlushPool	37.129.17.154	GREEN
5	SlushPool	37.139.19.57	GREEN
5	SlushPool	52.19.206.69	GREEN
5	SlushPool	52.31.186.94	GREEN
5	SlushPool	52.90.128.18	GREEN
5	SlushPool	54.204.233.72	GREEN
5	SlushPool	54.205.207.15	GREEN
5	SlushPool	82.196.10.199	GREEN
5	SlushPool	159.65.36.78	GREEN
5	SlushPool	159.89.190.157	GREEN
5	SlushPool	167.99.182.160	GREEN
5	SlushPool	167.99.182.205	GREEN
5	SlushPool	206.189.181.30	GREEN
5	SlushPool	206.189.184.132	GREEN
5	SlushPool	206.189.185.78	GREEN
5	SlushPool	34.207.252.164	GREEN
5	SlushPool	34.230.31.195	GREEN
5	SlushPool	13.231.35.92	GREEN
5	SlushPool	39.104.95.211	GREEN
5	SlushPool	39.104.114.74	GREEN
5	SlushPool	139.198.16.112	GREEN
5	SlushPool	139.198.16.189	GREEN
5	SlushPool	139.198.17.28	GREEN
5	SlushPool	178.128.101.5	GREEN
5	SlushPool	54.254.223.213	GREEN
7	ViaBTC	47.91.65.162	TEAL
7	ViaBTC	47.91.67.143	TEAL
10	dpool	47.95.127.114	BLUE
10	dpool	120.78.31.61	BLUE
10	dpool	47.252.50.24	BLUE
10	dpool	47.252.50.24	BLUE
12	Huobi.pool	39.97.6.46	NAVY
12	Huobi.pool	39.97.6.45	NAVY
12	Huobi.pool	39.197.191.38	NAVY
12	Huobi.pool	39.105.227.145	NAVY
13	Bitcoin.com	47.93.164.114	PURPLE
13	Bitcoin.com	60.205.132.187	PURPLE
13	Bitcoin.com	47.93.164.112	PURPLE
13	Bitcoin.com	47.93.164.133	PURPLE
15	Bixin	60.205.146.217	LIME
15	Bixin	60.205.146.222	LIME
15	Bixin	60.205.146.151	LIME

Table F.2: Mining Pools the IP address of the gateways.

F.2 Maps

F.2.1 BTC.com

Figure F.1: BTC.com pool server/gateway locations (May 7 2019).

F.2.2 AntPool

Figure F.2: AntPool pool server/gateway locations (May 7 2019).

F.2.3 F2Pool

Figure F.3: F2Pool pool server/gateway locations (May 7 2019).

F.2.4 Poolin

Figure F.4: Poolin pool server/gateway locations (May 7 2019).

F.2.5 SlushPool

Figure F.5: SlushPool pool server/gateway locations (May 7 2019).

Figure F.6: SlushPool pool server/gateway locations based on IP addresses (May 7 2019).

F.2.6 ViaBTC

Figure F.7: ViaBTC pool server/gateway locations (May 7 2019).

F.2.7 DPool

Figure F.8: DPool pool server/gateway locations (May 7 2019).

F.2.8 Huobi.pool

Figure F.9: Huobi.pool pool server/gateway locations (May 7 2019).

Figure F.10: Huobi.pool pool server/gateway locations based on IP addresses (May 7 2019).

nin

F.2.9 Bitcoin.com

Figure F.11: Bitcoin.com pool server/gateway locations (May 7 2019).

F.2.10 Bixin

Figure F.12: Bixin pool server/gateway locations (May 7 2019).

Bibliography

[1] Dong He, Karl Friedrich Habermeier, Ross B Leckow, Vikram Haksar, Yasmin Almeida, Mikari Kashima, Nadim Kyriakos-Saad, Hiroko Oura, Tahsin Saadi Sedik, and Natalia Stetsenko. Virtual currencies and beyond: initial considerations. Report, International Monetary Fund, 2016.

[2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *https://bitcoin.org/bitcoin.pdf*, 2008.

[3] Harald Vranken. Sustainability of bitcoin and blockchains. *Current Opinion in Environmental Sustainability*, 28:1–9, 2017.

[4] Ruben Recabarren and Bogdan Carbunar. Hardening stratum, the bitcoin pool mining protocol. *Proceedings on Privacy Enhancing Technologies*, 2017(3):57–74, 2017.

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

[6] Sergi Delgado-Segura, Cristina Pérez-Solà, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joan Borrell. Cryptocurrency networks: A new p2p paradigm. *Mobile Information Systems*, 2018:1–16, 2018.

[7] Varun Deshpande, Hakim Badis, and Laurent George. Btcmap: Mapping bitcoin peer-to-peer network topology. In *BTCmap: Mapping Bitcoin Peer-to-Peer Network Topology*, pages 1–6, 09 2018.

[8] Joan Antoni Donet Donet, Cristina Pérez-Solà, and Jordi Herrera-Joancomartí. The bitcoin p2p network. *Financial Cryptography and Data Security*, 8438, 2014.

[9] Alex Biryukov and Ivan Pustogarov. Bitcoin over tor isn't a good idea. In *2015 IEEE Symposium on Security and Privacy*, pages 122–134. IEEE, 2015.

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

[11] 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, Washington, D.C., 2015. USENIX Association.

[12] Giulia Fanti and Pramod Viswanath. Deanonymization in the bitcoin p2p network. In *Advances in Neural Information Processing Systems*, pages 1364–1373, 2014.

[13] Mirco; Werner Martin Feld, Sebastian; Schönfeld. Analyzing the deployment of bitcoin's p2p network under an as-level perspective. In *International Workshop on Secure Peer-to-Peer Intelligent Networks & Systems (SPINS-2014)*, page 1121 – 1126. Elsevier, 2014.

[14] Indra Deep Mastan and Souradyuti Paul. A new approach to deanonymization of unreachable bitcoin nodes. *Cryptology ePrint Archive*, 2018.

[15] Merve Can Kus Khalilov and Albert Levi. A survey on anonymity and privacy in bitcoin-like digital cash systems. *IEEE Communications Surveys & Tutorials*, 2018.

[16] Anna c Dupont, Jules; Squicciarini. Toward de-anonymizing bitcoin by mapping users location. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 139–141. ACM, 2015.

[17] Péter L Juhász, József Stéger, Dániel Kondor, and Gábor Vattay. A bayesian approach to identify bitcoin users. *arXiv preprint arXiv:1612.06747*, 2016.

[18] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. Bitiodine: Extracting intelligence from the bitcoin network. In *International Conference on Financial Cryptography and Data Security*, pages 457–468. Springer, 2014.

[19] Arif Sari and Seyfullah Kilic. Exploiting cryptocurrency miners with oisnt techniques. *Transactions on Networks and Communications*, 5(6):62, 2017.

[20] Garrick Hileman and Michel Rauchs. 2017 global cryptocurrency benchmarking study. *SSRN Electronic Journal*, 01 2017.

[21] Blandin Apolline Klein Kristina Pieters Gina Recanatini Martino Zhang Bryan Rauchs, Michel. 2nd global cryptoasset benchmarking study. https://www.jbs.cam.ac.uk/fileadmin/user_upload/research/centres/alternative-finance/downloads/2018-12-ccaf-2nd-global-cryptoasset-benchmarking.pdf, 12 2018.

[22] K.J. O'Dwyer and D. Malone. Bitcoin mining and its energy footprint. In *Irish Signals & Systems Conference and China–Ireland International Conference on Information and Communications Technologies*, pages 280–285, 2014.

[23] Muhammad Anas Imtiaz, David Starobinski, Ari Trachtenberg, and Nabeel Younis. Churn in the bitcoin network: Characterization and impact. In *IEEE International Conference on Blockchain and Cryptocurrency*, May 2019.

[24] Jan-Jelle Kester. Comparing the accuracy of ipv4 and ipv6 geolocation databases. *Methodology*, 10(11):12–17, 2016.

[25] S. Zu, X. Luo, S. Liu, Y. Liu, and F. Liu. City-level ip geolocation algorithm based on pop network topology. *IEEE Access*, 6:64867–64875, 2018.

[26] Ovidiu Dan, Vaibhav Parikh, and Brian D Davison. Ip geolocation through reverse dns. *arXiv preprint arXiv:1811.04288*, 2018.

[27] Wei Xu, Yaodong Tao, and Xin Guan. Experimental comparison of free ip geolocation services. In *International Conference on Security with Intelligent Computing and Big-data Services*, pages 198–208. Springer, 2018.

[28] Zhihao Wang, Hong Li, Qiang Li, Wei Li, Hongsong Zhu, and Limin Sun. Towards ip geolocation with intermediate routers based on topology discovery. *Cybersecurity*, 2(1):13, 2019.

[29] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. Decentralization in bitcoin and ethereum networks. *arXiv preprint arXiv:1801.03998*, 2018.

[30] Charles J Krebs et al. Ecological methodology. Technical report, Harper & Row New York, 1989.

[31] Annegret Grimm, Bernd Gruber, and Klaus Henle. Reliability of different mark-recapture methods for population size estimation tested against reference population sizes constructed from field data. *PLoS One*, 9(6):e98840, 2014.

[32] Nicola Accettura, Giovanni Neglia, and Luigi Alfredo Grieco. The capture-recapture approach for population estimation in computer networks. *Computer Networks*, 89:107–122, 2015.

[33] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by internet-wide scanning. In *ACM Conference on Computer and Communications Security*, 2015.

Online Sources

[34] Cryptocurrency market cap. https://coinmarketcap.com. Accessed June 2019.

[35] Hash functions. https://csrc.nist.gov/Projects/Hash-Functions. Accessed September 2018.

[36] Bitcoin block reward. https://www.bitcoinblockhalf.com/. Accessed October 2018.

[37] Bitcoin core. https://bitcoin.org/en/bitcoin-core/. Accessed October 2018.

[38] Bitcoin core source code. https://github.com/bitcoin/bitcoin. Accessed October 2018.

[39] Size of bitcoin blockchain. https://www.blockchain.com/en/charts/blocks-size. Accessed May 2019.

[40] Bitcoin network protocl. https://en.bitcoin.it/wiki/Network. Accessed September 2018.

[41] Stratum mining protocol. https://en.bitcoin.it/wiki/Stratum mining protocol. Accessed Augustus 2018.

[42] Stratum mining protocol. https://slushpool.com/help/stratum-protocol/#stratum. Accessed Augustus 2018.

[43] Bitcoin reference client protocol documentation. https://en.bitcoin.it/wiki/Protocol_documentation. Accessed April 2019.

[44] Bitcoin core address management module addrman.h (github). https://github.com/bitcoin/bitcoin/blob/master/src/addrman.h. Accessed February 2019.

[45] Fast relay network. http://bitcoinrelaynetwork.org. Accessed September 2018.

[46] Stan Higgins. Bitcoin's 'nervous system' gets an upgrade with fibre network. https://www.coindesk.com/bitcoins-nervous-system-getting-upgrade. Accessed April 2019.

[47] Fibre network. http://bitcoinfibre.org. Accessed September 2018.

[48] M. Corallo. Fast internet bitcoin relay engine. https://github.com/bitcoinfibre/bitcoinfibre. Accessed September 2018.

[49] Bitcoin improvement proposal (bip)-0152. https://github.com/bitcoin/bips/blob/ master/bip-0152.mediawiki. Accessed May 2019.

[50] Falcon relay network. https://www.falcon-net.org/. Accessed September 2018.

[51] Bitnodes source code. https://github.com/ayeowch/bitnodes, 2018. Accessed December, 2018.

[52] How much electricity (kwh) does an antminer s9 consume in a day? https://www.quora.com/How-much-electricity-kWh-does-an-Antminer-S9-consume-in-a-day. Accessed April 2019.

[53] Asicminervalue.com. https://www.asicminervalue.com/. Accessed May 2019.

[54] Cost of electricity by source. https://en.wikipedia.org/wiki/Cost_of_electricity_by_source. Accessed January 2019.

[55] Blockchain. https://www.blockchain.com/en/. Accessed May 2019.

[56] Docker. https://www.docker.com. Accessed March 2019.

[57] What is a container? https://www.docker.com/resources/what-container. Accessed April 2019.

[58] Rightscale-2019-state-of-the-cloud-report. https://resources.flexera.com/web/pdf/RightScale-2019-State-of-the-Cloud-Report-from-Flexera.pdf. Accessed March 2019.

[59] A bitcoin-core docker image. https://github.com/ruimarinho/docker-bitcoin-core. Accessed October 2018.

[60] Gevent tutorial. http://sdiehl.github.io/gevent-tutorial/. Accessed December 2018.

[61] Asynchronous programming in python. https://dbader.org/blog/understanding-asynchronous-programming-in-python. Accessed December 2018.

[62] Elasticsearch. https://www.elastic.co/products/elasticsearch. Accessed February 2019.

[63] Elastic Inc. Elasticsearch docker image. https://hub.docker.com/_/elasticsearch. Accessed September 2018.

[64] Elasticsearch-head. https://mobz.github.io/elasticsearch-head/. Accessed February 2019.

[65] Elastichq. http://www.elastichq.org/. Accessed February 2019.

[66] Kibana. https://www.elastic.co/products/kibana. Accessed February 2019.

[67] Elastic Inc. Kibana docker image. https://hub.docker.com/_/kibana. Accessed September 2018.

[68] Elasticsearch, logstash and kibana (elk) stack. https://www.elastic.co/elk-stack. Accessed June 2019.

[69] Maxmind. https://www.maxmind.com/en/home. Accessed April 2019.

[70] Geoip2 city accuracy. https://www.quora.com/How-accurate-are-GeoIP-methods. Accessed April 2019.

[71] Logging python module. https://docs.python.org/3/library/logging.html. Accessed April 2019.

[72] Click python module. https://click.palletsprojects.com/en/7.x/. Accessed April 2019.

[73] Wireshark network protocol analyser. https://www.wireshark.org/. Accessed April 2019.

[74] Global bitcoin nodes distribution. https://bitnodes.earn.com/. Accessed September 2018.

[75] Bitcoin core protocol module protocol.h (github). https://github.com/bitcoin/bitcoin/blob/master/src/protocol.h. Accessed April 2019.

[76] Mark and recapture. https://en.wikipedia.org/wiki/Mark_and_recapture. Accessed April 2019.

[77] Mark and recapture statistical software. http://www.phidot.org/software/. Accessed June 2019.

[78] Shodan search engine. https://shodan.io. Accessed March 2019.

[79] Shodan and censys: the ominous guides through the internet of things. https://www.kaspersky.com/blog/shodan-censys/11430. Accessed March 2019.

[80] Censys search engine. https://censys.io. Accessed March 2019.

[81] List of tcp and udp port numbers). https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers. Accessed May 2019.

[82] Hypertext transfer protocol (http). https://en.wikipedia.org/wiki/ Hypertext_Transfer_Protocol#Request_methods. Accessed February 2019.

[83] An introductory guide to using google bigquery to query censys. https://support.censys.io/google-bigquery/bigquery-introduction. Accessed February 2019.

[84] Google bigquery. https://cloud.google.com/bigquery/. Accessed February 2019.

[85] Bitnodes api v1.0. https://bitnodes.earn.com/api/. Accessed September 2018.

[86] Btc.com - mining pool server setup. https://help.pool.btc.com/hc/en-us/articles/360020105472-Mining-Address. Accessed April 2019.

[87] Antpool - mining pool server setup. https://www.antpool.com/support.htm. Accessed April 2019.

[88] F2pool - mining pool server setup. https://blog.f2pool.com/en/mining-tutorial-en/btc_en. Accessed April 2019.

[89] Poolin - mining pool server setup. https://help.poolin.com/hc/en-us/articles/360006872672-BTC-Mining-URL. Accessed April 2019.

[90] Slushpool - mining pool server setup. https://slushpool.com/help/bitcoin-setup/. Accessed April 2019.

[91] Viabtc - mining pool server setup. https://support.viabtc.com/hc/en-us/articles/360003420932-How-should-I-configure-the-mining-rig-. Accessed April 2019.

[92] Dpool - mining pool server setup. https://support.dpool.top/hc/zh-cn/articles/360015486632. Accessed April 2019.

[93] Huobi pool - mining pool server setup. https://www.poolhb.com/pow/help/1/137. Accessed April 2019.

[94] Bitcoin.com - mining pool server setup. https://news.bitcoin.com/bitcoin-com-pool-members-now-have-the-option-to-mine-bitcoin-cash/. Accessed April 2019.

[95] Bixin - mining pool server setup. https://bixin.com/post/pool-help/. Accessed April 2019.

[96] Proof of stake (focus on bitcoin). https://en.bitcoin.it/wiki/Proof_of_Stake. Accessed May 2019.

[97] Proof of stake (focus on ethereum). https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ. Accessed May 2019.

[98] Core conference ranking. http://portal.core.edu.au/conf-ranks/1551/. Accessed June 2019.

Sources on the location of Bitcoin mining activity

[99] Bitmain draft ipo application. http://www.hkexnews.hk/APP/SEHK/2018/2018092406/Documents/SEHK201809260017.pdf (INACTIVE!). Accessed December 2018.

[100] Canaan draft ipo application. http://www.hkexnews.hk/APP/SEHK/2018/2018051401/Documents/SEHK201805150005.pdf (INACTIVE!). Accessed December 2018.

[101] Ebang draft ipo application. http://www.hkexnews.hk/app/SEHK/2018/2018122001/Documents/SEHK201812200005.pdf. Accessed May 2019.

[102] Bitmain. https://www.bitmain.com/. Accessed January 2019.

[103] David Hamilton. The top 5 largest mining operations in the world.

[104] Morgen E. Peck. Why the biggest bitcoin mines are in china. https://spectrum.ieee.org/computing/networks/why-the-biggest-bitcoin-mines-are-in-china, Oct 2017.

[105] Inner mongolia bitcoin mine. https://www.techinasia.com/inner-mongolia-bitcoin-mine. Accessed January 2018.

[106] Inside the world of chinese bitcoin mining. https://www.aljazeera.com/indepth/inpictures/world-chinese-bitcoin-mining-180116112117869.html, Jan 2018.

[107] Genesis mining. https://www.genesis-mining.com/. Accessed January 2019.

[108] The top three countries for bitcoin miners. https://www.redherring.com/top-story/the-top-three-countries-for-bitcoin-miners/. Accessed January 2018.

[109] Iceland set to use more energy mining bitcoin than powering homes. https://www.telegraph.co.uk/news/2018/02/12/iceland-set-use-energy-mining-bitcoin-powering-homes/. Accessed January 2018.

[110] Iceland's big bitcoin heist showed how strong the country's crypto economy really is. https://www.redherring.com/opinions/icelands-big-bitcoin-heist-showed-how-strong-the-countrys-crypto-economy-really-is/, 2018.

[111] Bitfury. https://bitfury.com/. Accessed January 2019.

[112] Liz Alderman. Despite bitcoin's dive, a former soviet republic is still betting big on it. https://www.nytimes.com/2019/01/22/business/georgia-bitcoin-blockchain-bitfury.html, Jan 2019.

[113] How the tiny nation of georgia became a bitcoin behemoth. https://www.npr.org/sections/parallels/2018/04/23/597780405/how-the-tiny-nation-of-georgia-became-a-bitcoin-behemoth, April 23, 2018.

[114] Hut 8 mining corp. https://hut8mining.com/operations. Accessed January 2019.

[115] Bloomberg company overview of hut 8 mining corp. https://www.bloomberg.com/research/stocks/private/snapshot.asp?privcapId=546209022. Accessed January 2019.

[116] Hans Hauge. Introducing hut 8 mining. https://seekingalpha.com/article/4200170-introducing-hut-8-mining, Aug 2018.

[117] Dmg. https://dmgblockchain.com/. Accessed May 2019.

[118] Giga watt. https://giga-watt.com/. Accessed January 2019.

[119] Salcido enterprises. https://www.salcidoenterprises.tech/work. Accessed January 2019.

[120] Paul Roberts. This is what happens when bitcoin miners take over your town. https://www.politico.com/magazine/story/2018/03/09/bitcoin-mining-energy-prices-smalltown-feature-217230, Mar 2018.

[121] Olga Kharif. Bitcoin miner consolidation increases risk of more crypto mayhem. https://www.bloomberg.com/news/articles/2018-12-03/bitcoin-miner-consolidation-increases-risk-of-more-crypto-mayhem, Dec 2018.

[122] Paul Roberts. Bitcoin backlash as 'miners' suck up electricity, stress power grids in central washington. https://www.seattletimes.com/business/bitcoin-backlash-as-miners-suck-up-electricity-stress-power-grids-in-central-washington/, May 2018.

[123] Coinmint. https://www.coinmint.io/. Accessed May 2019.

[124] Friday: Layoffs at massena bitcoin miner. https://www.wwnytv.com/story/39776225/friday-layoffs-at-massena-bitcoin-miner, Jan 2019.

[125] Samantha Chang. $700 million bitcoin mining farm coming to upstate new york. https://www.ccn.com/700-million-bitcoin-mining-farm-coming-to-upstate-new-york, Jun 2018.

[126] Patrick McGeehan. Bitcoin miners flock to new york's remote corners, but get chilly reception. https://www.nytimes.com/2018/09/19/nyregion/bitcoin-mining-new-york-electricity.html, Sep 2018.

[127] Gmo. https://mining.gmo.jp/en/. Accessed May 2019.

[128] Wolfie Zhao. Gmo internet's crypto mining facility is up and running. https://www.coindesk.com/gmo-internets-crypto-mining-facility-is-up-and-running, Dec 2017.

[129] Tim Alper. This japanese company mined almost 600 btc "somewhere in scandinavia. https://cryptonews.com/news/this-japanese-company-mined-almost-600-btc-somewhere-in-scan-2611.htm, Sep 2018.

[130] Cryptocurrency mining business: Monthly report for december 2018 and termination of the monthly report. https://ir.gmo.jp/en/pdf/irlibrary/gmo_disclose_info20190108_e.pdf, Jan 2019.

[131] Nick Marinoff. Bitfury expands to norway with $35 million bitcoin data center. https://www.nasdaq.com/article/bitfury-expands-to-norway-with-35-million-bitcoin-data-center-cm939115, Mar 2018.

[132] Stephen Jewkes Christoph Steitz. Cryptocurrency miners seek cheap energy in norway and sweden. https://www.reuters.com/article/us-cryptocurrencies-nordics-energy/cryptocurrency-miners-seek-cheap-energy-in-norway-and-sweden-idUSKBN1HH13L, April 2018.

[133] Canaan. https://canaan.io/datacenter/. Accessed January 2019.

[134] Blockbase. https://blockbasemining.com/mining-farm-sweden/. Accessed May 2019.

[135] Hydro66. https://www.hydro66.com/. Accessed May 2019.

[136] Boden is investing in the blockchain technology. https://bodenbusinesspark.com/en/about/press-media/150352D7700E3C61F1–boden-is-investing-in-the-blockchain-technology, May 2018.

[137] Mark J.Valek Demelza Kelso Hays. Crypto research report april 2019. Web page, April 2019.

[138] Hydro66 provides update on data center colocation strategy. https://www.hydro66.com/news/hydro66-provides-update-on-data-center-colocation-strategy/, April 2019.

[139] Lynne Marek. Chicago's crypto industry takes a hit. https://www.chicagobusiness.com/finance-banking/chicagos-crypto-industry-takes-hit, Apr 2019.

[140] Andy Fox. Sound barrier could help lessen noise problems from major bitcoin facility in vb. https://www.wavy.com/news/local-news/virginia-beach/sound-barrier-could-help-lessen-noise-problems-from-major-bitcoin-facility-in-vb/1394092193, Aug 2018.

[141] Bit patagonia. http://bitpatagonia.com/. Accessed March 2019.

[142] Carlos Terenzi. Patagonia grows as a cryptocurrency mining centre. https://www.coinstaker.com/patagonia-argentina-mining-farm/, Aug 2018.

[143] Pablo Wende. La reconversión de tierra del fuego: la nueva meca para minar bitcoin. https://www.infobae.com/economia/2018/08/20/la-reconversion-de-tierra-del-fuego-la-nueva-meca-para-minar-bitcoin/, Aug 2018.

[144] Leigh Cuen. There's no crypto winter in argentina, where startups ramp up to meet demand. https://www.coindesk.com/crypto-startups-in-argentina-inflation-bitcoin, Feb 2019.

[145] Minery. https://minery.io/en. Accessed May 2019.

[146] Bitriver. https://bitriver.farm/en.html. Accessed May 2019.

[147] Russia is ready for the large scale legal bitcoin mining operation. https://www.globalbankingandfinance.com/russia-is-ready-for-the-large-scale-legal-bitcoin-mining-operation/, May 2018.

[148] Andrew Rossow. Russia prepares for large-scale bitcoin mining operation following minery launch. https://themerkle.com/russia-prepares-for-large-scale-bitcoin-mining-operation-in-wake-of-minery-launch/, May 2018.

[149] Наталья Кузнецова. Энергия для крупнейшего майнинг-центра. https://www.if24.ru/energiya-dlya-majning-tsentra/, Mar 2019.

[150] Cryptouniverse. https://cryptouniverse.io/. Accessed May 2019.

[151] Kevin Helms. Regional government announces opening of large crypto mining farm in russia. https://news.bitcoin.com/largest-crypto-mining-farm-russia/, Aug 2018.

[152] Bhushan Akolkar. Russia's leningrad region unveils the largest crypto mining farm. https://bitcoinist.com/russias-leningrad-region-unveils-the-largest-crypto-mining-farm/, Aug 2018.

[153] Rossana Prezioso. Cryptouniverse, a russian mining farm. https://cryptonomist.ch/en/2018/08/23/russian-mining-farm/, Aug 2018.

[154] В Ленобласти ввели в эксплуатацию первую очередь центра для майнинга. https://ria.ru/20180820/1526874305.html, Aug 2018.

[155] Bitbaza. https://bitbaza.com/. Accessed May 2019.

[156] Russia to open huge crypto-mining farm in siberia. https://russiabusinesstoday.com/technology/russia-to-open-huge-crypto-mining-farm-in-siberia/, Apr 2018.

[157] Allen Scott. Russia's bitcoin mining industry takes root in siberia. https://bitcoinist.com/siberia-russia-largest-mining-farms/, Apr 2018.

[158] В Красноярске планируют создать одну из самых крупных майнинг-ферм в России. https://tass.ru/ekonomika/5119640, Apr 2018.