Measuring Polkadot

The Impact of Tor and a VPN on Polkadot's Performance and Security

Just van Stam

Measuring Polkadot

The Impact of Tor and a VPN on Polkadot's Performance and Security

by

Just van Stam

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on December 5th, 2022 at 11.00h.

Student number:	4473280
Project duration:	March 15th, 2022 – November 28th, 2022

Thesis committee

Chairman: Prof. dr. ir. J. Pouwelse,	TU Delft, EEMCS, Distributed Systems
1^{st} supervisor: Prof. dr. ir. S. Roos,	TU Delft, EEMCS, Distributed Systems
2^{nd} supervisor: Prof. dr. ir. K. Liang,	TU Delft, EEMCS, Cybersecurity

Preface

It is a pleasure to present this thesis to fulfill the requirements for my master of Computer Science. It has been a fantastic opportunity to learn, explore and enjoy the world of Polkadot.

I want to thank Stefanie for the opportunity to work on this project and her guidance throughout this process. Her patience and commitment to helping me improve this thesis have been invaluable.

Secondly, I want to thank my friends and family, who have encouraged me throughout this process.

Finally, I want to thank Marijn for being my rock. I could not have done it without you!

Just van Stam
Delft, November 2022

Abstract

Begun in 2020, Polkadot is one of the largest blockchains in market capitalization and development. However, privacy on the Polkadot network has yet to be one of the key focus points. Especially unlinkability between the user's IP address and Polkadot address is essential. Without this unlinkability, users are vulnerable to targeted ads, manipulation, blackmail, reputational damage, financial loss, physical harm, discrimination, and more. This thesis investigates the viability of Tor or a VPN with Polkadot as external privacy-enhancing tools to hide the user's IP address, as users aiming to achieve unlinkability cannot easily change the Polkadot code.

To analyze the viability, we set up a measurement study to examine the performance of a Polkadot full node behind Tor or a VPN. We investigated, among other things, the latency, throughput, and the number of discovered and connected peers to determine the performance of three Polkadot full nodes located in London, Seoul, and North California. Furthermore, we did a security analysis to determine any vulnerabilities that could emerge from using Polkadot with either of the network environments. And we investigated in-depth the susceptibility of the Polkadot node to an Eclipse attack, as previous research has shown that Bitcoin with Tor was vulnerable to an Eclipse attack.

Our results show that a Polkadot node with Tor has considerably high latency and cannot maintain long-lasting connections. The short connection time decreases the time to perform an Eclipse attack on a Polkadot node from a couple of months and weeks for the normal and VPN environment to potentially six days or less for the Tor environment. We calculated the cost of running an Eclipse attack to be approximately €482 per week. The Polkadot node behind the VPN does perform considerably better. The Polkadot node in London, behind the VPN located in Frankfurt, performed similarly in terms of latency to the Polkadot node in a normal network environment. However, the Polkadot nodes in both the Tor and VPN environment have only outgoing connections. If too many nodes ran behind one of these environments, fewer peers would be able to establish connections with one another, resulting in network partitions or network failure.

This study emphasizes the importance of unlinkability between a Polkadot user's address and IP. However, using Tor or a VPN as privacy-enhancing tools could impact the security of the Polkadot node and the whole Polkadot network. So users should avoid using Tor with Polkadot and carefully consider the tradeoff between privacy and security when using a VPN.

The security issues mentioned in this thesis should be further investigated and tested. Furthermore, a default solution built into the Polkadot source code should be investigated.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General Introduction

Blockchain has become a hot research topic in the last few years. Many papers have been written on the security, scalability, and privacy of blockchain-based systems. The first blockchain system, Bitcoin, was created in 2008 [50]. Since then, many blockchain-based systems have emerged. These systems aim to improve the current system designs, and they are expected to disrupt businesses in different sectors such as finance, healthcare, logistics, energy, internet of things, and identity [14] [49].

While the first blockchain-based systems were only global payment networks, with the introduction of smart contracts [13], it became possible to automate business workflows and rules on the blockchain. A smart contract is a self-executing program that is stored on the blockchain. The program is executed when certain conditions are met. Smart contracts are typically used to automatically execute an agreement between parties without the involvement of an intermediate party. They can also automate workflows by triggering the next action when certain conditions are met. The transactions that are made by the smart contract are irreversible and transparent.

Ethereum [13] implemented the first smart contract-based system. Since then, many blockchain-based systems have been deployed, offering different use cases. One of the problems is that the current blockchain landscape is heavily fragmented. These systems are self-contained networks that cannot communicate with each other [63]. Interoperability between blockchains is an important development in the blockchain landscape to enable cross-blockchain asset transfer and smart contract invocation and interaction.

One of the new blockchain-based systems that tackle the problem of interoperability is called Polkadot. Polkadot, launched in 2020, is an interoperable blockchain protocol that enables the transfer of data or assets between different blockchain networks, so-called parachains. Parachains are user-created blockchain networks that can be customized for different use cases. The Polkadot blockchain enables communication between the parachains, creating an actual multi-chain system ([73]).

1.2 Motivation

The development of Polkadot has been going fast in the last couple of years as they are currently one of the largest blockchain ecosystems in terms of market capitalization[23] and developer contributions [45]. Nevertheless, privacy on the Polkadot network has yet to be one of the key focus points. Little information can be found on the development of privacy-enhancing solutions on the Polkadot network. At the same time, privacy is a critical problem in cryptocurrencies. Research has been done in linking transactions to accounts, linking accounts to cryptocurrency users, and linking cryptocurrency users to real-world identities [48], [3], [32], [26], [9], [7]. As a Polkadot user, this can be a problem. A link between someone's transaction history and real-world identity can be used for targeted ads, manipulation, blackmail, reputational damage, financial loss, physical harm, discrimination, and more [1][30].

One of the biggest risks to privacy is the link between the user's account and IP address. Institutions have requested ISPs to reveal the identities of people linked to an IP address [38]. This thesis investigates methods to improve a user's privacy on the Polkadot Network. More specifically, this paper focuses on the linkability of a Polkadot user to their IP address. Different techniques to improve privacy are considered, and a VPN and Tor are investigated in more detail as privacy-enhancing solutions for this problem. The following section describes the different privacy-enhancing solutions. Furthermore, the use of a VPN and Tor is explained.

1.2.1 Privacy-Enhancing Solutions

Different privacy-enhancing solutions for network layer privacy exist in literature and deployment. These solutions can enhance the user's privacy in different aspects. For example, the user's IP address can be obfuscated using Tor [28], I2p, or a VPN service. Alternatively, there are theoretical approaches to anonymize a user's network traffic by using dc-nets [17] or mix-nets [18].

Tor

Tor[28] is based on the concept of onion routing. A client creates a path through the Tor network, consisting of three nodes. Each packet has multiple layers of encryption, and each node decrypts off one of these layers. The last node, the exit node, sends the packet to the destination. Research has shown that Tor is susceptible to traffic correlation attacks [37] but is still considered one of the best ways to hide a user's IP address from an adversary.

VPN

A Virtual Private Network server relays a user's traffic to the public internet. The user creates an encrypted tunnel to the VPN, and the VPN relays the traffic to the destination, thereby hiding the user's actual IP address. The most common programs for creating a VPN tunnel are OpenVPN [12] and Wireguard [29]. There are multiple ways for a user to use a VPN. Most users connect to a paid VPN hosted by VPN providers like NordVPN[1], ExpressVPN[2], or Surfshark[3]. Another way is to connect to free VPN services. One of the privacy risks of using a third-party hosted VPN server is that the VPN third-party can store all the network traffic. An adversary can request or acquire these logs and figure out the user's IP address. For a paid VPN, it is also possible to link the user's real-world identity by looking at the user's payments. A user can also run his own VPN. The privacy risk for running a user-owned VPN server is that the server instance can be linked to the user. VPNs are considered the easiest and fastest way to hide a user's IP address. However, they are also considered the least secure in terms of privacy.

DC-nets

DC-nets are anonymization networks that are based on the dining cryptographers problem [17]. Most work on DC-nets is limited to theoretical research. There is no publicly available network deployed that can be used.

Mix-nets

Mix-nets are anonymization networks that use multiple proxies to relay network traffic. Mix nets were first described by David Chaum in [18]. Mix networks create hard-to-trace communication by chaining different proxy servers. Each proxy collects network packets, shuffles them, and relays them in random order. Mix-nets are considered high-latency networks, as it can take multiple seconds to hours before the messages are relayed. Different mix networks are described in the literature, like mixminion [25]

[1]https://nordvpn.com/
[2]https://www.expressvpn.com/
[3]https://surfshark.com/

and loopix [53]. However, no publicly available network is deployed, so mix-nets cannot be used for this thesis. [27] is currently working on the first globally deployed network promising low-latency mixing. However, they are still in the testing phase.

I2P

I2P [35] is an anonymization network similar to Tor. I2P was not designed to access the internet but as a self-sustained peer-to-peer network. I2P faces bandwidth limitations at the out proxy, making it difficult to use the network for accessing the internet [36]. This is the reason why I2P is not used in this thesis.

1.3 Research Question

1.3.1 Problem Statement

Section 2.2.6 describes three ways for a user to interact with the Polkadot network, namely via a full node, a light client, or a third-party hosted RPC. Currently, most users use third-party hosted RPC nodes. It is the easiest way to interact with the Polkadot blockchain. This thesis does not analyze third-party-hosted RPC nodes for two reasons. First, the core concept of blockchain is decentralization. If most users interact with the blockchain through a small number of nodes, it harms the decentralization concept. Secondly, third-party hosted nodes are the least secure way to interact with the blockchain. The user must trust that the third-party node will send their transaction to the network or that it returns the correct information. The light clients are also not analyzed in this thesis, as there are no stable release light clients available for Polkadot at the time of writing. So this thesis focuses on the third way, Polkadot full nodes.

In this thesis, users interact with the blockchain via their own full node. The assumption is made that a user's transactions can be traced back to the full node, based on previous research [55] [7], [9]. Another assumption is made that a user can be deanonymized based on the IP address of the full node. So if an attacker can link the node's communication traffic to its IP address, the attacker can deanonymize the user.

Section 2.3 defines a user's privacy in two aspects: anonymity and unlinkability. This thesis focuses on improving the unlinkability between the user's full node and its IP address, as there are currently different network layer privacy-enhancing solutions available as described in Section 1.2.1. The solutions that are considered are Tor and a VPN. A user should be able to use a solution without any alterations to the Polkadot source code. This offers a possible solution for users looking for privacy improvements on the Polkadot network without having to wait for Polkadot developers to implement native solutions.

1.3.2 Aims and Assumptions of Research Project

Aims

1. Gain insights into using Tor and VPNs to increase unlinkability between a Polkadot full node communication and its IP address.
2. Gain insights into the performance costs of Tor and a VPN.
3. Gain insights into possible security vectors introduced by using Tor or a VPN in combination with a Polkadot full node.

Assumptions and Requirements

1. A user's transactions can be traced back to its full node
2. A user can be deanonymized based on the IP address of their full node

The assumptions listed above are made to scope the research project. The first assumption also aims to offer a

1.3.3 Research Questions

1. How can Tor or a VPN be used to enhance the user's privacy concerning the unlinkability between the user's full node and its IP address?
2. Which of the two unlinkability solutions is preferable for the user based on the following:
   (a) The difficulty of linking the node's network traffic to its IP address
   (b) Performance of the node regarding throughput and latency
   (c) Ease of implementation
3. Are there security vulnerabilities introduced by using each solution?

1.3.4 Contributions

This thesis has the following scientific contributions:

- This thesis is, to our knowledge, the first public research into network layer privacy of the Polkadot network.
- A measurement study of the Polkadot network.
- A practical implementation of the use of Tor or a VPN with Polkadot.
- Analysis of security concerns when running Polkadot nodes behind Tor or A VPN.
- An in-depth conceptual analysis of Eclipse attacks on the Polkadot network.

Chapter 2

Background

2.1 Blockchain

A ledger is a collection of financial records that have a chronological order. Blockchain[67] is a technology used to maintain a digital ledger[22] in a distributed system. Blockchain technology handles sharing, replicating, and synchronizing the ledger data between multiple computers worldwide. In the most basic blockchain system, the ledger consists of groups of transactions. These transactions are grouped into blocks, and each block is linked to a previously created block. Linking the blocks creates a chronological chain. This chain of blocks is distributed over a few to tens of thousands of computer systems.

A consensus algorithm [4] ensures that each honest node in the network eventually agrees on the correctness and the order of blocks. When a new transaction is issued, it is shared with all the nodes in the network. Block-producing nodes in the network add this transaction to a new block together with other new transactions. This block is again shared with all nodes in the network, and the block is added to the chain. The nodes can receive different candidate blocks at any time, as they function in a decentralized network, making it possible for different nodes to have varying views of the blockchain. The consensus ensures that these nodes eventually agree on part of the view. At any point in time, all honest nodes agree on the blocks $1...n - T$ with overwhelming probability P where T and P depend on the type of consensus algorithm. A formal definition of blockchain properties is given in [41]. Consensus algorithms are explained in more detail in Section 2.1.3.

2.1.1 Cryptography

Different types of cryptography are used in blockchain technology, such as public-key cryptography and cryptographic hashing. The public key of public-key cryptography is used to determine an address on the blockchain. A user can prove that he owns a particular address with the corresponding private key. On the other hand, cryptographic hashing is used, among other things, to create a digital fingerprint for newly created blocks. These fingerprints can be made with a hash function. Hash functions are one-way functions, meaning creating a hash from specific data is easy. However, it is impossible to reverse the process. Well-designed hash functions have three properties [24]: Preimage resistance, Second-preimage resistance, and Collision resistance. The preimage of a hash function is the set of all input values that produce the same specific hash. For a hash function to be preimage resistant, it should be computationally infeasible to find a value x that results in y for $H(x) = y$. Computational infeasibility means a computation that, although computable, would take far too many resources actually to compute. A hash function can have multiple values of x that result in y. A hash function is second-preimage resistant if it is computationally infeasible to find a second value for x that results in y. A hash function is collision-resistant if it is virtually impossible for hash collisions to occur. A hash collision occurs when two values for x result in the same y. Adding a digital fingerprint of all data inside a block makes it nearly impossible to alter the data without altering the fingerprint, as the digital fingerprint is produced by a hash function that is second-preimage resistant.

2.1.2 Blocks

A block is a data structure in a blockchain system that contains a set of transactions and all relevant metadata for a blockchain to function properly. Figure 2.1 gives an example of a Block in Bitcoin [50]. Each block contains a hash of all data in the block's header. Each block also contains the hash of a previous block. All blocks in the blockchain are linked to each other using this previous block hash. A simple example is given in Figure 2.2. Using the *prev_block* field as data for the block's hash makes it nearly impossible to alter a block (see Section 2.1.1) somewhere in the chain without altering all blocks proceeding that block. So it becomes harder to alter a block as more blocks are built on top of it.

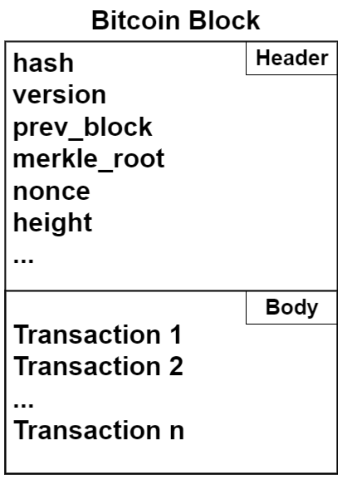


Figure 2.1: An simplified example of a Bitcoin block without all metadata.

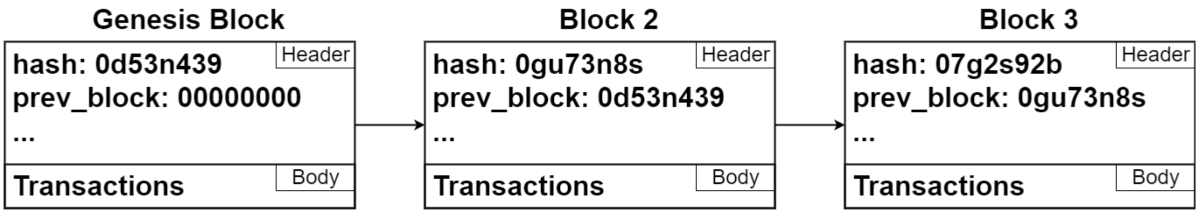


Figure 2.2: An example of a blockchain, by linking blocks via the prev_block field.

Another important value in a block header is the Merkle root [1]. The Merkle root is the root of a Merkle tree data structure. For each transaction in a block, the Merkle tree has the hash value of a transaction as a leaf node. Each inner node in the tree is a hash of the concatenated values of the child nodes. Figure 2.3 gives an example of a Merkle tree. A Merkle tree root is essentially a compressed hash of all transactions. Using the Merkle root in the block header makes it impossible to alter transaction data without altering the Merkle root. The Merkle root is part of the header data used for the block hash, so it becomes nearly impossible to alter the transaction data without altering the block and all subsequent blocks. So a transaction is considered final when it is part of a block in the blockchain. Finalization is discussed in more detail in Section 2.1.3.

The transactions discussed above are the most basic transactions a blockchain can process. However, newer blockchain systems extend the types of transactions, making programming on the blockchain possible. Most of these blockchain systems use smart contracts[68], pieces of code on the blockchain that can be executed. Smart contracts enable developers to write applications for the blockchain. Ethereum, for example, has three types of transactions [71]: Regular transactions, smart contract de-

[1] `https://en.wikipedia.org/wiki/Merkle_tree`

ployment transactions, and smart contract execution transactions. The second and third transaction types deploy code to the blockchain and interact with these pieces of code.

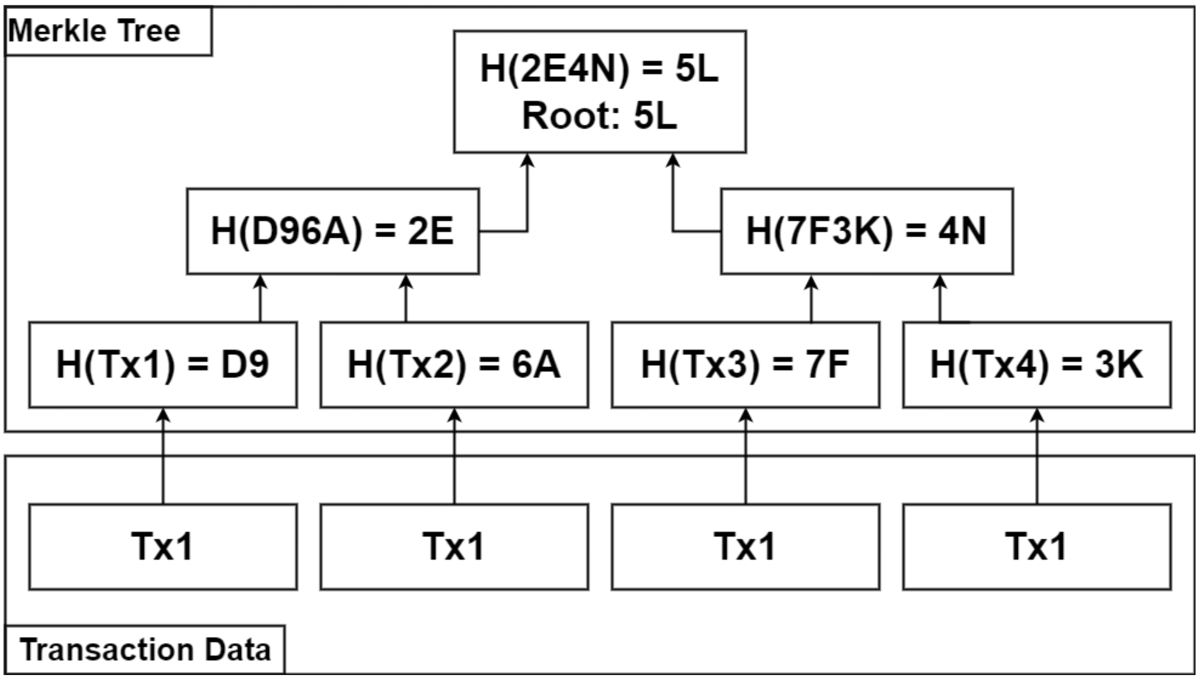


Figure 2.3: An example of a Merkle tree constructed from the transactions in a block. The Merkle tree root is a field in the block's metadata.

2.1.3 Consensus and Decentralization

Decentralization is an essential part of any blockchain system. Blockchain engineers have to make many choices for their blockchain system regarding decentralization. Examples of decentralization choices are: Who maintains and updates the ledger? Who determines what transactions are valid? Who has the power to create new tokens? Furthermore, who determines when and how the rules of the system change? These choices impact the blockchain's decentralization, security, and scalability and are often a trade-off between these three. This trade-off is called the blockchain trilemma [39].

One of the most important choices on decentralization is the question: Who maintains and updates the ledger? Most blockchain systems take a decentralized approach. Anyone can participate in maintaining and updating the ledger. This process consists of validating transactions, blocks, and the order of the blocks. For this to happen, all parties must reach a consensus over the ”true” state of the ledger. These nodes can use different approaches to reach a consensus, as there are a lot of different consensus algorithms. Nevertheless, most of the larger public blockchain systems in terms of market capitalization use a form of the Proof of Work or Proof of Stake consensus algorithm[5].

Proof of Work is a consensus algorithm that works with computationally hard-to-solve hash puzzles to determine which new block should be added to the blockchain. In Bitcoin [50], for example, the puzzle is based on the hash of a block. The hash of a block is calculated from all the block's header data, as explained in Section 2.1.2. Each Bitcoin block contains a nonce in the header. Particular nodes in the Bitcoin network, called miners, try to create a valid block by finding a nonce such that the block's hash is below a certain threshold. When a miner finds such a nonce, it sends the valid block to all other nodes in the network. The other nodes check the block's validity and add it to their chain if it is considered valid.

In proof of work, multiple miners can find valid blocks simultaneously. In this case, there are multiple 'truths' in the network, and the blockchain is forked in multiple chains. Each miner now has a choice to work on one of these forks. The chance of finding a new block is based on the computational power of the miners mining for that fork. For example, if 20% of the miners' computation power is used to work on fork one, and 80% of the computational power is used to work on fork two, the chance that a new block is added to fork two before fork one is 80%. Eventually, one of the forks is longer than the others. This fork is considered the main chain. An essential aspect of forking is that there is no hundred percent guarantee that a transaction is final, as it can be part of a block in one of the side chains. Proof of Work works with probabilistic finality, meaning that the chance that a transaction is final is based on the depth of a block in a chain. For Bitcoin, a transaction is considered final if its block has a depth of six [21], meaning that five other blocks are built on top of it. The chance that a transaction is reversible is a function of the attacker's computational power, and the number of blocks builds on top of the transaction's block (confirmations). The function is described in the original Bitcoin whitepaper[50]. If an attacker has 10% of the computational power, and a transaction has six confirmations, the chance that a transaction is reversed is 0.1%.

Instead of computational work, in Proof of Stake[61], the node that may produce a block is determined based on a financial stake in the system. Nodes that want to participate in the consensus algorithm have to lock tokens. The chance that a node can produce a block is based on the percentage of locked tokens compared to the total number of locked tokens. So if a node has a 20% stake in the network, the node has a 20% chance to produce the next block.

Two main variants of proof of stake exist. Both take a different approach to finalizing produced blocks. The first approach is a Byzantine fault tolerance-based proof of stake system. In this variant, all participating nodes have to vote on the validity of the new block. If two-thirds of the nodes find the block valid, the block is considered final. This type of finality is called absolute finality.

The second type of Proof of Stake system is a chain-based proof of stake system. This system has probabilistic finality, like proof of work systems. The finality of a block is also based on the depth of the block. Both variants of Proof of Stake have advantages and disadvantages. Each blockchain system that uses Proof of Stake has to make a trade-off between the two types.

2.1.4 Attack on Blockchain System: Eclipse attack

A relevant attack in this thesis is an Eclipse attack[64]. In an Eclipse attack, an adversary occupies all the incoming and outgoing connections of a node in the blockchain network, thereby isolating the node from the rest of the network. The adversary can subsequently prevent their target from attaining a true picture of real network activity and the current blockchain state. For network efficiency, nodes in a blockchain network only connect to a select group of peers instead of all peers. This makes an Eclipse attack possible. Bitcoin[8] and Ethereum[44] only have 8 and 13 outgoing connections, respectively.

2.2 Polkadot

Polkadot [73] is a unique blockchain network that focuses on interoperability. Most blockchain systems are closed systems, i.e., it is easy to communicate within the network but hard to communicate with other blockchain systems. Polkadot tries to solve this problem by building a core blockchain via which other specialized chains can communicate. This makes the specialized chains interoperable. The core chain is called the relay chain, and the connected chains are called parachains.

2.2.1 Relay Chain and Parachains

Most blockchain systems have a single blockchain that handles all types of applications. However, these applications have different requirements. For example, a salary payout system could require high security and privacy. On the other hand, a social media application could need a high transaction throughput, so scalability is the primary concern. Polkadot takes another approach. Instead of building one chain to handle all applications, the Polkadot network consists of multiple chains with different characteristics.

At the core of the Polkadot network lies the relay chain. The relay chain is the leading chain in Polkadot, and the primary function of the relay chain is to coordinate the whole system. Among other things, the relay chain acts as a message broker between different parachains and manages the security of the whole system. The relay chain has minimal functionality. For example, smart contracts are not supported. More advanced functionalities are outsourced to the different parachains. A parachain is a specialized blockchain that can handle different use cases. Some Parachains can specialize in different aspects, such as scalability, privacy, or smart contracts. Others will be application specific, for example, a distributed storage service.

2.2.2 Consensus

As explained in Section 2.1.3, there are two approaches to Proof of Stake. Polkadot's consensus algorithm is a hybrid consensus algorithm that uses both approaches by splitting the block production and finality gadget into two different algorithms. This variant is called Nominated Proof of Stake [73].

Important actors in Nominated Proof of Stake are validator nodes. Validators act as block producers and verify blocks produced by other validators. Only a subset of 297 validators are elected to the active validator set and can participate in the consensus algorithm. Polkadot uses an election algorithm called Phragmms[16] to select the active validators. The selection process is based on the total stake a validator has acquired. A validator can supply the stake themselves or receive it from other users. These users are called nominators, and the role of the nominators is explained later in this section. A validator requires better hardware than a normal node to run specialized node software. This hardware is needed to ensure they can process blocks in time. If a validator cannot process blocks in time, they receive fewer rewards or are even punished by getting their stake slashed.

For block production, an algorithm is used called *BABE* (Blind Assignment for Blockchain Extension) [2]. The block production process is split into slots. In each slot, one of the active validators can produce a block. This validator is randomly chosen from the active validator set based on the amount of DOT (native currency on Polkadot) staked. Each slot takes around 6 seconds to ensure the security of the BABE algorithm. BABE is resistant to 2.796 seconds network delay with a slot duration of 6 seconds if 85% of the validators respond in time. These results are based on mathematical experiments[2] in the BABE paper[2].

Because of the randomness, multiple or no validators may be selected for a slot. In case multiple validators are selected, it is a race between them who can send their block to most of the network first. These blocks may create forks in the chain. Validators work on these new forks, but only one of the created forks is chosen in the finalization part of the consensus algorithm. If no validator is chosen, a round-robin style selection algorithm is used to find a secondary validator for that slot. So a slot is never empty as the slot is either filled with the block of the first chosen validators or the secondary validator. The block from the secondary validator is ignored if there is a block from the first validators. The blocks have probabilistic finality.

A new block-producing algorithm called *SASSAFRAS* [11] is being developed that should solve the problems of having zero or multiple validators per slot in the *BABE* algorithm. The algorithm uses particular cryptography to ensure that exactly one validator is chosen per slot.

For the finality part of the consensus process, an algorithm called *GRANDPA* (GHOST-based Recursive ANcestor Deriving Prefix Agreement) [65] is used. It is based on the Practical Byzantine Fault Tolerant algorithm (PBFT) [15]. As with PBFT, *GRANDPA* is secure as long as 2/3 of the validators are honest. Instead of reaching finality on blocks, the *GRANDPA* algorithm reaches finality on chains, speeding up the finality process. So when a chain is finalized, all blocks leading up to that chain have absolute finality. The combination of chain finalization and the creation of primary and secondary blocks

[2]`https://github.com/w3f/research/blob/master/experiments/parameters/babe_median.py`

by the BABE algorithm makes a unique fork choice rule. The *BABE* algorithm produces blocks for a fork built on the chain finalized by *GRANDPA* with the most primary blocks. This rule differs from other blockchain systems' longest chain rule.

A problem in Proof of Stake systems is centralization in the validator set. For scalability reasons, most PoS systems choose a limited validator set. Polkadot also has a limited active validator set, i.e., the set of validators participating in the consensus algorithm. Wealthy users have a higher chance of being chosen as a validator as they have a higher stake, so they have more power to influence the network. Another problem is that only a limited number of users have the technical knowledge and equipment to run a validator. Polkadot solves this problem with its Nominated Proof of Stake algorithm. Users can participate in the validation process by voting on validators with their stake as voting power. When a validator participates in the consensus algorithm, the users get a share of the rewards that the validator receives. Users that participate in the consensus algorithm by nominating validators are called Nominators. They play an essential role in the network's security. If the validator is not chosen, the user does not receive any rewards for that part of their stake. They can redistribute their stake for the next validator choosing round.

2.2.3 Shared Security

One of the primary security goals in blockchain systems is the integrity of the blockchain data. Integrity means that all honest nodes in a blockchain system eventually agree on the same view of the blockchain. When the blockchain data is agreed upon, it is easy to append new blocks but nearly impossible to change existing blocks. A consensus algorithm is used to reach a consensus on a blockchain view. A large part of the security of a consensus algorithm is based on the power of malicious nodes in the system. Blockchain data integrity cannot be guaranteed when there are too many malicious nodes. For example, in Bitcoin, the consensus algorithm is secure as long as 51% of the mining power is honest. The *GRANDPA* algorithm is secure as long as two-thirds of the validators are honest. So to disincentivize malicious validators, it should be expensive to gain power in the network. The expenses for getting 51% of the mining power are high for Bitcoin.

At the time of writing, getting more than half of the mining power of the Bitcoin network would require billions of dollars of mining equipment, making it probably too expensive and challenging to execute a 51% attack. For Polkadot, a malicious user needs more than one-third of all staked DOT to execute a majority attack, which is also expensive. Therefore the Bitcoin and Polkadot networks are considered economically secure. For smaller blockchain networks, it is considerably harder to stay economically secure, as it is easier to gain a majority of the power. Attacks have already been made on smaller networks such as Ethereum Classic [3].

Parachains in Polkadot are blockchains and would normally have their own validator sets for the PoS algorithm, making them potentially vulnerable to attacks. However, Polkadot solves the issue of economic security for its parachains with the notion of shared security. Parachains benefit from the security of the validator set of Polkadot. A part of the Polkadot validators validates new blocks for the parachain. As validators are randomly assigned to a parachain, it is economically expensive for a malicious actor to obtain a majority of the validators of a certain parachain.

Essential actors in the shared security model are collators. Collators are full nodes on the Polkadot network and a parachain network. They collect user transactions for a parachain, aggregate the transactions in parachain proposal blocks and create state transition proofs based on these blocks. A state transition proof is a summary of final account balances that are changed by the transactions. The proposal blocks and state transition proofs are sent to the validators of the relay chain, who verify the parachain proposal blocks. When the block is verified, and all is well, the block is sent to the other validators. Then all parachain proposal blocks are gathered in a relay chain proposal block. When the validators reach a consensus over the relay chain proposal block, the block is shared with all validators and collators. Based on the new relay chain block, collators can build a new parachain proposal block.

[3] `https://www.coindesk.com/markets/2020/08/29/ethereum-classic-hit-by-third-51-attack-in-a-month/`

In this way, parachains are secured by the relay chain.

2.2.4 Networking

The Polkadot networking stack [43] used for communication between nodes in the network is built on top of the libp2p [4] library. libp2p is a modular peer-to-peer networking stack that handles most networking functionalities such as peer discovery, connection establishment, node identities, encryption, and multiplexing.

Node Identities

Each node in the Polkadot network has a node identity based on an ED25519 key pair. This key pair is used to establish a secure connection between a pair of nodes. From the public key, a *PeerId*[5] is created as an identity for each node. This identity is used in the peer discovery protocol.

Peer Discovery

Polkadot has three phases in the node discovery protocol [43]:

- Each node has a list of hard-coded bootstrap nodes maintained by the Web3[6] foundation and parity tech[7]. These nodes are included in the chain specification. The chain specification is a collection of information that describes the Polkadot blockchain network. It consists, among other things, of the bootstrap nodes and the initial state of the network that all nodes must agree on. All blocks in the Polkadot network are built on this initial state.
- The mDNS protocol is a protocol provided by the libp2p library to discover nodes in a LAN. The node broadcasts a request to the local network, and other nodes can respond. This phase of node discovery is optional and can be disabled.
- The last phase for node discovery is the Kademlia[46] algorithm for node discovery. Kademlia asks known nodes for a list of available peers. New peers from this list are added to the list of known nodes.

The node keeps looking for peers to connect to till it reaches a limit. This limit is the default number of outgoing connections a node tries to maintain + 15. For Polkadot, the default number of outgoing connections is 25, so a node stops with peer discovery when it reaches 40+ connections. The Kademlia query schedule is an exponentially increasing delay, capped at 60 seconds. So for most of the run time, there should be 60 Kademlia queries scheduled per hour if the connection limit is not reached.

When a node reaches 40+ connections, it stops searching for new peers, but peers can still try to connect to the node, thereby making themselves known. This way, the number of discovered peers can still increase after the connection limit is reached.

A Polkadot node also has a method of dealing with outdated peer information. A discovered peer is forgotten after an hour if it does not establish a connection in the last hour. A peer can also be dropped for different reasons. For example, a peer does not respond in time to a ping request, refuses to connect, or supplies incorrect information.

Connection Establishment

When a Polkadot node wants to communicate with a peer, it first establishes a TCP connection to the peer. Three networking protocols in libp2p can be used to establish a TCP connection.

[4]https://libp2p.io/
[5]https://github.com/libp2p/specs/blob/master/peer-ids/peer-ids.md
[6]https://web3.foundation/
[7]https://www.parity.io/

- TCP/IP connection is established by providing the remote node's address in the form of */ip4/{IP Address}/tcp/{Port}*.
- A WebSocket connection negotiated within a TCP connection. The address is in the form of */ip4/{IP Address}/tcp/{Port}/ws*.
- Both previously mentioned protocols can also be established via a DNS address. The addresses are in the forms of */dns/example.com/tcp/{Port}* and */dns/example.com/tcp/{Port}/ws/*

Once the TCP connection is established, an encryption layer is negotiated. A multiplex layer is then built on the encryption layer to enable substreams over the connection. Polkadot uses a lot of different protocols, and each protocol has a substream, so a substream consists of all the packets belonging to a certain protocol. Each established connection is either an incoming or outgoing connection. When a peer receives a connection request, the connection is incoming, and the peer is listening. When a peer sends a connection request, the connection is outgoing, and the peer is dialing.

Encryption Layer

For establishing an encrypted layer on top of the TCP connection, the libp2p *Noise*[8] framework is used. The Noise protocol performs a Diffie-Hellman key exchange to create a symmetric key that can be used for all encrypted data. The protocol is defined in detail in the Noise specification.

Multiplexing

After the encryption layer has been established, a multiplex layer is negotiated. A multiplex layer allows a Polkadot node to create different substreams over one TCP connection. These substreams are used for different Polkadot protocols. The multiplex layer is established by either the libp2p *mplex*[9] protocol or the *yamux*[10] protocol.

Polkadot Protocols

Polkadot has different application-specific protocols, and each protocol has a substream. Some essential protocols are pinging and identifying a node, syncing and sharing of blocks, transaction transfers, and Kademlia communication [43]. All protocols are listed in the Polkadot network specification. When two peers establish a connection, the dialing peer sends 32 bytes of random binary data via the ping protocol. The listening peer echoes the data back. The dialing peer verifies the response and measures the latency between the request and response. Then the peers exchange information using the identify protocol. This contains the node's listening address, the peer's observed address, and the protocols that the node supports. The peer uses the observed address to determine if its connection to the node goes through NAT. The supported protocols information is used to determine which Polkadot functionalities the peers support. Validators, for example, support more protocols for the validation process than normal full nodes.

There are two types of protocol substreams, request-response substreams, and notification substreams. For the request-response substream, a single request is sent over the substream. The substream is immediately closed(code does not process packets related to this substream anymore) once the requester receives a response or the request timed out. The initiator of a notification substream sends a single handshake message. The responder can either accept the substream by sending its own handshake or reject it by closing it. After the substream has been accepted, the initiator can send an unbound number of individual messages. The responder keeps its sending side of the substream open, despite not sending anything anymore, and can later close it to signal to the initiator that it no longer wishes to communicate. The most important Polkadot protocols supported by a full node are listed below:

[8] `https://github.com/libp2p/specs/tree/master/noise`
[9] `https://docs.libp2p.io/concepts/stream-multiplexing/#mplex`
[10] `https://docs.libp2p.io/concepts/stream-multiplexing/#yamux`

- $.../sync/2$[11] - a request and response protocol that allows the Polkadot Host to request information about blocks.
- $.../transactions/1$ - a notification protocol which sends transactions to connected peers.
- $.../grandpa/1$ - a notification protocol that sends GRANDPA votes to connected peers.

2.2.5 Substrate

Substrate[12] is a software development kit (SDK) for building blockchains. Polkadot is built on top of substrate, and the same team develops Substrate and Polkadot. Substrate can be used to develop parachains or stand-alone blockchains. It brings blockchain essentials and advanced blockchain features such as storage, consensus, networking, smart contracts, staking, and identity. Substrate is an important subject in this paper as it lays at the core of the Polkadot network and implements the libp2p networking stack.

2.2.6 User Definition

In Polkadot, a user is defined as an entity interacting with the Polkadot blockchain or one of its parachains. A user has three ways to interact with the blockchain.

Full Node

Polkadot is a permissionless blockchain. Thus, it is possible to connect to the blockchain by running a full node. A full node is part of the Polkadot network and can perform all Polkadot functionalities. A full node can interact with other nodes, verify transactions and blocks, and broadcast transactions to other nodes. A full node is the most secure way for a user to interact with the blockchain because a full node downloads a copy of the blockchain and verifies every transaction. By running a full node, a user is not dependent on other nodes for broadcasting transactions or querying blockchain data. A user can run a full node for the Polkadot main network or a node for one of the Polkadot parachains. A full node requires the most resources (Standard hardware: 64GB ram and 80-160GB SSD [59]) as it verifies and stores the whole blockchain. Initialization of a full node can take a couple of hours to a few days. A full-node solution is mainly for companies and institutions with high-security specifications or individuals who prefer a high level of security.

Light Client

A light client [69] is a restricted Polkadot node that connects to full nodes in the Polkadot network. A light client is restricted as it only verifies the authenticity of blocks and does not store any blocks. A user cannot query the blockchain directly while using a light client because the blockchain is not stored locally. A light client has to send the query to a full node to receive the queried data, such as new block headers or balances. Light clients make it possible to interact directly with the blockchain in a trustless manner. This means that a light client can verify the information provided by the full node. So the light client does not have to trust the full node to give the correct information because it can prove that the information is correct. This makes a light client interesting for regular blockchain users. The initialization of a light client only takes up to 10 seconds, and the hardware requirements are low compared to a full node, which makes it possible for every user to run a light client on standard hardware such as laptops or phones.

Third-Party Hosted RPC Nodes

Most users use dapps (decentralized apps) to interact with the Polkadot blockchain. Dapps are applications that interact with the blockchain through smart contracts. Unlike traditional apps, dapps are

[11] The dots in each protocol are replaced with an identifier for the Polkadot network: /91b171bb158e2d3848fa23a9f1c25182fb8e20313b2c1eb49219da7a70ce90c3. Dots are used for readability

[12] `https://github.com/paritytech/substrate`

not controlled by a single entity, making them decentralized. Most of these dapps interact with the blockchain through publicly available full nodes via the RPC protocol [56]. Most decentralized applications on the Polkadot network currently connect to remote full nodes hosted by third parties [51]. It is convenient for most users to interact with public nodes as they do not have to run their own Polkadot node. However, users cannot interact with the blockchain in a trustless manner using public nodes, as these nodes have to be trusted to return the correct information.

2.3 Privacy Definitions

A user's privacy can be divided into two main aspects as described in [52]. These aspects are anonymity and linkability. In this section, both aspects are defined, and a distinction is made between the user's network layer privacy and application layer privacy as described in [20]. The network layer model comprises all messages between entities (senders and recipients). In this model, the attackers can observe network traffic and control some entities. The application layer model comprises all the application-specific information. The information can be almost everything (Usernames, email addresses, account IDs, credit card information, age, sex). In the following sections, the aspects are defined. The distinction between the application layer and the network layer is made for each aspect.

Anonymity

Definition. The anonymity of a subject means that it is hard to identify the subject with certainty within a set of subjects, the anonymity set.

In the Polkadot network, a user is not identified by their real name but by an account address based on a public/private key pair. This public address represents a user on the blockchain and, together with a private key, gives the user access to the user's assets. The public/private key pair is generated using the ed25519 or sr25519 signature schemes [62]. These keys can be generated online or offline. There is no direct link between a user and their key pair. A user can own multiple key pairs and therefore have multiple identities on the Polkadot network.

Polkadot users interact with the Polkadot blockchain using these public addresses on the application layer. Each transaction that a user makes is coupled to this address. Users are, therefore, not anonymous. The transactions on the blockchain are easily coupled to the user's address because the sender's and receiver's public addresses are linked to a transaction. Each transaction is linked to a recipient's public address [31]. A user has sender anonymity if an attacker cannot sufficiently identify which user sends the transaction. A user has receiver anonymity if an attacker cannot sufficiently identify which user is the recipient of the transaction. So a Polkadot user has no sender anonymity, nor does he/she have recipient anonymity on the Application layer.

On the network layer, a user is identified by their IP address. Each packet a user sends to the Polkadot network has the user's IP address as the sender's address. So it is easy to identify a user based on their IP address. All network traffic also contains a recipient's address. Hence, a Polkadot user has no sender or recipient anonymity on the Network layer.

Unlinkability

Definition. Unlinkability of two or more items of interest (IOIs, e.g., subjects, messages, actions, and more) means that within the system, it is hard to distinguish whether these IOIs are related or not.

There are many items of interest in Polkadot, such as the user's IP address, public key, transactions, events, and extrinsic. Some of these IOI are easily linked because of system design. Others can be linked through analysis of the blockchain. [57] and [48], for example, show that transactions in the Bitcoin network can be linked through the Bitcoin transaction graph. The same types of analysis should be possible for the Polkadot blockchain. The unlinkability of some of these IOI is preferable to improve the user's privacy.

Most user interactions with the blockchain on the application layer are linked to their public key. Because of this system design, it is easy to track a user's blockchain interactions. One of the most important cases on the network layer is the link between the user's IP address and the user's public key. If an attacker can link a user's IP address to their public key, they can identify all the user's interactions with the blockchain. [7] and [42] show that for Bitcoin, it is possible to link transactions to users' IP addresses.

2.4 Tor

Tor [28] is an anonymization network that can help users to anonymize their network traffic. Tor is an overlay network that is built on top of the internet. A user can connect to a Tor client. The client routes the user's internet traffic through the Tor network, making it harder for an adversary to trace its internet activity. Tor encrypts internet traffic with multiple layers of encryption and routes the traffic around the globe through the Tor network, a global network of relays run by volunteers. Routing multi-encrypted internet traffic through multiple relays is called onion routing. Tor uses, by default, a three-hop strategy, sending the traffic through three relays.

The name onion routing is based on the multiple layers of encryption used to encrypt a network packet. Each relay on the routing path decrypts one layer of security. The last relay decrypts the last layer of encryption and sends the original packet to the preferred destination. The packet flow works in reverse when the exit node receives a response. Each node first encrypts the message and then sends the message to the next node on the reversed routing path. The user finally receives a triple-encrypted message that it can decrypt. Tor can also bypass censorship blockades by using special relays called bridge relays. Tor is explained in more detail in the following paragraphs.

2.4.1 Circuits

A Tor client routes all network traffic through three different relays in the Tor network. The first relay is called the guard relay. This relay receives fully encrypted network packets from the user. It is the only node that knows the user's IP address. The second relay is a general relay that only relays data packets from the guard relay to the exit node. It does not know anything about the user or the traffic destination. The last relay is the exit relay. The exit relay sends the user's original packet to the preferred destination. Only the exit relay knows the destination of the internet traffic but does not know anything about the user.

Such a path through the Tor network is called a circuit. Each circuit is identified by the Tor client using a CircID. All the user's network traffic is routed through a circuit for roughly ten minutes. After ten minutes, the Tor client creates a new circuit, making it harder to track a user over time. A user can open multiple TCP streams (see Section 2.4.3) over one circuit. If a user keeps a TCP stream open for longer than ten minutes, the circuit handling the stream remains open. The new circuit handles new streams.

Constructing and coordinating a circuit is difficult, but Tor has a nifty protocol. When a user wants to open a new circuit, the Tor client first requests a list of active relays from one of nine Tor directory nodes. Independent persons or organizations run these nodes and keep a consensus on the list of active relays. Every hour the consensus is calculated and signed by each directory node, ensuring that every Tor client has the same network view while maintaining trust in the directory nodes.

After receiving a complete list of Tor nodes, the Tor client can start constructing a circuit. The Tor client first contacts a chosen guard relay to check if it is available. The client negotiates a TLS session with this relay if the node can receive requests. The next step is connecting to the second (or middle) relay. As one of Tor's privacy guarantees, Tor prevents the middle and exit relay from knowing the client's IP address. The client cannot directly contact these relays, not even to set up a TLS session. The TLS session between the client and the middle relay is negotiated through the guard relay. This way, the client and middle relay will never directly communicate. When the TLS negotiation is complete, the circuit is extended by one relay, and the client can select an exit relay. The TLS negotiation process is the same, but now the negotiation with the exit node is done via both the guard and middle relay. After this negotiation, the client created a different symmetrical TLS session key for each relay. The client can now send and receive triple-encrypted packets (cells) through the circuit.

2.4.2 Cells

An encrypted packet sent through a Tor circuit is called a cell [70]. Tor contains two types of cells. Fixed-length cells and variable-length cells. Fixed-length cells use padding to maintain their fixed length. Fixed-length cells are used to prevent correlation attacks, where an adversary can correlate packets through the Tor network by their packet size. Fixed-length cells are used, among other things, for creating, extending, and destroying circuits and for relaying end-to-end stream data via relay cells. Variable-length cells are only used for a part of the handshake protocol traffic and only cover a small part of the Tor cells.

Relay cells are fixed-length cells used to set up a stream (see Section 2.4.3) between the Tor client and the exit relay and tunnel end-to-end stream data through a circuit. While both sides of the circuit can send data and commands, only the Tor client can initialize a stream. The structure of a relay cell payload as constructed by the Tor client is shown in Figure 2.5 [70]. The relay cell payload is encrypted in the opposite order of the circuit path. So for the last layer of encryption, the cell is encrypted with the symmetrical key of the guard relay, as the guard relay is the first to receive the relay cell. The exit relay is the only relay able to decrypt the original payload. The exit node is the only relay that knows to which stream (see Section 2.4.3) the cell belongs.

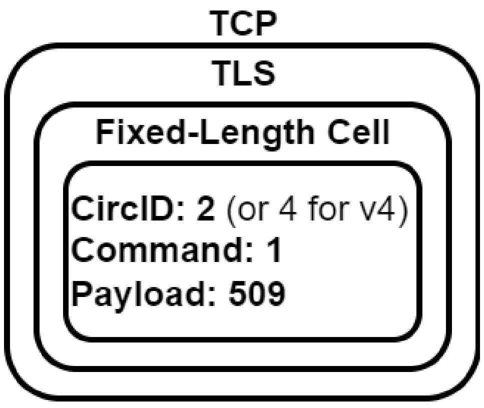


Figure 2.4: cell

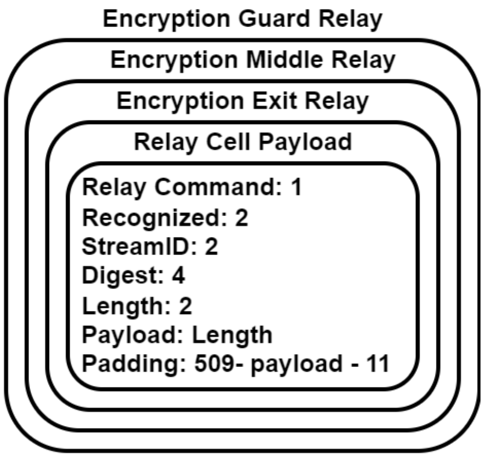


Figure 2.5: relay cell

2.4.3 Streams

Tor can be used to forward TCP streams around the world. When a user starts a Tor client, a SOCKS5 proxy is started. The user can use this proxy to send TCP data through the Tor network. Tor is, for example, used by the Tor browser to send and receive HTTP(S) traffic through Tor. Any application that can use a SOCKS5 proxy and uses TCP traffic should be able to send the traffic through Tor.

When a user wants to open a TCP connection, the user asks the Tor client to set up that connection. The Tor client then initializes a new stream by sending a *RELAY_BEGIN* cell to the exit relay selected from the newest open circuit. The exit relay sends back a *RELAY_CONNECTED* cell to confirm that a connection is established. The user is now able to send TCP packets through Tor. The packets are routed through the established connection via *RELAY_DATA* cells. Each relay in the circuit decrypts its layer of the cell. Using the circID, the relay forwards the packet to the next relay in the circuit. Relays can detect if the cell is still encrypted by checking the *recognized* field. If the *recognized* field is zero, the cell is fully decrypted and should have reached the endpoint of the end-to-end stream.

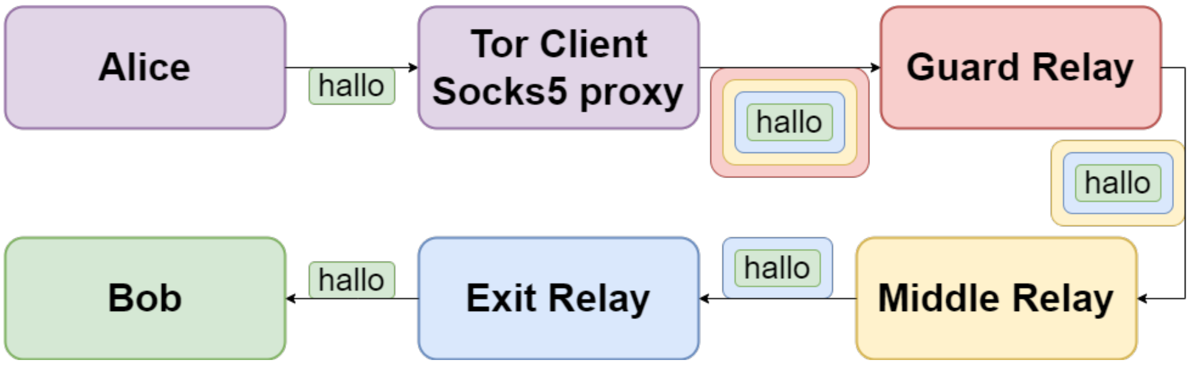


Figure 2.6: An example of the encryption and decryption of a Tor cell on the Tor stream

2.4.4 Channels

Each pair of relays opens only one TCP connection between the two. Such a connection is called a channel. A channel prevents two relays from opening a new TCP connection for each end-to-end TCP stream they are a part of. A channel is responsible for sending cells between a pair of relays. The cells sent through a channel can be from different end-to-end TCP streams. The channel multiplexes these different streams. Channels significantly reduce the number of open connections in the Tor network compared to a TCP connection for each stream.

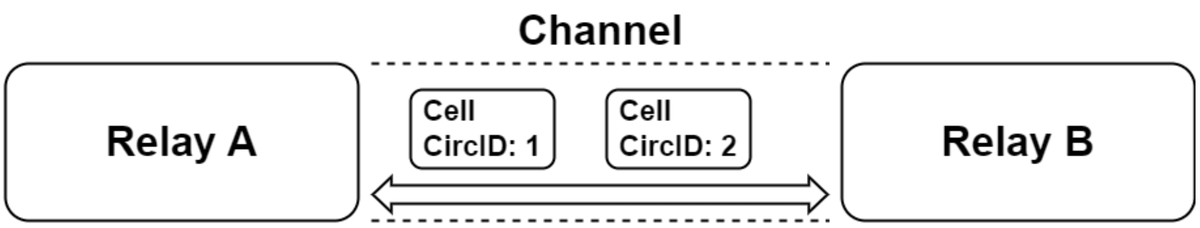


Figure 2.7: Tor channel

2.4.5 Bridges

To prevent users from using Tor, governments, companies, or adversaries can block access to the Tor relays, as the list of all relay nodes is openly available. The Tor project found a solution by using special relays called bridges. Bridges are relays that are not publicly known. They forward a user's traffic to their chosen guard node. Therefore, a bridge is essentially a proxy into the Tor network. To receive a list of a few bridge relays, a user must email the Tor project or request a bridge in the Tor browser. When a bridge is censored, the user can switch to one of the other bridges in the list. These bridges can, for example, be useful for people that want to run a Polkadot node with Tor in a country that censors either Polkadot or Tor.

2.4.6 Attacks on Tor

Through the years, many papers have been written on attacks on Tor. As Tor is an anonymization network, most attacks against Tor try to de-anonymize Tor users. Other attacks focus on disrupting the Tor services, primarily by using denial-of-service attacks on single or multiple Tor nodes. Both types of attacks are relevant for this thesis as they can de-anonymize a user or prevent the user from using Tor. Tor attacks can be divided into groups. Karunanayake et al.[40] created a taxonomy of attacks on Tor. On the top level, they divided the attacks into four groups.

1. De-anonymization attacks
2. Network disruption attacks

3. Censorship attacks
4. Generic attacks

Then they divide the de-anonymization attacks into four categories, each of which is divided into active and passive attacks. In active attacks, the adversary can observe and actively manipulate the traffic. In passive attacks, the adversary can only passively observe network traffic.

De-anonymization Attacks

In de-anonymization attacks, the adversary aims to link a client's IP address to the server's IP address with which they are communicating. These attacks can be divided into different categories, and they mainly depend on the type of resources the adversary can observe or control:

1. Entry and Exit Routers
2. Onion Router, Onion Proxy, or destination server
3. Other resources
4. Hybrid

In entry and exit router attacks, the attacker controls the guard and exit node of the circuit used by the client and the server or can observe the traffic to and from these nodes. The attacker can correlate the traffic entering the guard node to the traffic exiting the exit node using different types of analysis. The guard node knows the client's IP address, and the exit node knows the server's IP address, so the attacker can confirm that the client and server are communicating.

This attack is passive when the attacker can correlate the traffic using the available data. The attacker could also manipulate the data by manipulating the inter-packet arrival time, the packet rate, or the latency. Traffic can then be correlated based on active manipulations of the traffic. In this case, it is an active attack. Tor is vulnerable to these attacks as it is a low-latency anonymity network. Tor does not delay or mix network packets, making it easier to correlate traffic than, for example, a mix network. Tor does also not obfuscate network traffic, making it easier for adversaries to detect and filter out Tor traffic.

Correlation attacks are challenging to execute in the actual world, as the attacker needs many resources to control guard and exit nodes. The Tor network has grown significantly through the years, making entry and exit router attacks more difficult to execute. The attacker needs access to many nodes to have a significant share of the entry and exit nodes. Else the adversary's probability of controlling a user's entry and exit node is extremely low. Only a highly resourceful global adversary should be able to execute these attacks.

For attacks in the Onion Router, Onion Proxy, or destination server category, the attacker only needs to control a single Tor network component. Most attacks are irrelevant for this thesis as they focus on de-anonymizing onion services[13]. The other attacks are designed for earlier versions of the Tor network and created for using Tor in combination with specific applications, making them irrelevant.

In other-resourced attacks, an adversary uses resources other than the Tor network resources. The most commonly used other-resourced attacks use the network link between the client and the guard node. ISPs and different kinds of agencies have access to these resources. These attacks focus on traffic analysis to link the user's IP address to the traffic destination. Tor is not resistant to a global adversary that can analyze global network traffic. However, it is improbable for regular adversaries or agencies to execute other-resourced attacks as they do not have access to these resources.

Hybrid attacks combine previously mentioned resources to de-anonymize a user or hidden services. Most attacks in this category focus on hidden services and are irrelevant to this thesis.

[13] `https://community.torproject.org/onion-services/overview/`

Network Disruption Attacks

In Network Disruption attacks, an adversary focuses on disrupting the network, making it unusable by users. Most of these attacks fall in the Denial-of-Service category. In a Denial-of-Service attack, an adversary overloads the client's network resources, limiting or disabling those resources. By overloading guard or exit nodes with large amounts of traffic or connection attempts, these nodes cannot handle the client's traffic at the standard capacity. DoS attacks can be targeted to a single or small group of nodes or can target the whole Tor network. DoS attacking the entire Tor network would require many resources, so it is less probable to happen.

Censorship and General Attacks

Many users use Tor as a censorship circumvention tool. Censorship attacks focus on blocking access to the Tor network. Many of these attacks are executed by governments with excessive censorship. All attacks that are not easily categorized fall in the General attack category. One of these types of attacks is fingerprinting attacks.

In a fingerprinting attack, an adversary uses the aspect that most network traffic has distinct characteristics. This property can be used to determine what kind of service a client is communicating. An attacker first builds a fingerprint database and compares the client's traffic to that database. This traffic can then be analyzed for specific characteristics. For example, webpages can be fingerprinted by their files.

2.5 VPN

Virtual private networks, also known as VPNs, are a service that creates an encrypted private connection over a public network. Users can use VPNs to improve their online privacy or circumvent geolocation-based blocking or censorship. Users connect their devices via a VPN client to a VPN. The VPN port forwards its network data making it seem like the VPN's IP address and location is the user's IP and location.

VPNs are used in different ways. Regular internet users can use a VPN to access geographically blocked content or hide their IP address and browser history. A VPN can also make it difficult for advertisers to target individual ads. Organizations can use a VPN in different ways. An organization can connect multiple private networks via a VPN or can use a VPN to connect remote employees via an authorized and encrypted channel to their private network.

2.5.1 Tunneling

A VPN uses tunneling software to set up a private encrypted connection between two devices. Data tunneling is a way to send data between two private nodes over a public network without the nodes in the public network knowing that the traffic is part of the private network. Tunneling consists of two main concepts, namely encapsulation and encryption. Encryption ensures that only the sender and intended recipient can read the private network traffic and that the recipient of a message can authenticate that the sender sent the message. Encapsulation hides data packets inside other data packets. The private network data is fully encrypted and then encapsulated in a public data packet to make it look like regular network traffic. There are different tunneling protocols to secure and encrypt data. The tunneling protocol used in this thesis is Wireguard [29].

Chapter 3

Related Work

The related work in this thesis is split into two parts. The first part of the related work is in this section. The second part is in Section 7.1.1 and discusses three Eclipse attacks performed on Bitcoin[8], Ethereum[44], and IPFS[54] in depth.

This section discusses various research that has been done on privacy-related issues and solutions for blockchain technologies. Especially Bitcoin is the most researched cryptocurrency in terms of privacy. Different privacy-related issues exist. The issues can be divided into two categories. The first category includes problems with the linkability of the user's public key to their real-life identity. The second category includes problems with the linkability and anonymity of a user's transactions.

3.1 Linkability between Blockchain Address and User's Identity

A user's identity can be linked to their cryptocurrency address in different ways. Online stores can link a user's email address or shipping information to their cryptocurrency address when this cryptocurrency is used as a payment method. [32] shows how third-party web trackers can use cookies to deanonymize users of cryptocurrencies. They found that many web merchants leak user information to trackers, such as names, emails, and shipping addresses, that can link web profiles and real identities to cryptocurrency users.

[42] shows that it is possible to link Bitcoin transactions to the IP address of the server that proposed the transaction. The research does not discuss the implications of a peer using the Tor anonymity network [28]. Bitcoin developers proposed using Tor [28] as a countermeasure. In [7], researchers show that Bitcoin users can be deanonymized by linking their public keys to their public IP address, even if they are behind a firewall or NAT. They also show that their method even works when users use Tor to anonymize their traffic by abusing Bitcoin's anti-DoS countermeasures. A bitcoin node can ban traffic from a certain IP address (for example, a Tor exit node) if this IP sends malformed messages. An attacker can send malformed messages from exit nodes not under his control such that the Bitcoin node bans these exit nodes, forcing a user to route his traffic through the attacker's controlled exit nodes. This attack is not possible in the Polkadot network, as Polkadot nodes do not implement an IP blocklist as an anti-DoS countermeasure. [7] also shows that linking multiple public keys together is possible if they are used during a single session.

In [8], the security risks of using Bitcoin with Tor are discussed. They show that it is possible to perform a stealthy man-in-the-middle attack, controlling all information flows between users using bitcoin via Tor. They also show that they can fingerprint a user by setting an address cookie. This cookie can be used to track users over multiple sessions. Users may directly connect to the bitcoin network in the new session, revealing their IP address.

Dandelion [72] is proposed as a solution for the linkability of transaction messages to the source node's IP address. Dandelion implements a new networking policy that provides network-wide anonymity. By first sending the transaction message through multiple nodes in the anonymity phase, it becomes hard to pinpoint the source node in the message's spreading phase. Dandelion is implemented as a networking policy for the bitcoin core. Such an implementation is only possible for Polkadot by changing Polkadot's code.

3.2 Linkability of Transactions and Sender/Receiver Anonymity

[57] and [48] showed that users' transactions could be linked to each other in the Bitcoin network. By using statistical analysis [57] analyzed user behavior in the Bitcoin network, and the flow of bitcoins through the network. They managed to track transaction flows from users that use different wallets to protect their privacy. [48] used heuristic clustering to cluster bitcoin wallets in groups, then re-identification attacks were used to classify the cluster operators. Different solutions have been proposed to prevent transaction linkage and allow sender anonymity, so-called mixing, or tumbler services. Mixcoin [10] and Coinshuffle [58] are two examples of mixing services implementation in Bitcoin. Möbius [47] is a tumbling service implemented on Ethereum that uses smart contracts instead of a centralized mixing service to do the mixing autonomously.

While mixing services help anonymize native tokens like Bitcoin and Ethereum, different standalone cryptocurrencies exist that provide better privacy guarantees. Mixing services break the link between the sender of a transaction and the receiver of this transaction. Standalone cryptocurrencies such as Monero [60] and Zcash [6] use different methods to not only break up the link between sender and receiver but also provide sender and receiver anonymity.

Monero [60] uses ring signatures to provide unlinkability of transactions. With RCT (ring confidential transactions), the amounts involved in the transaction are hidden. Monero also enables sender and receiver anonymity. Zcash [6] uses another cryptographic approach, namely Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARKs). This allows users to prove a transaction is valid without revealing information about it (sender, receiver, amount). Zcash allows for unlinkability between transactions and sender and receiver anonymity.

Recently, a new privacy-focused framework called Manta [19] was proposed. Like Zcash, Manta uses zk-SNARKs to provide privacy. Manta implements efficient zk-SNARKs improving upon the performance of Zcash. Manta also supports a multi-asset anonymous payment protocol improving upon Monero and Zcash, which only supply a mono-asset private cryptocurrency. Manta can be implemented as a Polkadot parachain, making it possible to privatize different cryptocurrencies that are connected to the Polkadot network. Manta also provides the world's first Decentralized Anonymous Exchange (DAX), making it possible to swap assets anonymously.

The previously mentioned mixing services and privacy-focused standalone cryptocurrencies focus on application layer privacy. [9] shows that it is still possible to deanonymize and link transactions in Bitcoin, Monero, and Zcash using network analysis. This paper focuses on the privacy of Polkadot on the network layer instead of application-layer privacy like Manta does as application-layer privacy would require code changes in the Polkadot source code.

Chapter 4

Design and Implementation

Different analyses are done to investigate the usability of Tor and a VPN to enhance the IP unlinkability of users running a Polkadot node. One key aspect is the performance of such technologies. Both technologies are compared with a standard environment to determine their performance. In Section 4.1, the design of the performance analysis is explained. In Section 4.2, the implementation of the experiments is explained in detail.

4.1 Design Performance Measurements

In this section, the design of the measurement study is explained. The goal is to determine the performance impact of Tor and a VPN on the performance of a Polkadot node. Key aspects such as throughput, number of connections, and latency are examined to determine the overall performance. The study consists of different experiments designed to support the possible performance differences and rule out certain biases.

4.1.1 Environment Experiment

The first experiment is set up to determine the working of each environment. These environments consist of a Polkadot node in a standard environment, a node behind a self-hosted VPN server, and a node behind the Tor network. Each option is run on the same server to minimize server impact on the performance results. The experiment is run to compare the performance of the VPN and Tor environment to the normal environment and to determine the viability of the Tor en VPN environment as network environments.

4.1.2 Location Impact Experiment

The second experiment determines the server location's impact on a Polkadot node's performance. All three environments are used in different locations worldwide. Three servers are deployed in different locations. The first server is deployed in Europe, the Second in North America, and the Third in North-East Asia. The location's performance impact can depend on the location of other servers in the Polkadot network. If most Polkadot nodes are located in Europe, the node in Europe probably performs better in terms of latency. The information provided by the Telemetry service[1] gives an indication on the location distribution of the Polkadot nodes. The map shows a considerable number of nodes in Europe compared to the rest of the world. However, it is hard to exactly determine the location distribution. If the server location significantly impacts the performance, it could be helpful to set a location-based exit policy for Tor. However, the impact on the number of connections could be limited.

[1] `https://telemetry.polkadot.io/#/0x91b171bb158e2d3848fa23a9f1c25182fb8e20313b2c1eb49219da7a70ce90c3`

4.1.3 Metric Design

A wide range of metrics is considered to determine the performance of a Polkadot node. Each metric is chosen to gain insights into the stability and performance of the node. This section describes the importance of each metric. Section 4.2.3 describes the implementation details of the metric computations. The metrics that are considered are:

- Latency of the network communication
- Number of connected peers over time
- Number of discovered peers
- Disconnection rate
- Throughput
- Number of Kademlia queries
- Total number of connected peers
- Average connection time

Latency of the Network Communication

The network latency of the node is the time it takes for a message to travel over the internet to a peer. In this thesis, the network latency is measured as the time it takes to receive a response for a request, meaning the time it takes for a packet to travel to and from a peer, plus some request processing time. Implementation details of the network latency are discussed in Section 4.2.3.

Network latency between nodes is an essential factor when looking at blockchain networks. When latency is high, it takes a long time for nodes to communicate and, therefore, to synchronize their blockchain states. This has an impact on the performance of a blockchain. The faster the nodes can synchronize their states, the more state changes (transactions) they can handle. For example, Polkadot currently has a block time of 6 seconds, so every 6 seconds, a new block is generated. As discussed in Section 2.2.2, the BABE algorithm is resistant to a network delay of 2.796 seconds if the block time is 6 seconds and 85% of validators respond in time. If the block time is reduced to 4 seconds, the Babe algorithm is resistant to a network delay of 1.53 seconds if the other variables stay the same. If every node in the network could reduce its latency, Polkadot could potentially reduce its block time. This increases the number of transactions Polkadot could handle, as more blocks could be created per hour.

As previously mentioned, the BABE algorithm is secure if 85% of the validators respond within 2.796 seconds. So the latency results of nodes behind a VPN or Tor could impact the network's security. Besides the impact of network latency on network performance, network latency can also cause different types of timeout errors. These timeouts can disconnect nodes, destabilizing the blockchain network. Timeouts are discussed in more detail in Section 5.1.4.

Number of Connected Peers over Time

The number of connected peers over time is the number of peers that the Polkadot node has an established incoming or outgoing connection with measured in 15-second intervals, meaning that every 15 seconds, the number of connected peers is logged. The number of connected peers is an important metric for a node in a distributed network. A low number of connected peers can have security implications as it can be easier to perform Eclipse attacks. It has been shown that Bitcoin[8], Ethereum [44], and IPFS [54] are vulnerable to Eclipse attacks. The larger the number of connected peers, the harder it is for an attacker to eclipse a node in the network. Polkadot's maximum number of connected peers is set to 50, which should make it considerably hard to eclipse a node. The implementation of the number of connected peers over time is discussed in Section 4.2.3

Number of Discovered Peers

The number of discovered peers is the number of peers that established a connection, attempted to establish a connection, or were discovered during the node discovery. The number of connected peers over time is a subset of the number of discovered peers. The number of discovered peers is an important metric to determine the node's connectivity. The more peers a node discovers, the larger its network view. If the number of discovered peers is low, it could indicate that the node is the target of an Eclipse attack, as the attacker forces a particular view on the node. A low number of discovered peers could also make a node more vulnerable to Eclipse attacks if the attacker can include its nodes in the smaller network view. The chance that the node connects to the attacker's nodes is higher. On the other hand, an Eclipse attack is more challenging to execute if the attacker cannot include its nodes in the limited view of the victim, as the node can not connect to nodes it does not know. This is further discussed in Section 7.1.2. The implementation of the number of discovered peers is discussed in Section 4.2.3.

Disconnection Rate

The disconnection rate is the number of the node's connections that are disconnected per hour. The disconnection rate of a node can say something about the node's network stability. If a node has a high disconnection rate, it could mean it cannot maintain connections. This could result in the connection attempts by the node being blocked by peers, so the node attempts to connect with other peers. So a high disconnection rate could result in a high total number of connected peers. This can increase the likelihood of an Eclipse attack.

Throughput

Another metric to look at is the throughput of a Polkadot node. The throughput is the total number of inbound and outbound bytes that the node sends and receives. If the network throughput of a Polkadot node is too low, it cannot deliver all messages. The network drops messages, or the messages are not delivered in time. This can cause different types of errors and timeouts. These errors and timeouts can result in connections being dropped. So the throughput of a node is an important metric to determine network stability. Throughput could be problematic for the nodes behind the Tor network, as the Tor network can struggle with network throughput.

Number of Kademlia Queries

The number of Kademlia queries is the number of Kademlia lookups the node performs. The number of Kademlia queries is an indication of the discovery rate of the Polkadot node. As with the number of discovered peers, the discovery rate is important to determine the node's connectivity. As described in Section 2.2.4, Kademlia lookups are stopped when the node reaches 40+ connections. Else the node should perform around 60 queries per hour.

Total Number of Connected Peers

The total number of connected peers is the total number of peers with which the node has had an established incoming or outgoing connection. The total number of connected peers gives an indication of the stability of a Polkadot node. A high total number of connected peers indicates that the Polkadot node struggles with maintaining connections. It could also indicate that peers are dropping the Polkadot node if it is not functioning properly. This can happen when the node's connection to the peer times out. There are also possible errors on the transport layer, resulting in a disconnection. A high total number of connected peers could help an adversary perform an Eclipse attack. This is discussed in more detail in Section 7.1.2.

Average Connection Time

The average connection time is the average time that an established connection lasts. As with the total number of connections, the average connection time gives an indication of the stability of the peer connections. A short average connection time means that a node has unstable connections. Each time a node disconnects, it has an extra slot for a new connection. A low average connection time can create security issues, as there are more opportunities for an adversary to connect to the node. This is, for example, useful for an Eclipse attack. These security issues are discussed in more detail in Section 7.1.2.

4.2 Implementation Performance Measurements

This section gives a detailed explanation about the implementation of the experiments mentioned in Section 4.1. The general Polkadot server setup is explained in Section 4.2.1. Section 4.2.2 explains the tools used to collect and analyze the performance data. Finally, Sections 4.2.4 and 4.2.5 dives into the setup of the VPN and Tor environments.

4.2.1 General Setup

In this section, the general server setup is discussed. Polkadot has several settings. Some of these settings are used to enable Polkadot to work with Tor. In the following sections, the server and node specifications are given. Moreover, the setting options are substantiated.

Node Specifications

The Polkadot[2] v0.9.27 release is used. The node is run as a full node, meaning that only a part of the chain is stored. Storing the entire chain currently takes around 473GB, while running a full node only takes up 93GB of storage.

The node software is easily cloned from the Polkadot node releases. The installation guide for installing a Polkadot full node is used as an installation instruction[3]. As previously mentioned, Polkadot has a lot of different run settings.

The `--no-telemetry` option is used to enhance the privacy of the Polkadot node. The `--no-mdns` and `--chain` options are used to make the Polkadot node work with Tor. These settings are explained in more detail in the following sections. In this thesis, the Polkadot node is run with the following settings:

```
--no-mdns
--no-telemetry
--chain=custom_spec.json
```

The command for downloading and running a Polkadot node is given below:

```
# VERSION is the Polkadot version. Example is v0.9.21
curl -sL
 ↪  https://github.com/paritytech/polkadot/releases/download/*VERSION*/polkadot
 ↪  -o polkadot

# Make Polkadot executable
sudo chmod +x polkadot

# Run Polkadot node with options
./polkadot --no-mdns --no-telemetry --chain=custom_spec.json
```

[2]https://github.com/paritytech/polkadot/releases
[3]https://wiki.polkadot.network/docs/maintain-sync

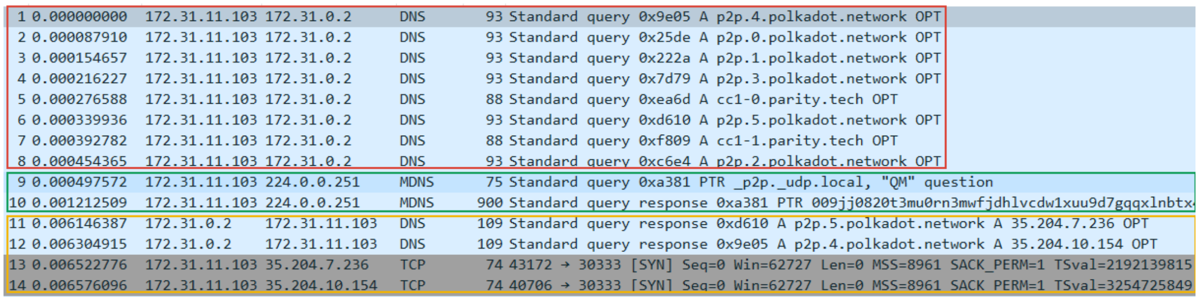
Telemetry

Telemetry is a service to send Polkadot monitor data to a monitoring server. The Polkadot client connects to the public Polkadot telemetry server[4]. This server records, among other things, the node version, the node's network id, the node's location, and the number of peers. Connecting a Polkadot node to the public telemetry service may expose information that puts a node at higher risk of being attacked. The telemetry service could also expose the node's IP address. The telemetry service is disabled using the `--no-telemetry` option to prevent these problems.

No mDNS Setting

By analyzing the network traffic of a Polkadot node, the different stages of the node discovery protocol can be investigated. For the network analysis, Tshark[5] is used to capture the network packages on the Polkadot server, and Wireshark[6] is used to analyze this network data. Figure 4.1 shows the network traffic of a normal Polkadot node. The green-boxed packets show an mDNS request and response. These packets are part of the second stage of the Polkadot node discovery protocol, as explained in Section 2.2.4. The mDNS protocol looks for other Polkadot nodes in the local network.

The Tor network can only serve TCP streams, so users cannot use UDP protocols with Tor. Tor is unable to make mDNS requests. For each experiment, the mDNS option is disabled to rule out the performance impact of mDNS. As the mDNS phase is optional, it is disabled with the `--no-mdns` run option. The mDNS protocol tries to establish connections with nodes in a LAN. Disabling this option could improve the node's security as it could decrease the clustering of connected nodes. Node clustering could be dangerous. It could be possible for an adversary to disconnect a cluster of nodes from the rest of the network if these nodes are mainly connected to each other, with few connections to the rest of the network. On the other side, disabling mDNS could increase the average latency of the connected nodes, as the node does not focus on establishing connections in the LAN. Both the security and latency impact completely depends on the number of nodes that are located in the LAN.



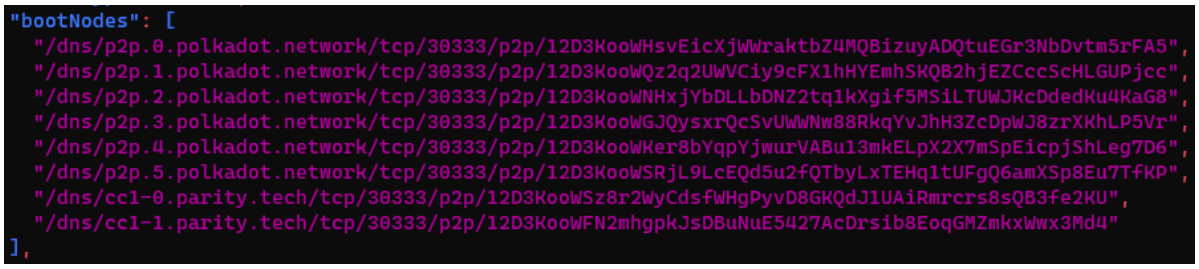
```
 1 0.000000000  172.31.11.103 172.31.0.2     DNS       93 Standard query 0x9e05 A p2p.4.polkadot.network OPT
 2 0.000087910  172.31.11.103 172.31.0.2     DNS       93 Standard query 0x25de A p2p.0.polkadot.network OPT
 3 0.000154657  172.31.11.103 172.31.0.2     DNS       93 Standard query 0x222a A p2p.1.polkadot.network OPT
 4 0.000216227  172.31.11.103 172.31.0.2     DNS       93 Standard query 0x7d79 A p2p.3.polkadot.network OPT
 5 0.000276588  172.31.11.103 172.31.0.2     DNS       88 Standard query 0xea6d A cc1-0.parity.tech OPT
 6 0.000339936  172.31.11.103 172.31.0.2     DNS       93 Standard query 0xd610 A p2p.5.polkadot.network OPT
 7 0.000392782  172.31.11.103 172.31.0.2     DNS       88 Standard query 0xf809 A cc1-1.parity.tech OPT
 8 0.000454365  172.31.11.103 172.31.0.2     DNS       93 Standard query 0xc6e4 A p2p.2.polkadot.network OPT
 9 0.000497572  172.31.11.103 224.0.0.251    MDNS      75 Standard query 0xa381 PTR _p2p._udp.local, "QM" question
10 0.001212509  172.31.11.103 224.0.0.251    MDNS     900 Standard query response 0xa381 PTR 009jj0820t3mu0rn3mwfjdhlvcdw1xuu9d7gqqxlnbtx
11 0.006146387  172.31.0.2    172.31.11.103  DNS      109 Standard query response 0xd610 A p2p.5.polkadot.network A 35.204.7.236 OPT
12 0.006304915  172.31.0.2    172.31.11.103  DNS      109 Standard query response 0x9e05 A p2p.4.polkadot.network A 35.204.10.154 OPT
13 0.006522776  172.31.11.103 35.204.7.236   TCP       74 43172 → 30333 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=2192139815
14 0.006576096  172.31.11.103 35.204.10.154  TCP       74 40706 → 30333 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=3254725849
```

Figure 4.1: Wireshark network capture of normal Polkadot node during startup. In red are the DNS request for the bootstrap nodes. The green packets show the mDNS requests and responses to and from the local network. Yellow shows DNS response packets and the first TCP connection packets to two bootstrap nodes.

Custom Chain Specifications

The first Polkadot node discovery phase (see Section 2.2.4) consists of connecting with the hard-coded bootstrap nodes. These nodes are hard-coded in the chain specification JSON file. Figure 4.2 gives an example of the bootstrap nodes. On startup, the Polkadot node tries to resolve the bootstrap nodes' hostnames using the DNS protocol. Figure 4.1 shows the DNS queries in the red box. In the yellow box, the IP addresses of two bootstrap nodes are returned. When the node receives the IP addresses, it tries to connect to these bootstrap nodes, as shown in the yellow box.

[4]`https://telemetry.polkadot.io/`
[5]`https://linux.die.net/man/1/tshark`
[6]`https://www.wireshark.org/`



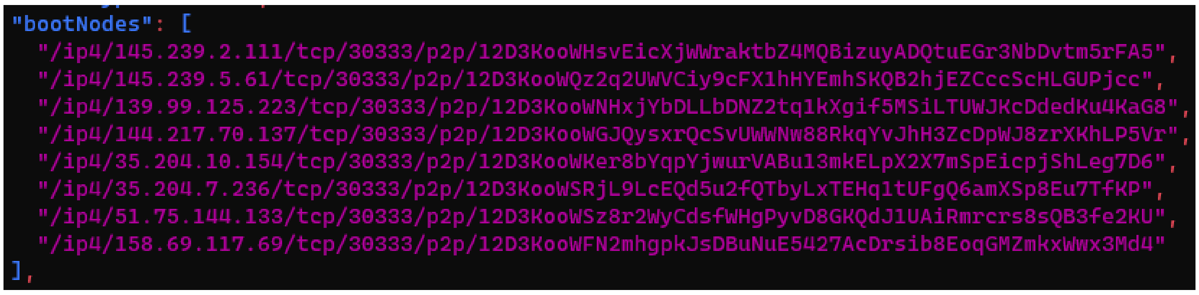
```
"bootNodes": [
  "/dns/p2p.0.polkadot.network/tcp/30333/p2p/12D3KooWHsvEicXjWWraktbZ4MQBizuyADQtuEGr3NbDvtm5rFA5",
  "/dns/p2p.1.polkadot.network/tcp/30333/p2p/12D3KooWQz2q2UWVCiy9cFX1hHYEmhSKQB2hjEZCccScHLGUPjcc",
  "/dns/p2p.2.polkadot.network/tcp/30333/p2p/12D3KooWNHxjYbDLLbDNZ2tq1kXgif5MSiLTUWJKcDdedKu4KaG8",
  "/dns/p2p.3.polkadot.network/tcp/30333/p2p/12D3KooWGJQysxrQcSvUWWNw88RkqYvJhH3ZcDpWJ8zrXKhLP5Vr",
  "/dns/p2p.4.polkadot.network/tcp/30333/p2p/12D3KooWKer8bYqpYjwurVABu13mkELpX2X7mSpEicpjShLeg7D6",
  "/dns/p2p.5.polkadot.network/tcp/30333/p2p/12D3KooWSRjL9LcEQd5u2fQTbyLxTEHq1tUFgQ6amXSp8Eu7TfKP",
  "/dns/cc1-0.parity.tech/tcp/30333/p2p/12D3KooWSz8r2WyCdsfWHgPyvD8GKQdJ1UAiRmrcrs8sQB3fe2KU",
  "/dns/cc1-1.parity.tech/tcp/30333/p2p/12D3KooWFN2mhgpkJsDBuNuE5427AcDrsib8EoqGMZmkxWwx3Md4"
],
```

Figure 4.2: Hard-coded bootstrap nodes in the chain spec file. These nodes are used in the first phase of the node discovery protocol.

While Tor can do DNS queries, some privacy concerns are mentioned in [33]. The Polkadot client also failed to resolve the hostnames of the bootstrap nodes. For these reasons, the decision was made to resolve the hostnames manually in the chain spec file. First, a custom chain spec file was made using `./polkadot build-spec >> custom_spec.json`. This command makes a copy of Polkadot's default chain spec. Secondly, the hostnames had to be resolved, as shown in Figure 4.2. This was done using the python script listed in Listing 1. The result is shown in Figure 4.3. Polkadot can use this custom chain spec file to directly connect to the bootstrap nodes by using the `--chain=custom_spec.json` option, making it possible to use Polkadot with Tor.

```python
import JSON
import socket
import sys

def resolveDNS(filename):
    with open(filename, 'r+') as f:

        spec = json.load(f)
        parsedDNS = []
        for bootnode in spec['bootNodes']:
            segments = bootnode.split('/')
            segments[1] = "ip4"
            try:
                segments[2] = socket.gethostbyname(segments[2])
                address = '/'.join(segments)
                parsedDNS.append(address)
            except:
                print(f'could not resolve host: {segments[2]}')
        spec['bootNodes'] = parsedDNS

        f.seek(0)
        JSON.dump(spec, f, indent=2)
        f.truncate()

if __name__ == '__main__':
    resolveDNS(sys.argv[1])
```

Listing 1: Python script to resolve bootstrap hostnames in a Polkadot chain spec file



```
"bootNodes": [
  "/ip4/145.239.2.111/tcp/30333/p2p/12D3KooWHsvEicXjWWraktbZ4MQBizuyADQtuEGr3NbDvtm5rFA5",
  "/ip4/145.239.5.61/tcp/30333/p2p/12D3KooWQz2q2UWVCiy9cFX1hHYEmhSKQB2hjEZCccScHLGUPjcc",
  "/ip4/139.99.125.223/tcp/30333/p2p/12D3KooWNHxjYbDLLbDNZ2tq1kXgif5MSiLTUWJKcDdedKu4KaG8",
  "/ip4/144.217.70.137/tcp/30333/p2p/12D3KooWGJQysxrQcSvUWWNw88RkqYvJhH3ZcDpWJ8zrXKhLP5Vr",
  "/ip4/35.204.10.154/tcp/30333/p2p/12D3KooWKer8bYqpYjwurVABu13mkELpX2X7mSpEicpjShLeg7D6",
  "/ip4/35.204.7.236/tcp/30333/p2p/12D3KooWSRjL9LcEQd5u2fQTbyLxTEHq1tUFgQ6amXSp8Eu7TfKP",
  "/ip4/51.75.144.133/tcp/30333/p2p/12D3KooWSz8r2WyCdsfWHgPyvD8GKQdJ1UAiRmrcrs8sQB3fe2KU",
  "/ip4/158.69.117.69/tcp/30333/p2p/12D3KooWFN2mhgpkJsDBuNuE5427AcDrsib8EoqGMZmkxWwx3Md4"
],
```

Figure 4.3: Hard-coded bootstrap nodes in the chain spec file. The hostnames are manually resolved to get the IP4 addresses.

Server Specifications

The Polkadot nodes are run on AWS servers with the following specs:

- AWS T3 Large instance
- Ubuntu 20.04.4
- 160 GB GP2 SSD

For the location impact experiment in Section 4.1.2, the following AWS regions are used:

- EU-west-2 (London)
- us-west-1 (N. California)
- ap-northeast-2 (Seoul)

4.2.2 Data Collection

For each experiment, different metrics are analyzed to determine their performance impact. Polkadot uses open-source software for collecting metric data called Prometheus [7]. This data can be analyzed and visualized with a Grafana [8] dashboard. This section explains the use of Prometheus and Grafana isin more detail. For the second set of metrics, custom log files are created and analyzed. Section 4.2.2 discusses the collection of these log files. In Section 4.2.3, the different analyzed metrics are explained.

Prometheus

Prometheus is an open-source monitoring stack designed to collect and store metrics. The metrics are recorded in time series. Prometheus can scrape and combine these metrics from different sources. The metrics can, for example, come from different applications or servers. The metrics are scraped in specific intervals. Polkadot records metrics as Prometheus metrics using a Prometheus exporter[9]. This exporter makes it easy to collect and analyze Polkadot performance data.

Setting up the monitoring process is explained in the Polkadot documentation[10]. Using Prometheus in combination with a VPN took some extra configuration. These steps are explained in Section 4.2.4. The config file used for this thesis is given in Listing 2. The Polkadot Prometheus exporter is exposed by the Substrate framework and is scraped every 15 seconds. Prometheus exposes an endpoint for data querying at port 9090. This endpoint is used for data visualization in the Grafana dashboard.

[7] https://prometheus.io/
[8] https://grafana.com/
[9] https://prometheus.io/docs/instrumenting/exporters/#exporters-and-integrations
[10] https://wiki.polkadot.network/docs/maintain-guides-how-to-monitor-your-node

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: ”Prometheus”
    scrape_interval: 15s
    static_configs:
      - targets: [”127.0.0.1:9090”]
  - job_name: ”substrate_node”
    scrape_interval: 15s
    static_configs:
      - targets: [”127.0.0.1:9615”]
```

Listing 2: Prometheus config file

Grafana

Grafana is a data visualization and analytics framework easily integrated with the Prometheus endpoint. Grafana makes it possible to query, visualize and explore data no matter where it is stored. In this thesis, Grafana is used to analyze the Polkadot performance data.

The following tutorial can be used[11] to set up Grafana with the Polkadot Prometheus source. For network-related metrics, the Substrate Networking dashboard[12] is used. Custom graphs are made for the other metrics. The metrics are explained in more detail in the next section.

Custom Log Files

For each run of the Polkadot nodes, custom logs are created. The logs contain timestamps in 5-minute intervals, and a list of PeerIDs of the connected nodes at that time for each timestamp. The log files are created using the script in Listing 3 that is run every 5 minutes using a cron job. The script queries the node’s `system_unstable_networkState` RPC endpoint using the `substrateinterface` package. This package can be used to interface with a substrate RPC like the Polkadot RPC. The RPC endpoint returns, among other things, a list of the currently connected peers with their peerID. In the script, the peerIDs are extracted and logged with the current time.

[11] https://wiki.polkadot.network/docs/maintain-guides-how-to-monitor-your-node#installing-grafana

[12] https://github.com/paritytech/substrate/blob/master/scripts/ci/monitoring/grafana-dashboards/substrate-networking.json

```python
#! /usr/bin/python3
from substrateinterface import *
import json
from datetime import datetime

def call_rpcs():
    sub = SubstrateInterface(url="http://localhost:9933")
    netstat = sub.rpc_request(method="system_unstable_networkState",
     ↪  params=[])['result']

    connected_peers = netstat['connectedPeers'].keys()
    with open('stats.csv', 'a') as stats_file:
        stats = json.dumps(connected_peers)
        time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        stats_file.write(f'{time}; {stats}\n')

if __name__ == '__main__':
    call_rpcs()
```

Listing 3: Python script to log the currently connected peers. The script is run every 5 minutes using a cron job

4.2.3 Metric Implementation

The performance of the Polkadot nodes is determined based on different metrics. The first set of metrics is gathered by Prometheus and analyzed in the Grafana dashboard. These include the number of discovered nodes, the number of connected peers, the number of Kademlia queries, the disconnection rate, and the request-response time (used as a latency metric). The second set of metrics is gathered by analyzing the custom log files. These metrics include the total number of connected nodes and the average connection time. In the following sections, the implementation of each metric is discussed.

Latency of the Network Communication

The network layer latency is based on the request answer time of the synchronization protocol (see Section 2.2.4 for the explanation of protocols). This protocol uses the /dot/sync/2 substream to share blocks between nodes. This metric is used as an approximation of the network layer latency. The request answer time is round trip time plus request processing time. Round trip time is the time it takes to send and receive a packet over the internet. The request processing time is, on average, approximately the same for each set of connected peers. So the difference in latency between the nodes and network environments is mainly influenced by the round trip time. The median and 99th percentile of the network latency are analyzed. This metric shows the upper time limit for 50% and 99% of the requests. This means that for the 99th percentile, 99% of the requests receive an answer before this time. These metrics help gain insights into a node's average and upper-limit network latency without the 1% most extreme outliers.

The Polkadot node stores the response time of every successful request in a histogram metric type[13]. The values are stored in different buckets. For the response time metric, the values are distributed over 16 buckets. The lowest bucket is 0.001 seconds, and each following bucket exponentially increases with a factor of 2. Each bucket corresponds with a time interval; The first bucket covers 0 to 1 ms, the second bucket covers 1ms to 2ms and the third covers 2ms to 4ms.

The Prometheus queries used to calculate the median and 99th percentile are given in Listing 5. The queries request the values in 5-minute intervals. Then the `histogram_quantile` function calculates the median or 99th percentile of the response times in the interval. For a deeper explanation of the

[13] `https://prometheus.io/docs/concepts/metric_types/#histogram`

query, see the `histogram_quantile` documentation[14]. The queries return a time series with a single value for each 5-minute interval. To gain insight into the latency of the full run, the average of these values is calculated. So the average of the 5-minute interval medians and 99th percentiles.

Number of Connected Peers over Time

The Polkadot node keeps a list of the peers that are connected. Every 15 seconds, Prometheus stores the number of connected peers. The Prometheus query used in the Grafana dashboard is given in Listing 5.

Number of Discovered Peers

The Polkadot node keeps a list of the peers that it has discovered. This list consists of all the peers that established a connection, attempted to establish a connection, or were discovered during the node discovery. Prometheus stores the number of discovered nodes every 15 seconds. The Prometheus query used in the Grafana dashboard is given in Listing 5.

Disconnection Rate

A Polkadot node can have different reasons for disconnecting either an incoming or outgoing connection. Polkadot keeps a Prometheus counter that stores the number of disconnects together with a disconnect reason and the direction of the connection (either incoming or outgoing). The following reasons can cause a disconnect:

- transport-error: An error occurred on the transport[15] layer
- ping-timeout: The connected peer did not respond in time to a ping request
- sync-notifications-clogged: Channel of synchronous notifications is full
- protocol-error: A protocol level error
- keep-alive-timeout: The connection keep-alive timeout expired

Prometheus stores the state of the counter every 15 seconds. In the Grafana dashboard, the hourly disconnection rate is calculated by calculating the disconnection rate per second and multiplying it by 3600. The per second rate is calculated using the rate[16] function. The rate function calculates the average rate per second over a certain rolling time window. So if the window is an hour, the value at 10:00 is the average rate per second over the time window from 9:00 to 10:00. The Prometheus query used in the Grafana dashboard is given in Listing 5.

Throughput

The Polkadot node stores both the total number of inbound and outbound bytes in two counters. The node obtains the values using the libp2p `BandwidthSinks` struct that returns the number of inbound and outbound bytes that passed through all the opened connections. Prometheus stores the state of the counters every 15 seconds. In the Grafana dashboard, the throughput per second is calculated using the rate function with a window of an hour. The Prometheus query used in the Grafana dashboard is given in Listing 5.

Number of Kademlia Queries

The Polkadot node keeps a Prometheus counter for the number of Kademlia lookups done. Prometheus stores the state of the counter every 15 seconds. As with the disconnection rate, the number of Kademlia queries per hour is calculated by calculating the number of queries per second and multiplying it by

[14] `https://prometheus.io/docs/prometheus/latest/querying/functions/#histogram_quantile`
[15] `https://docs.rs/libp2p/0.12.0/libp2p/trait.Transport.html`
[16] `https://prometheus.io/docs/prometheus/latest/querying/functions/#rate`

3600. The rate window is an hour. The Prometheus query used in the Grafana dashboard is given in Listing 5.

Total Number of Connected Peers

As described in Section 4.2.2, custom logs are created for the Polkadot node. The logs store a time series of the set of connected peers in 5-minute intervals. This time series can be used to determine the distinct peers that the node was connected to during the run time by unioning each of the sets.

Average Connection Time

As with the total number of connected peers, the custom logs are used to calculate the average connection time. The logs contain the PeerIDs of the connected nodes in 5-minute intervals. These logs are used to calculate the connection times for each peer using their PeerID. If a set contains a peerID, the connection to the peer is considered open. The connection is considered closed if one of the following sets does not contain the peerID anymore. Using the timestamps, the time between these two events can be calculated. The average connection time can be calculated using all the connection times for all the peers. As the connected peer sets are only collected once every 5 minutes, the connection times are not exact. The connection times can have an error of $\pm 5m$. A connection time of 5m is used when no succeeding sets contain the peerID. A peerID can have multiple connections in the log file.

4.2.4 VPN Node

Implementing a Polkadot node behind a VPN takes some steps. In this section, these steps are explained.

The VPN used in the experiments is personally owned, so no VPN providers are used. The VPN server is hosted at linode.com using the 1GB Nanode instance. The instance runs in Frankfurt. As only one VPN is used for all experiments, the VPN node has a higher latency for the American and Asian Polkadot nodes than the European node. This impact is taken into account during the data analysis. The following section explains the implementation of a Wireguard VPN.

Wireguard

For the installation of Wireguard, an install script[17] is used. This script installs and configures a basic Wireguard VPN and creates client configurations. This thesis uses a Star network topology, where all Polkadot nodes connect to the VPN server. Each client has a pre-shared key[18] with the server that is used for data encryption. The file is copied to the client's server when the client configuration settings are generated. The Wireguard software can then be used to connect to the VPN server. A secure tunnel is set up between the Polkadot server and the VPN server. All data communication between these servers is sent through the tunnel.

The default configuration of Wireguard poses some problems, as all network outgoing network data of the Polkadot server is sent through the VPN. Directly connecting to the Polkadot server via ssh becomes impossible with the default configuration, as all returning ssh network traffic is incorrectly sent via the VPN. This makes it impossible to set up an ssh connection. Also, querying the Prometheus endpoint is impossible. The Grafana software cannot connect to the Prometheus endpoint because a TCP connection cannot be set up. A sketch of the problem is given in Figure 4.4.

[17]https://github.com/Nyr/wireguard-install
[18]https://www.wireguard.com/protocol/#primitives

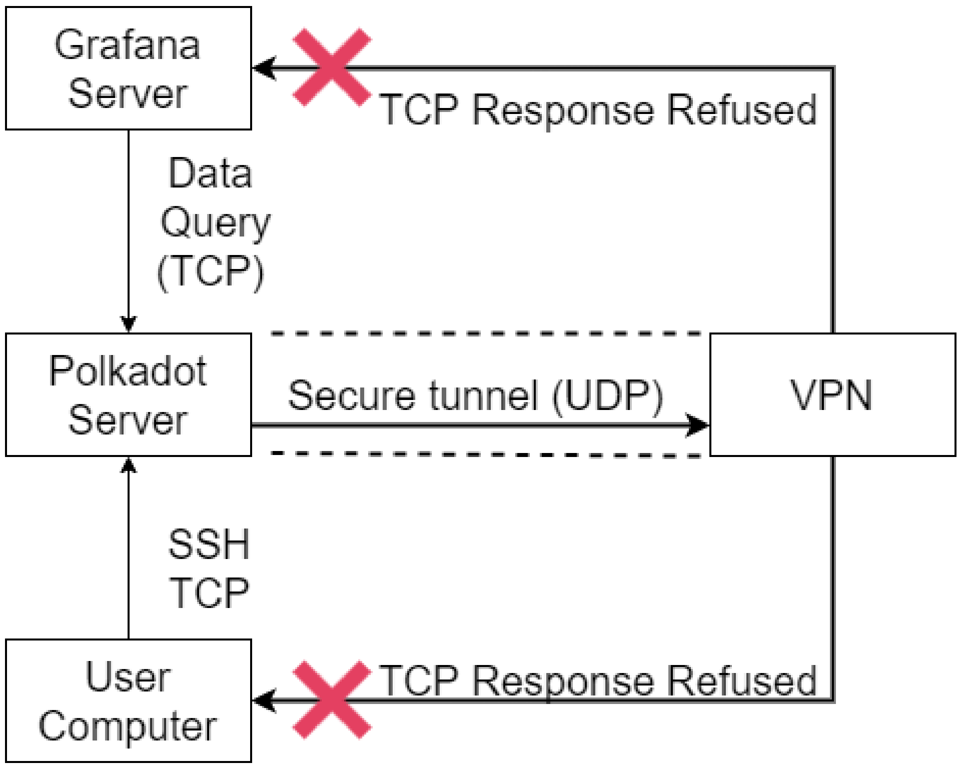


Figure 4.4: Grafana and SSH connection problem when using the default VPN settings.

This problem is solved by editing the default Wireguard configuration to send the SSH and Grafana query network data directly over the internet. For this network data, the VPN is not used. This configuration is accomplished by editing the Linux routing table configurations. These changes are applied when the VPN connection is started. When the VPN connection is disconnected, the changes to the routing table are reverted. The changes are given in Listing 4. All traffic with source port 22 (SSH) or 9090 (Prometheus endpoint) is routed to the ens5 network interface, thereby routing it directly to the internet instead of routing it via the wg0 network interface (Wireguard tunnel).

```
PostUp = ip route add default via 172.31.0.1 dev ens5 table ssh
PostUp = ip rule add fwmark 0x2 table ssh
PostUp = /sbin/iptables -A OUTPUT -t mangle -m multiport -o wg0 -p tcp
 ↪  --sports 22,9090 -j MARK --set-mark 2

PreDown = /sbin/iptables -D OUTPUT -t mangle -m multiport -o wg0 -p tcp
 ↪  --sports 22,9090 -j MARK --set-mark 2
PreDown = ip rule del fwmark 0x2 table ssh
PreDown = ip route del default via 172.31.0.1 dev ens5 table ssh
```

Listing 4: Wireguard configuration settings for sending ssh and Prometheus endpoint data directly instead of via the VPN. PostUp commands are run after the VPN connection is set up. PreDown commands are run before the VPN connection is brought down

4.2.5 Tor

This section explains the implementation of a Polkadot Node with Tor. The TorGhost[19] script is used to implement a Tor environment in Linux. TorGhost manages the installation and configuration of Tor on a Linux system. It configures the IP routing tables to force all network traffic through Tor. It also prevents DNS leaks by sending DNS requests through Tor. Lastly, it blocks all network traffic that can not be sent

[19]https://github.com/SusmithKrishnan/torghost

through Tor, such as UDP packets. A Tor daemon is easily started and stopped using `torghost -s` and `torghost -x`. It is important to use the Polkadot settings, as explained in Sections 4.2.1 and 4.2.1, to make Polkadot compatible with Tor as it disables Polkadot's UDP traffic.

Chapter 5

Results

This chapter discusses the results of the experiments mentioned in Section 4.2. Table 5.1 shows the run times of the experiments for each environment. The VPN environment experiment ran for four days instead of three, as it ran through the weekend. The Tor experiment was run in October, as the experiment in September had some configuration errors, making the results unusable.

5.1 Metric Analysis

The different Polkadot metrics, described in Section 4.2.3, are analyzed for each network environment. Multiple experiments are done as described in Section 4.1 to evaluate the performance of each of the environments. The normal, the Tor, and the VPN network environments are each tested in three data center locations to take into account the impact of the node's location. The figures in this chapter only show the metric results for the European node in the normal, Tor, and VPN environments. The results for each location per environment are listed in Appendix A.

5.1.1 Measuring Network Latency

This section discusses the latency results for each of the environments. The design and implementation of the latency metric are discussed in Sections 4.1.3 and 4.2.3. The results for the European node are shown in Figure 5.1.

Network Latency: Normal

The latency measurement of the nodes in the normal environment can be used as a benchmark for the other environments. The results are shown in Figure A.1. There are some interesting differences in latency between the different nodes. The median latency of the European node is noticeably lower than the other two nodes. The node has an average response time of 60ms.

In contrast, the North American and Asian nodes receive responses at an average of 310 and 380 ms. This could indicate that most of the Polkadot nodes are in the European regions. This corresponds with the location distribution estimation described in Section 4.1.2 based on Polkadot's telemetry service. However, the 99th percentile average of the European node and the US node is nearly the same at

Environment	Start Time	End Time
Normal	5/sep/2022 9:30	8/sep/2022 9:30
VPN	8/sep/2022 10:30	12/sep/2022 8:30
Tor	7/oct/2022 8:00	10/oct2022 8:00

Table 5.1: Run times of the experiments.

728ms and 741ms. The Asian node has an average 99th percentile of 1.28 seconds, indicating that the node is physically located near the edge of the network.

Network Latency: Tor

The network latency for the Tor nodes is noticeably different from the normal nodes. The average median latency of the European node is 509ms. The North American and Asian node reaches 1.37s and 1.17s. It is important to note that the North American node had to download a part of the blockchain on startup (the other two nodes had synchronized the blockchain during a previous small experiment), so the latency was way higher in the first hours, impacting the averages. On the time frame excluding the downloading hours, it has an average median request answer time of 1.15, matching the time of the Asian node. During this time, the North American node functioned the same as the other two nodes (only synchronizing instead of downloading)

As shown in Table 5.2, the chance to select Tor nodes in Europe for a Tor circuit is considerably higher than in other regions. This impacts the latency between the Polkadot node and the Tor guard node. On average, the Asian and North American nodes have a higher latency to the guard node than the European node. This results in a higher median and 99th percentile latency for the North American and Asian nodes. All three nodes have a noticeably high 99th-percentile network latency. The European, North American, and Asian nodes reach an average of 6.04s, 8.15s (excluding the downloading hours), and 7.62s 99th percentile latency, respectively.

A latency above 6 seconds posts problems as the block creation time is 6 seconds. This could mean that validator nodes do not receive a previously created block in time before they have to submit a new block on top of it. The experiments in the BABE paper[2] show that the BABE algorithm should be resistant against 2.796 seconds network delay if 85% of the validators have a lower network delay. This would mean that validators behind the Tor network would cause potential security problems for the BABE algorithm. Besides, validators could miss their block creation time slot. As a punishment, they could get kicked out of the active validator set, or part of their stake could be slashed. A high latency could also have an impact on the grandpa consensus protocol. If too many protocol messages are delayed by a couple of seconds, it impacts voting. The validator nodes could have a long delay in reaching a consensus on previous blocks. This delay could result in an added delay to the block production protocol, impacting the blockchain performance even more.

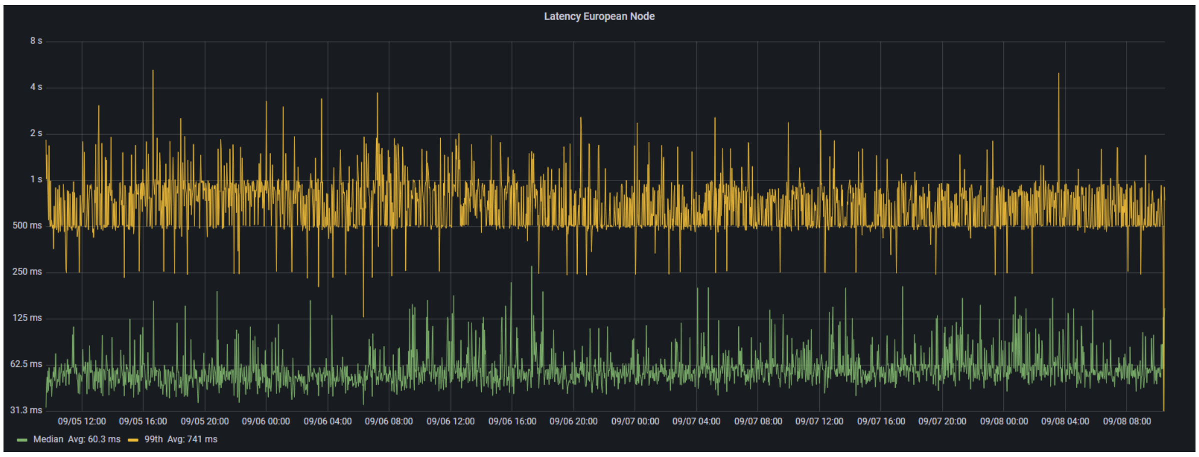
For full nodes, high latency is not that big of a problem if they belong to a small set of full nodes with high network latency. Other nodes with lower latency will deliver important protocol messages in time. However, a large set of full nodes with high latency would impact the overall network performance and could cause the abovementioned problems. Active validators with a high network latency would seriously impact the network performance and security as they actively participate in the consensus algorithm.

Network Latency: VPN

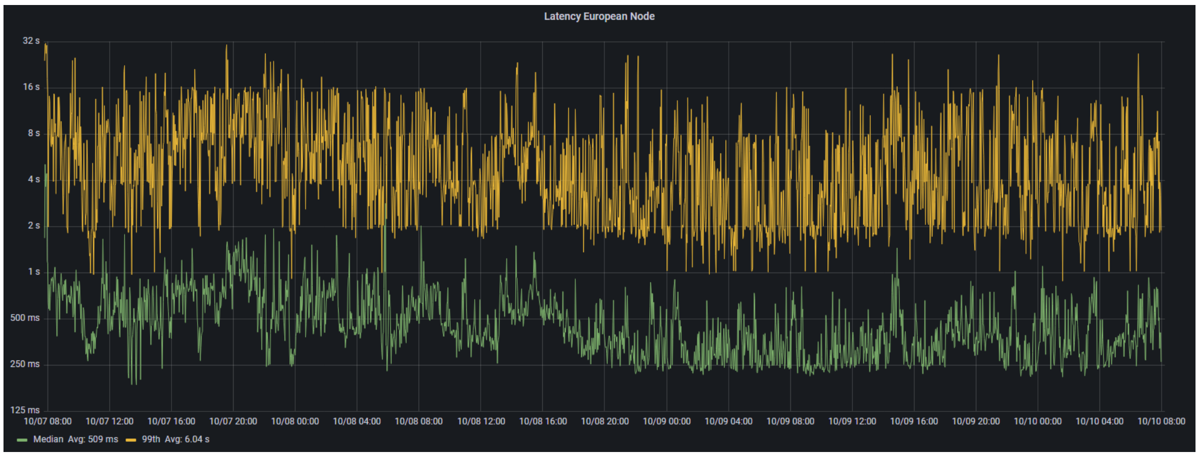
The nodes in the VPN environment perform considerably better in terms of network latency than those in the Tor environment. The North American and Asian nodes have an average median network latency of 380ms and 699ms. The European node even has an average median network latency of 56ms. It reaches a lower latency than the European node in the normal environment.

This result was unexpected. The European node should have an increased latency as all traffic has an extra hop through the VPN. A possible explanation could be that the nodes in the VPN environment eventually only connect to nodes with fast and stable network connectivity as they only establish outgoing connections, while the nodes in the normal environment receive mostly incoming connections. These incoming connections could be poorer, thus increasing the latency. This may explain the nearly identical network latency for the North American node as well.

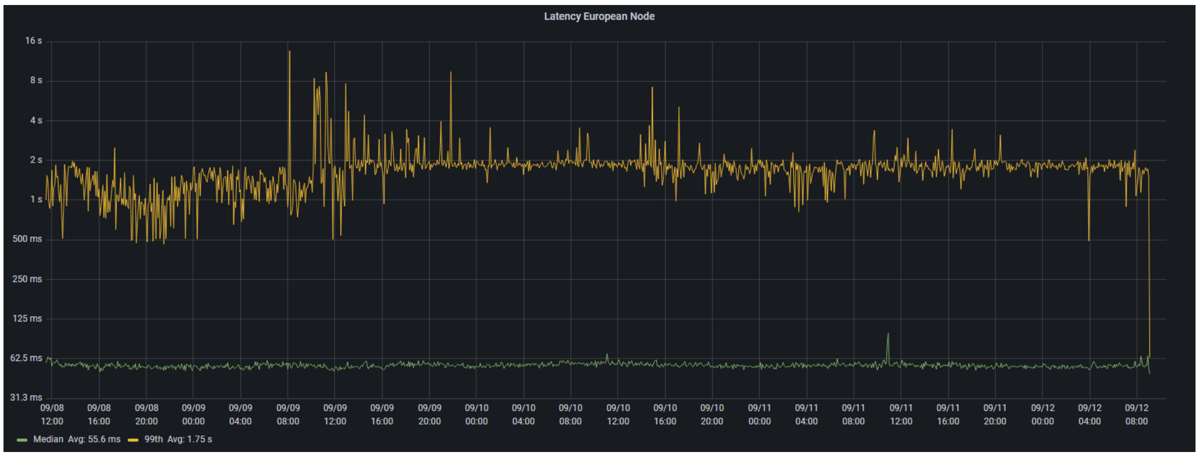
Only the Asian node performs poorer than the Asian node in the normal environment, nearly dou-



(a) Normal Environment



(b) Tor Environment



(c) VPN Environment

Figure 5.1: Request answer time for block synchronization requests for the European node in the normal, Tor, and VPN environments. Shown are the 50% and 99% percentiles.

bling the average median network latency. The impact of the latency between the node and the VPN is clearly visible.

The VPN has a more severe impact on the average 99th percentile latency, increasing it between 0.8 and 1.2 seconds. The Asian node has, with an average 99th percentile of 2.11 seconds, the highest latency but is still within limits of the BABE security experiments[2].

5.1.2 Measuring the Number of Connected Peers

This section discusses the connected peers results for each of the environments. The design and implementation of the metric are discussed in Sections 4.1.3 and 4.2.3. The results for the European node are shown in Figure 5.2.

Connected Peers: Normal

Figure A.4 shows that a node in a normal network environment easily connects to the maximum number of peers and consistently keeps a connection to 50 peers. The European-based node takes approximately 30 minutes to reach the maximum number of peers, while the American and Asian-based nodes take around 1 hour and 2 hours. In a second run for the three nodes, they all take approximately 2 hours to reach the 50 peers limit. So the differences are not considerable, and there is likely just variance in this data.

Connected Peers: Tor

The number of connected peers for the nodes in the Tor network environment differs from the nodes in the normal network environment. Figure A.5 shows the number of connected peers in the Tor environment. All three nodes cannot reach the maximum number of connected peers. The European node starts with around 12 peers and then slowly climbs to around 25 peers and keeps oscillating around 25 peers. The American node oscillates around 16 peers, and the Asian node only around 13. The difference between the average peers of the European node and the Asian node is considerably large. The number of connected peers is also not stable, where each node shows drops and increases of 20 peers in just a couple of hours while the nodes in the normal environment stay at 50 peers. Unstable peers make a node with a Tor network environment potentially vulnerable to Eclipse attacks, further discussed in Section 7.1.2.

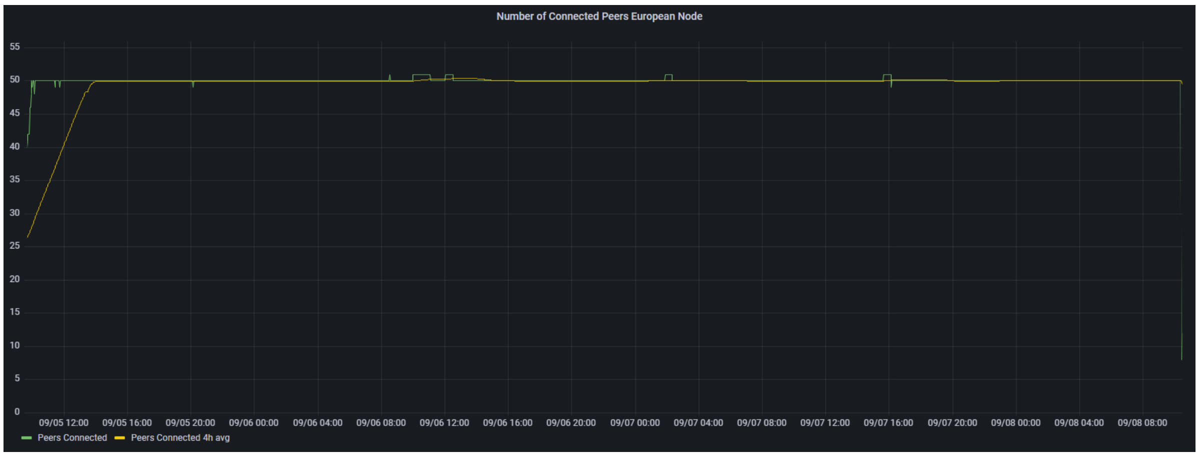
The difference between the number of connected nodes of the European and the Asian node may be due to the distribution of Tor nodes. Table 5.2 shows the top ten countries in terms of the bandwidth of the Tor nodes in those countries [1]. It also shows the chance of selecting a guard, middle, or exit node hosted in these countries. Nine of the ten countries are in Europe, and one is in North America. The chance that the Asian Polkadot node selects a guard node in one of these ten countries is 84.43%. Therefore the Asian node has a higher latency to the guard node than the European node, which can affect the established connections with peers.

Another critical factor in the number of connected nodes could be that the node in the Tor environment cannot receive incoming connections. Peers send connection requests to the nodes' known addresses, which are all Tor exit nodes. These exit nodes do not know where to route these requests, so the Polkadot node never receives them.

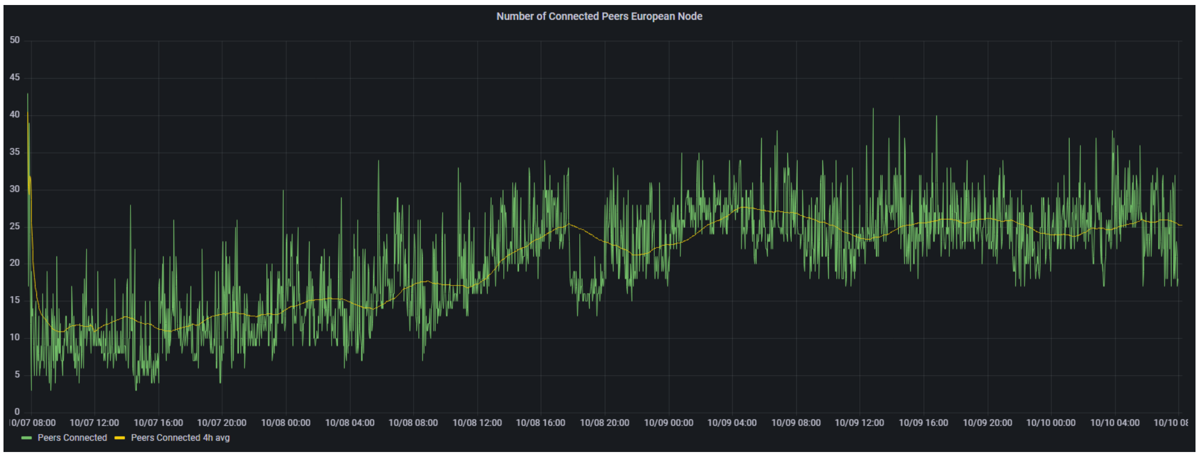
Connected Peers: VPN

The node in the VPN environment cannot reach the 50 peers threshold. In this current setup, the VPN cannot relay incoming connections, as the VPN does not know to which node to relay the connection requests. All three nodes seem to converge to around 40 connected peers, as shown in Figure A.6. The European and Asian nodes have noticeable drops in the number of connected nodes during the

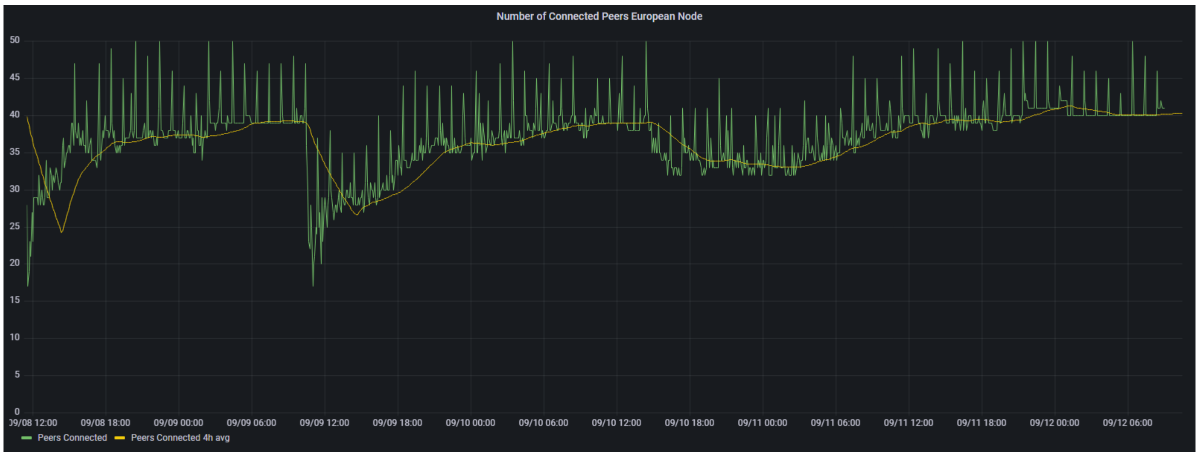
[1] `https://metrics.torproject.org/`



(a) Normal Environment



(b) Tor Environment



(c) VPN Environment

Figure 5.2: Number of connected peers for the European node in the normal, Tor, and VPN environments.

Country	Guard Probability	Middle Probability	Exit Probability
Germany	33.1631%	33.3643%	43.8361%
United States	16.6772%	13.8090%	9.0311%
Netherlands	6.4683%	10.1296%	12.5572%
France	8.1660%	6.7842%	1.7443%
Finland	7.0394%	5.2613%	0.4410%
Poland	3.9443%	3.9432%	2.8283%
Switzerland	2.3110%	3.4722%	2.7658%
Sweden	2.6754%	2.5382%	3.0988%
United Kingdom	2.7689%	3.3432%	1.1617%
Norway	1.2208%	0.9810%	4.9199%
Top 10 Total	84.4344%	83.6262%	82.3842%

Table 5.2: Top ten countries running Tor node in terms of bandwidth. The chances of selecting a guard, middle, or exit node for each country are shown.

run time. These drops are at different moments in time. From the logs, it is unclear why these drops in the number of connected nodes occur. It looks like these events are not correlated with considerable changes in the number of connected peers of the other nodes, so the possibility that there were VPN-wide connection errors can be ruled out.

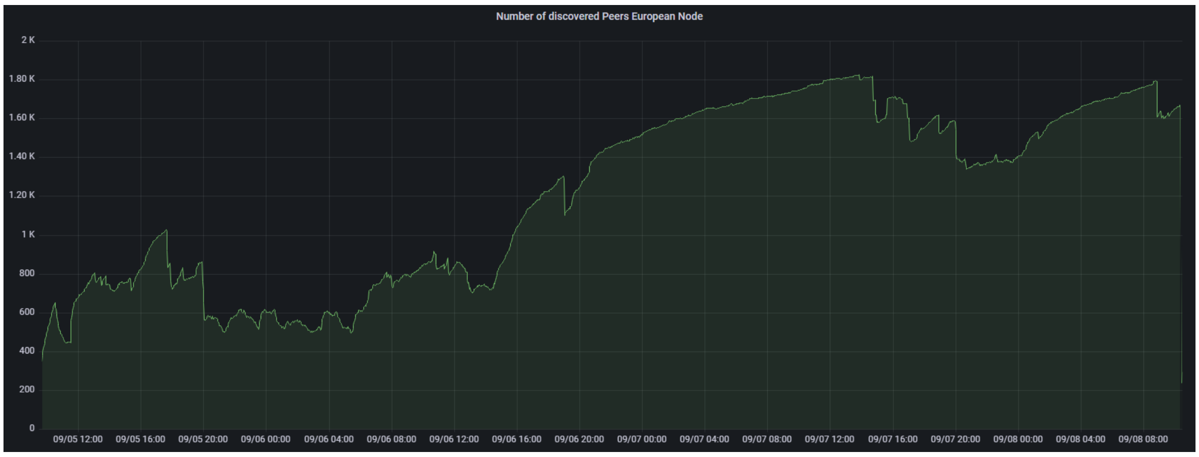
5.1.3 Measuring the Number of Discovered Peers

This section discusses the discovered peers results for each of the environments. The design and implementation of the metric are discussed in Sections 4.1.3 and 4.2.3. Section 2.2.4 describes the node discovery protocols. The results for the European node are shown in Figure 5.3. The number of discovered peers can increase and decrease based on the mentioned protocols. Until the connected peer limit is reached, the node should query connected peers for new peers at a rate of 60 queries per minute.

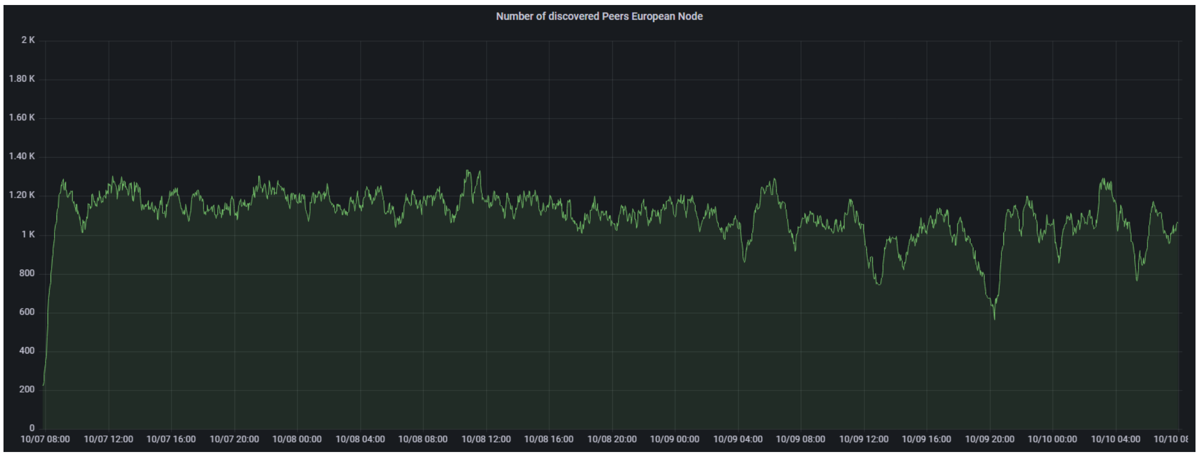
Interesting to note is that all nodes in the different environments never reach 60 queries but converge to 30. This difference is caused by the connection limit and connection timeouts. After each query, the node opens connections to several discovered nodes. It keeps the connections open till they reach a certain timeout. The next Kademlia query is scheduled for that period. However, the connection limit is reached, so the query is not executed. When the connections timeout, they are dropped, and the number of open connections drops below the limit again. So for the next scheduled Kademlia query, the limit is not reached. The number of Kademlia queries per hour converges to around 30 queries because this process keeps repeating.

Discovered Peers: Normal

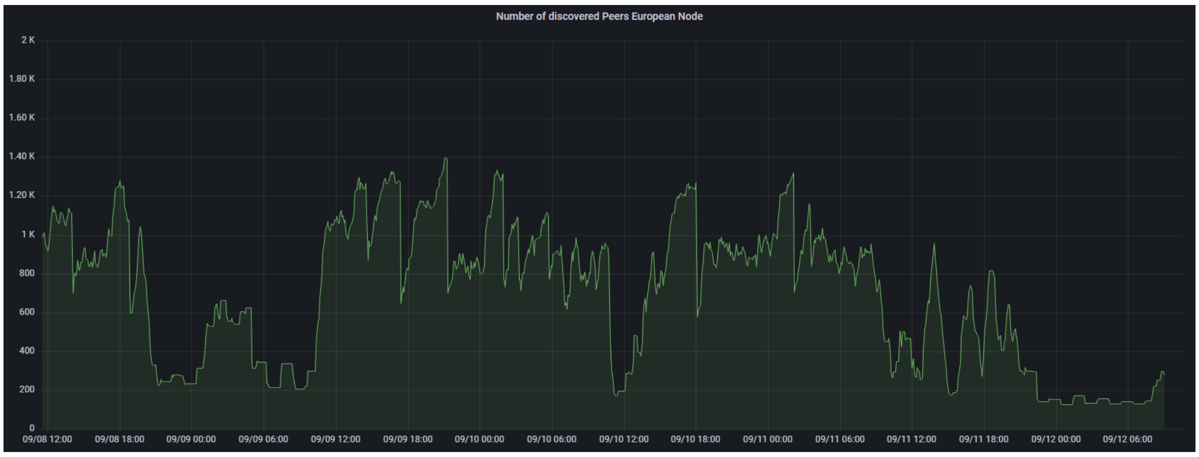
The nodes in the normal environment all have the largest number of discovered nodes compared to the Tor and VPN environment. The Asian node has the most discovered peers, with nearly 2800 nodes at the end of the run time. The European and North American nodes reach around 1800 and 2000 discovered peers, respectively.
An exciting thing to notice is that all three nodes have periods where the number of discovered peers decreases. As shown in Figure A.4, all normal nodes connect to 50 peers in a considerably short time. So the node stops looking for new peers, but the number of discovered peers increases over the rest of the run time. Figure A.19 shows for each point in time the number of Kademlia queries in the previous hour for the normal environment. It shows that each node only does a couple of queries. After that, the number of connections reaches at least 40+ peers, as shown in Figure A.16. The first hour of data is calculated with data from the previous run, causing the steeply decreasing line.



(a) Normal Environment



(b) Tor Environment



(c) VPN Environment

Figure 5.3: The number of discovered peers for the European node in the normal, Tor, and VPN environments.

The only reason that the number of discovered peers keeps increasing is that peers try to connect to the nodes. Figure A.10 shows the number of disconnections per hour recorded for each node in the normal environment. It clearly shows many disconnects for nodes that were not actively searching for peers. The European node has, on average, around 2500 disconnects. The North American and Asian nodes average around 4500 and 6500. All these disconnects correspond to established connections, meaning that nearly all new connections are actively disconnected because the nodes have already reached the maximum connected peers limit.

It is interesting to note that the Asian node has the highest number of discovered peers and connection logs. There is a clear connection between the number of logs and the number of discovered peers. It is less clear why the Asian node receives the most connection requests. It is likely due to some variance in the data. During a succeeding run, the North American node received the most connection requests showing that there is variance in the data.

Discovered Peers: Tor

The number of discovered peers for the Tor environment average around 1100 for each node, as shown in Figure A.8. It is interesting that each node averages around the same number of peers. It looks like the nodes reach between 1000 and 1400 peers as they keep querying for peers and dropping peers simultaneously. The logs show that the nodes sometimes reach the connection limit, so they stop querying for new peers until some connections are lost, dropping below the threshold. The nodes can have different reasons for losing connections, but the main reason is that they drop peers due to timeouts. Querying for peers and disconnecting from peers keeps repeating, showing a steady stream of Kademlia queries in Figure A.20. Only the European node can stay connected long enough to reach the connection limit for a longer period, resulting in drops in the number of Kademlia queries per hour and the number of discovered nodes. As previously described, these results are related, as can be seen in Figures A.8 and A.20.

Discovered Peers: VPN

The number of discovered peers for the VPN environment has the same trajectory in the beginning as the nodes in the Tor environment. The number of discovered peers reaches around 1100 peers. The nodes then manage to keep more open connections than the Tor node, indicated by the more substantial number of peer slots filled, a decreasing number of discovered peers, and the number of Kademlia queries dropping to zero for specific periods.

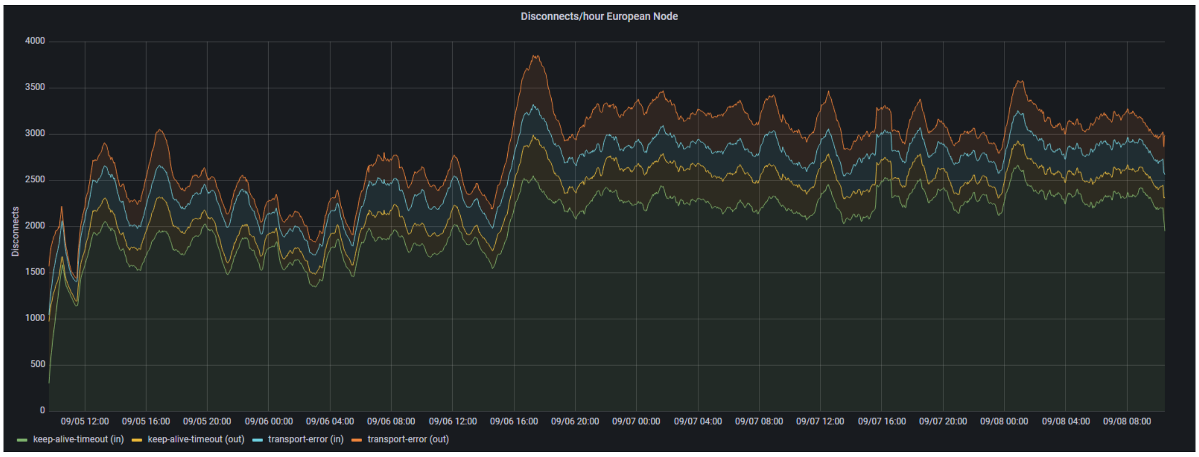
While the nodes in the Tor and normal environment had more than a thousand plus discovered peers over most of the run time, the nodes in the VPN environment had their number of discovered peers drop below 200. This is caused by not having incoming connections and opening enough stable connections to reach the connection limit for Kademlia queries (see Figure A.18). The nodes in the VPN environment have the smallest network view of the three environments.

5.1.4 Measuring the Disconnection Rate

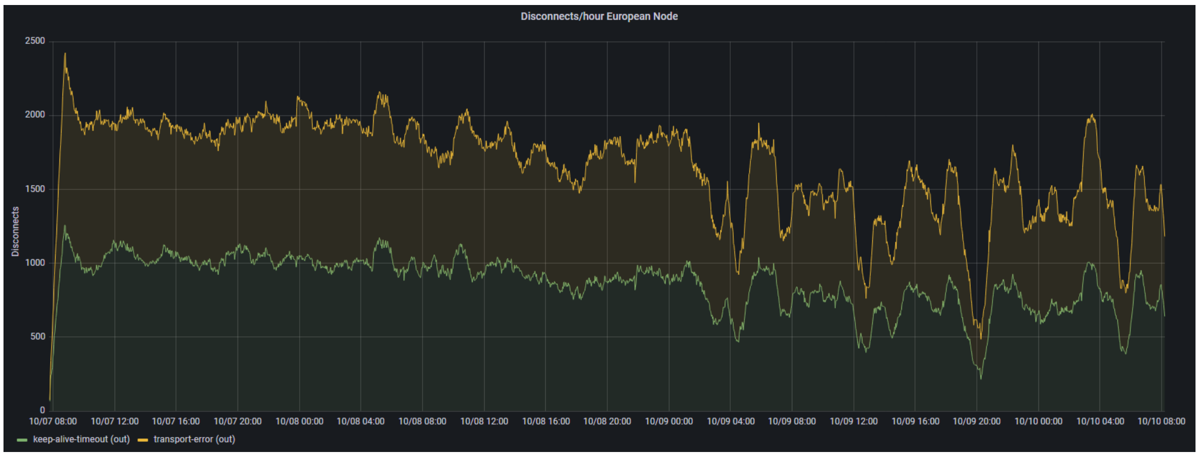
This section discusses the disconnection rate results for each of the environments. The design and implementation of the metric are discussed in Sections 4.1.3 and 4.2.3. The results for the European node are shown in Figure 5.4. The main reasons for disconnecting are keep-alive timeouts and transport layer errors. While there are multiple other errors and timeouts, as discussed in Section 4.1.3, the previously mentioned reasons are the only considerable reasons. Therefore these metrics are the only ones analyzed, and the rest is excluded.

Disconnection Rate: Normal

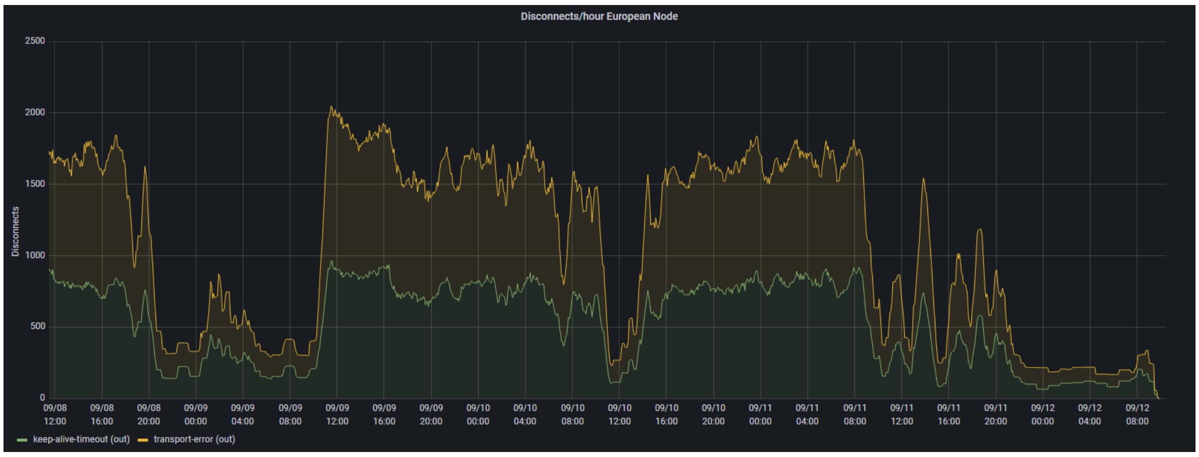
The main disconnect reason for nodes in the normal environment is keep-alive timeouts on incoming connections. A keep-alive timeout is thrown when a connection has been idle for longer than the



(a) Normal Environment



(b) Tor Environment



(c) VPN Environment

Figure 5.4: The hourly disconnection rate for the European node in the normal, Tor, and VPN environments. Shown are the keep-alive-timeout and transport-error reasons for the incoming and outgoing connections.

'KeepAlive' time. As shown in Figure A.16, the nodes in the normal environment have mainly incoming connections, causing most of the keep-alive timeouts to be on incoming connections. Transport errors have a smaller share in the total number of disconnect reasons but are still noticeably common. Transport errors occur when there are errors on the transport layer. Libp2p has multiple layers in the transport protocols, as mentioned in 2.2.4. These layers include a raw TCP connection, an encryption layer, and a multiplexing layer. If there is an error in one of these layers, Polkadot records a transport error in Prometheus.

When comparing the number of disconnects between the different nodes, the European node has considerably fewer disconnects than the other two nodes, with an average of around 2500. The Asian node has an average of 6500 disconnects per hour, and the North American node has around 4500. Figure A.10 shows that the Asian node has an average of 160 open connections, while the European node and North American node have an average of around 80 and 110. So the number of open connections explains a part of the difference in the number of disconnects. More open connects mean more potential disconnect errors.

Interesting to note is the number of incoming transport errors of the European node compared to the Asian and North American nodes. The other nodes have four times as many incoming transport errors as the European node. It is hard to explain why this is happening exactly, but it could have something to do with the distribution of nodes worldwide and the latency between nodes. If most of the Polkadot nodes are located in Europe, network latency between the European node and the other nodes is relatively low if we compare it to the network latency of the Asian and North American nodes. Section 5.1.1 compares the network latency for each of the nodes and shows that the request-response time for the block sync protocol has a median of around 300 and 400 milliseconds for the North American and Asian nodes, while the European node has a median of around 60 milliseconds.

Latency could substantially impact the disconnection rate of nodes, as the Asian node has the highest latency and number of disconnects.

Disconnection Rate: Tor

Nodes in the Tor environment only have outgoing connections, so the only disconnect reasons are outgoing connection errors and timeouts. As with the normal node environment, the keep alive-timeout and transport-error metrics are analyzed as they are the only considerable disconnection reasons. All three nodes in the Tor environment reach around the same number of disconnects per hour. They all have, on average, around 1000 keep-alive timeouts and 1000 transport errors, as shown in Figure A.11. Only the European node has drops in the number of disconnections. These drops correspond with drops in the number of Kademlia queries. Kademlia queries result in new outgoing connections. As the establishment of new connections can result in timeouts and errors, the drops in disconnects positively correlate to the number of Kademlia queries.

Disconnection Rate: VPN

As with the nodes in the Tor environment, the nodes in the VPN environment only have outgoing connections. However, the number of disconnects looks quite different. All three nodes in the VPN environment have periods with a steady set of connections equal to or over the connection limit. This decreases the number of Kademlia queries, as explained in 5.1.3. As with the Tor nodes, a low number of Kademlia queries equals a lower number of disconnects. The disconnection rate is comparable with the Tor environment when the number of connected peers drops below the connection limit.

Interesting to note is that the European node has the most prolonged periods of high disconnection rates, even though the VPN server is in Europe. In a succeeding run, the European node has the least prolonged periods of high disconnection rates, so the results have some variance.

5.1.5 Measuring the Throughput

This section discusses the throughput results for each of the environments. The design and implementation of the metric are discussed in Sections 4.1.3 and 4.2.3. The results for the European node are shown in Figure 5.5.

Throughput: Normal

The nodes in the normal environment all reach around the same amount of throughput. The nodes have an input and output throughput of around 300KB/s, as shown in Figure A.13. Only the European node has a lower input throughput of around 180KB/s. The European node may have the most stable connections, so the least amount of input packet retransmissions. There is no other apparent explanation for these results. The European node's output throughput also increases at a particular time to around 450 KB/s. It is possible that the European node connected to a peer that started downloading the chain. This peer would request more blocks than others, increasing the output upload throughput. The 300KB/s average is taken as a baseline to compare to the other environments.

Throughput: Tor

The throughput for nodes in the Tor environment is considerably lower than the throughput baseline of the nodes in the normal environment. The input and output throughput is for all three nodes nearly the same, but the total throughput fluctuates a lot. All three nodes reach an average lower limit of around 20-30 KB/s and an average upper limit of around 120-150KB/s excluding the peaks.

There is a clear connection between the number of connected peers and the throughput. More connected peers mean more peers to communicate with. This connection is evident when comparing the throughput and the number of connected peers of the European node. When there is an increase in the number of connected peers, there is an increase in the amount of throughput used. The correlations between the throughput and the number of peers are given in Table 5.3. It shows a strong correlation for the European node. However, the correlation is less for the North American and Asian nodes.

So, the lower number of peers does not explain all the difference in throughput. It is also possible that the latency of nodes in the Tor network throttles the throughput. The Asian node has the highest latency to the Tor network. It has the lowest correlation between the number of peers and the throughput.

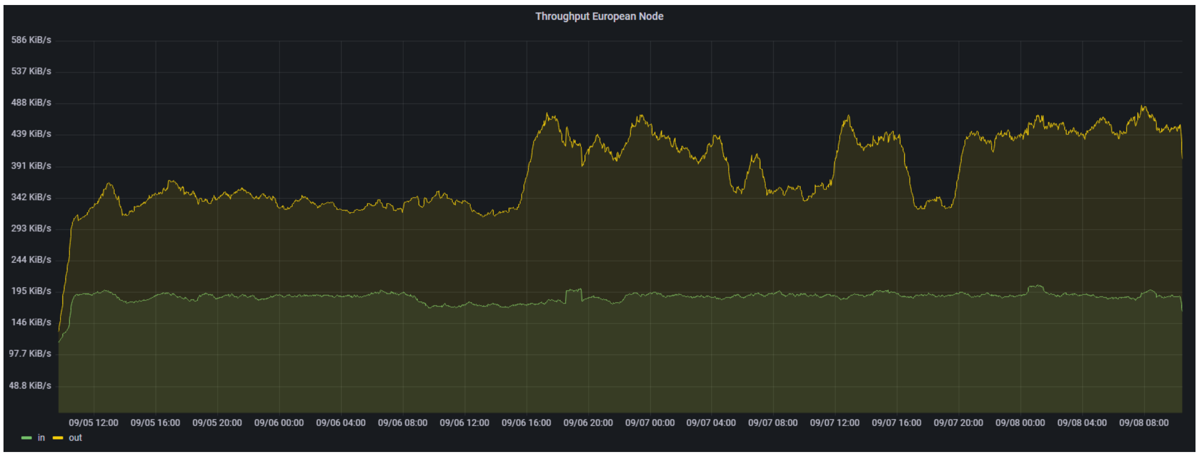
The throughput for the nodes in the normal environment also shows that the number of peers is not the only influence in the fluctuating throughput, as the number of peers for those nodes is stable at 50. Fluctuations in the throughput can also depend on network usage. If the Polkadot blockchain is heavily used, more data is included in each block, increasing the block's size and the throughput used to distribute these blocks.

Node	In/Out	In/Peers	Out/Peers
Asian	0,78	0,64	0,43
European	0,90	0,90	0,81
North American	0,91	0,86	0,76

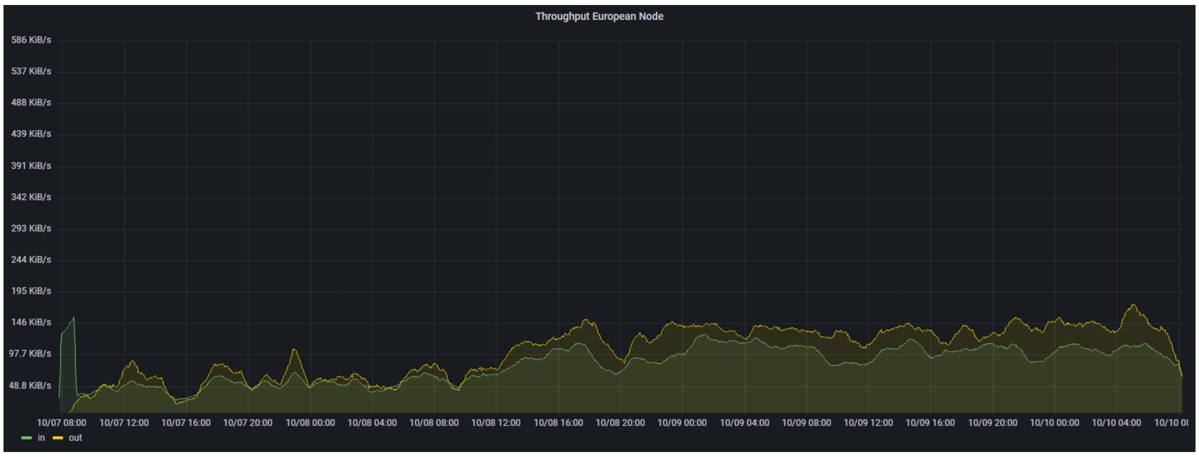
Table 5.3: Correlation between In/out throughput and number of connected peers in the Tor environment

Throughput: VPN

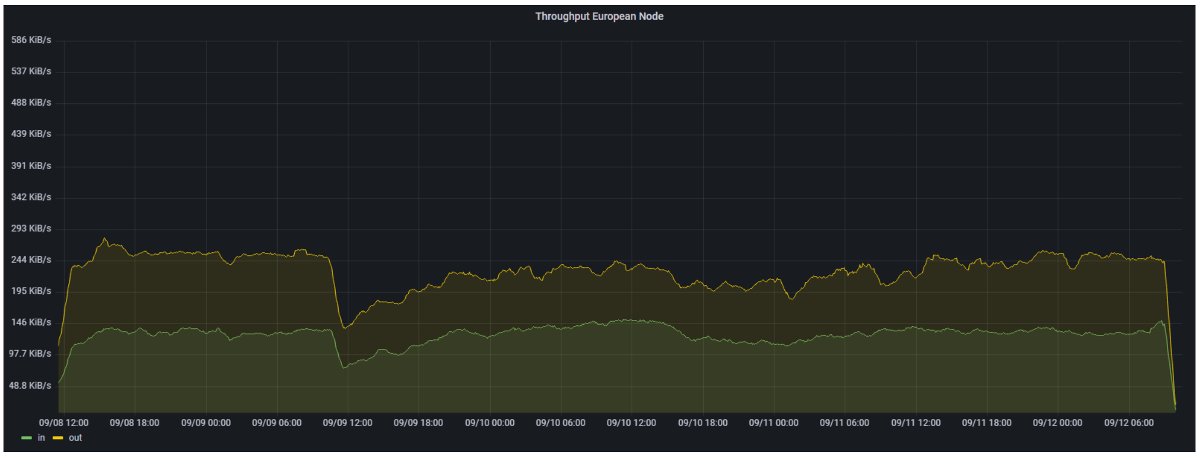
The throughput of the nodes in the VPN environment is slightly lower than that of the nodes in the normal environment. The North American node averages around 200 KB/s input and output throughput. The European node has an average of around 150KB/s input and 250KB/s output throughput, with a noticeable drop in throughput at a particular time. This drop corresponds with a drop in the number of connected peers. The Asian node has an average of around 180KB/s output and 250KB/s throughput. The Asian node also has a noticeable drop in throughput, like the European node, corresponding with



(a) Normal Environment



(b) Tor Environment



(c) VPN Environment

Figure 5.5: Throughput for the European node in the normal, Tor, and VPN environments.

a drop in the number of connected peers. Interesting to note is that the input and output throughputs can differ quite a lot, but the total throughput is around 400 for each node.

The correlation calculations in Table 5.4 show that the input throughput is clearly correlated with the number of peers connected. Especially the Asian node reaches a 0.96 correlation. The output throughput is less correlated. It is unclear why the Asian node has the highest correlation for the throughput with the number of connected peers. However, as with some of the other metrics, there is some variance in the data.

Node	In/Out	In/Peers	Out/Peers
Asian	0,75	0,96	0,76
European	0,66	0,79	0,74
North American	0,69	0,87	0,70

Table 5.4: Correlation between In/out throughput and number of connected peers in the VPN environment

5.1.6 Measuring Total Number of Connected Peers

This section discusses the total number of connected peers for each environment. The design and implementation of the metric are discussed in Sections 4.1.3 and 4.2.3. In Table 5.5, the total number of connections is shown for each network environment and each node location. There is a clear difference between the different network environments. The normal node has the lowest number of connections, while the Tor environment has the highest. The nodes are run for approximately the same time for each environment, so it is clear that the nodes in the Tor environment have connected to considerably more peers. This is expected when looking at the disconnection rate and the number of Kademlia queries. These nodes cannot keep long-lasting connections and never reach the connection limit. So they keep looking for new peers. The nodes in the normal environment manage to establish more long-lasting connections and eventually connect in total to one-tenth of the peers the nodes in the Tor environment connect. As previously mentioned, this could be an important factor for an Eclipse attack.

	Asian	European	North American
Normal	268	165	171
Tor	1407	1306	1293
VPN	501	540	427

Table 5.5: The total number of connections for each node in every environment.

5.1.7 Measuring Average Connection Time

This section discusses the average connection time of the peers for each of the environments. The design and implementation of the metric are discussed in Sections 4.1.3 and 4.2.3. Table 5.6 shows the average connection time for each environment and location.

The results show a substantial difference between the Tor environment and the other environments. The Tor nodes have an average connection time of between 10 and 20 minutes. The VPN and normal environments reach 3-5 hours and 8-11 hours, respectively.

	Asian	European	North American
Normal	8:55:19	10:41:49	8:04:53
Tor	0:11:59	0:19:55	0:14:59
VPN	4:25:01	3:03:08	5:02:17

Table 5.6: The average connection time for each node in every environment.

Chapter 6

Privacy

6.1 Threat model

A threat model is defined to analyze the privacy and security implications of using Tor or a VPN with Polkadot. The threat model is based on the threat model used by the Tor network[1]. Our threat model assumes the following:

- The Tor network cannot defend against a global passive adversary. So in this thesis, the assumption is made that there is no global passive adversary.
- An adversary may control one or multiple guard and exit nodes.
- An adversary may control one or multiple Polkadot nodes.
- An adversary may have control of the LAN or ISP
- An adversary may inject malicious content using their malicious Polkadot nodes.
- An adversary may reject or block connections to their controlled Tor nodes or Polkadot nodes
- An adversary may know the PeerID of the target Polkadot node and can identify a node using the PeerID on the application layer.
- An adversary may drop packets between two non-malicious Polkadot nodes if they control an element on the connection path.
- An adversary cannot alter the communication between two non-malicious Polkadot nodes as the nodes encrypt and authenticate their communication.

6.2 VPN

This section investigates the privacy aspects of a Polkadot node in a VPN environment. There are multiple ways to use a VPN. Most VPN users use a paid or free VPN service. This is the easiest way to use a VPN server. However, there are some serious privacy concerns. These VPN servers are not self-hosted, so the user is not in control of the information being stored on these servers. Most free and many paid VPN servers store user information, such as IP addresses, device information, connection time, and visited websites. Adversaries can obtain this information by compromising the server or by buying the information from the VPN providers [34]. Even months after connecting to a VPN, users could be deanonymized.

A way to prevent a VPN from storing information is by self-hosting one. Self-hosting a VPN server has some advantages compared to VPN servers hosted by a VPN provider. With a self-hosted VPN server, a user can choose the level of performance. The user can also determine what level of logging

[1]`https://2019.www.torproject.org/projects/torbrowser/design/#adversary`

the VPN server should use. Moreover, the user is responsible for the security of the VPN server. This also introduces a disadvantage to running a self-hosted VPN server. Running it requires quite some technical knowledge, and only some people can set it up.

A self-hosted VPN Server does not offer a high level of privacy. First, the VPN server user could be identified by the payments to the hosting provider. These payments could be subpoenaed or be sold by the hosting provider [34]. The privacy of the user also depends on how the server is used. If they are the sole user of the server, their origin client IP address is easily identified by an ISP or law enforcement agency. Also, a hosting provider could be asked to supply logs of the network traffic from and to the server, making it easy to identify the origin of the network traffic. However, for regular users, it could be challenging to determine the origin of the client behind a VPN, so against basic adversaries, a VPN could offer some level of privacy.

This thesis uses a single self-hosted VPN to connect the Polkadot nodes to the rest of the network. Peers do only know the IP address of the VPN server. The VPN offers some level of privacy against basic adversaries, like other Polkadot node operators. Nevertheless, if privacy against governments, ISP operators, or law-enforcement agencies is required, a VPN is not a secure method to anonymize a node's network traffic.

6.3 Tor

In this thesis, Tor is used as an anonymization method to anonymize the network traffic of a Polkadot node. As described in Section 2.4, Tor helps anonymize users' network traffic by routing it through multiple nodes. Each network packet is first sent through the Tor network before reaching a peer. Each Tor node only knows about the previous and next hop in the path, so none of the Tor nodes in the path knows the source and the destination of the network packet. Only the guard node knows the source, and only the exit node knows the destination. This makes it hard for adversaries to find the Polkadot node's IP address.

As described in Section 2.4.6, there are different possible attacks to deanonymize Tor clients. The two types of attacks most relevant to this thesis are Entry and Exit Router attacks and Other-Resourced attacks. In both types of attacks, the adversary tries to determine a link between a Tor user and a destination server. In this case, it would be to determine a link between a Polkadot node and its peers. If an adversary can establish a link, it can determine the Polkadot node's IP address and possibly deanonymize the user using the node.

As described in Section 2.4.6, these attacks require considerable resources or access to global networking information. Only global adversaries could probably have access to global networking information. As described in the threat model (Section 6.1), global adversaries are not considered for this Thesis. Using Tor as an anonymization tool against normal adversaries could be effective. However, certain negative interactions between Polkadot and Tor could impact the effectiveness of these attacks.

Polkadot uses a PeerId to identify nodes instead of an IP address. This uncouples a Polkadot node's identification from its network location. So a Polkadot node could move to a new location without losing its identity and reputation in the network. All encrypted network communication between nodes is based on this PeerId, and nodes are identified based on the PeerId. While this is a useful feature, it negatively interacts with the Tor network. Because nodes are identified by their PeerId, their network traffic is easily tracked over multiple Tor exit nodes. The Polkadot node announces the different exit nodes it is using in Polkadot's identity protocol. This information could be leveraged to increase the effectiveness of certain correlation attacks. An adversary only has to be connected to the Polkadot node with a single peer to receive a list of 30 (the complete list is truncated to max 30) of the Tor exit nodes the node is using. The adversary knows if the Polkadot node has a connection through one of their exit nodes if the IP address of the exit node appears in the list. So the peerID helps an attacker determine if the target is connected through one of their exit nodes. To perform an Entry and Exit Router attack, the adversary still has to control the Polkadot node's guard node. This is hard as the Tor network currently has nearly 2000 guard nodes.

Another negative interaction between Polkadot and Tor is the ports that Polkadot uses and the Tor Exit policy. Tor exit node operators can specify specific exit policies. They specify which ports can be used with their Tor exit node. Many Tor exit nodes have a reduced exit policy. They only allow some ports to prevent malicious usage of the Tor network. Polkadot uses port 30333 to establish connections between peers. Port 30333 is not included in the reduced exit policy. This limits the number of exit nodes that a Polkadot node can use. This negative interaction can benefit an adversary. By running exit nodes that allow port 30333, they increase the chance of being chosen for a circuit for the Polkadot node. This can increase the effectiveness of Entry and Exit Router attacks. Details about all Tor exit nodes can be gathered using onionoo[2]. From the nearly 1600 exit nodes, only 270 allow port 30333, making it nearly six times more likely that an adversary can control the exit node of a Polkadot node using Tor. Nevertheless, the adversary still has to control the Polkadot node's guard node to use Entry and Exit Router attacks. The last possible negative interaction between Polkadot and Tor is the predictability of the network traffic. Most of the traffic of a Polkadot node is related to the consensus algorithm and block synchronization. The consensus algorithm has a clear timing schedule. Every 6 seconds, a new block is produced. The Grandpa algorithm then finalizes this block. This schedule could make correlation attacks based on timing, where the adversary controls the guard node, more effective. The Polkadot node has different circuits to different peers. If a guard node sees a request on multiple circuits simultaneously, it could identify the request as a Polkadot node request.

These three aspects could have an important impact on the effectiveness of correlation attacks. This does not alter the fact that executing these attacks requires resourceful adversaries. The chance that an adversary controls either only the guard node or both the guard and exit node depends on the number of nodes they can inject into the Tor network. For basic adversaries, these attacks have become improbable because of the growth of the Tor network. The Tor network currently has nearly 2000 guard nodes and 1600 exit nodes. Using the port allowance technique, the adversary can shrink the pool of exit nodes to 270 but controlling the guard node remains a problem. As a network anonymization tool, Tor could be useful depending on the user's privacy requirements.

One risk of running a Polkadot node behind Tor is forgetting to enable Tor. If a user forgets to enable Tor only once, all peers that it has connected with know the original IP address of the Polkadot node. These peers will remember this IP address even when the user has enabled Tor again. In the Bitcoin over Tor paper[8], the researchers showed that using a specific fingerprint, a Bitcoin node's session could be tracked between Tor and a normal environment. For Polkadot, this is even simpler as the node supplies its own fingerprint as its PeerID.

[2]`https://onionoo.torproject.org/details?flag=exit`

Chapter 7

Security

This chapter discusses different security risks for the Polkadot node. The chapter starts with an in-depth analysis of an Eclipse attack. Three papers describing Eclipse attacks on the bitcoin[8], Ethereum[44], and IPFS[54] networks are compared to the Polkadot network. Eclipse attacks are especially relevant in this thesis, as the bitcoin paper describes an Eclipse attack on the bitcoin node behind the Tor network, and we run a Polkadot node behind the Tor network. And the other two papers describe networks similar in structure to the Polkadot network. After the paper comparison, the Polkadot system is analyzed for the potential of an Eclipse attack, and some considerations are discussed why the attack was not tested. After that, other security risks for the Polkadot node in the VPN and Tor environments are discussed.

7.1 Eclipse Attack

7.1.1 Eclipse Attack Comparison

This section compares the Polkadot system to other systems vulnerable to Eclipse attacks. Each leveraged mechanism in the other papers is analyzed to determine if Polkadot could be vulnerable to an Eclipse attack. In table 7.1, some key aspects of each network are compared.

	Polkadot	Bitcoin	Ethereum	IPFS
Max Connections (out)	50 (25)	125 (8)	25 (13)	900 (900)
Network structure	Kademlia	Unstructured (random graph)	Kademlia	Kademlia
Node identification	PeerId (based on ed25519 Pub key)	IP address	ECDSA Pub key	PeerId (based on ed25519 Pub key)
Message Encryption and Authentication	Both	None	Both	Both
Peer information storage	In memory	On disk	On disk	In memory

Table 7.1: Key comparisons between Polkadot, Bitcoin, Ethereum, and IPFS.

Bitcoin over Tor

The first paper analyzed is the Bitcoin over Tor paper[8]. In this paper, the researchers found that users using a Bitcoin node over Tor were vulnerable to Eclipse attacks. Furthermore, they used a fingerprint attack to track the user over multiple sessions.

The main vulnerability used for the Eclipse attack is Bitcoin's anti-DOS protection mechanism. Peers in the Bitcoin system are identified by their IP addresses. When a peer sends malicious data, it is banned for 24 hours. In this paper, a malformed coinbase transaction is used. Bitcoin uses this system to prevent Denial-of-Service attacks.

When a user connects to the Bitcoin network through Tor, the IP address used to identify the user is of a Tor exit node. The researchers used the anti-DOS mechanism to ban all non-malicious exit nodes. Meanwhile, they inserted multiple exit nodes into the Tor network, forcing users to connect to the Bitcoin network through their exit nodes. They also inserted bitcoin peers into the network. A user can only connect to the attacker's bitcoin peers or through the attacker's Tor exit nodes. So the attacker can fully eclipse a Bitcoin node connected through Tor.

Polkadot uses a reputation-based peer system. This system also bans nodes that send malicious data. However, in Polkadot, this particular attack would not be possible as Polkadot identifies nodes with a PeerId based on an ed25519 public key. So an attacker cannot force bans on Tor exit nodes as the Polkadot system does not care where the traffic is coming from but from which public key is coming. On the assumption that an attacker can not steal a user's ed25519 key, the attacker can not forge a user's network traffic, as all network data is encrypted and authenticated with the user's private key.

Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network

In this Eclipse attack, the researchers could fully eclipse an Ethereum node. The eclipsed node did not run in a special network environment, like behind the Tor network. However, the Ethereum network is more similar to the Polkadot network than the Bitcoin network. The Ethereum network and the Polkadot network structure are based on Kademlia. Both use public-key node identification and use encrypted and authenticated network traffic. Therefore, the vulnerabilities used in the paper are analyzed to determine if the same vulnerabilities could be used in the Polkadot network.

In the Ethereum Eclipse attack paper, the researchers leveraged the Kademlia peer discovery system and an early UPD connection listener. There are two attacks used, a simple and a more advanced one. In the simple attack, the researchers leveraged an Ethereum node that reboots and accepts incoming connections before any node discovery process has started. The Ethereum node allows all established connections to be incoming connections. The researchers quickly occupied all the victim's connections during the startup process, completely eclipsing the node.

For the more advanced Eclipse attack, the researchers use the early listener and the Kademlia node discovery mechanism to occupy all the victim's incoming and outgoing connections.

An Ethereum node uses the Kademlia algorithm to find new peers. The peer information is stored in two different data structures. The first structure is a long-term database stored on disk. The second data structure is a routing table containing 256 buckets. Each bucket can contain information about 16 peers. The discovered peers are inserted in the long-term database and one of the buckets if the bucket is not already full. The specific bucket is selected based on the distance between the Ethereum node and the peer. The distance is calculated using the Xor of the sha3 hash values of the node's public key and the peer's public key. The distance between two nodes is 256 minus the number of significant bits of the hash values that are equal. So if the two hash values are identical, the distance would be zero, and the peer would be placed in the first bucket.

The researchers discovered that most honest nodes were placed in the last 16 buckets. So if they could occupy the last 16 buckets with IDs of the attacker's nodes, the victim would only connect to these nodes. Node IDs are easily created by generating ECDSA pub keys. It takes around 15 minutes on a general computer for an attacker to generate enough node IDs to occupy the last 16 buckets. The attacker then creates an Ethereum node for each node id and inserts all the node IDs in the victim's long-term database by pinging the victim from these nodes. Ethereum only authorizes nodes based on their node ID so that the attacker can run all the attacker nodes from one IP address.

When the nodes are inserted in the victim's database, the attacker waits for or forces the victim to reboot. The victim's routing table is empty on startup, so the attacker pings the victim from all the malicious nodes. The victim checks its long-term storage database for the node IDs. If the node IDs are present, the victim immediately inserts the node ID in its routing table. The attacker already filled the victim's long-term storage with the attacker node IDs in the previous step and thus immediately fills up the first 16 buckets of the victim's routing table. The victim then picks one of the attacker's nodes to establish an outgoing connection, thereby completely isolating itself from the rest of the network. The researchers managed to eclipse the victim 88% of the time successfully.

For both attacks, the researchers used the listener that accepted incoming connections before the routing table seeding process began. One of the countermeasures mentioned in the paper is first seeding the routing table before accepting incoming connections. This makes it significantly harder to successfully Eclipse attack a victim, as they already have legitimate node IDs in their routing table and possibly already established outgoing connections with honest nodes. The researchers also leveraged the long-term storage database for the second attack to quickly insert node IDs in the routing table.

Polkadot is not exposed to these specific vulnerabilities. First, a Polkadot node does not accept only incoming connections. A Polkadot node accepts 25 incoming connections and 25 outgoing connections by default. If a peer tries to establish an incoming connection while the 25 limit is reached, Polkadot's peerset manager refuses the connection. So there are always slots available for outgoing connections. Secondly, Polkadot only has a Kademlia routing table in memory and does not store any node information on disk. So discovered nodes are not persistent over multiple sessions. Thirdly, Polkadot inserts all bootstrap nodes (hardcoded in the genesis file) into its routing table before it starts establishing connections. Lastly, running Polkadot nodes requires many resources, making the attack more expensive. The attack on the Ethereum node only required two servers, while an attack on a Polkadot node would require hundreds of servers. The cost of running an Eclipse attack in Polkadot is described in Section7.1.2

Especially for the Polkadot nodes in the Tor and VPN environment, this particular Eclipse attack is impossible, as the node cannot establish incoming connections.

Total Eclipse of the Heart – Disrupting the InterPlanetary File System

In this paper, the researchers entirely eclipse regular and bootstrap IPFS nodes. Like the Ethereum nodes, the nodes were not running in a special environment. This paper is interesting as the IPFS nodes use the same networking stack as the Polkadot nodes (Libp2p). The researchers used vulnerabilities in the Libp2p software and specifically in the connection manager to quickly eclipse nodes.

The connection manager is responsible for managing the number of connections. When too many connections are opened with the node, the manager has to decide which connections to drop. The upper limit of connections is called the highWater mark. When the High Watermark is reached, it starts trimming connections till it reaches the lowWater mark. The trimming process is based on a scoring system. The idea behind the scoring system is to keep beneficial connections open. The researchers leveraged this scoring system to make malicious connections less likely to be trimmed.

To eclipse a target node, an attacker first generates many node IDs covering the lowest 33+ buckets of the target node. The attacker exploits the connection manager by opening as many connections to the victim as possible until the highWater mark is reached. The IDs of the malicious nodes are inserted into the victim's Kademlia buckets if there is room in the buckets.

The attacker nodes then execute certain behavior to inflate their connection score, making their connections more important than other legitimate connections. When the highWater mark is reached, the connection manager starts trimming the connections to the low water mark, mainly dropping connections to legitimate peers as they have lower scores than the malicious nodes. When these peers are disconnected, the Kademlia routing table eventually removes them to replace them with connected peers. The attacker can thus exploit the connection manager to evict peers from the routing table and populate the routing table with malicious nodes. This process is repeated until the victim is completely eclipsed. The attacker can then maintain the eclipse by keeping four malicious connections alive. When there are fewer connections, the target node starts to connect to the bootstrap nodes.

The researchers show that it is possible to entirely eclipse a target node within an hour using the abovementioned exploits. They created a security model that indicates the costs of running an Eclipse attack. The costs for eclipsing all reachable nodes in the IPFS network are calculated based on the following parameters:

- n: network size (=number of nodes)
- s_{100}: Number of node Identities that need to be generated to fill a target's routing table to eclipse it.
- e: Number of Sybil nodes that need to be run to fill these routing table spots
- $c_{s_{100}}$: cost of node identify generation
- c_e: cost of operating Sybil nodes required during an actual attack on a single target
- c_i: total cost of attacking an individual node
- c_t total cost of attacking the whole network

Based on the theoretical study by Prünster et al., s_{100} and e can be estimated for a Kademlia-like network design.

$$s_{100} \approxeq 2 * k * n \tag{7.1}$$

$$e \approxeq k * log_2(n) \tag{7.2}$$

The researchers approximated the costs of eclipsing a single node and the entire network for the IPFS network. For the attack previously described, c_i and c_t are $€2000 + €0.031/h$ and $€2000 + €0.031/h * n$ where n is 3000, and the attack takes one hour. They also calculate the costs of running a naive Eclipse attack without using the exploits. The cost of running this attack for an individual node is $€0.4/h$ or $€67.20/week$ and could span multiple weeks. Eclipsing the whole network would cost more than $€200.000/week$. These calculations are also applied in Section 7.1.2 to approximate the cost of running a naive Eclipse attack on a Polkadot node.

The previously mentioned vulnerabilities do not apply to Polkadot. Polkadot also uses Libp2p and Kademlia but handles connections differently. Regular Polkadot nodes have a peerset manager that manages the incoming and outgoing connection. The maximum number of incoming and outgoing connections is 25, totaling a maximum of 50 connections. The peerset manager also tries to maintain the same number of incoming and outgoing connections. When a Polkadot node receives a new incoming connection, it refuses this connection if it already has established 50 connections. This attack is impossible for the Polkadot nodes in the Tor environment as the nodes cannot establish incoming connections.

Polkadot also uses a reputation system to score peers. This reputation system is different from the scoring system used by IPFS. The reputation is based on connection stability and different system protocols. A peer receives, for example, a negative reputation for sending incorrect information or ignoring requests. IPFS scores peers based on different metrics, for example, the distance of the peer in the Kademia routing table. In Polkadot, peers are mainly punished for misbehaving, while in IPFS, the peers could inflate their score based on a particular behavior. In section 7.1.2, the Polkadot reputation system and other features are analyzed in more depth to determine if Polkadot is vulnerable to an Eclipse attack.

7.1.2 Eclipse Attack on Polkadot

In the previous section, three different Eclipse attacks are compared for vulnerabilities that could be used in an Eclipse attack. For each of these vulnerabilities, a reason has been given why Polkadot in the Tor and VPN environments or Polkadot, in general, is not vulnerable. In this section, other possible vulnerabilities are analyzed for a Polkadot node to determine if the node could be eclipsed. The threat model in Section 6.1 also applies to the Eclipse attacks described in this section.

One of the main reasons the Polkadot node in the Tor and VPN environment is not vulnerable to many of the previously mentioned vulnerabilities is that it can not receive any incoming connections. It is difficult for an attacker to establish a connection to the node as it has to wait for the node to establish a connection. The chance that this happens depends on the number of nodes the attacker controls in the network.

Once one of the attacker's nodes establishes a connection with the node, it can be queried by the Kademlia discovery mechanism for new peers. Polkadot uses a random walk strategy for peer discovery, meaning it generates a random peerID and asks the 20 closest peers to return their closest peers to the random peerID. The chance that the malicious node is selected as one of the 20 closest peers is approximately $1/i * 20$, where i is the total number of nodes in the routing table. The newly discovered peers are not automatically inserted into the routing table but must be explicitly added.

The discovered nodes are sent to the peerset manager. The peerset manager manages the set of connected peers and determines to which peer the node should try to establish a connection. It tries to establish by default a maximum of 50 connections, 25 outgoing and 25 ingoing connections. All newly discovered nodes are placed in the peer set with a reputation of 0. Each second, the peerset manager signals to open a connection to the non-connected peer with the highest reputation. If multiple nodes have the same reputation, one is chosen randomly. If there is a free outgoing connection slot available, the Polkadot node tries to establish a functional connection to the peer. The peer is added to the Kademlia routing table if it establishes a functional connection.

The Polkadot node tries to maintain connections to peers as long as possible. However, the node disconnects from a peer in certain situations. When a peer misbehaves, the peer's reputation is lowered, and the peer is possibly disconnected, depending on the type of misbehavior. It is also possible that the peer reaches a certain ban threshold. Once the peer reaches the threshold, its connection is dropped. It is also possible that the peer initializes a disconnect. Nonetheless, if the peer behaves correctly, the Polkadot node keeps the connection open.

The combination of the previously mentioned systems should make it hard for an attacker to perform an Eclipse attack on a node in the Tor and VPN environment. First, it is difficult to establish the initial malicious connection. Then it is hard to insert more malicious peers in the node's routing table because the malicious peer has the same chance as the others to be part of a Kademlia query. Once the malicious peer is selected for a Kademlia query, it can respond with the IDs of other malicious peers. These peers are then added to the peerset manager, and the chance that they are chosen for a connection is the same as the other discovered peers. Once the node establishes a connection to a malicious peer, it is inserted into the routing table bucket if the bucket still needs to be filled.

Eclipse Attack on Polkadot Node in Tor Environment

The primary concern for the nodes in the Tor environment is the high disconnection rate. As previously mentioned, the Polkadot node in the Tor environment establishes connections to 1300-1400 distinct nodes, while the nodes in the VPN and normal environment connect to 400-500 and 150-250 distinct nodes. The disconnection rate indicates that the nodes in the Tor environment struggle with establishing lasting connections. This gives an attacker an opportunity. If the attacker can establish lasting connections, they might be able to eclipse a target node eventually. It is not known how easy it is to maintain a connection due to the Tor environment between the target and the attacker. A connection over Tor might time out, resulting in a disconnection. Only the target can re-establish the connection.

The re-establishment depends on the reputation of the malicious peer, as the peerset manager chooses the peer with the highest reputation.

A malicious peer should fully function to maintain a good reputation. For example, missing a block request from the target already decreases the malicious peer's reputation by 536870912, and the connection to the peer is dropped. Most reputation changes are between 128 and 4096, so a peer is heavily punished for not functioning fully. The malicious peer does not receive a re-establishment request as other peers have higher reputation scores. So a malicious node should function properly but can be adapted to prioritize a connection to the target node.

The high disconnection rate also increases the attacker's chance of establishing the first malicious connection. Once the first connection is established, the malicious peer can return more malicious peer IDs. The chance that another connection to a malicious peer is established increases as the number of malicious peer IDs in the peerset increases. If the attacker can establish long-lasting connections with the target node, they could eventually eclipse it.

The cost calculations from Section 7.1.1 are used to approximate the cost of running an Eclipse attack against a Polkadot node in a Tor environment. The maximum number of discovered nodes in Section 5.1.3 is 2700. It is unclear if these are all reachable nodes or if some are running behind a NAT/Firewall. So as an indication of the number of reachable nodes in the network, the number of nodes announced on Polkadot Telemetry[1] service is taken as these are nodes that publicly announce themselves. Currently, around 2000 nodes are announced on the Telemetry website, so parameter n is 2000. Using Equation 7.2 e is then approximated to:

$$e_{Polkadot} \approxeq k * log_2(n) = 20 * log_2(2000) \approxeq 220$$

As previously described, the attacker's nodes should be fully operational. Running a Polkadot full node requires considerable resources. In the IPFS paper, the researchers used a cheap hosting provider[2] to host their nodes. The same hosting provider is used for these cost calculations. By comparing the hosting specs used in Section 4.2.1 to the specs of the cheap hosting provider, two fully functional Polkadot nodes should be able to run on the CPX31 instance. A CPX31 instance costs $€0.0261/h$, so the cost of running the malicious peers to eclipse a single target is:

$$c_e = \frac{e}{2} * €0.0261/h \approxeq €2.87/h$$

It is not clear, if at all possible, how long it would take to eclipse a target node entirely. During the measurements, the Polkadot nodes in the Tor environment established around 1400-1500 connections to distinct peers over three days. As previously mentioned, the maximum number of discovered nodes in Section 5.1.3 is 2700. This could indicate that the node would establish connections to every discovered node in approximately six days, making it possible to eclipse a node in less than six days if the attacker can keep long-lasting connections. Otherwise, the attacker is not able to perform an Eclipse attack.

The time to eclipse could be lower, as malicious nodes only return other malicious nodes for the Kademlia lookup. In contrast, honest nodes could return both malicious and non-malicious nodes. This could increase the percentage of malicious nodes in the list of discovered nodes over time. This makes it more likely that the next established connection is to a malicious node. It is hard to approximate the effect of this process on the overall time to eclipse.

Nevertheless, the operating costs are $€482/week$, so attempting an Eclipse attack is rather expensive. The costs of running an Eclipse attack scale linearly with the network size. The attack could become even more expensive in the future as more Polkadot nodes join the network.

[1]`https://telemetry.polkadot.io/`
[2]`https://www.hetzner.com/cloud`

Eclipse Attack on Polkadot Node in the VPN environment

The Eclipse attack on a Polkadot node in the VPN environment is comparable to that of a node in the Tor environment. They are both only establishing outgoing connections, so an attacker has to wait for the node to establish outgoing connections to the malicious peers. However, the VPN environment is different, as the disconnection rate is considerably lower. The nodes in the VPN environment establish around 450 to 550 connections to distinct peers over three days. The node would need at least three weeks to establish connections to all peers. However, the experiments in the results section show that the nodes in the VPN environment can establish longer-lasting connections to the nodes in the Tor environment. There are periods when the nodes reach the connection limit, so they stop performing Kademlia queries. The nodes establish most of the 450-550 connections at the beginning of the run time. Once the nodes found good-performing nodes, they could establish lasting connections. The nodes in the VPN environment established connections lasting around 4 hours. In contrast, the nodes in the Tor environment could only establish connections lasting on average 15 minutes.

This indicates that the long-term connection rate of the nodes in the VPN environment is much higher than that of the nodes in the Tor environment. It could take weeks to months to eclipse a node in the VPN environment. The cost calculations of running an Eclipse attack are the same as for the Tor and VPN environments. With an operating cost of $€482/week$, an Eclipse attack on a node in the VPN environment costs thousands of euros.

Eclipse Attack on Polkadot Node in the Normal Environment

The Eclipse attack on a Polkadot node in the normal environment is different than on the nodes in the VPN or Tor environment. A normal environment node can establish incoming connections, like the nodes in the Ethereum and IPFS papers described in Sections 7.1.1 and 7.1.1. This could make them vulnerable to these Eclipse attacks described in the papers. However, as described in Sections 7.1.1 and 7.1.1, Polkadot is not vulnerable to these particular attacks because of the implementation of the Peerset manager.

As described in Section 2.2.4, a Polkadot node first tries to establish outgoing connections with the boot nodes. There are currently 17 boot nodes hard-coded in the Chain specification file. After that, the attacker could try to occupy all the victim's incoming connections by quickly sending connection requests during the startup phase. This is only possible if the attacker knows the victim's IP address and when the victim is starting up. It is nearly impossible to know the victim's IP address if it is the first time the node joins the network. The attacker could know the IP address if the node has joined the network. The attacker then has to wait for the victim to restart or force a restart, as described in the Ethereum paper. The attacker then has to compete for the incoming connection slots with honest peers that try to reconnect. So an attacker could occupy all the incoming slots of a victim if they are faster than other honest peers.

Nevertheless, the attacker has only established incoming connections and has to occupy all outgoing connections for a full Eclipse attack. Establishing all the outgoing connections is difficult, as the Polkadot node already established connections to some boot nodes. The peerset manager chooses new outgoing connections from the list of discovered peers. Peers discovered during the Kademlia random lookup and peers that try to establish incoming connections are inserted into the discovered nodes list with a reputation of 0. The new outgoing peer is selected based on the reputation system. The attacker cannot cheat the reputation system like in the IPFS paper, so each peer is as likely to be selected. The bootstrap nodes are already inserted into the discovered peers list during the startup phase. Moreover, most bootstrap nodes are queried for the first Kademlia requests as they are already inserted into the Kademlia routing table. The attacker's peers must first identify themselves with the Identity protocol before being inserted into the empty routing table slots. So during the startup phase, most of the peers in the discovered peers list are honest. A new outgoing connection is presumably established with an honest peer.

As with the other environments, an attacker has to wait for the victim to establish outgoing connections with their malicious nodes. The cost of running this attack is €$482/week$. The big difference between this environment and the Tor environment is that nodes in this environment maintain connections on average for eight to ten hours, while the nodes in the Tor environment only manage to maintain connections for around 15 minutes. So the rate at which the node establishes connections to new peers is considerably lower. The time it takes to successfully eclipse a node increases, making an Eclipse attack on a node in the normal environment quite expensive.

Eclipse Attack Overview

The previous sections discuss an Eclipse attack on a Polkadot node in each environment. No vulnerabilities were found for each of the environments that would make it easier to perform an Eclipse attack, like in the Bitcoin[8], Ethereum[44], and IPFS[54] papers. However, the nodes in the Tor environment show a high disconnection rate and low average connection time compared to the other two environments. This makes the execution of a standard Eclipse attack on a node in the Tor environment possibly faster. The time to eclipse a node in the Tor environment is approximately six days, while eclipsing nodes in the other environments could take weeks to months. These approximations exclude possible problems while eclipsing the node, like malicious connections being dropped due to unstable Tor or VPN connections.

Considerations and Decisions

We chose not to perform the Eclipse attack on each environment for multiple reasons. First, there is an ethical consideration. To perform the Eclipse attack, many peers must be inserted into the Polkadot network. From the calculations in Section 7.1.2, we calculated that we need at least 220 sybils in the Network to perform the attack. This could disrupt the network's stability during the attack or after the attack when these sybils exit the network.

A second reason is the cost of running an Eclipse attack. In Section 7.1.2, we calculated that the costs of performing an Eclipse attack are at least €$482/week$. We cannot say with certainty how long an attack would take, but it could be weeks, so it could cost thousands of euros to perform the attack.

7.2 VPN

This section discusses other security risks of running a Polkadot node in a VPN environment besides an Eclipse attack. First, the performance metrics are analyzed for security issues, and a problem with only outgoing connections is discussed. Then a possible Denial-of-Service attack is discussed.

7.2.1 Performance Metrics

The metrics discussed in Chapter 5.1 show no security risks for the Polkadot node in the VPN environment. The average median latency of the nodes in the VPN environment is higher for the North American and Asian nodes. This is expected as their network traffic must also travel to and from the VPN server. However, the increase in latency is not enough to start causing latency-related issues. The average 99th percentile latency is around a second higher for the nodes in the VPN environment. Nevertheless, the latency still is within an acceptable limit of the BABE security experiments as mentioned in Section 5.1.1.

7.2.2 Only Outgoing Connections

Running a Polkadot node behind a VPN could create a problem for the network in general. The node is not reachable by other peers as it only has outgoing connections. This does not pose a problem if only a small set of nodes are running behind a VPN connection. However, suppose a large number of nodes can only establish outgoing connections. In that case, this creates a problem in the overall network connectivity. Nodes would only be able to establish connections with peers that can receive incoming connections. If too many nodes can only establish outgoing connections, it could disrupt the whole network. New nodes would not be able to connect to existing nodes. Core connections would not be established, causing the network to partition.

Furthermore, in an extreme case, the network would stop working as no node could communicate with another. So this security problem depends on the network environments of all nodes in the network. If every node used the VPN network environment, no one could communicate with each other.

7.2.3 Denial of Service Attack

A possible security issue for the VPN setup could be a Denial-of-Service attack on the VPN server. Recent research shows that both the Wireguard and OpenVPN implementations are susceptible to DOS attacks[66]. They show that a Wireguard and an OpenVPN connection could be disrupted with 300 Mb/s and 100Mb/s attack traffic, respectively. The experiments are performed on a high-performance server, so the attack traffic needed to disrupt the VPN server used in this thesis could even be lower. This attack creates a serious security concern for running a Polkadot node behind a VPN. The IP address of the VPN is easily determined, as the node shares this address with other Polkadot peers. So it is easy to determine the targeted IP address.

7.3 Tor

This section discusses other security risks of running a Polkadot node in a Tor environment besides an Eclipse attack. First, the performance metrics are discussed. Then multiple attack scenarios in three papers [8][44][54] are compared to the Polkadot with Tor setup in this thesis. Finally, several components in the Polkadot system are analyzed to determine if an Eclipse attack on a Polkadot node is possible.

7.3.1 Performance Analysis

This section analyzes each metric for security problems for the node or the network. As with the nodes in the VPN environment, the nodes in the Tor environment establish only outgoing connections. Section 7.2.2 discusses possible security issues related to only establishing outgoing connections. The same security issues are relevant for the nodes in the Tor environment.

There is a serious increase in network latency for the nodes in the Tor environment compared to the other environments. As explained in Section 5.1.1, the latency of the nodes in the Tor environment could impact the overall network stability, security, and performance. The experiments in the BABE paper[2] show that the BABE algorithm is resistant to a network delay of 2.796 seconds if 85% of validators respond in time. However, the nodes in the Tor environment show 99th-percentile network delays of around 6 to 8 seconds. If more than 15% of validators ran behind the Tor network, the security of the BABE algorithm would not be guaranteed.

Using the script[3] from the paper with the same parameters, the network would be resistant to a network delay of 8 seconds if the slot duration is 18 seconds. So the network performance would be one-third of the current network.

[3]https://github.com/w3f/research/blob/master/experiments/parameters/babe_median.py

Chapter 8

Conclusion

Polkadot is a novel multi-chain blockchain protocol that aims to solve the scalability and interoperability issues many blockchain-based systems face. While Polkadot is relatively new (launched in 2020), it has received much attention from investors and developers. The development of the Polkadot system has been going fast. Nevertheless, privacy on the Polkadot network has yet to be one of the key focus points.

Privacy in a blockchain system is important, as users' real-world identities can be linked to their transaction history, making them vulnerable to targeted ads, manipulation, blackmail, reputational damage, financial loss, physical harm, discrimination, and more. Especially the unlinkability of a Polkadot user to their IP address is important. This thesis discusses the linkability of a user's IP address and their Polkadot interactions. We also analyze Tor and a VPN as privacy-enhancing solutions. The following research questions are defined to gain insights into Tor and a VPN as network layer privacy-enhancing solutions.

Q1. How can Tor or a VPN be used to enhance the user's privacy concerning the unlinkability between the user's full node and IP address?

In this thesis, a Polkadot user is implemented as a Polkadot full node. The node communicates with other nodes and broadcasts the user's transactions. As long as an adversary cannot find the node's IP address, the user's transactions and IP address are unlinkable. A VPN and Tor are used to hide the IP address of the full node's server. All network communication is sent through Tor or a VPN making it harder for an adversary to discover the node's IP address.

Q2.1 Which of the two unlinkability solutions is preferable for the user based on the difficulty of linking the node's traffic to its IP address

Both Tor and a VPN can hide the node's IP address but offer different levels of privacy. This thesis uses a self-hosted VPN server, as VPN services may store and sell user information. Self-hosted VPN services have some drawbacks. Firstly, the user could be identified based on the payments to the hosting provider that hosts the VPN server. Secondly, an ISP or law enforcement could passively monitor the network traffic from and to the VPN or subpoena the network logs from the hosting provider. The Tor network, on the other hand, is run by volunteers in a decentralized fashion. A user does not have to host anything and cannot be de-anonymized based on payments to a hosting provider. With Tor, the traffic is also routed through three relays, so there is no single place on the traffic's route that would reveal both the source and destination of the traffic. Tor is, therefore, the preferred solution in terms of the difficulty of linking the node's traffic to its IP address.

Q2.2 Which of the two unlinkability solutions is preferable for the user based on the performance of the node regarding throughput and latency.

Based on the location of the VPN server, the Polkadot node in the VPN environment can reach nearly the same median latency as the Polkadot node in the normal environment. At the 99th percentile latency, the node in the VPN environment is considerably slower than the node in the normal environment, but this does not impact the overall performance noticeably.

However, the Polkadot node in the Tor environment has a considerable latency difference compared to the node in the Normal environment. The median latency is over a second, and the 99th percentile latency can go up to 6-8 seconds. This has serious implications for the stability of the Polkadot node. The results of the experiments show that the nodes in the Tor environment cannot reach the maximum number of connected nodes and maintain long-lasting connections. This can impact the node's security and the network's security in general. These security concerns are further discussed in the answer to the last research question.

The results show that using Tor or a VPN does not limit a Polkadot node's throughput noticeably. Both environments show a lower throughput. However, this is mainly due to a lower number of connected nodes, as the throughput is heavily correlated to the number of connected nodes.

Mainly based on the latency of both environments, a VPN is the preferable solution for the user.

Q2.3 Which of the two unlinkability solutions is preferable for the user based on the ease of implementation.

With some changes to Polkadot's mDNS and bootstrap node settings, the Tor environment is easily set up using the TorGhost script. The user does not need to set up any other service like with the VPN environment, where the user needs to set up a VPN server. The setup of the VPN environment required more steps and custom configuration changes of the wireguard configuration files. So Tor is the preferable solution based on the ease of implementation.

Q3 Are security vulnerabilities introduced by using each solution?

There are some security vulnerabilities for both the VPN and Tor environments. Polkadot nodes in both environments only are potentially vulnerable to Eclipse attacks. The costs of running such an attack are estimated to be $€482/week$. The run times are approximately six days and weeks to months for an Eclipse attack on a node in the Tor and VPN environment, respectively.

The Polkadot nodes in both environments establish only outgoing connections. This is not a security concern for the node but for the network in general, as it can create a problem for the overall network connectivity. If too many nodes ran behind one of the two environments, the network could partition or stop working.

Furthermore, the VPN server of the node in the VPN environment could potentially be vulnerable to a Denial-of-Service attack, disrupting the Polkadot node's connectivity.

Lastly, the latency of the Polkadot node in the Tor environment could pose problems for the BABE block production algorithm. It mainly poses a problem if multiple users run a validator node behind the Tor network. The BABE algorithm is resistant to a network delay of 2.796 seconds if 85% of validators respond in time. Validators behind the Tor network would not belong to the 85%. So if too many validators ran in a Tor environment, the BABE algorithm with the current configuration would no longer be secure. In this thesis, a Polkadot user is modeled as a full node, not a validator node. High latency for a Polkadot full node does not pose immediate security concerns. However, it is possibly related to a high disconnection rate. A high disconnection rate makes a successful Eclipse attack more likely. However, if many Polkadot full nodes have high latency, it can impact the network's performance.

Concluding, running a Polkadot node behind Tor or a VPN poses problems. In terms of network layer privacy and implementation ease, Tor is the best option. However, Tor creates serious security concerns due to its latency. The VPN only offers basic network layer privacy and is harder to implement. Furthermore, the VPN environment has some security concerns regarding the lack of incoming connections and the potential Denial-of-Service attack against the VPN server. However, a VPN could offer basic network layer privacy for individual Polkadot nodes. However, running a Polkadot node behind either of the environments is not recommended as it has a severe security and performance impact on the whole Polkadot network.

8.1 Ethical considerations

In this section, some ethical considerations are discussed. The decision was made to run the measurements only with three nodes to impact the Polkadot network as little as possible while still investigating the location impact on the performance metrics. Furthermore, only three nodes were used to minimize the load on the Tor network, as it is used by many people who require anonymity. As discussed in Section 7.1.2, the decision was made not to perform the Eclipse attack on each environment, as it would require hundreds of Polkadot nodes. This could disrupt the Polkadot network's stability during the attack or after the attack when the nodes exit the network.

8.2 Contributions

This thesis has the following scientific contributions:

- This thesis is, to our knowledge, the first public research into network layer privacy of the Polkadot network.
- A measurement study of the Polkadot network.
- A practical implementation of Tor or a VPN with Polkadot.
- Analysis of security concerns when running Polkadot nodes behind Tor or A VPN.
- An in-depth conceptual analysis of an Eclipse attack on a Polkadot node.

8.3 Future work

This thesis asks for further research into network layer privacy for the Polkadot network. Some of the future work is discussed below.

- **Implementing Eclipse attack:** The analysis of the Eclipse attack in Section 7.1.2 discusses that the node in the Tor environment could potentially be eclipsed in less than a week. A practical implementation of the Eclipse attack could show if this is the case or if other implementation details would prevent a successful Eclipse attack.
- **Investigate Denial-of-Service attack on VPN:** As mentioned in Section 7.2.3, the wireguard VPN could potentially be vulnerable to a DoS attack [66]. Investigating the attack mentioned in the VPN DoS paper is important, as it would pose a serious security concern for Polkadot nodes running behind a VPN. Especially validator nodes would be vulnerable to DoS attacks, as they could be punished for unresponsiveness.
- **Performance impact of Tor on Polkadot network.** This thesis has investigated the performance of running a single Node behind Tor. The latency issues for a Polkadot node in a Tor environment could impact the whole network, as mentioned in Section 7.3. Especially if many nodes would start running behind the Tor network. It could be interesting to investigate the impact of multiple Polkadot nodes running behind Tor on the network in general. This could be done using a private test network.
- **Built-in network level privacy:** There is some research into network layer privacy built into the blockchain networking stack. Dandelion[72] is an example of a solution for the Bitcoin network. Investigating built-in solutions for the Polkadot network could be an exciting topic for future work.

Bibliography

[1] Hal Abelson, Ken Ledeen, and Harry Lewis. *Blown to bits: your life, liberty, and happiness after the digital explosion*. Addison-Wesley Professional, 2012.

[2] Handan Kilinc Alper. *Babe*. URL: `https://research.web3.foundation/en/latest/polkadot/block-production/Babe.html`.

[3] Elli Androulaki et al. *Evaluating User Privacy in Bitcoin*.

[4] Leo Maxim Bach, Branko Mihaljevic, and Mario Zagar. "Comparative analysis of blockchain consensus algorithms". In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Ieee. 2018, pp. 1545–1550.

[5] Seyed Mojtaba Hosseini Bamakan, Amirhossein Motavali, and Alireza Babaei Bondarti. "A survey of blockchain consensus algorithms performance evaluation criteria". In: *Expert Systems with Applications* 154 (2020), p. 113385.

[6] Eli Ben-Sasson et al. "Zerocash: Decentralized anonymous payments from bitcoin". In: Institute of Electrical and Electronics Engineers Inc., Nov. 2014, pp. 459–474. ISBN: 9781479946860. DOI: `10.1109/SP.2014.36`.

[7] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. "Deanonymisation of clients in bitcoin P2P network". In: Association for Computing Machinery, Nov. 2014, pp. 15–29. ISBN: 9781450329576. DOI: `10.1145/2660267.2660379`.

[8] Alex Biryukov and Ivan Pustogarov. "Bitcoin over Tor isn't a good idea". In: (Oct. 2014). URL: `http://arxiv.org/abs/1410.6079`.

[9] Alex Biryukov and Sergei Tikhomirov. "Deanonymization and linkability of cryptocurrency transactions based on network analysis". In: Institute of Electrical and Electronics Engineers Inc., June 2019, pp. 172–184. ISBN: 9781728111476. DOI: `10.1109/EuroSP.2019.00022`.

[10] Joseph Bonneau et al. "Mixcoin: Anonymity for bitcoin with accountable mixes". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2014, pp. 486–504.

[11] Jeff Burdges et al. *Sassafras*. URL: `https://research.web3.foundation/en/latest/polkadot/block-production/SASSAFRAS.html`.

[12] *Business VPN: Next-Gen VPN*. Nov. 2021. URL: `https://openvpn.net/`.

[13] Vitalik Buterin et al. "A next-generation smart contract and decentralized application platform". In: *white paper* 3.37 (2014).

[14] Michael Casey et al. "The impact of blockchain technology on finance: A catalyst for change". In: (2018).

[15] Miguel Castro, Barbara Liskov, et al. "Practical byzantine fault tolerance". In: *OsDI*. Vol. 99. 1999. 1999, pp. 173–186.

[16] Alfonso Cevallos and Alistair Stewart. "A verifiably secure and proportional committee election rule". In: *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 2021, pp. 29–42.

[17] David Chaum. *The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability*. 1988, pp. 65–75.

[18] David L Chaum. *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*. 1981.

[19] Shumo Chu, Yu Xia, and Zhenfei Zhang. *Manta: a Plug and Play Private DeFi Stack*. 2021.

[20] Sebastian Clauß and Stefan Schiffner. “Structuring anonymity metrics”. In: *Proceedings of the second ACM workshop on Digital identity management*. 2006, pp. 55–62.

[21] *Confirmation*. Mar. 2018. URL: `https://en.bitcoin.it/wiki/Confirmation`.

[22] *Crypto FAQ: What is distributed ledger technology (DLT)?* URL: `https://cryptocurrencyworks.com/faq/what-is-distributed-ledger-tech.html`.

[23] *Cryptocurrency prices, charts and market capitalizations*. URL: `https://coinmarketcap.com/`.

[24] *Cryptographic hash function*. Sept. 2022. URL: `https://en.wikipedia.org/wiki/Cryptographic_hash_function#Properties`.

[25] George Danezis, Roger Dingledine, and Nick Mathewson. *Mixminion: Design of a Type III Anonymous Remailer Protocol*. 2003.

[26] Erik Daniel, Elias Rohrer, and Florian Tschorsch. “Map-Z: Exposing the Zcash Network in Times of Transition”. In: (July 2019). URL: `http://arxiv.org/abs/1907.09755`.

[27] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. *The Nym Network*. 2021. URL: `https://nymtech.net/nym-whitepaper.pdf`.

[28] Roger Dingledine. *Tor: The Second-Generation Onion Router*.

[29] Jason A. Donenfeld. *Fast, modern, secure VPN Tunnel*. URL: `https://www.wireguard.com/`.

[30] David M Douglas. “Doxing: a conceptual analysis”. In: *Ethics and information technology* 18.3 (2016), pp. 199–210.

[31] *Extrinsics*. URL: `https://polkadot.js.org/docs/substrate/extrinsics#balances`.

[32] Steven Goldfeder et al. “When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies”. In: (Aug. 2017). URL: `http://arxiv.org/abs/1708.04748`.

[33] Benjamin Greschbach et al. “The effect of DNS on Tor’s anonymity”. In: *arXiv preprint arXiv:1609.08187* (2016).

[34] Nguyen Phong Hoang and Davar Pishva. “Anonymous communication and its importance in social networking”. In: *16th International Conference on Advanced Communication Technology*. IEEE. 2014, pp. 34–39.

[35] *i2p anonymous network*. URL: `https://geti2p.net/en/`.

[36] Paula Iacoban. “Measuring Accessibility of Popular Websites While Using the I2P Anonymity Network”. In: (2021).

[37] Aaron Johnson et al. “Users get routed: Traffic correlation on Tor by realistic adversaries”. In: 2013, pp. 337–348. ISBN: 9781450324779. DOI: `10.1145/2508859.2516651`.

[38] Alice Kao. *RIAA v. Verizon: Applying the Subpoena Provision of the DMCA*. 2004, pp. 405–426.

[39] Enis Karaarslan and Enis Konacaklı. “Data storage in the decentralized world: Blockchain and derivatives”. In: *arXiv preprint arXiv:2012.10253* (2020).

[40] Ishan Karunanayake et al. “De-anonymisation attacks on Tor: A Survey”. In: *IEEE Communications Surveys & Tutorials* 23.4 (2021), pp. 2324–2350.

[41] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. “A better method to analyze blockchain consistency”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 729–744.

[42] Philip Koshy, Diana Koshy, and Patrick McDaniel. “An analysis of anonymity in bitcoin using p2p network traffic”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2014, pp. 469–485.

[43] Fabio Lama, Florian Franzen, and Syed Hosseini. *Polkadot protocol specification*. July 2022. URL: `https://spec.polkadot.network/`.

[44] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. “Low-resource eclipse attacks on ethereum’s peer-to-peer network”. In: *Cryptology ePrint Archive* (2018).

[45] Mudit Marda. *Blockchain Development Trends 2021*. Feb. 2021. URL: `https://outlierventures.io/wp-content/uploads/2021/02/OV-Blockchain-Dev-Q1-2021-_v7.pdf`.

[46] Petar Maymounkov and David Mazieres. “Kademlia: A peer-to-peer information system based on the xor metric”. In: *International Workshop on Peer-to-Peer Systems*. Springer. 2002, pp. 53–65.

[47] Sarah Meiklejohn and Rebekah Mercer. *Möbius: Trustless Tumbling for Transaction Privacy*.

[48] Sarah Meiklejohn et al. “A fistful of bitcoins: Characterizing payments among men with no names”. In: 2013, pp. 127–139. ISBN: 9781450319539. DOI: `10.1145/2504730.2504747`.

[49] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. “An overview of smart contract and use cases in blockchain technology”. In: *2018 9th international conference on computing, communication and networking technologies (ICCCNT)*. IEEE. 2018, pp. 1–4.

[50] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: *Decentralized Business Review* (2008), p. 21260.

[51] *Node Endpoints*. Dec. 2021. URL: `https://wiki.polkadot.network/docs/maintain-endpoints`.

[52] Andreas Pfitzmann and Marit Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. 2010. URL: `http://dud.inf.tu-dresden.de/Anon_Terminology.shtml`.

[53] Ania M Piotrowska et al. “The loopix anonymity system”. In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 1199–1216.

[54] Bernd Prünster, Alexander Marsalek, and Thomas Zefferer. *Total Eclipse of the Heart-Disrupting the InterPlanetary File System*. URL: `https://ipfs.io`.

[55] Fergal Reid and Martin Harrigan. *An Analysis of Anonymity in the Bitcoin System*. URL: `http://www.theatlantic.com/technology/archive/2011/06/libertarian-dream-a-site-`.

[56] *Remote procedure call*. Nov. 2020. URL: `https://nl.wikipedia.org/wiki/Remote_procedure_call`.

[57] Dorit Ron and Adi Shamir. “Quantitative analysis of the full bitcoin transaction graph”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2013, pp. 6–24.

[58] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. “Coinshuffle: Practical decentralized coin mixing for bitcoin”. In: *European Symposium on Research in Computer Security*. Springer. 2014, pp. 345–364.

[59] *Run a validator (polkadot)*. Feb. 2022. URL: `https://wiki.polkadot.network/docs/maintain-guides-how-to-validate-polkadot`.

[60] Nicolas Van Saberhagen. *Monero (CryptoNote v 2.0)*. 2013. URL: `https://www.getmonero.org/ru/resources/research-lab/pubs/whitepaper_annotated.pdf`.

[61] Fahad Saleh. “Blockchain without waste: Proof-of-stake”. In: *The Review of financial studies* 34.3 (2021), pp. 1156–1190.

[62] Danny Salman. *Polkadot Keys · Polkadot Wiki*. Apr. 2022. URL: `https://wiki.polkadot.network/docs/learn-keys`.

[63] Stefan Schulte et al. “Towards blockchain interoperability”. In: *International conference on business process management*. Springer. 2019, pp. 3–10.

[64] Atul Singh et al. “Eclipse attacks on overlay networks: Threats and defenses”. In: *In IEEE INFOCOM*. Citeseer. 2006.

[65] Alistair Stewart and Eleftherios Kokoris-Kogia. “GRANDPA: a Byzantine finality gadget”. In: *arXiv preprint arXiv:2007.01560* (2020).

[66] Fabio Streun, Joel Wanner, and Adrian Perrig. “Evaluating Susceptibility of VPN Implementations to DoS Attacks Using Adversarial Testing”. In: *arXiv preprint arXiv:2110.00407* (2021).

[67] Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[68] Nick Szabo. *The Idea of Smart Contracts*. URL: `https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html`.

[69] Parity Technologies. *Paritytech/smoldot: Alternative client for substrate-based chains.* Oct. 2021. URL: `https://github.com/paritytech/smoldot`.

[70] Torproject. *Tor Specification*. May 2022. URL: `https://github.com/torproject/torspec/blob/main/tor-spec.txt`.

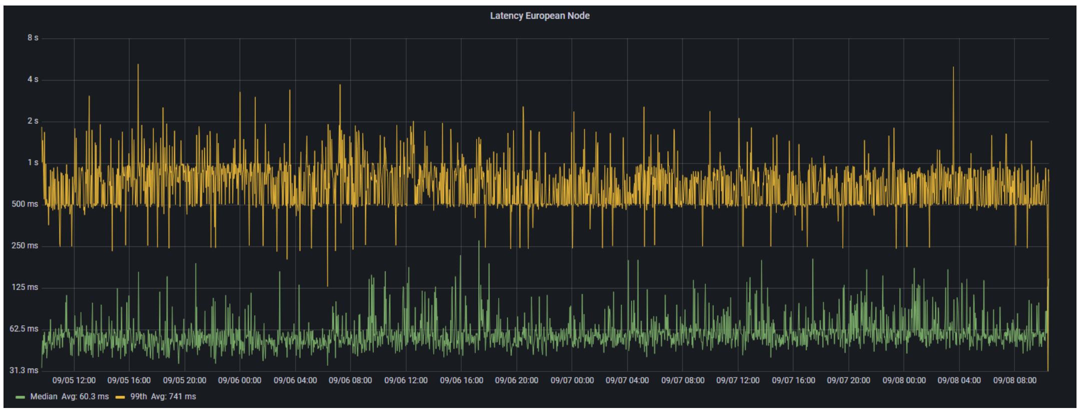
[71] *Transactions*. June 2022. URL: `https://ethereum.org/en/developers/docs/transactions/#types-of-transactions`.

[72] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. "Dandelion". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1 (1 June 2017), pp. 1–34. DOI: `10.1145/3084459`.

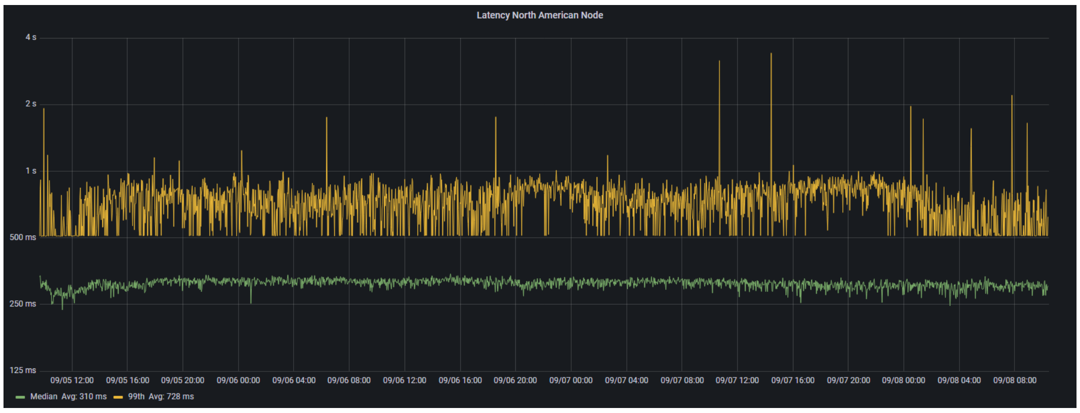
[73] Gavin Wood. "Polkadot: Vision for a heterogeneous multi-chain framework". In: *White Paper* 21 (2016), pp. 2327–4662.

Appendix A

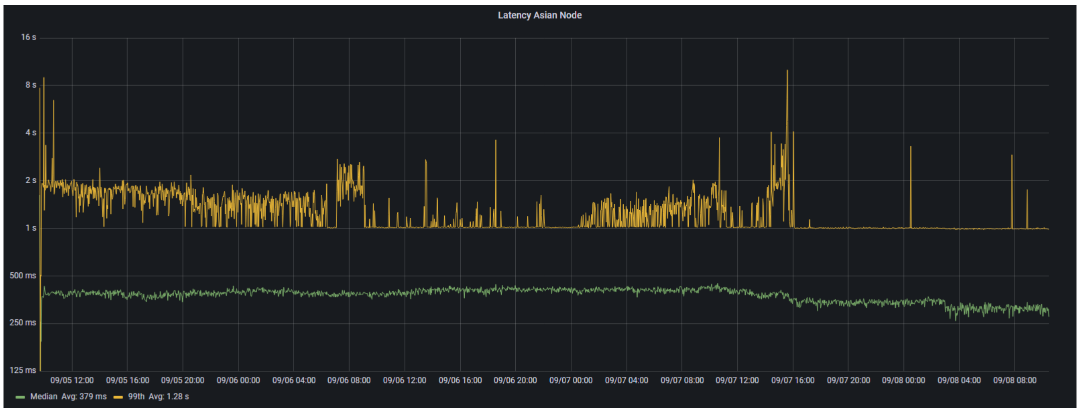
A.1 Latency of the Network Communication



(a) European Node

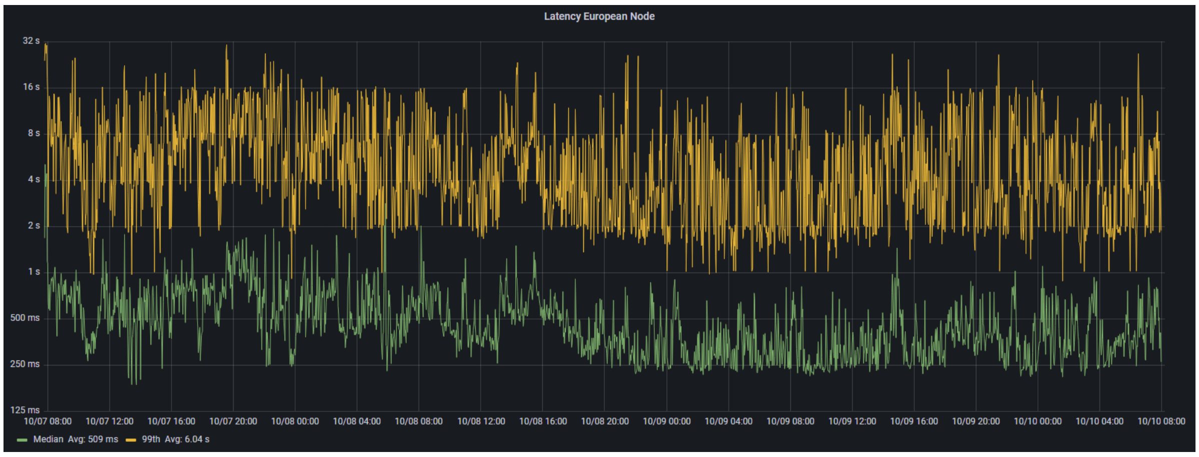


(b) North American Node

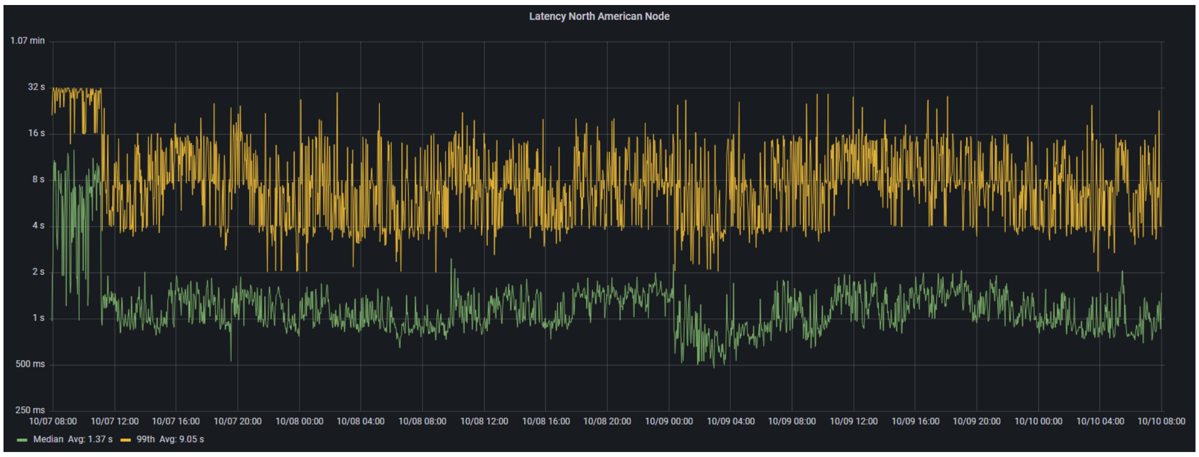


(c) Asian Node

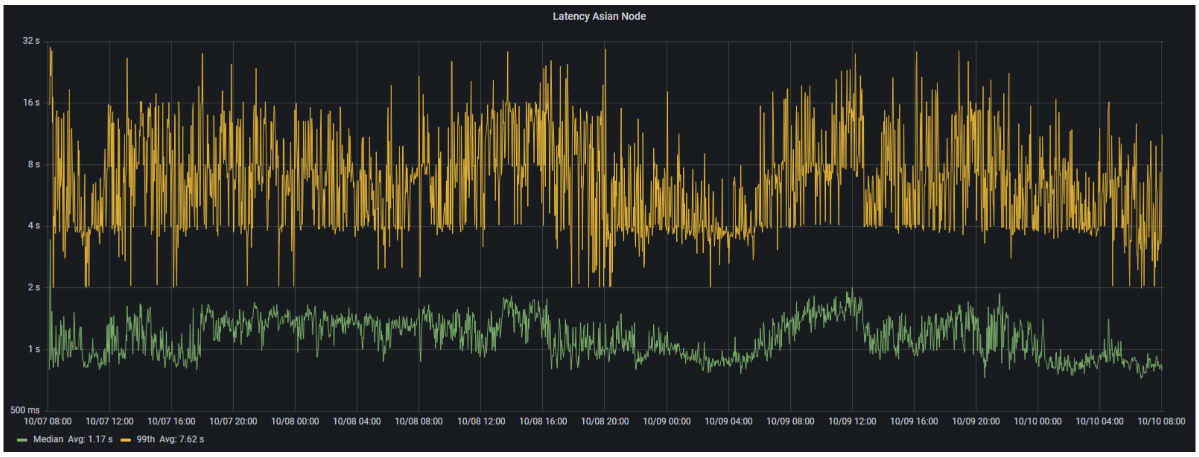
Figure A.1: Request answer time for block synchronization requests in the normal environment. Shown are the 50% and 99% percentiles.



(a) European Node

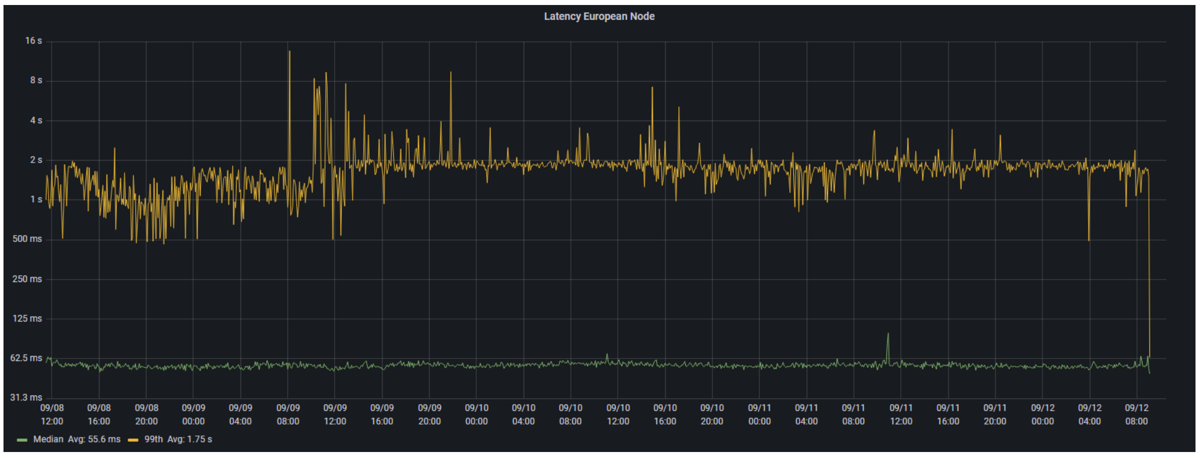


(b) North American Node

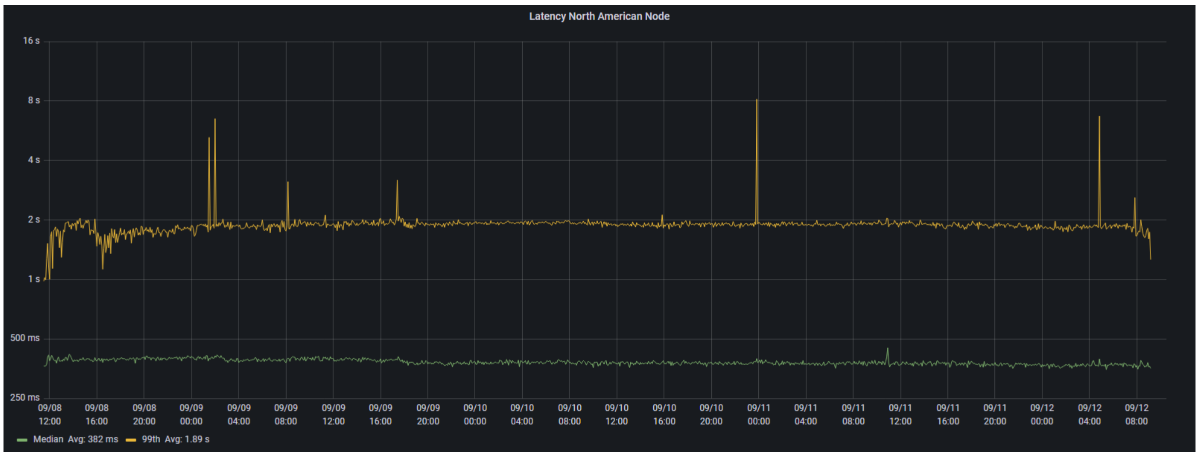


(c) Asian Node

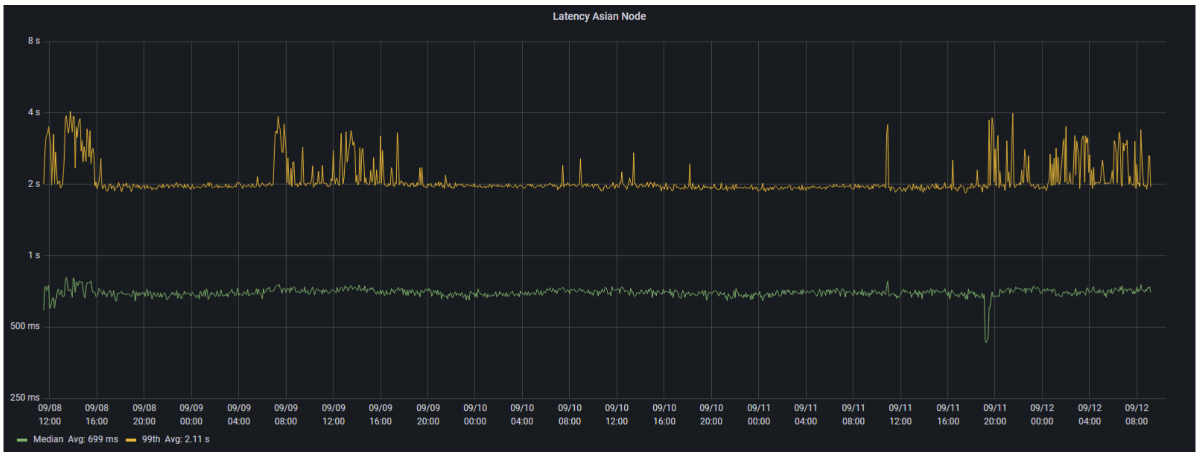
Figure A.2: Request answer time for block synchronization requests in the Tor environment. Shown are the 50% and 99% percentiles.



(a) European Node



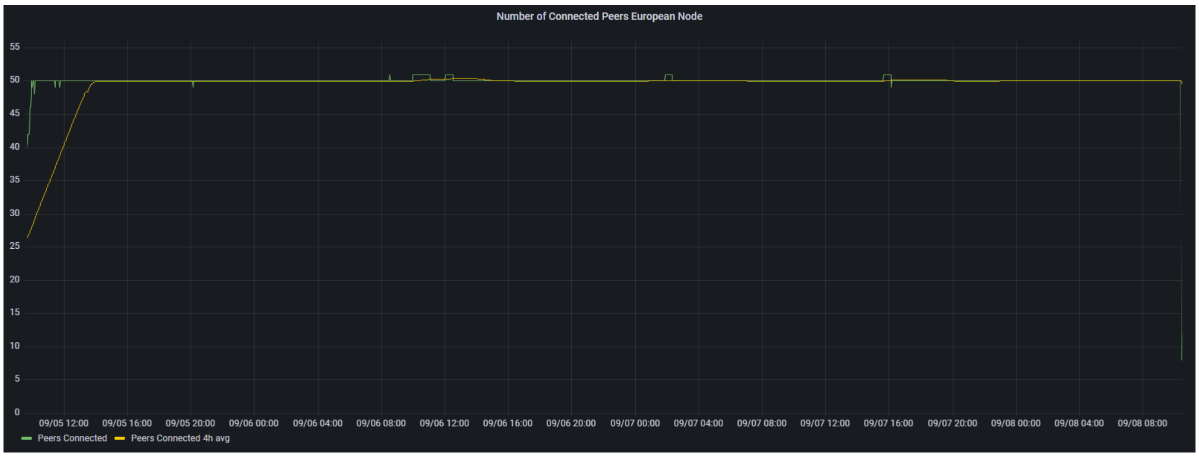
(b) North American Node



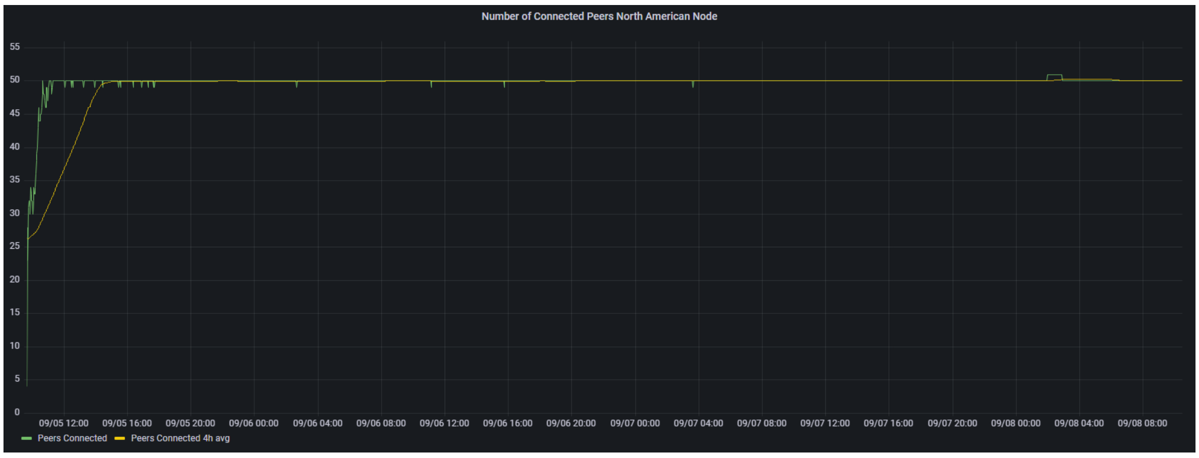
(c) Asian Node

Figure A.3: Request answer time for block synchronization requests in the VPN environment. Shown are the 50% and 99% percentiles.

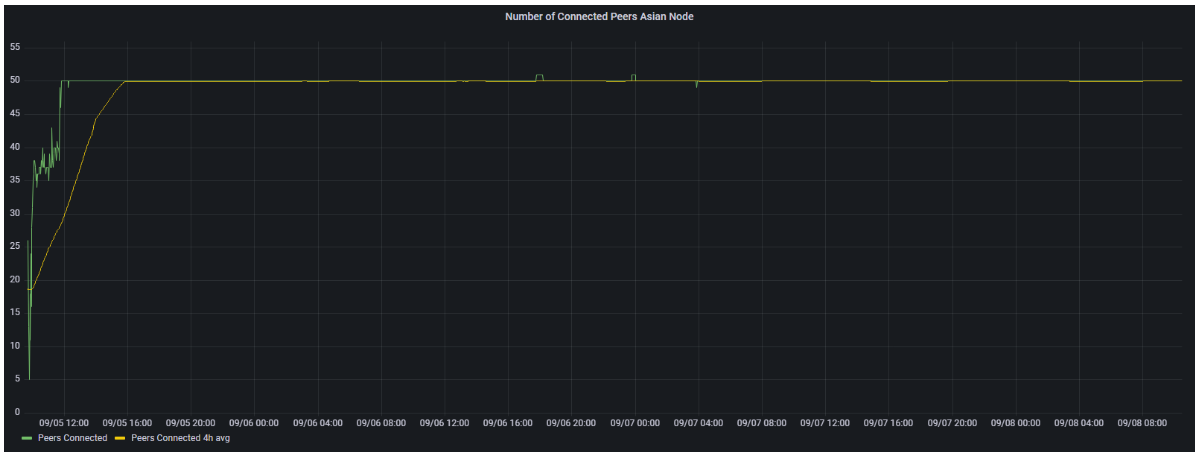
A.2 Number of Connected Peers over Time



(a) European Node

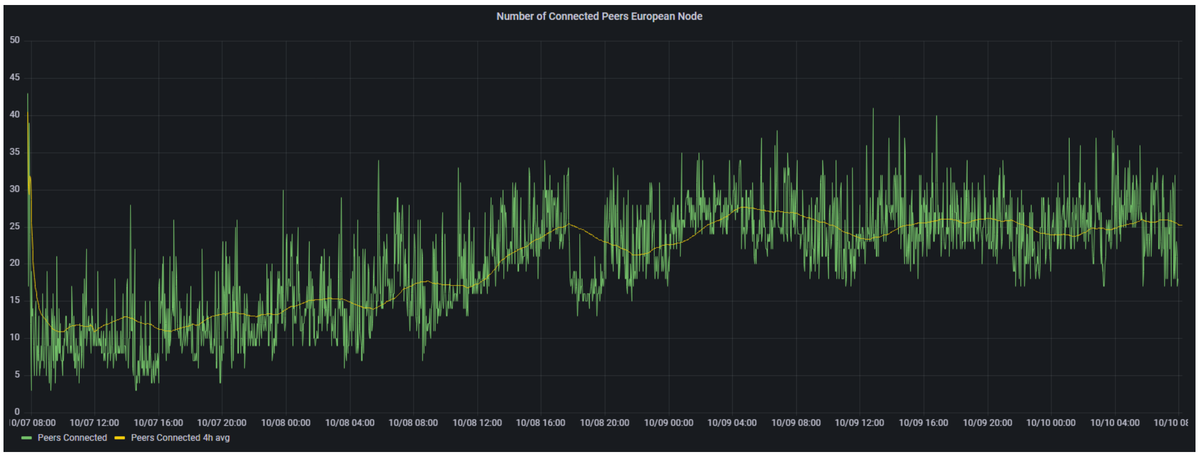


(b) North American Node

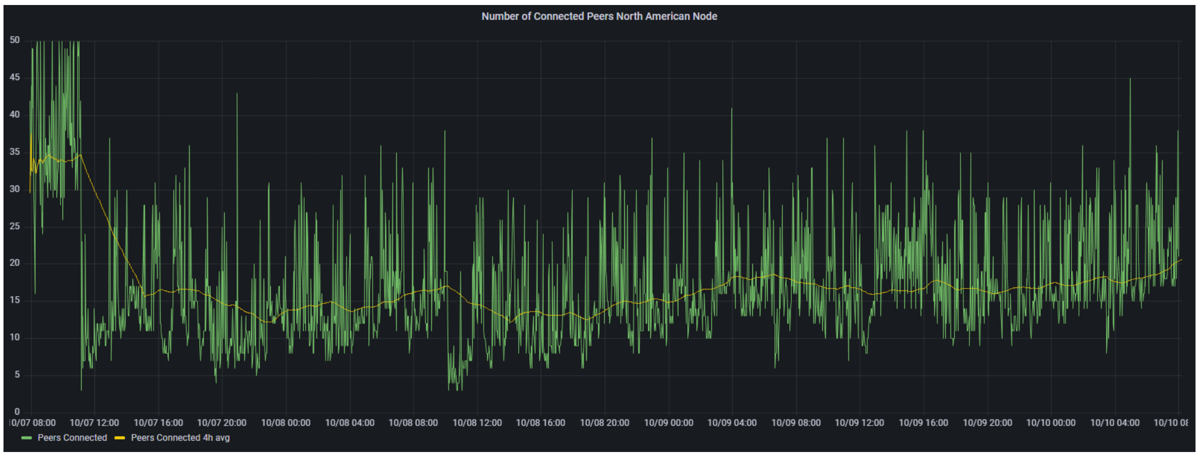


(c) Asian Node

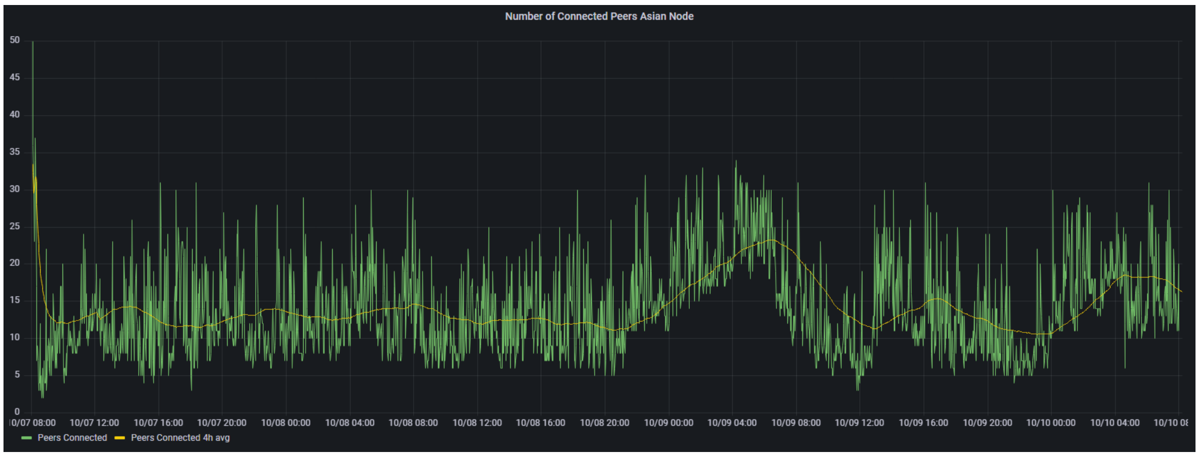
Figure A.4: Number of connected peers in the normal environment.



(a) European Node

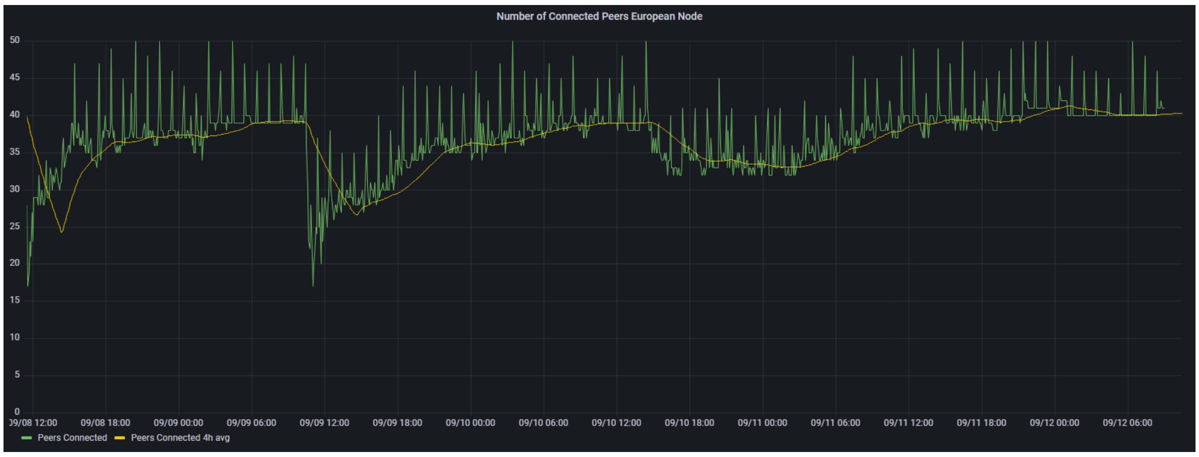


(b) North American Node

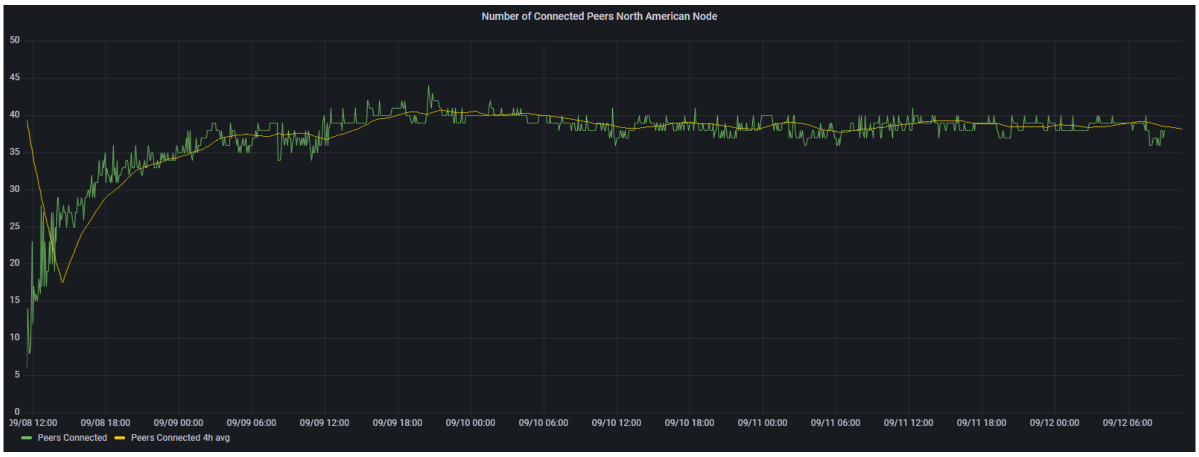


(c) Asian Node

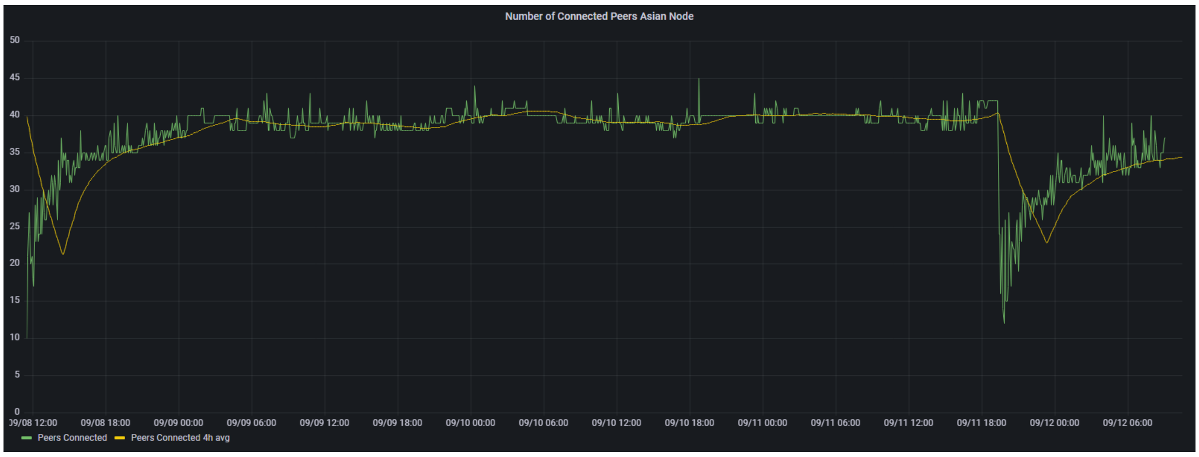
Figure A.5: Number of connected peers in the Tor environment.



(a) European Node



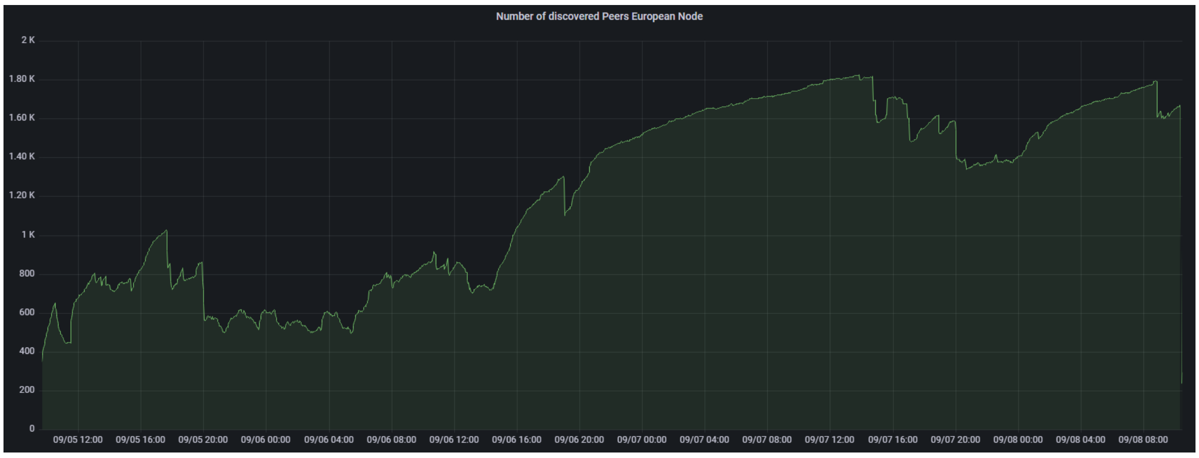
(b) North American Node



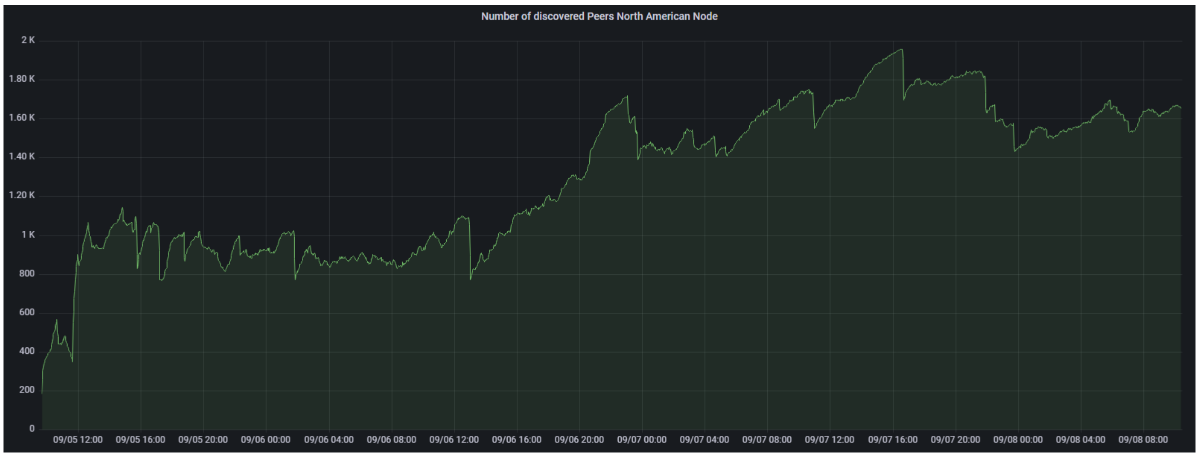
(c) Asian Node

Figure A.6: Number of connected peers in the VPN environment.

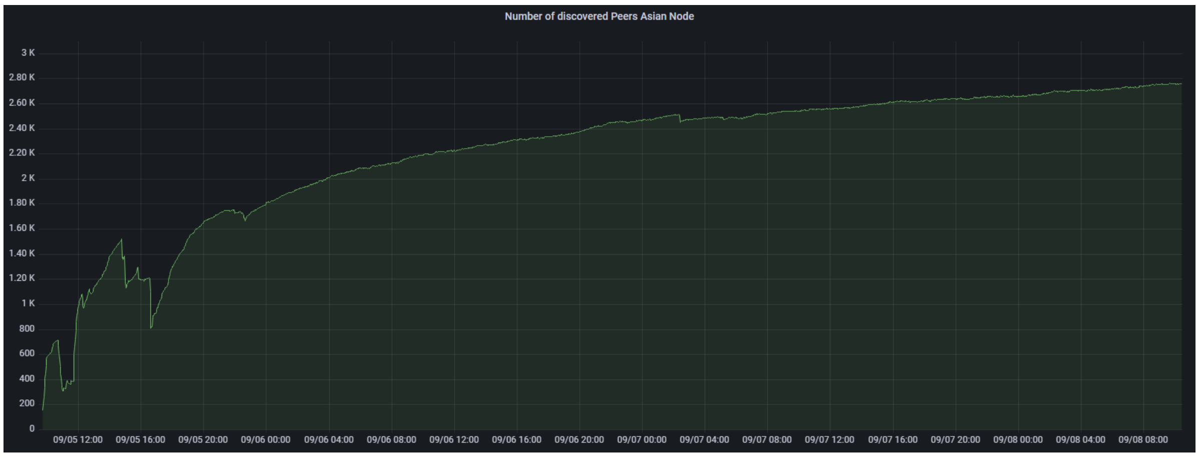
A.3 Number of Discovered Peers



(a) European Node

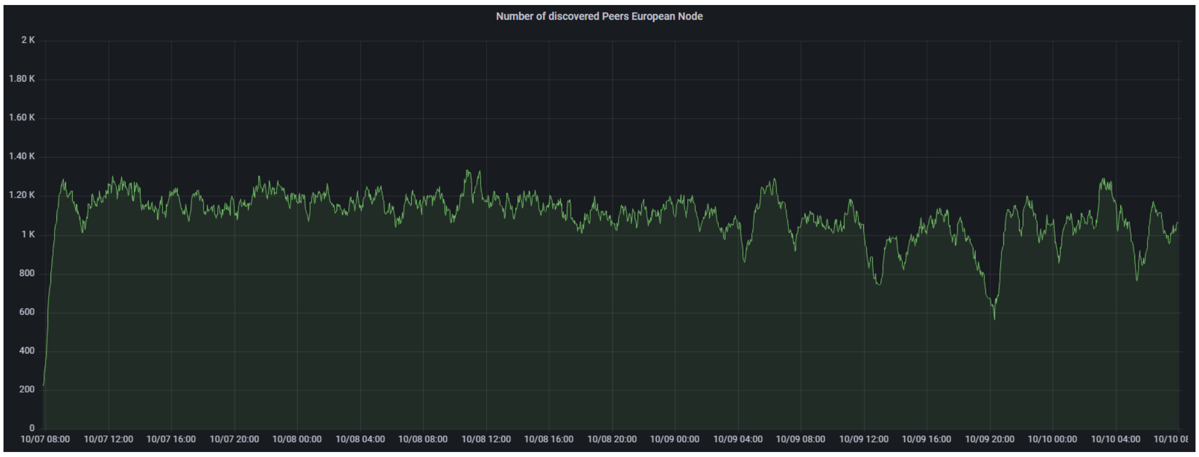


(b) North American Node

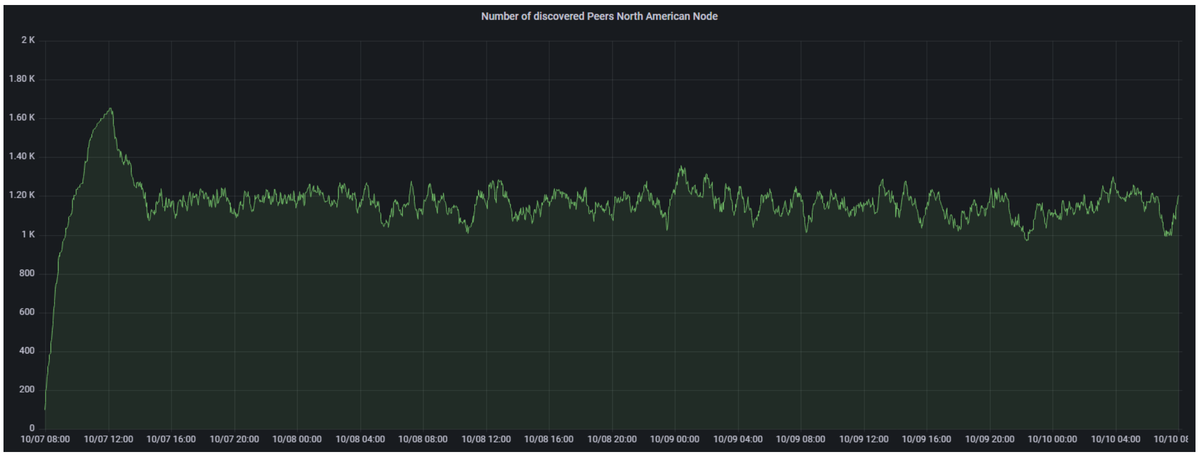


(c) Asian Node

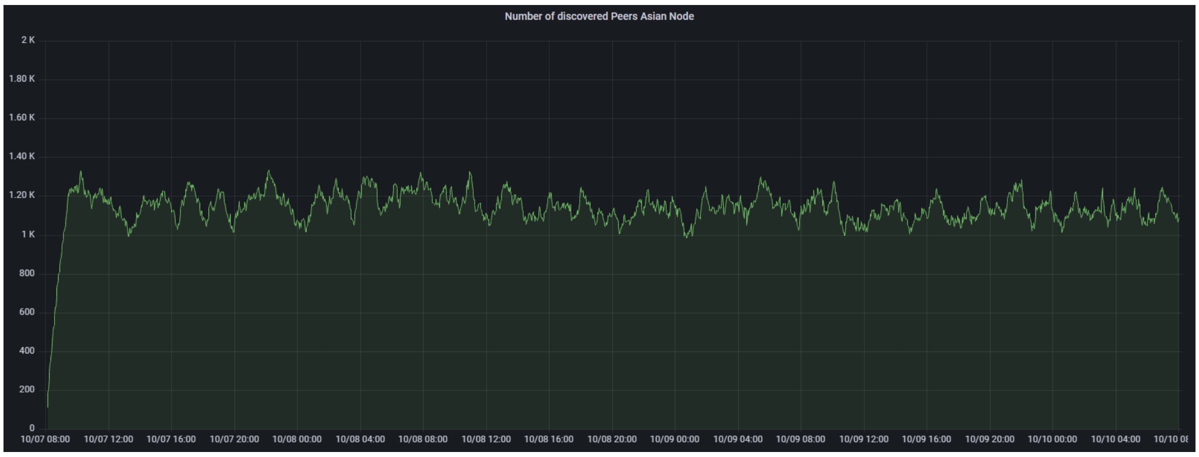
Figure A.7: Number of discovered peers in the normal environment.



(a) European Node

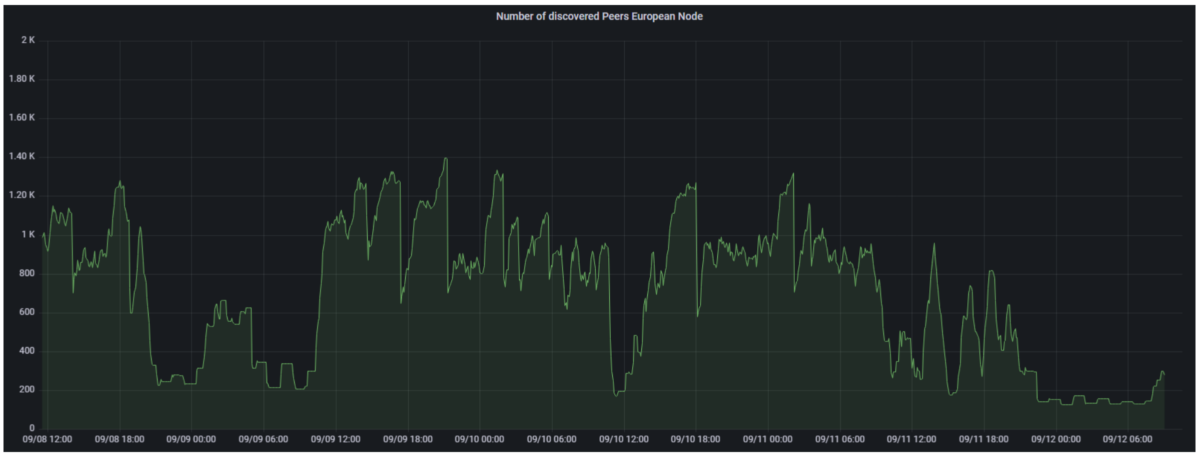


(b) North American Node

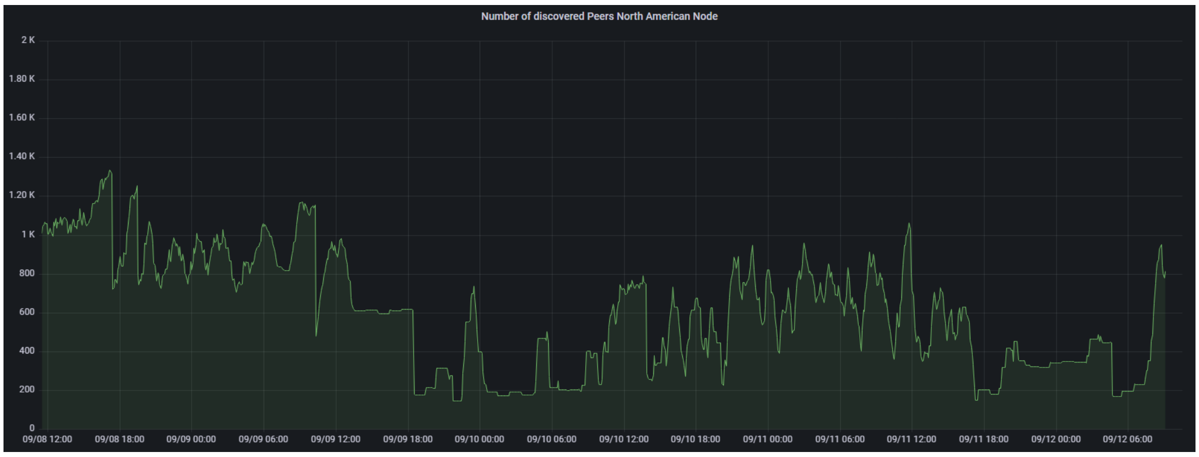


(c) Asian Node

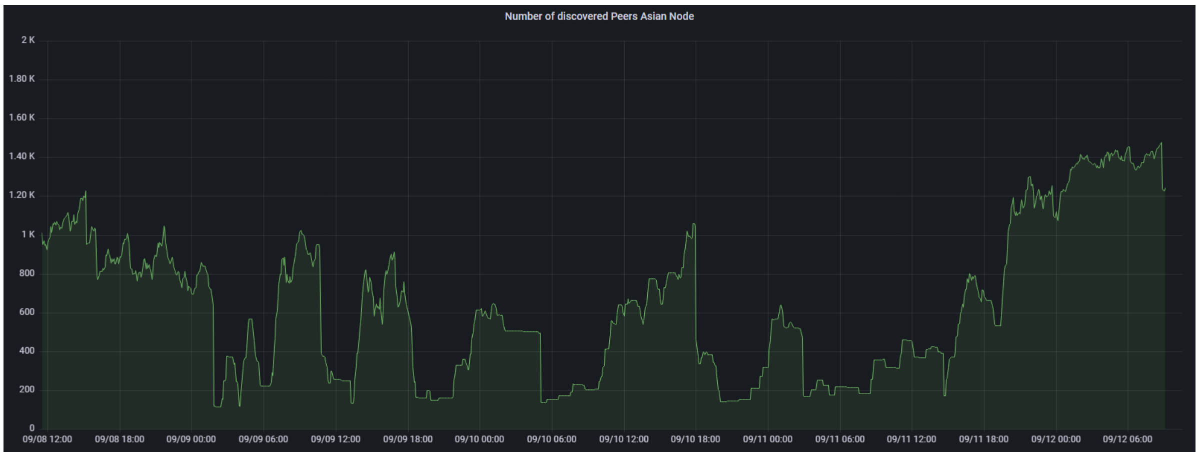
Figure A.8: Number of discovered peers in the Tor environment.



(a) European Node



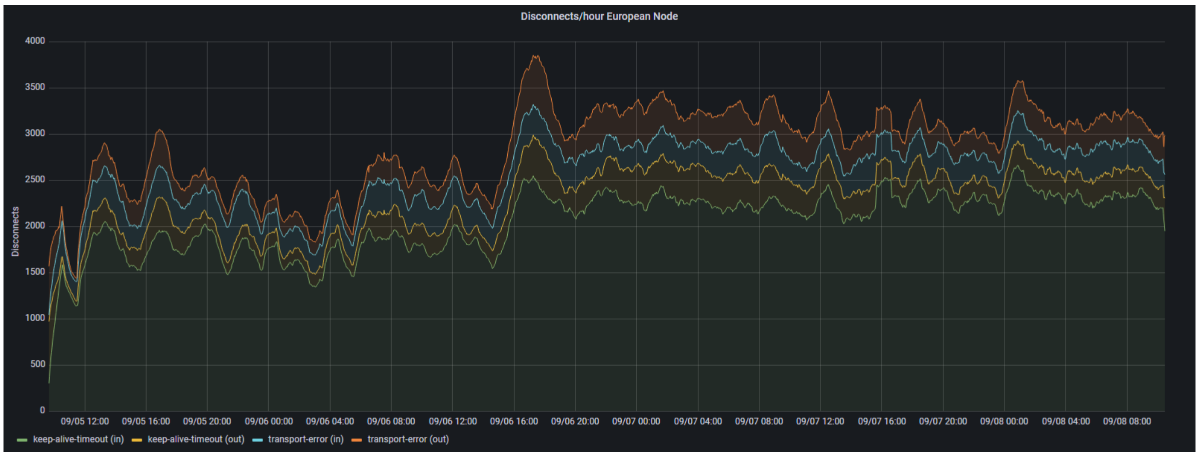
(b) North American Node



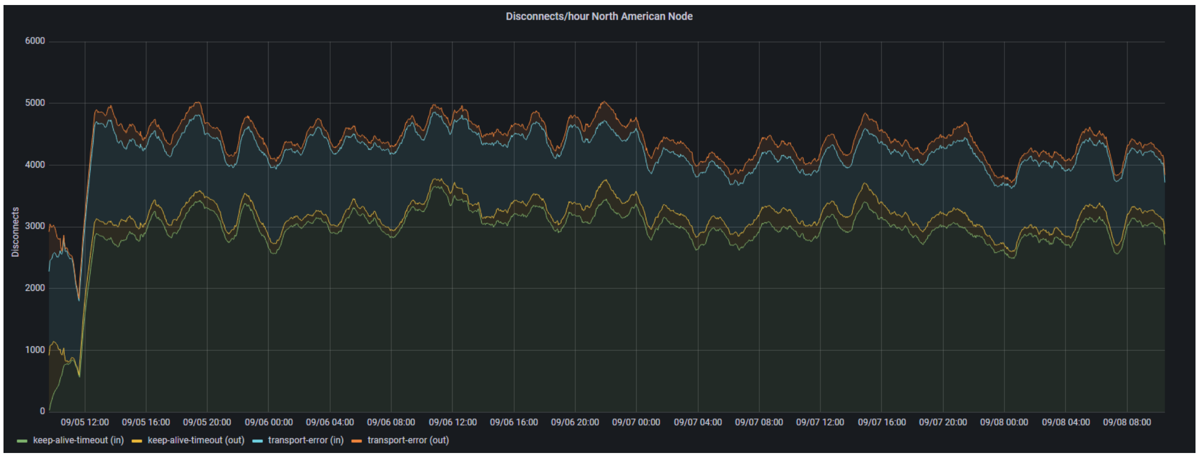
(c) Asian Node

Figure A.9: Number of discovered peers in the VPN environment.

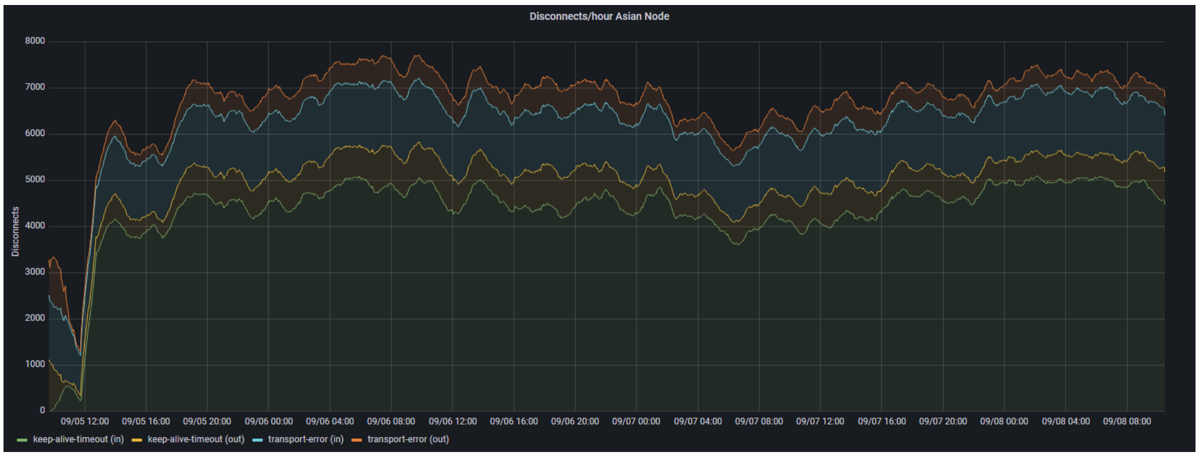
A.4 Disconnection Rate



(a) European Node

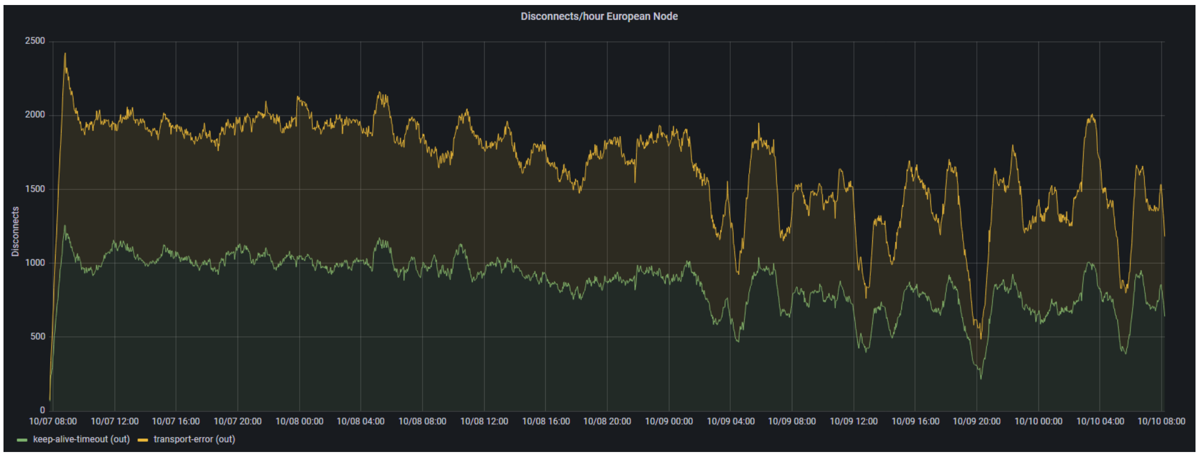


(b) North American Node

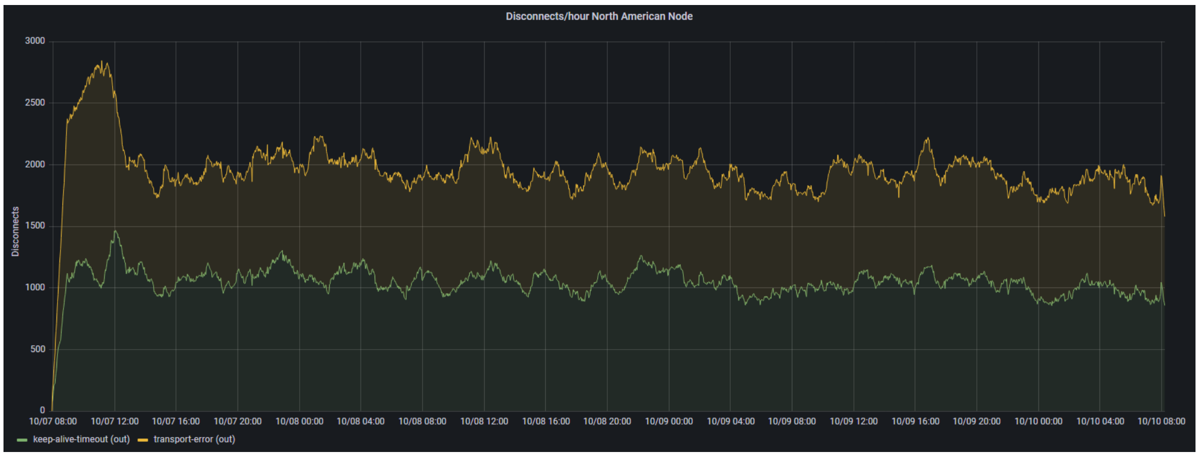


(c) Asian Node

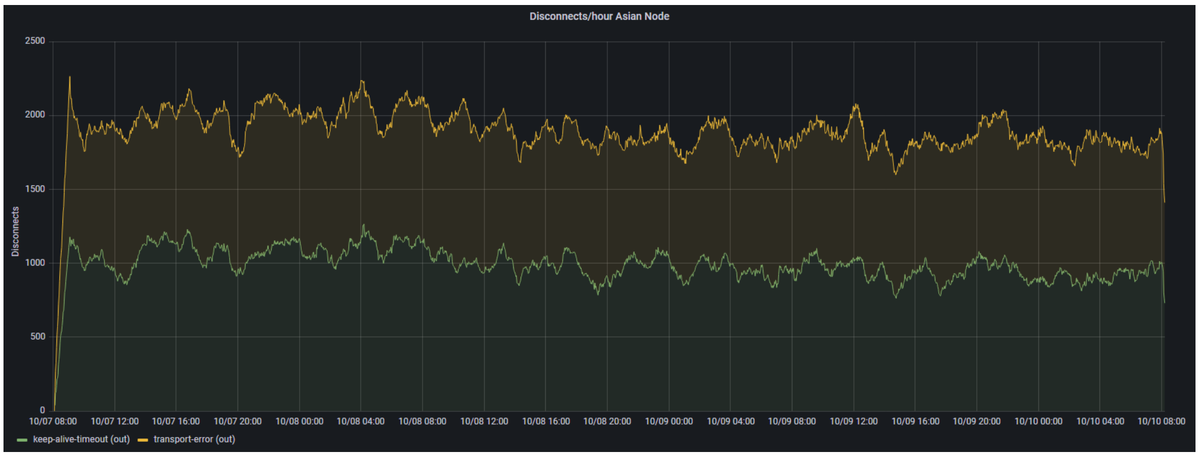
Figure A.10: Disconnection rate measured in disconnects/hour in the normal environment.



(a) European Node

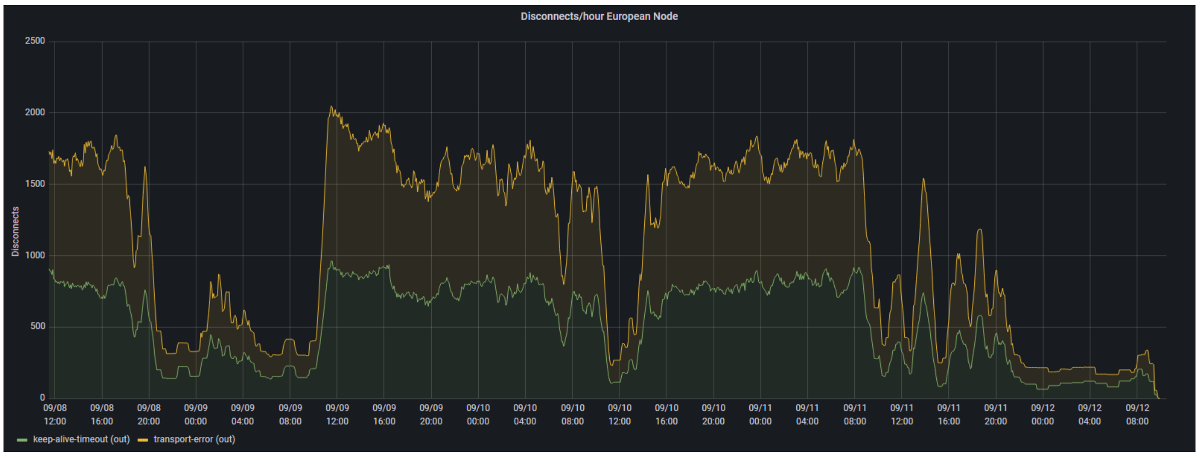


(b) North American Node

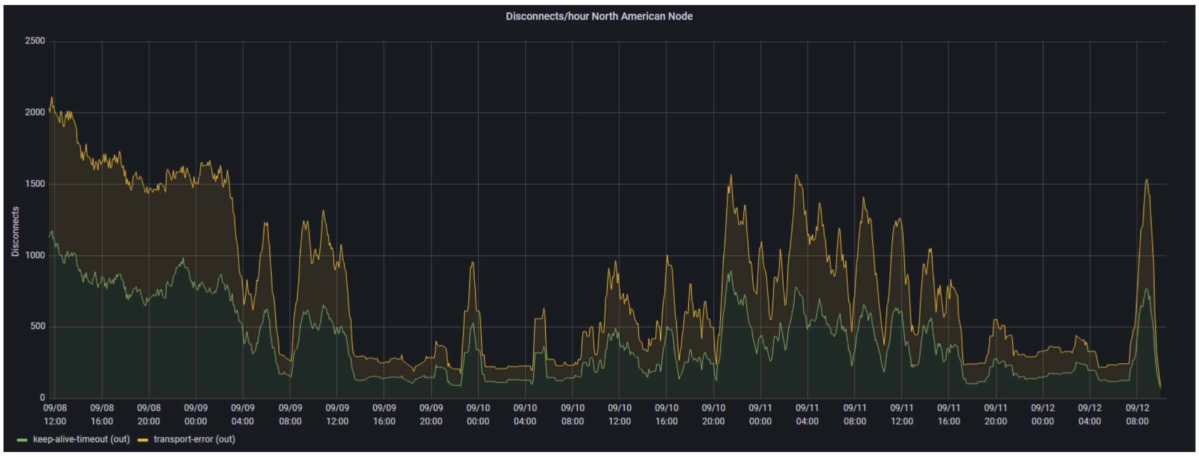


(c) Asian Node

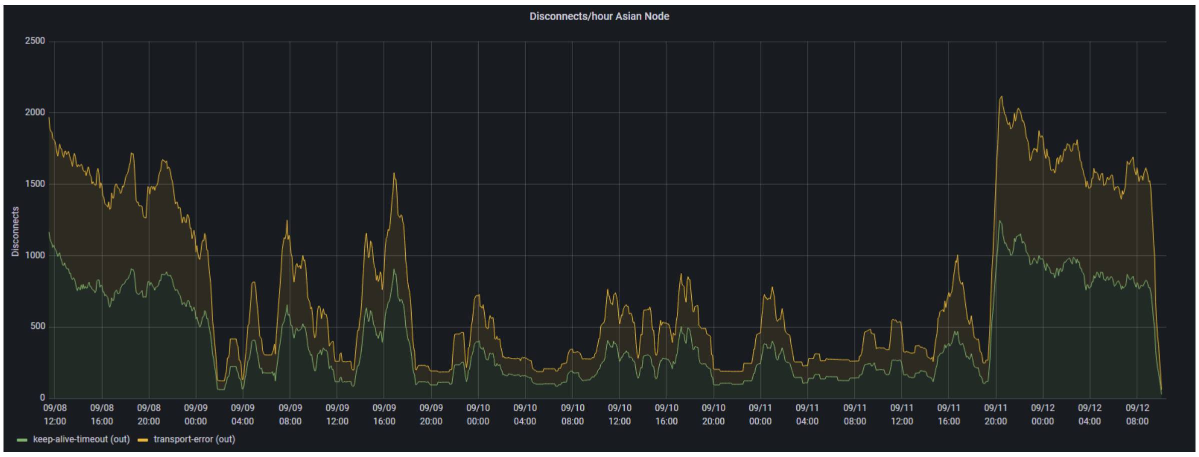
Figure A.11: Disconnection rate measured in disconnects/hour in the Tor environment.



(a) European Node



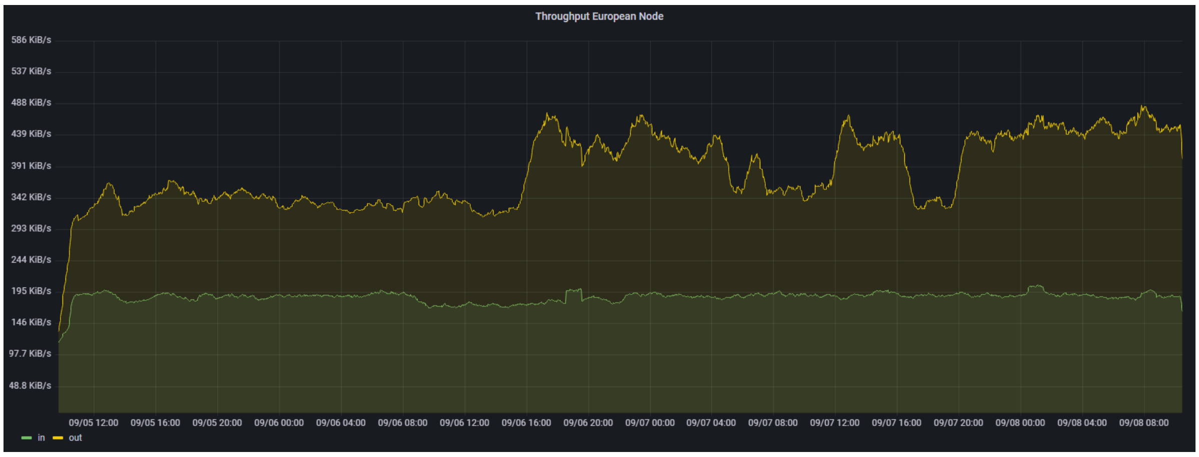
(b) North American Node



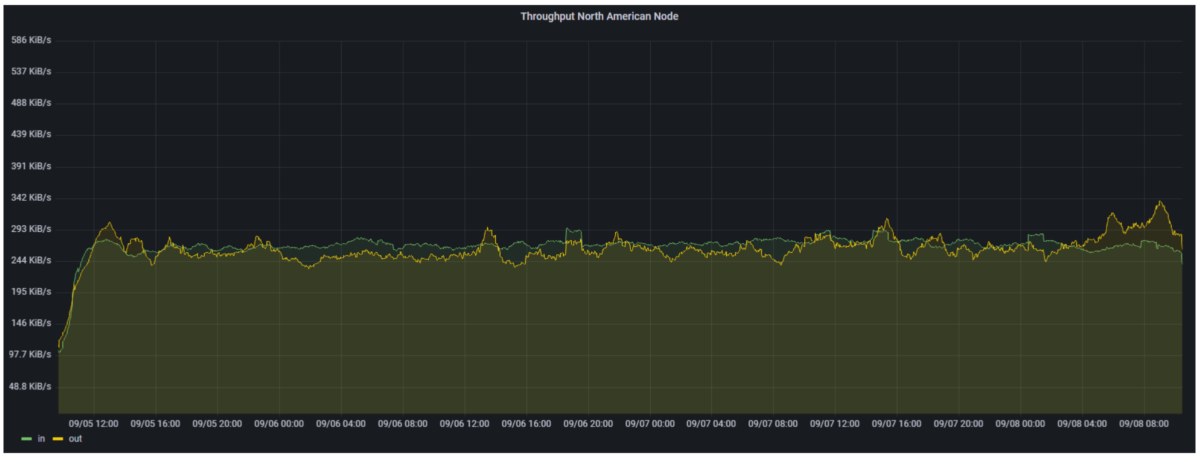
(c) Asian Node

Figure A.12: Disconnection rate measured in disconnects/hour in the VPN environment.

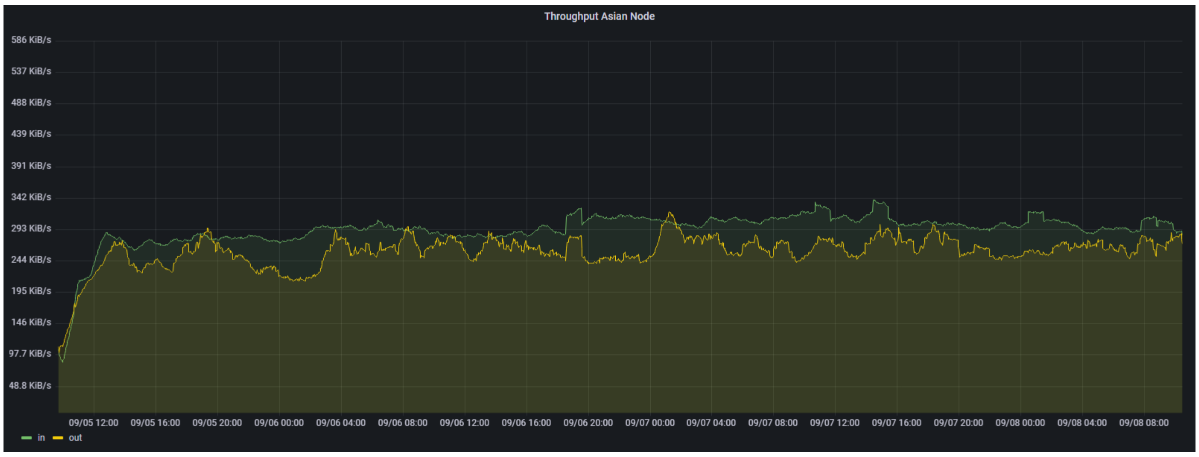
A.5 Throughput



(a) European Node

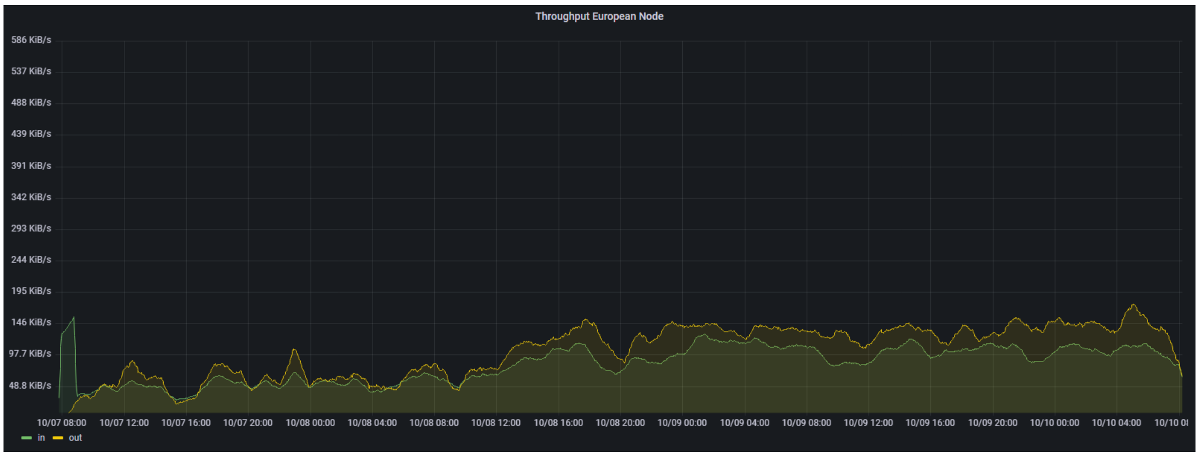


(b) North American Node

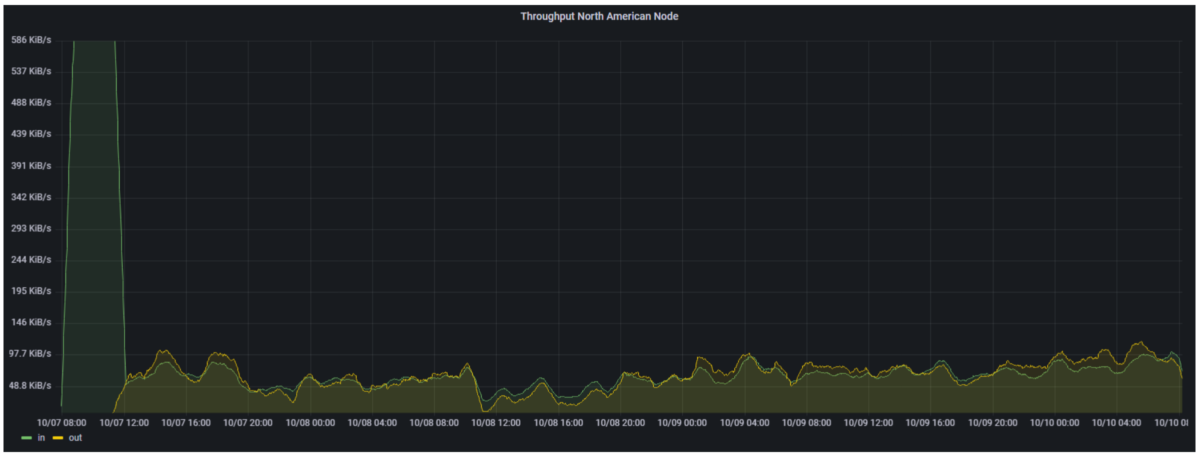


(c) Asian Node

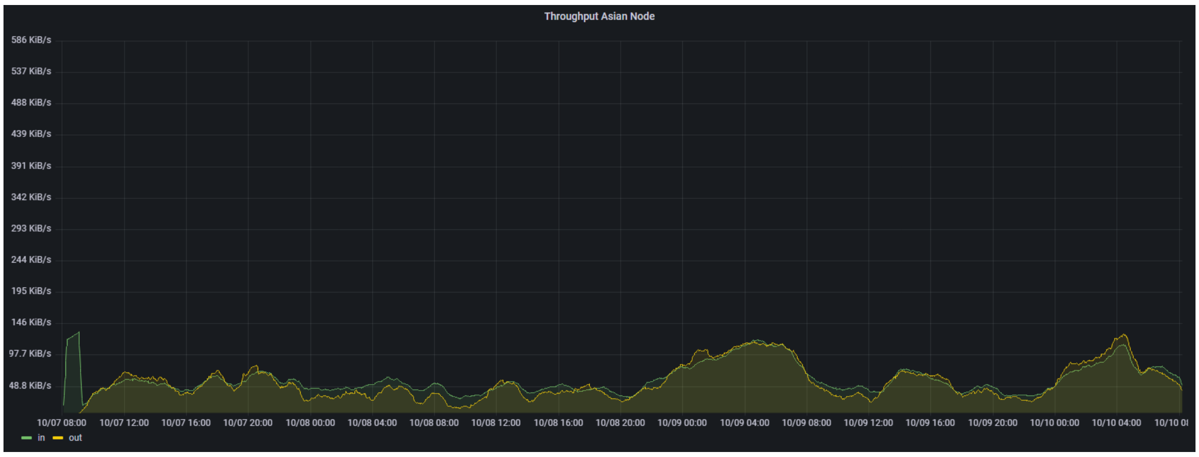
Figure A.13: Throughput in the normal environment.



(a) European Node

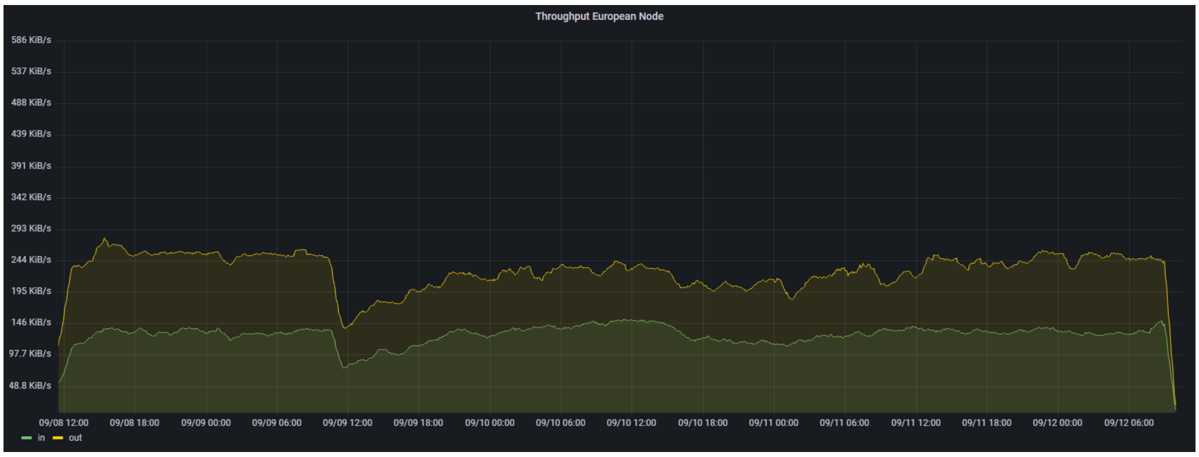


(b) North American Node

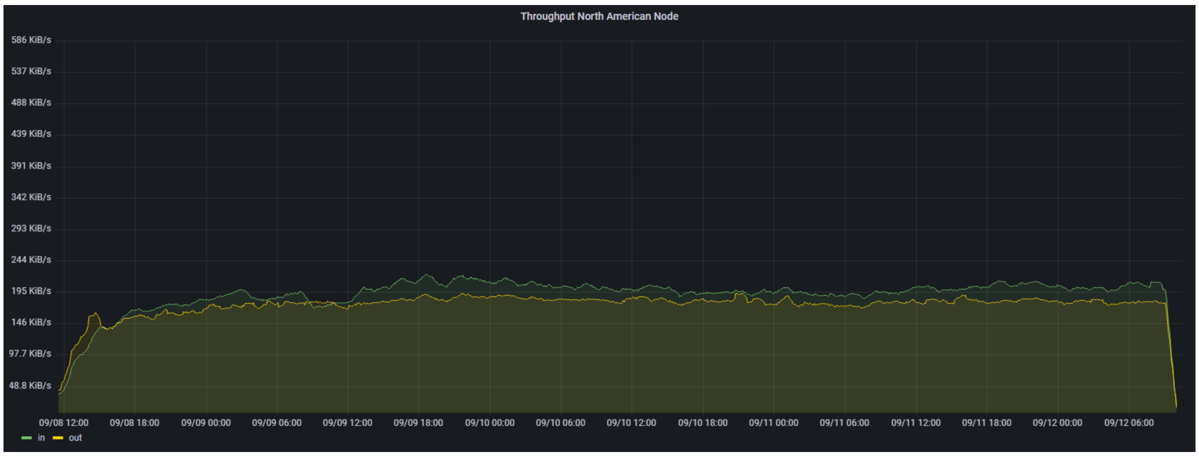


(c) Asian Node

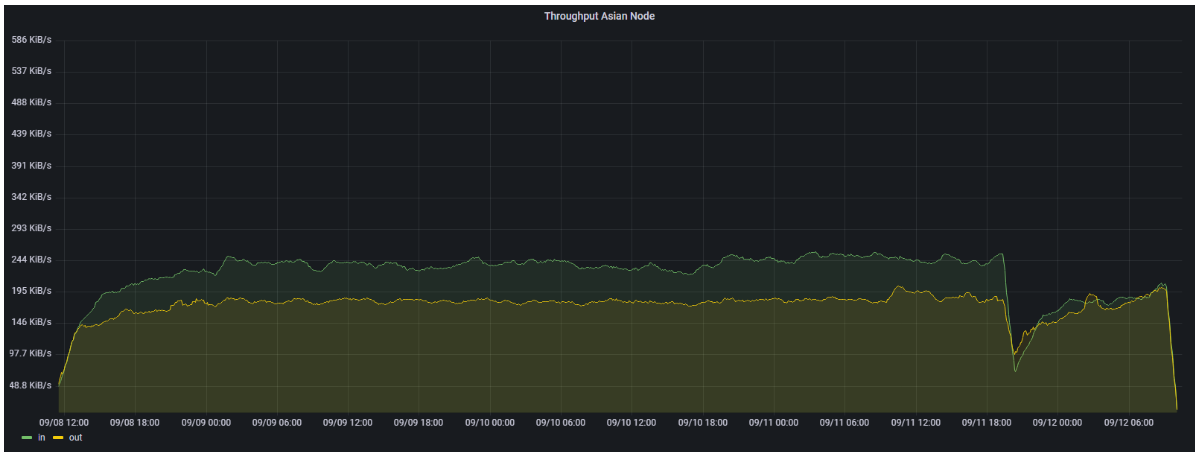
Figure A.14: Throughput in the Tor environment.



(a) European Node



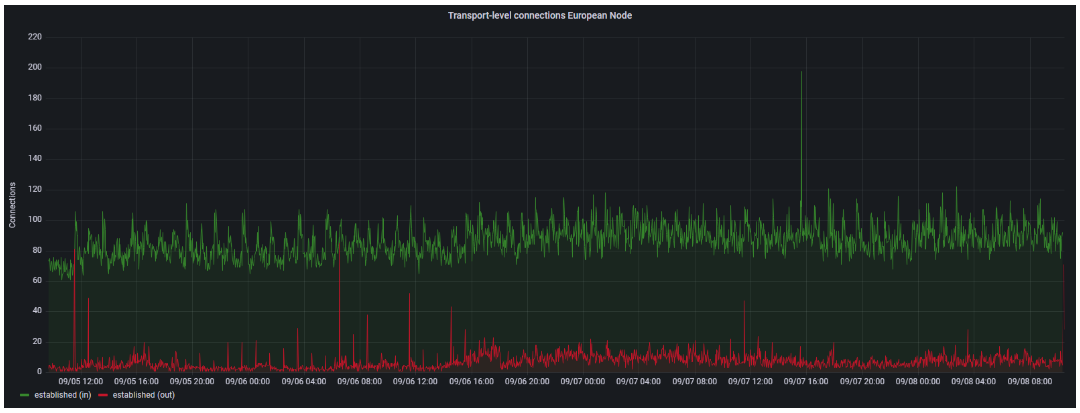
(b) North American Node



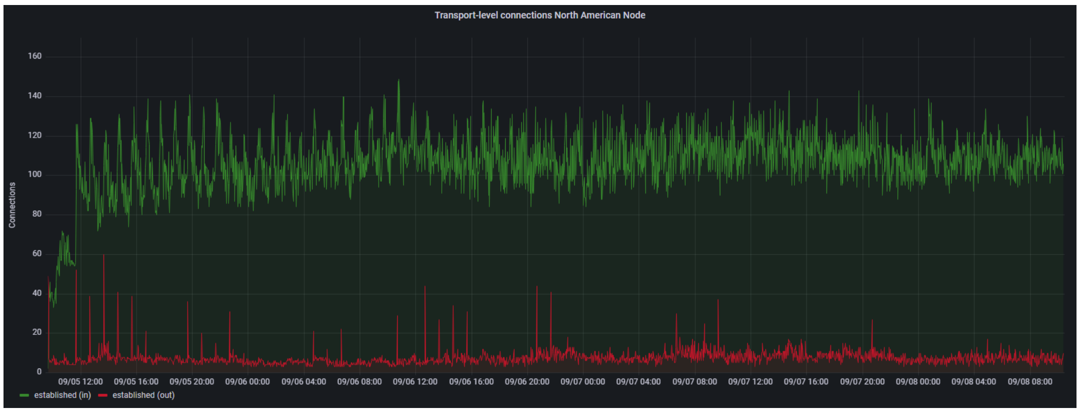
(c) Asian Node

Figure A.15: Throughput in the VPN environment.

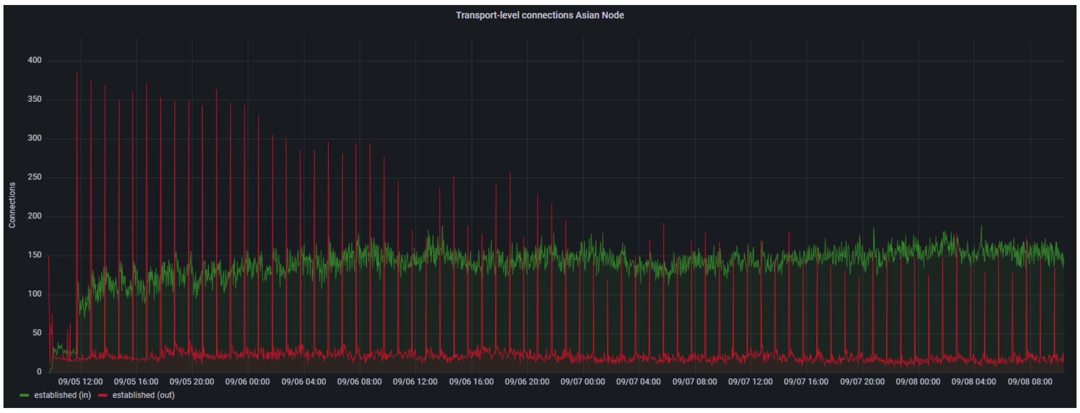
A.6 Number of Established Connections



(a) European Node

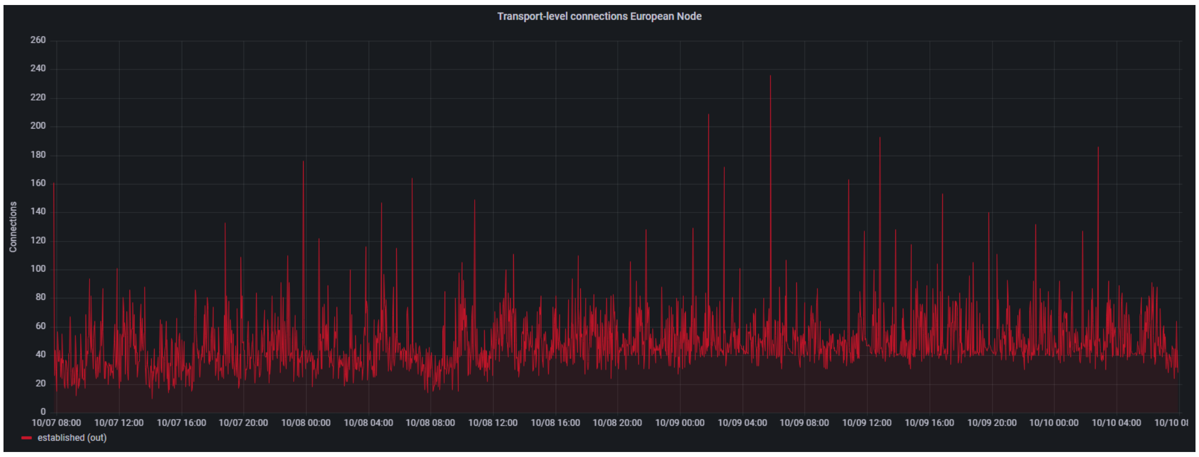


(b) North American Node

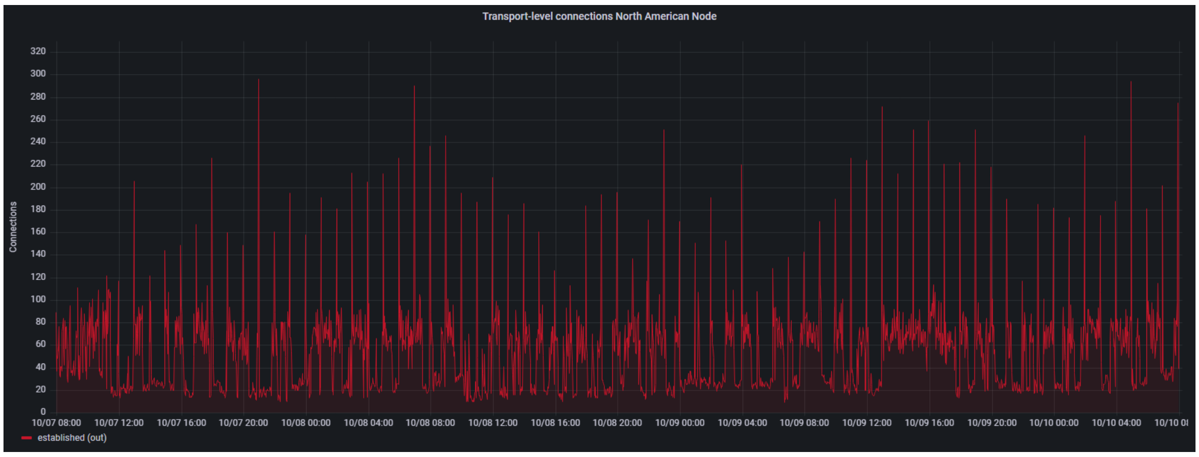


(c) Asian Node

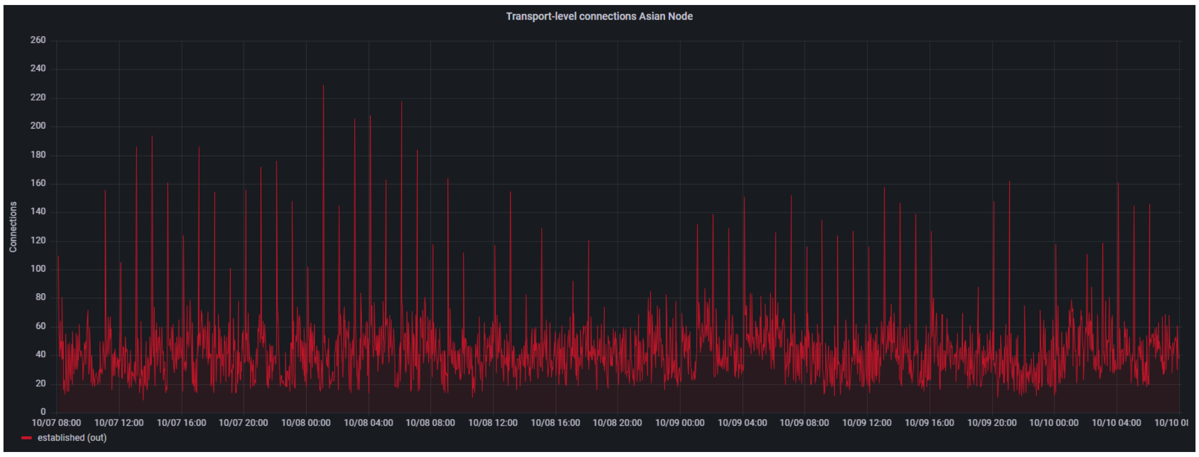
Figure A.16: Number of established connections in the normal environment.



(a) European Node

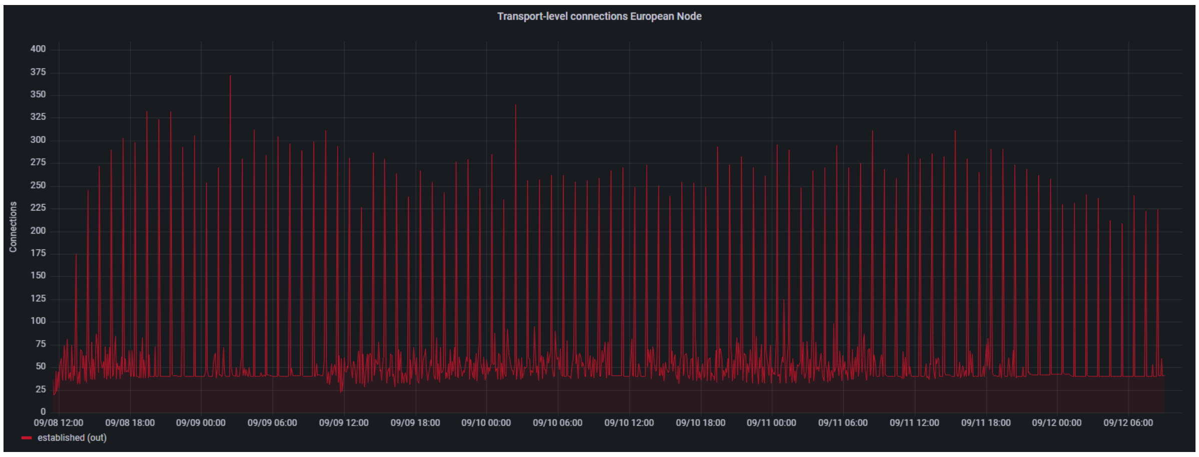


(b) North American Node

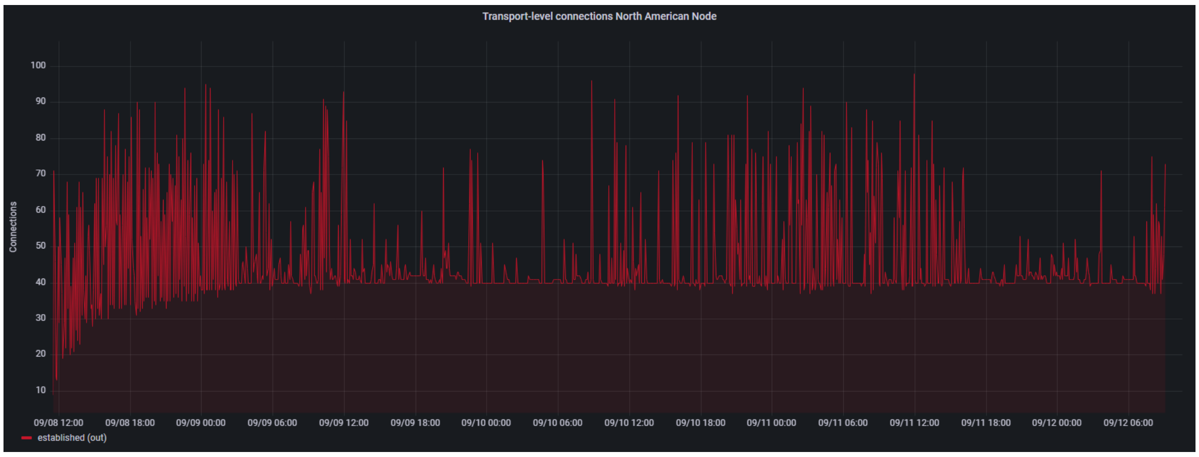


(c) Asian Node

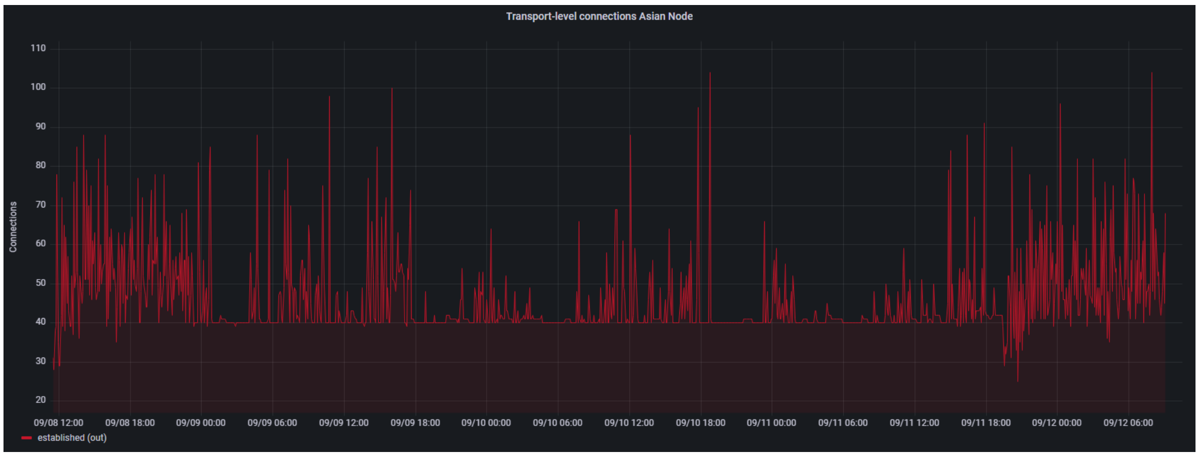
Figure A.17: Number of established connections in the Tor environment.



(a) European Node



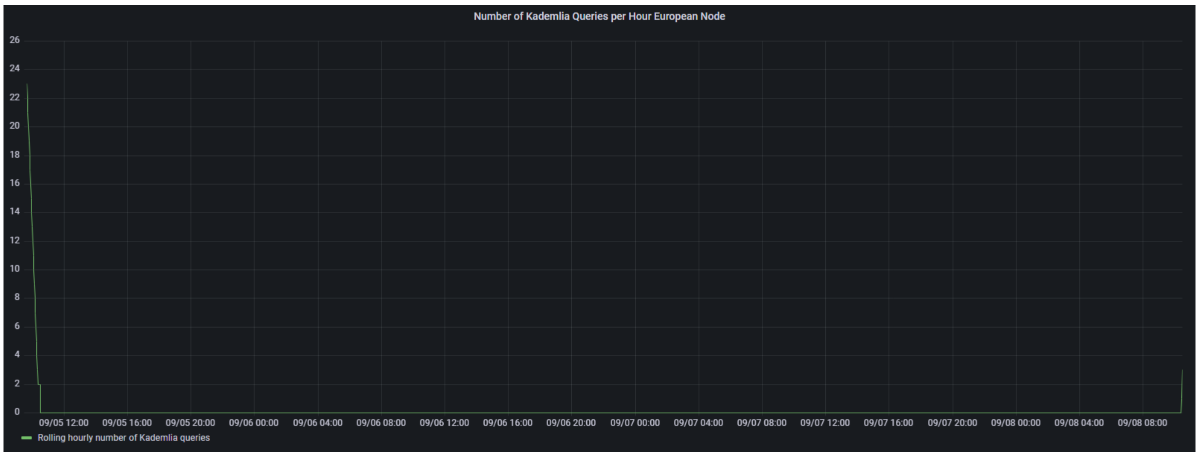
(b) North American Node



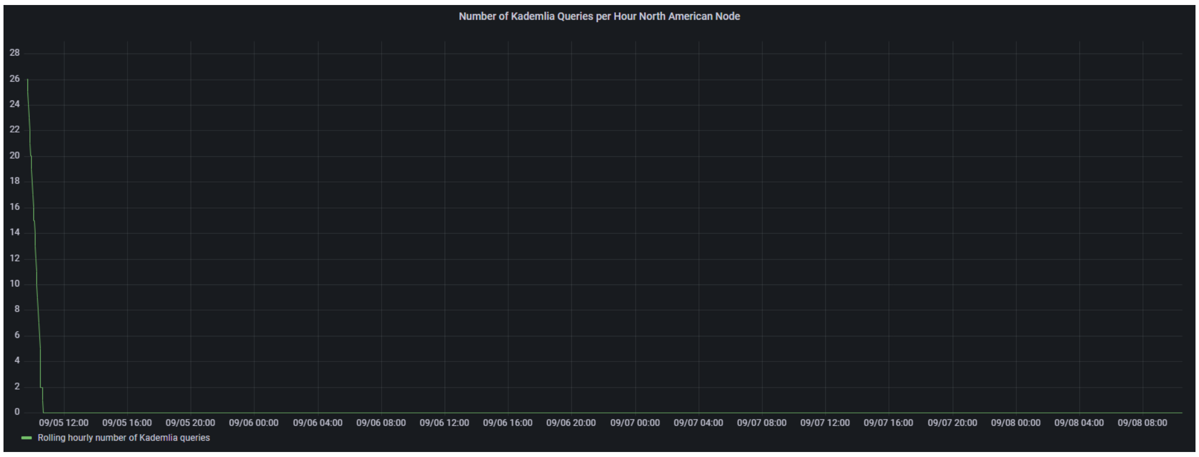
(c) Asian Node

Figure A.18: Number of established connections in the VPN environment.

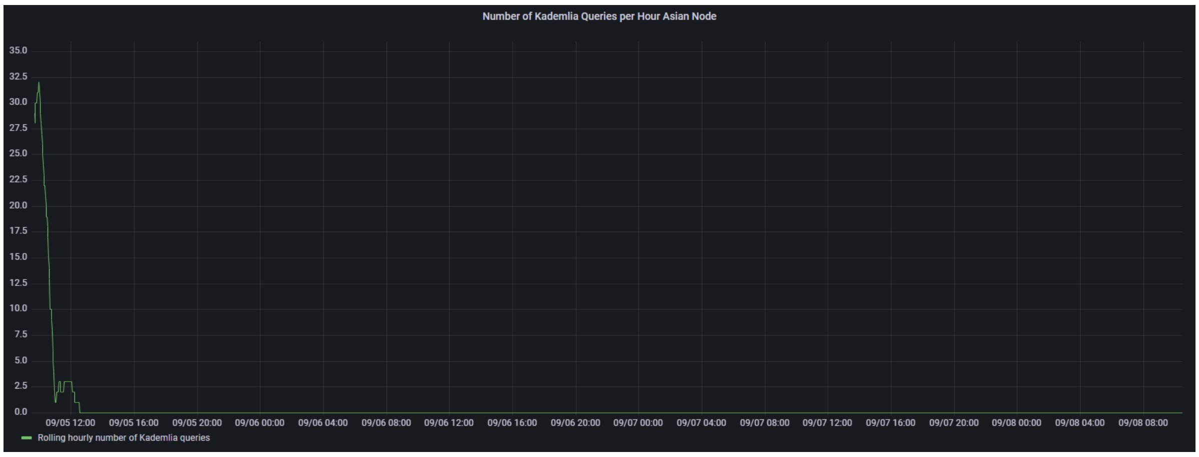
A.7 Number of Kademlia Queries per hour



(a) European Node

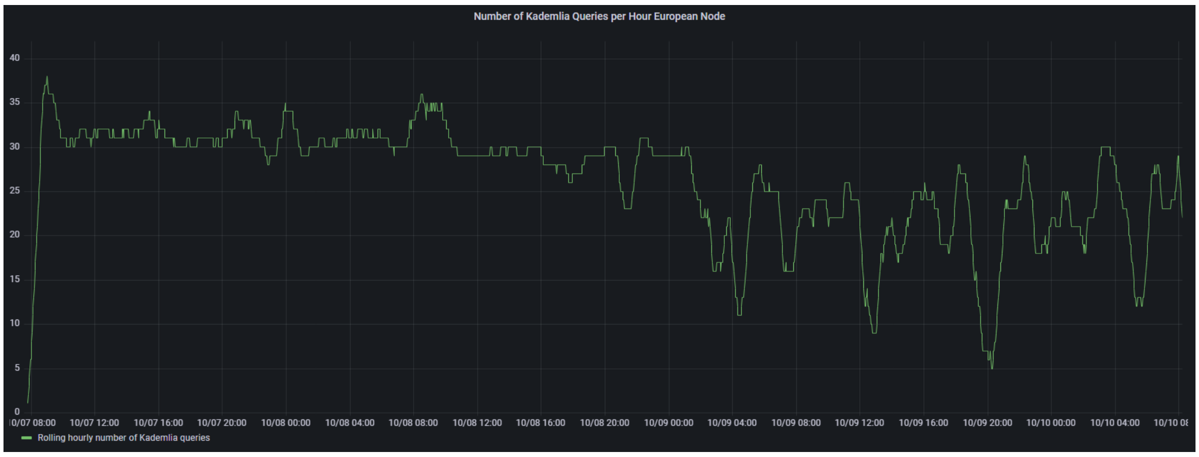


(b) North American Node

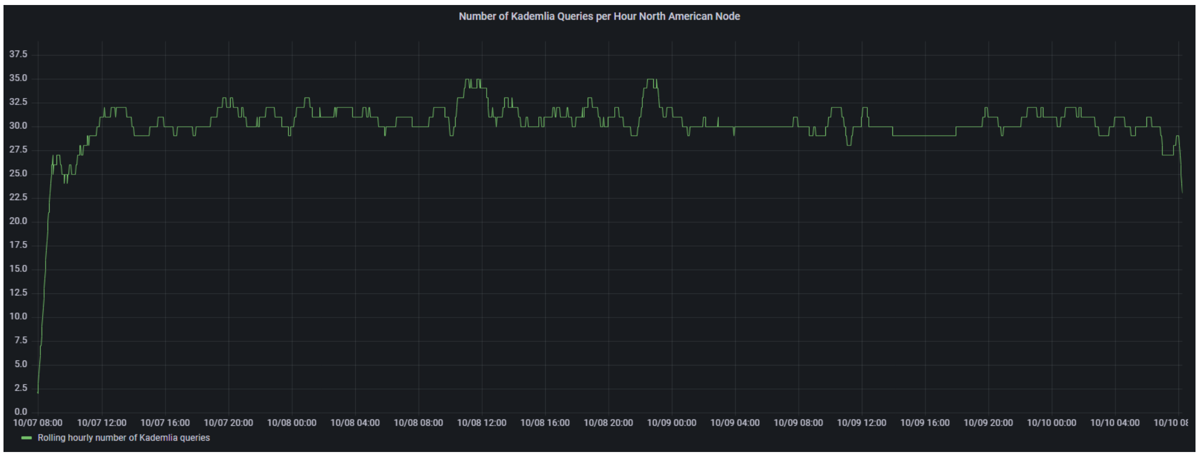


(c) Asian Node

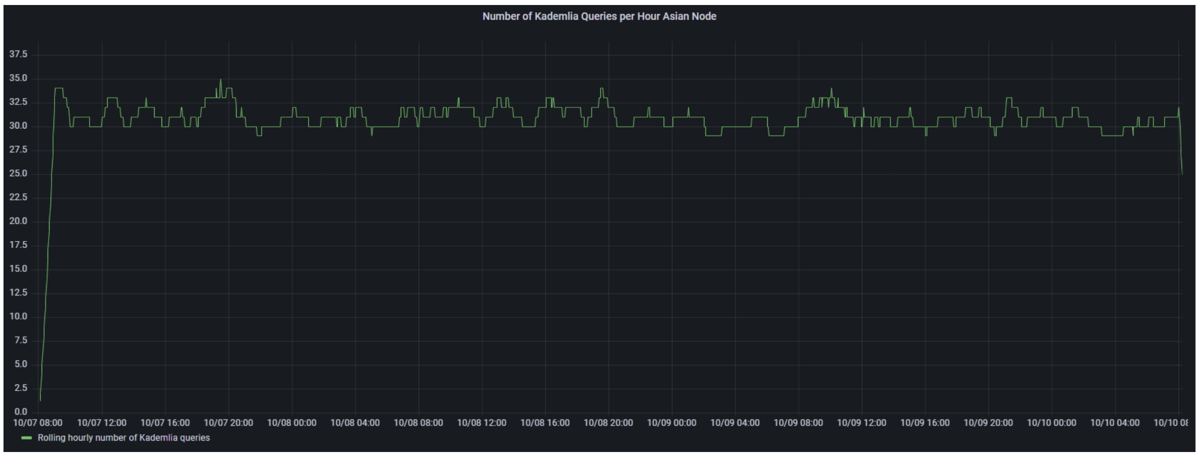
Figure A.19: Number of Kademlia queries in the normal environment.



(a) European Node

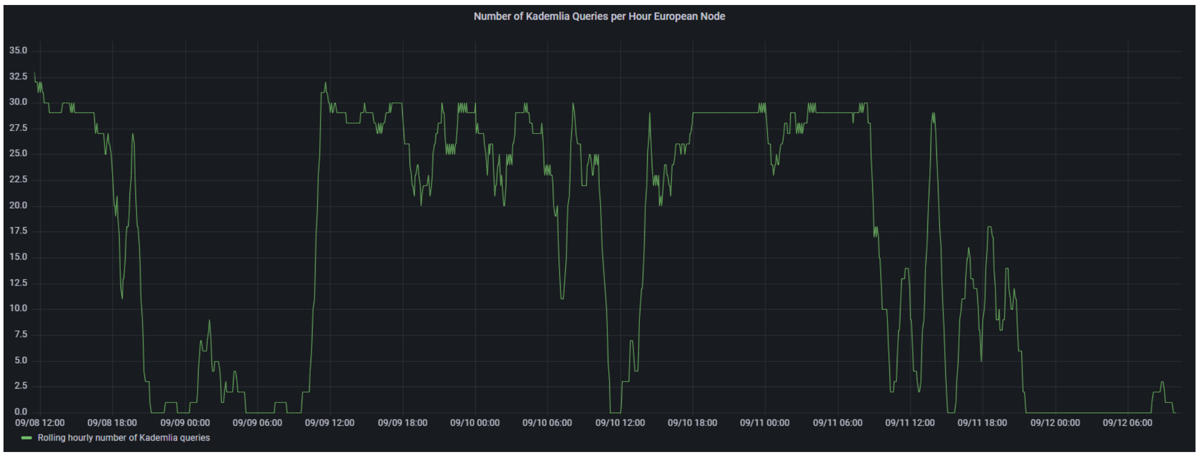


(b) North American Node

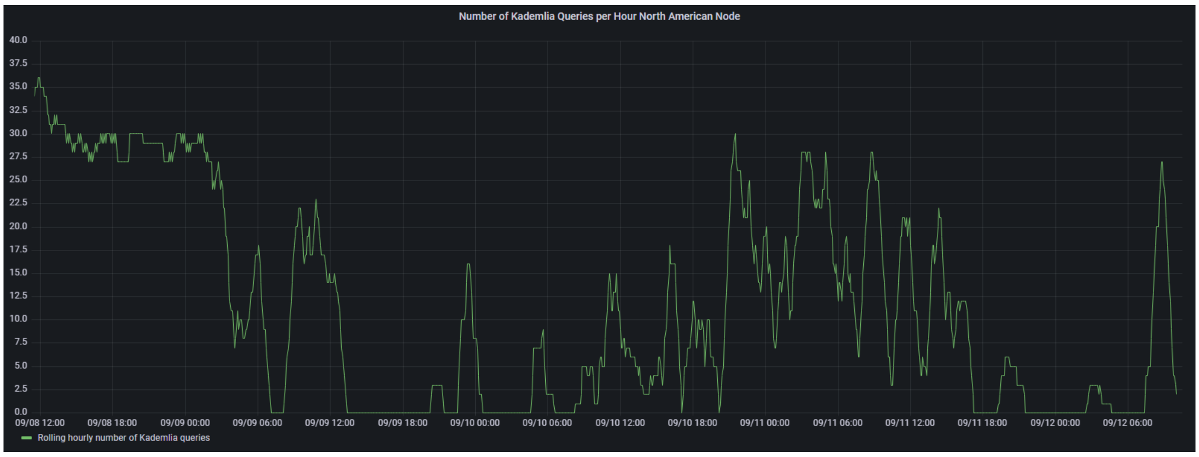


(c) Asian Node

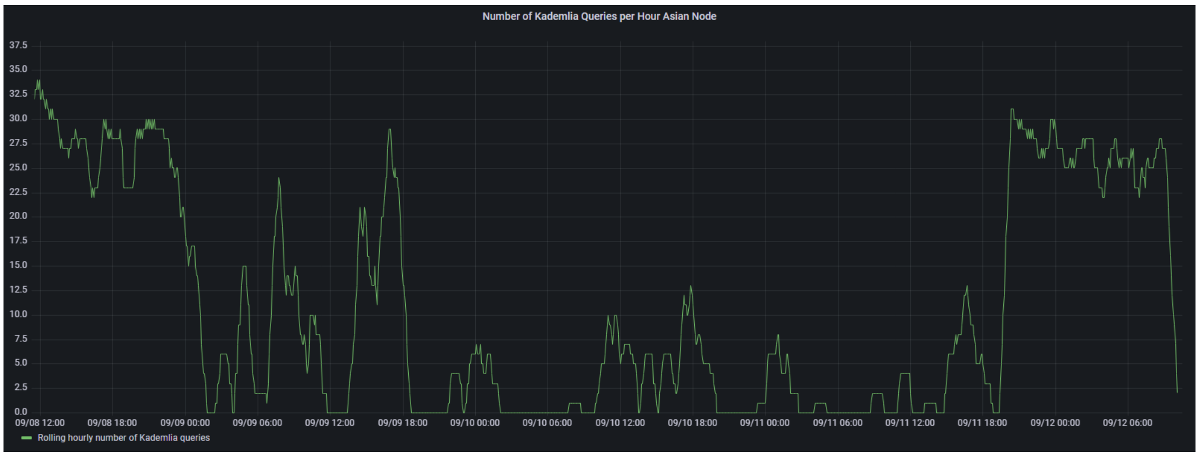
Figure A.20: Number of Kademlia queries in the Tor environment.



(a) European Node



(b) North American Node



(c) Asian Node

Figure A.21: Number of Kademlia queries in the VPN environment.

Appendix B

B.1 Grafana Queries

```
# Latency of the Network Communication
# Median
histogram_quantile(0.5,
    sum(rate(
        substrate_sub_libp2p_requests_out_success_total_bucket{
            instance=~"${nodename}",
            protocol="${request_protocol_out}"
        }
    [$__rate_interval])) by (instance, le)
) > 0

# 99th
histogram_quantile(0.99,
    sum(rate(
        substrate_sub_libp2p_requests_out_success_total_bucket{
            instance=~"${nodename}",
            protocol="${request_protocol_out}"
        }
    [$__rate_interval])) by (instance, le)
) > 0

# Number of Connected Peers over Time
# Peers Connected
substrate_sub_libp2p_peers_count{instance=~"${nodename}.*"}

# Peers Connected 4h avg
avg_over_time(
    substrate_sub_libp2p_peers_count{instance=~"${nodename}.*"}[4h]
)

# Number of Discovered Peers
substrate_sub_libp2p_peerset_num_discovered
```

```
# Disconnection Rate
# Keep-alive-timeout (in and out)
rate(
    substrate_sub_libp2p_connections_closed_total{
        instance=~"${nodename}",
        reason="keep-alive-timeout"
    }[1h]
) * 3600

# Transport-error
rate(
    substrate_sub_libp2p_connections_closed_total{
        instance=~"${nodename}",
        reason="transport-error"
    }[1h]
) * 3600

# Throughput
rate(
    substrate_sub_libp2p_network_bytes_total{
    instance=~"${nodename}"
    }[1h]
)

# Number of Kademlia Queries
rate(
    substrate_sub_libp2p_kademlia_random_queries_total[1h]
) * 3600

#  Number of Transport Layer Connections
(
    sum(
        substrate_sub_libp2p_connections_opened_total{
            direction="in",
            instance=~"${nodename}"
        }
    ) by (instance) -
    sum(
        substrate_sub_libp2p_connections_closed_total{
            direction="in",
            instance=~"${nodename}"
        }
    ) by (instance)
)
```

Listing 5: The Prometheus queries used for the graphs in the Grafana dashboard.