Proof of Latency Using a Verifiable Delay Function

Master's Thesis
University of Turku
Department of Computing
2022
Jani Anttonen

UNIVERSITY OF TURKU
Department of Computing

Jani Anttonen: Proof of Latency Using a Verifiable Delay Function

Master's Thesis, 75 p., 0 app. p.
Department of Computing
April 2022

In this thesis I present an interactive public-coin protocol called Proof of Latency (PoL) that aims to improve connections in peer-to-peer networks by measuring latencies with logical clocks built from verifiable delay functions (VDF). PoL is a tuple of three algorithms, $Setup(e, \lambda)$, $VCOpen(c, e)$, and $Measure(g, T, l_p, l_v)$. Setup creates a vector commitment (VC), from which a vector commitment opening corresponding to a collaborator's public key is taken in VCOpen, which then gets used to create a common reference string used in Measure. If no collusion gets detected by neither party, a signed proof is ready for advertising.
PoL is agnostic in terms of the individual implementations of the VC or VDF used. This said, I present a proof of concept in the form of a state machine implemented in Rust that uses RSA-2048, Catalano-Fiore vector commitments and Wesolowski's VDF to demonstrate PoL.
As VDFs themselves have been shown to be useful in timestamping, they seem to work as a measurement of time in this context as well, albeit requiring a public performance metric for each peer to compare to during the measurement. I have imagined many use cases for PoL, like proving a geographical location, working as a benchmark query, or using the proofs to calculate VDFs with the latencies between peers themselves. As it stands, PoL works as a distance bounding protocol between two participants, considering their computing performance is relatively similar. More work is needed to verify the soundness of PoL as a publicly verifiable proof that a third party can believe in.

Keywords: verifiable delay functions, peer-to-peer networks, vector commitments, distance bounding protocols

TURUN YLIOPISTO
Teknillinen Tiedekunta

Jani Anttonen: Proof of Latency Using a Verifiable Delay Function

M.Sc., 75 s., 0 liites.

Huhtikuu 2022

Tässä tutkielmassa esitän interaktiivisen protokollan nimeltä Proof of latency (PoL), joka pyrkii parantamaan yhteyksiä vertaisverkoissa mittaamalla viivettä todennettavasta viivefunktiosta rakennetulla loogisella kellolla. Proof of latency koostuu kolmesta algoritmistä, $Setup(e, \lambda)$, $VCOpen(c, e)$ ja $Measure(g, T, l_p, l_v)$. Setup luo vektorisitoumuksen, josta luodaan avaus algoritmissa VCOpen avaamalla vektorisitoumus indeksistä, joka kuvautuu toisen mittaavan osapuolen julkiseen avaimeen. Tätä avausta käytetään luomaan yleinen viitemerkkijono, jota käytetään algoritmissa Measure alkupisteenä molempien osapuolien todennettavissa viivefunktioissa mittaamaan viivettä. Jos kumpikin osapuoli ei huomaa virheitä mittauksessa, on heidän allekirjoittama todistus valmis mainostettavaksi vertaisverkossa.
PoL ei ota kantaa sen käyttämien kryptografisten funktioiden implementaatioon. Tästä huolimatta olen ohjelmoinut protokollasta prototyypin Rust-ohjelmointikielellä käyttäen RSA-2048:tta, Catalano-Fiore–vektorisitoumuksia ja Wesolowskin todennettavaa viivefunktiota protokollan esittelyyn.
Todistettavat viivefunktiot ovat osoittaneet hyödyllisiksi aikaleimauksessa, mikä näyttäisi osoittavan niiden soveltumisen myös ajan mittaamiseen tässä konteksissa, huolimatta siitä että jokaisen osapuolen tulee ilmoittaa julkisesti teholukema, joka kuvaa niiden tehokkuutta viivefunktioiden laskemisessa. Toinen osapuoli käyttää tätä lukemaa arvioimaan, että valehteliko toinen viivemittauksessa. Olen kuvitellut monta käyttökohdetta PoL:lle, kuten maantieteellisen sijainnin todistaminen, suorituskykytestaus, tai itse viivetodistuksien käyttäminen uusien viivetodistusten laskemisessa vertaisverkon osallistujien välillä.
Tällä hetkellä PoL toimii etäisyydenmittausprotokollana kahden osallistujan välillä, jos niiden suorituskyvyt ovat tarpeeksi lähellä toisiaan. Protokolla tarvitsee lisätutkimusta sen suhteen, voiko se toimia uskottavana todistuksena kolmansille osapuolille kahden vertaisverkon osallistujan välisestä viiveestä.

Asiasanat: todennettavat viivefunktiot, vertaisverkot, vektorisitoumukset, etäisyydenmittausprotokollat

Contents

List of Figures

1 Introduction 1

1.1 Contribution 4

1.2 Related Work 6

1.2.1 FOAM 7

1.2.2 GoAT 7

2 Background 8

2.1 Cryptography 8

2.1.1 Cryptographic Proofs and Their Soundness 9

2.1.2 RSA 10

2.1.3 RSA Accumulators 13

2.1.4 Commitment Schemes 14

2.2 Peer-to-Peer Networking 16

2.3 Role of Latency in Distributed Systems 20

2.3.1 Ad Hoc and Zero Configuration Networking 22

2.3.2 Distributed Hash Tables 23

2.3.3 Eclipse Attacks 27

2.3.4 Sybil Attacks 28

2.4 Distance Bounding Protocols 28

2.4.1 Attacks . . . 31
2.5 Verifiable Delay Functions . . . 31
2.5.1 Use Cases . . . 33
2.5.2 Precursors to VDFs . . . 36
2.5.3 Variations . . . 37
2.5.4 Constructs Related to VDFs . . . 40
2.5.5 Hardware Developments . . . 41

3 Design and Architecture 43
3.1 Protocol Description . . . 45
3.1.1 Setup . . . 46
3.1.2 VC Opening . . . 47
3.1.3 Distance Bounding . . . 47
3.1.4 Finalizing . . . 50
3.2 Use Cases . . . 51
3.2.1 Dynamic P2P Routing . . . 52
3.2.2 Benchmark Queries . . . 52
3.2.3 VDF Calculation with Proved Latencies . . . 53
3.2.4 Proving Geographical Locations . . . 54
3.3 Security Analysis . . . 54
3.3.1 Advertising Dishonest Peers and Proof Spoofing . . . 54
3.3.2 Performance Matching . . . 55
3.4 Reducing the Attack Vectors . . . 56
3.4.1 Hiding Information of PoL Results . . . 56
3.4.2 Web of Trust . . . 56
3.4.3 Peer Scoring and Usage Optimizations . . . 57
3.4.4 Using a Third Party Validator . . . 57

3.4.5 Using in Conjunction with String Similarity Based Peer Discovery 57

4 Proof of Concept 59

4.1 Tests 63

5 Conclusion 64

5.1 Future Considerations 65

6 Aknowledgements 66

References 67

List of Figures

2.1 An example topography of an unstructured overlay network. 18

2.2 An example topography of a structured overlay network. 19

2.3 Polyring network topology.[45] 26

2.4 Uniring network topology.[45] 26

2.5 Geographically clustered polyring topology.[45] 27

2.6 A distance bounding protocol [50] 29

3.1 Example PoL network topology. Localized and highly connected clusters with high performance bridges, resulting in optimal routing . . . 43

3.2 A high abstraction view of Proof of Latency. 44

3.3 Distance bounding, Proof of Latency. 49

3.4 Possible states for the state machines. 50

3.5 Example suboptimal routes achieved by randomly selecting peers. . . 51

4.1 Software roles 60

4.2 Usage of Proof of Latency as a library in a P2P context. 61

Listings

4.1 VDF iteration logic 61
4.2 VDF proof generation 62
4.3 VDF proof verification 62

Chapter 1

Introduction

Networked computer applications, whether they are on the public Internet or on a private network, commonly use the client-server model of communication. In this model, the client application sends requests to the server, and the server responds. Applications are today, however, increasingly moving towards a distributed, peer-to-peer (P2P) networked model, where every peer, whether it is an entire computer or merely a process running on one, serves equally as the both sides of the client-server model. Many uses of P2P are not visible to the end user, because in many cases, P2P technologies serve to optimize resource utilization rather than as centerpoints of applications. For example, the music subscription service Spotify's protocol has been designed to combine server and P2P streaming to decrease the load on Spotify's servers and save bandwith [1], and thus improve the service's scalability. This means that whenever someone listens to a song that another Spotify user is listening to or has recently listened, the Spotify client program may download parts of the song from that user instead of the server. Naturally, P2P streaming comes in handy also whenever there is a short outage, as users might not experience any stoppages to the service, at least when listening to widely streamed content.

The aspects of peer-to-peer networking that have recently gotten attention are cryptocurrency and blockchain technologies, categorized under the umbrella term of

distributed ledger technologies. Prior to the emergence of ledger technologies, peer-to-peer systems were often seen in the public light as a technology to work around regulations and for doing lawless activities. This is partly because before their use in blockchain applications, P2P systems were popularized for their use in file sharing applications like Bittorrent. Obviously, this reputation was not earned for nothing, but peer-to-peer networking's use in torrents and other file sharing networks also demonstrates its potential in content delivery networks or CDNs for short.

Peer-to-peer networks are not without problems. One persistent source of problems is routing and peer discovery. Since the users of a P2P network do not necessarily connect to a centralized and optimized server infrastructure with vast bandwidth and computation resources, the individual connections between peers can be ephemeral. P2P networks are usually also random in terms of peer discovery because of the commonly used algorithms and parameters, such as Kademlia [2] used with random peer identifiers. This results in some routes between peers to being suboptimal, and possibly in bad performance to even false states in distributed ledgers. Slow data propagation in a P2P publish-subscribe network could prevent peers in participating in the forming of consensus due to latency.

Of course, when talking about any networking, latency is usually an issue, but it is particularly so in distributed systems, especially in systems that are designed to reach a consensus. Latency is not only caused by bad network bandwidth and signal loss due to copper cabling that has seen its heyday, but by a number of things ranging from data processing to protocol latency and suboptimal routing. Physical network latency is mostly not a computer science problem, but rather an engineering and materials science problem, as it depends on the physical medium the signal goes through and the distance between the sender and receiver. Processing latency means the latency that is created between the time a peer receives a request and sends a response. This latency is often caused by cryptographic encoding and decoding of

messages and fetching data from a database, but in well designed systems this cause of latency is usually negligible, and the problem is largely similar in regular or P2P network applications.

Routing latency, however, can be very different between regular networks and P2P networks. Routing on the Internet is handled by familiar devices – routers. Even in P2P overlay networks[1] routing is handled by the regular Internet building blocks like routers, as by definition overlay networks are abstractions on top of existing networks. Overlay P2P networks use IP routing because they use the Internet, but P2P networks need peer discovery since not every IP on the Internet participates in every P2P network available. There needs to be a way to tell which computers with which identifiers and with which IP addresses are a part of a P2P network, and a way of maintaining this info. Routing latency is created when data is routed between peers on the Internet or on a P2P network through relaying peers or slower routes than the global optimum.

The search for algorithmic solutions to routing latency has long been a part of P2P networks research. Since latency between peers in overlay P2P networks is a problem of the network topology, the solutions try to optimize the peer discovery mechanism that constructs the topology. Currently, advanced P2P networks usually use, in practice because of public key identifiers, randomized peer discovery and a high number of concurrent connections resulting in P2P networks with good peer discoverability and resource availability but unfortunately with subpar routes between peers that are not directly connected to each other. Randomized peer discovery might help with message propagation, as messages can reach far-away peers with fewer hops. For this reason, it might be useful to have a minimal amount of active connections to far-away peers at all times, while still optimizing for individual connections that are close-by. For applications like blockchains this

[1] Peer-to-peer networks that build on top of an existing network infrastructure

has mostly been ”good enough”, but P2P routing needs to improve with the applications it supports, otherwise it could become a bottleneck, as block times decrease and blockchain performance increases at large, and P2P utilization grows.

In this thesis, I propose a protocol for minimizing routing latency by measuring physical latency using processing latency caused by predefined hard mathematical problems, and saving these measurements in a manner that prevents spoofing.

1.1 Contribution

The issues this proposed protocol, Proof of Latency, is trying to solve are suboptimal routing in P2P networks and location spoofing. Too often in P2P networks today peers need to blindly trust one another in telling the truth in terms of their location or the peers they advertise to others. Even if a web of trust is used to score peers based on reputation, the system can be gamed and can be restrictive for new or ephemeral peers, as they have not had the time to gain trust enough to fully utilize the network. Lets imagine a situation where a peer told its individual latencies to other peers on its routing table. Couldn't this help with constructing an optimal distributed routing table, with peers only connecting to other peers if they are actually close-by or behind a fast connection?

Distance bounding protocols have tried, and to a certain extent, succeeded in creating secure protocols for measuring the latency between two peers that do not have to trust each other. They are a category of interactive protocols that are used when a simple ping will not suffice, as a prover could craft a distance fraud attack by responding with a pong before seeing the ping message. This type of fraud can lead to problems beyond suboptimal routing, like relay attacks. For example, an attacker could relay a signal from a car to a car key and vice versa, if the access control protocol is not relay attack resistant. Because contactless payments are becoming more and more common all over the world, the problem of relay attacks

is highly relevant. To the author's knowledge, there are no protocols in use today that could provide proofs of a close distance without a trusted verifier, i.e. that a third party could also trust the close distance between the prover and the verifier in a decentralized and trustless setting.

Now, if a third party could trust the given latency between the responder and a randomly selected peer, it could estimate the network topology and find the peers closest to it with a roughly optimal amount of queries, and the P2P network could converge towards an optimal topology. Such a proof of latency could also be used to prove a geographical location, which could be used to battle GPS spoofing. Usually, GPS is a user-facing technology and an individual tool for pathfinding, with only the user requiring the coordinates to be correct. Some applications might require some sort of a proof of a physical location, and this can be done in a centralized manner with bluetooth beacons or with something resembling COVID-19 contact tracing. In a trustless P2P setting, however, this has not been as easily achievable without introducing an incentive structure or requiring the use of trusted computation modules.

To remove the requirement for trust between peers in a P2P protocol is not a trivial task, and adds requirements for the protocol in multiple places. Most of these are restrictions, or rather, enforcements of causality with commitments to values. Other requirements involve requiring input from another peer before doing something, or providing proofs that a required computation has been done correctly, which can be verified without the verifier having to redo the whole computation itself.

Cryptographic commitments in short are a way of locking a value in place by publishing a derivative of it that only could have been created by the original value, thus the original value cannot be changed after the derivative is known by another participating peer. Committing to a bunch of future values is also sometimes a

requirement, and it can be achieved by a vector commitment, where instead of a single derivative one can create an arbitrary amount of them, called openings, all derived from the original starting point. This way, for example, once a peer commits to a certain processing power, it cannot change it without overwriting the whole vector commitment.

Using vector commitments plus two concurrent verifiable delay functions and a race between them, I propose a novel protocol for producing a publicly verifiable proof of network latency and difference in computation resources between two participants. The latency is denoted in the difference of iterations between the two verifiable delay functions. This proof can be used for dynamic routing to achieve better, faster routes between peers, or to prove a geographical location.

The protocol's security ultimately depends on each peer's ability to measure against a committed processing power metric of another peer based on their communication. If used with parameters described in this thesis, also all assumptions related to Catalano-Fiore vector commitments [3] and the RSA cryptosystem [4] must hold for the proofs to be valid.

1.2 Related Work

Multiple protocols have aimed at creating a reliable proof of location or latency with varied scopes in the past, starting with distance bounding protocols as a way to measure latency between two computers that don't have to trust each other [5]. The following are protocols that demand a mention, as they have tried to solve the same problem as Proof of Latency using different methods.

1.2.1 FOAM

FOAM [6] is an ongoing open source project that provides decentralized location services with a crowdsourced map that has been incentivized by using their own token on the Ethereum blockchain. Their motivation is that GPS is spoofable and largely unprotected against attacks but trusted by a wide array of critical infrastructure. The proofs of location are created in a web of trust manner, with compounding effects involving trusted peers, or so-called beacons. The proofs consist of a coordinate of their own format combined with a geohash[2] and the Ethereum address of the prover. These proofs are then saved on the Ethereum blockchain, but they can also be checked off-chain without requiring the verifier to use Ethereum.

1.2.2 GoAT

GoAT is a protocol that does file geolocation via anchor timestamping [7]. Although the imagined use case is different from Proof of Latency or FOAM, the protocol creates a similar proof to Proof of Latency by measuring an upper bound for latency between two peers that holds even for a third party verifier. GoAT uses timestamping services to enforce causality and to create a timeframe during which the proof must have been created. The resulting proof is meant to be used in a decentralized content delivery network context to ensure that a file has been correctly spread across the network for better availability and performance.

[2] A coordinate that is aunique identifier for a location on Earth.

Chapter 2

Background

2.1 Cryptography

Cryptography is a field of mathematics that uses computationally difficult problems to obfuscate, hide, split and verify data. Starting in the ancient times with the famous Caesar cipher, which relied on shifting the letters in a message in relation to an alphabet, there are multiple cryptographic protocols today for a variety of use cases, which are used in both public and private messaging. As mentioned earlier, while the familiar connotation of cryptography as a word brings secrecy and secure communications into mind, it can also be used to verify data, and to prove that something has happened. I will go over cryptography subjects that are relevant to the thesis in this chapter, like groups of unknown order [8], RSA [4] and cryptographic proofs in general.

Modern cryptography is based on relatively few mathematical fundamentals. The most dominant one during the age of computers has been prime numbers with modular multiplication, but also elliptic curve cryptography [9, 10] is used widely today because of its easy and less resource-intensive key generation method when compared to the prime number based RSA cryptosystem [11]. Both RSA and elliptic curve cryptography use modular arithmetic in their operations.

Modular[1] arithmetic with large composite semiprimes has a useful property that checking a factorization is quick but finding the two large prime factors is a hard problem. This is called the discrete logarithm problem [12]. With public key cryptography, participants can exchange encrypted data without knowing each other's private keys, but requiring a computation that is theoretically almost impossible to break due to discrete logarithm problem. These large numbers that are used as the modulo need to have unknown factorizations to create groups of unknown order, which is paramount to cryptographic systems that rely on the factorization problem for their safety. The order of a group means the amount of elements in it. Cryptographic proofs need to be created with groups of unknown orders for them to be valid, as knowing the order of a group means that the modulo used in the cryptographic operations is so small that it has a known factorization, and the security of the factorization problem is broken [13].

2.1.1 Cryptographic Proofs and Their Soundness

Cryptographic proofs are proofs that depend on the trapdoor nature of cryptographic functions. In other words, cryptographic proofs are arguments that can be trusted if the cryptographic assumptions they depend on are valid and not broken.

The most well-known proofs, which are not necessarily thought of as such, are digital signatures. Given a public key, a prover in possession of its corresponding private key can produce a signature of any given data, given to a verifier together with the original data, that the prover has seen and processed the data. If the private key pertaining to the public key gets leaked to a third party, its security is broken, and the unique identity is lost.

Proofs can be private or public. A cryptographic proof can be categorized as public, if a verifier can gather all information required to verify the proof from the

[1] Using the modulus operation; taking the remainder of a division. Useful in cryptography because it creates finite cyclical groups.

transcript of the proof itself, and verify the proof to be correct. Now, since classical cryptography is based on the fact that a computation is asymmetric — being harder to compute the other way around, a cryptographic proof is still probabilistic in nature, and the security of it based not only on the algorithm but also the parameters used.

Proofs need to be tested for soundness to be called as such. Soundness means that a prover cannot make a verifier accept a false statement, except for a really small probability. This means that a proof is dependent on context, the verifier. If a proof holds statistically, it is a proof. If it instead holds only computationally, it should be instead called an argument [14].

2.1.2 RSA

RSA [4] is named after its discoverers – Rivest, Shamir, and Adleman. It is an asymmetric public-key cryptosystem, meaning that it is based on a keypair in which one is public and the other is secret. The public key is used to encrypt and the secret key is used to decrypt. Everyone who has the public key can encrypt data and that encrypted data is practically impossible to decrypt without the secret key. This works due to the fact that it is hard to factor the product of two large prime numbers. RSA has been the most widely used cryptosystem since its creation, and it is easy to make it more secure by just increasing the size of the keys.

RSA uses an arithmetic trick that involves a mathematical object called a trapdoor permutation. Trapdoor permutation is a function that transforms a number x to a number y in the same range, in a way that computing y from x is easy using the public key but computing x from y is practically impossible without knowing the private key. The private key is the trapdoor [15].

RSA relies on the prime factorization problem and the strong RSA assumption for its security. Strong RSA groups [16] are groups of unknown order, meaning that

the number of elements in the group is unknown. RSA has played a big part in the development of cryptography, and even served as a motivation for the creation of verifiable delay functions, as it was used in the creation of time-lock puzzles closely related to VDFs [17, 18].

An RSA group, like other modular multiplicative groups, are commutative under their group operation, multiplication. This enables multiple interactive protocols and proofs, one of them being Diffie-Hellman key exchange. Diffie-Hellman key exchange enables two peers to create a shared key with their private keys in a modular multiplicative group without revealing their individual secrets because of the commutative property; the order of operations does not matter. An example run of Diffie-Hellman key exchange goes as follows:

$$A = 256^{1097} \mod 311$$
$$B = 256^{1091} \mod 311$$
$$A^B \equiv B^A \mod 311$$

RSA Setup

When using RSA, the users need to setup the public parameters, which need to be distributed between the involved peers to be used in encryption. This key generation process is quite resource intensive, and is rarely used in ephemeral contexts, unless one can skip the key generation process, or ceremony, and use a publicly available modulo that is created by a trusted party. These types of key generation processes are called trusted setups, and they produce so-called toxic waste that needs to be discarded to preserve the key safety [19]. To skip the trusted setup and thus the ”toxic waste”, one can use RSA numbers published by a trusted third party. This

is sometimes even used in production systems, but it is highly encouraged to do a trusted setup in cases where security is the utmost requirement. In this thesis and the Proof of Latency protocol's proof of concept I use RSA-2048 for brevity.

Public RSA Semiprimes

The company formed by the RSA algorithm's inventors, RSA Security LLC, published public RSA semiprimes[2], that are called RSA numbers, as a part of the RSA factoring challenge, from 1991 up until 2007 when the challenge ended [20]. The challenge was ended because it had reached its purpose of forwarding science and understanding of common symmetric-key and public-key algorithms. Despite this, still under a half of the RSA numbers have been factored, and the largest of them might take hundreds if not even thousands of years to break even when given extraordinary hardware to do it with. Using these factoring challenge RSA numbers requires the user to assume that nobody has access to the parameters. Given that the RSA numbers are said to have been created with a machine that was completely destroyed after their creation and the primes were never apparent to anyone, one needs to trust the company's claims if they had ever used these public challenge numbers. Fortunately, usually they are not used for encrypting large swathes of personal data, but in more ephemeral contexts like Proof of Latency, which wouldn't cause a huge stir if the cryptosystem was broken and needed changing. Also, groups of unknown order work similarly to each other, so in most cases they are relatively interchangeable if not for the trusted setup.

[2] A product of two prime numbers.

2.1.3 RSA Accumulators

An RSA accumulator is a set of elements built from cryptographic assumptions in groups of unknown order, like $\mathbb{Z}_N^*$[3]. It enables a prover to prove to a verifier that an element belongs or does not belong to a set by providing a succinct digest together with a proof [21, 22]. The digest is the latest state of the accumulator, and can thus be called the accumulator as well. This digest can be used to prove any previously accumulated elements' membership, or any non-membership. An interesting fact is that whenever an accumulator gets accumulated to, the latest accumulation is actually a binding commitment. This means that once an element has been accumulated to the set together with a proof, the proof is valid with every subsequent digest. This can't be changed unless the prover is able to roll back the accumulator to the state before the accumulation in question. In a P2P environment where proofs get broadcast and/or saved on multiple machines together with digital signatures done by the prover this is highly unlikely, so once an element gets accumulated, it has provably influenced the accumulator.

For the scheme to work, an RSA accumulator needs a function that can hash arbitrary input to a prime number. This is because every accumulated element needs to be unique for the proofs to work. In terms of code, this roughly means creating random odd numbers as fast as possible and taking a primality check of them until a prime number has been generated. This may sound bad performance-wise, but with modern processors and multithreading the problem is not as hard as it might seem, as guesswork can be highly parallellized. The generated prime l needs to be larger than the modulus N of the accumulated set, $l \notin \mathbb{Z}_N^*$, for the proofs to be secure.

[3] Multiplicative group of integers modulo N.

2.1.4 Commitment Schemes

Commitment schemes are a way to lock causality in a cryptographic protocol [23], and are among the most used primitives in cryptography. Commitments usually come in the form of commit-reveal, when at a later stage of the protocol the committer reveals the original message m they made their commitment C_m on. This has a great property — a committer cannot change their committed message after revealing their commitment to the verifier, because the commitment would not hold for a new message. Commitment schemes are usually used when an interactive protocol requires for a message to be hidden until the termination of the protocol.

The requirements for a commitment scheme are that it is hiding and binding. Hiding means that nothing can be known of the committed message by looking at the commitment, not until or if the committer reveals it. Binding means that once a commitment has been made, the committer cannot get the same commitment with different data— the commitment binds the committer to reveal the exact committed data they made the commitment with.

The function that is used to transform a message to a commitment could, for example, be a cryptographically secure hash function. In this case, a commit-reveal scheme works as follows:

1. Committer hashes the message m with a hash function $H(m)$
2. Committer publishes the hash $H(m)$
3. Third parties see the commitment, meaning that the committer cannot change the message and convince the third parties with another input, locking its decision without revealing m.
4. Committer publishes the original message m, convincing the third parties that the commitment was correct.

Vector Commitments

Vector commitments are a cryptographic primitive formalized in 2013 consisting of a vector commitment *com* and openings to the vector [3]. Vector commitments are related to cryptographic accumulators, but add a position binding commitment to them. This means that once a opening has been made at an index, no other opening can reside at that given index. The simplest form of a vector commitment (and a cryptographic accumulator!) is a merkle tree. Unfortunately, merkle proof size grows with input, and verifiers need to know the history instead of a concise digest of an accumulator. For example, Catalano-Fiore vector commitments improve upon the proof size. Catalano-Fiore vector commitments are called concise, which means that the proof is at most a fixed size [3]. In addition to the fixed size, the verifier only needs to know the proof and the digest[4] to verify the proof. In layman's terms, vector commitments enable a prover to prove that an opening *pi* to the vector commitment *com* has been made at the index i.

Vector commitments enable opening an index i from the vector commitment that corresponds to a message m. To get the index corresponding to the message, the message must be hashed to prime. This prime is the index i that can now be opened once and only once.

Formal Definition

Vector commitments are a tuple of four algorithms: VC.Setup, VC.Com, VC.Open, VC.Verify [24].

1. VC.Setup(λ, n, M) $\rightarrow$ *pp* Given security parameter λ, length n of the vector, and message space of vector components M, output public parameters *pp*, which are implicit inputs to all the following algorithms.

[4] latest output

2. VC.Com$(m) \rightarrow \tau, com$ Given an input $m = (m_1, ..., m_n)$ output a commitment com and advice τ.

3. VC.Open$(com, m, i, \tau) \rightarrow \pi$ On input $m \in M$ and $i \in [1, n]$, the commitment com, and advice τ output an opening π that proves m is the ith committed element of com.

4. VC.Verify$(com, m, i, \pi) \rightarrow 0/1$ On input commitment com, an index $i \in [n]$, and an opening proof π output 1 (accept) or 0 (reject).

Like regular commitment schemes, vector commitments are used to enforce causality. While regular commitment schemes make sure that a user of a protocol can't change an input or "skip the line" after the start, vector commitments create a causal relationship for multiple proofs. To wrap it up, if proofs are based on vector commitments, the commitments guarantee that the proofs' order can't change, unless the whole vector with all associated proofs are unvalidated.

2.2 Peer-to-Peer Networking

Peer-to-peer networking, P2P in short, means that two or more devices communicate with each other with both serving as a client and a server simultaneously. P2P networks vary in scope a lot. Some networks are global, while some are as small, a couple of devices in a local area network, which is the case in printer sharing and some IoT installations. P2P networks can be standalone or rely on some existing infrastructure, like the Internet. The P2P networks that rely on preexisting infrastructure are called overlay networks.

While overlay networks' addressing and routing is usually based on TCP/IP[5]

[5] Transmission Control Protocol (transport layer) / Internet Protocol (Internet layer), a connection-oriented protocol stack

or UDP/IP[6], they must introduce a separate routing scheme, like a distributed hash table or a DHT for short, that works as an index for peers on the network [25]. Distributed hash tables are a key-value mapping of peer identifiers to their addresses, which is stored in a deduplicated manner across peers on a P2P network. Since IP only focuses on addressing the computers and not the applications or resources they offer, it can't function as a peer discovery scheme by itself.

Overlay P2P networks must somehow signal the initial peers, also called bootstrap or introductory peers, to connect to at first. This can be done either by including the bootstrap peers' IP addresses in the application code itself or by providing the same information on a regular web page. The bootstrap peers can be also signaled by word-of-mouth between a group of people, or with wireless broadcast, which is discussed briefly in the next section. This said, usually every peer can function as a bootstrap peer, so knowing the connection information of a single peer on the network can potentially introduce a node into the network at large.

A simple way to create a P2P network would be to share a file containing all the IP adresses a given node has ever connected with with anyone who connects to that node. This is an example of an unstructured P2P network, which could work relatively well in a closed setting, like inside a LAN without support for multicast DNS[7], or mDNS in short.

In contrast to unstructured networks, where peers are connected to at random, structured networks use some logic to form a structured overlay network, rather than just connecting to peers as they come.

---

6 User Datagram Protocol (transport layer) / Internet Protocol (Internet layer), a connectionless protocol stack

7 A protocol that enables broadcasting all addresses in a local area network for easy discovery. Used commonly for discovering printers and such.

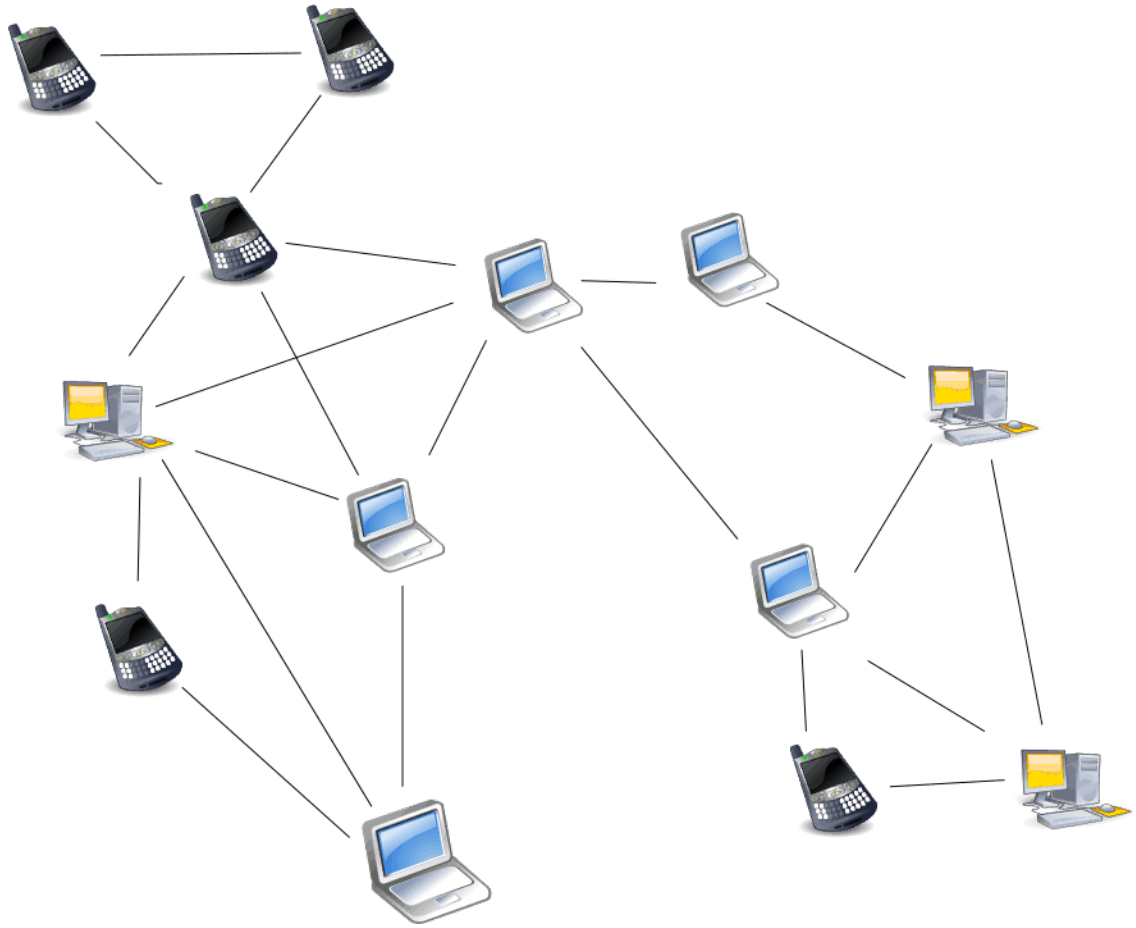

Figure 2.1: An example topography of an unstructured overlay network.

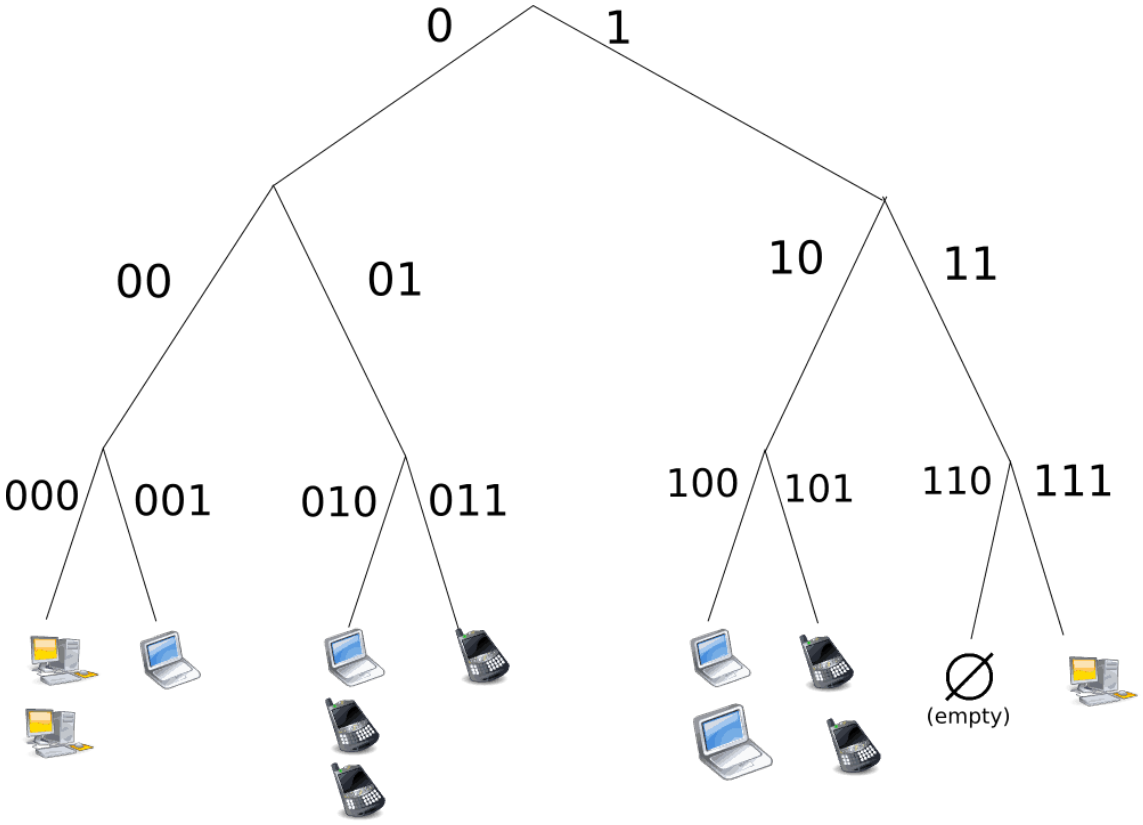

Figure 2.2: An example topography of a structured overlay network.

2.3 Role of Latency in Distributed Systems

It is hardly a surprise, but latency is an important consideration in distributed systems, especially trustless, decentralized ones. Latency is first constrained by the speed of light, and then by hardware and software along the way. In 2012, the global average round-trip delay time to Google's servers was around 100ms. [26]

In the new space age the maximum possible latency grows very fast, as there could be peers joining to a distributed network from other planets, space ships or stations. This might seem unnecessary to think about in the distributed P2P context for now, but before all that, we have global satellite mesh Internet providers, like Starlink. Elon Musk, the founder of SpaceX, which deploys the Starlink network of satellites, claims that there is going to be a round-trip[8] latency of about 20 milliseconds between a single satellite and the user [27]. In legacy satellite Internet access, the round-trip time even in perfect conditions is about 550 milliseconds [28]. This difference between legacy and newer satellite Internet comes from the difference in the satellites' orbits and the sheer amount of satellites involved. Legacy satellite Internet uses geostationary orbits, which are very high. These satellites beam on a single face of the earth at a time with limited bandwidth. Newer systems, like Starlink, use a low-earth-orbit, which requires more satellites, since they zoom by at such a speed that constantly changing which satellite one is connected to is a must. The low orbit also means less distance between the satellites and the user. The 20 millisecond latency claimed by Starlink seems like a stretch at first, but is believable when one takes into account that inter-satellite links are done by laser, and light can travel about 31 percent faster in a vacuum than in fiber optics [29]. Intercontinental latencies can become much lower because of this.

In blockchains, latency plays a role in the energy efficiency used to achieve consensus. Miners waste energy on a previous block in Proof of Work as long as they

[8] Including the user's initial request and received response

have not received information on the winner of the previous block race, as all the miners on the network are trying to find the correct answer to a puzzle at the same time. Thus, it results in energy wasted if miners drag behind the latest block, still trying to solve the previous one. Simulations have shown that if one calculates the round-trip time between the peers that are connected to each other and dropping the ones with larger latencies in favor of lower ones, one can achieve 50% improvement in average latency with 1 to 2 peers connected. When connectedness grows from the degree of just 1-2 peers up to 20 connected peers, the average latency improvements achieved drop to about 20% [30]. When connectedness is large, there are shorter routes simply by chance to peers one is not directly connected to. For example, in a situation like this, publish-subscribe schemes work faster, propagating messages to the whole network more reliably, because there is less relaying happening.

One cannot keep multiplexing connections[9] forever, as there are hardware and software related limitations to the amount of peers, and there is a Goldilock's zone for the most effective amount of connections. With IPFS[10] [31], for example, the protocol has been breaking users' routers [32] because of the high number of incoming connections that need to be routed through NAT[11].

Network hops in P2P systems are introduced when two peers are not directly connected to each other, but rather through one or many relays. There are network hops that cannot be easily avoided, like the hops between network routers on the Internet. Most of the P2P routing protocols used today are oblivious to the problem of introducing large hops to communications between two peers, trading network performance for network robustness and decentralization. Some DHT-based protocols, like Kademlia [?], make the assumption that their users have fast Internet

[9] Having multiple concurrent stateful connections.

[10] Interplanetary File System

[11] Network address translation. Hides the local area network from the Internet under a different subnet address.

access, and minimize the average latency by selecting connected peers basically at random [33].

While the randomness is great for preventing eclipse attacks[12] [34], they can introduce unnecessary geographical hops between two peers. If two peers are in the same WAN, for example, in Kademlia they might still connect to each other through a network hop going through another continent. This makes routing data between peers inefficient, resulting in preventable lag when communicating between peers that are not directly connected. Now, if we were to rely on IP address geolocation, we could more efficiently connect to peers that are close-by. This is unfortunately impossible in privacy-oriented P2P networks, like mixnets, which aim to hide as much of the packet routing information as possible by routing individual packets through different peers and hiding IP addresses of two connected peers from each other [35]. Obviously, this also has implications on routing, and thus Proof of Latency. Optimizing for privacy makes it difficult, if not impossible, to measure and advertise latencies between peers reliably and correctly, when packet routes are obfuscated.

2.3.1 Ad Hoc and Zero Configuration Networking

Multicast DNS was proposed by Apple in 2013 [36] as a way of discovering peers in a local area network in a zero-configuration manner. It is used today for resource sharing, such as sharing printers. Multicast DNS does not work outside local area networks, since it works by associating names with IP addresses, like regular DNS does. The problem is that these names are not quaranteed to be unique, and they can therefore be spoofed. If there are two clients with the same name, the first one to respond with its IP address to a query wins [37]. The security of zero-configuration and ad hoc networking must rely on cryptographic identities, so that a peer can

[12] Eclipsing a peer means purposefully advertising it malicious peers that isolate it from the network

verify itself with public-key cryptography. This makes the peers on the network practically unique and thus hard to spoof.

Zero-configuration networking in an unconstrained, global setting is possible with radios, using either dedicated meshnet radios like GoTenna [38] or Helium [39], or by using antennas inside mobile devices to form a network. These types of networks are usually called meshnets or ad hoc networks. Walkie-Talkies are the simplest form of a P2P network. Problems can arise due to geography blocking signals, or when one wants to cross large distances with low transmitting capability. Mesh networking has been used most famously in protests worldwide by using a smartphone app called Firechat [40].

Ad hoc mesh networks have a natural metric for latency: signal strength. They can rely on Bluetooth RSSI[13], or triangulate distances by cooperating with multiple peers. These methods are used to locate emergency calls and in contact tracing [41]. Mesh networks, while being peer to peer and not relying on existing infrastructure like overlay networks, still need routing and multiple hops if one wants to reach peers that are not in the operating range of the communication method used. Even ad hoc networks could thus benefit from using Proof of Latency, although the proofs should be updated way more frequently because of the network's mobile nature.

2.3.2 Distributed Hash Tables

Distributed hash tables (DHT) are a way of addressing content to peers in a distributed network. In addition to indexing content in content-addressed networks like IPFS [31], they can function as routing tables, and have been developed to remove bottlenecks in peer search.

A hash table is a key-value mapping from a to b. What makes a hash table distributed is the fact that the data stored is meant to be distributed between peers,

[13] Received Signal Strength Indication

with no single peer keeping all the available data in its hash table, but relaying queries for resources it does not have to other peers on the network. There are multiple versions of DHTs with different methods for prioritizing certain peers: using tree structures, sorting by identifiers, using computational trust, et cetera.

In addition to identifier closeness, DHTs can force a certain network behavior by peer scoring and constructing a web of trust. For example, a peer could only advertise peers that have been connected over a period of time, or enforce reconnecting to disconnected peers that have a good reputation. A widely used trust system is TrustGuard, implemented in the blockchain framework Tendermint. [42, 43]

Most of the DHT algorithms were invented in the early 2000s, with Kademlia being one of them. DHTs mostly differ just by how distance between peers is defined, and how neighbors are chosen. [34]

Kademlia

Kademlia is a DHT designed by Petar Maymounkov and David Mazières in 2002 [?]. It is based on a tree of identifiers that are split across peers on a network. The identifiers are 160 bits, e.g., a SHA-1 hash of some larger data. Kademlia tries to improve upon previous DHT-based routing algorithms by introducing a symmetric XOR metric for distance between node IDs in the key space [?]. These IDs are sorted in a binary search tree, with each node's position determined by the shortest unique prefix of its ID, like shown in the diagram 2.2 on page 19. Kademlia makes sure that any node in the network can locate any other node by its ID by ensuring that each node knows at least one of the nodes in each subtree.

A single query in Kademlia has been shown in real-world tests to result in an average of three network hops, meaning that the query gets relayed through two peers before reaching the requested resource [44]. Network hops are a necessary evil in distributed systems, and Kademlia does well in requiring on average $log(n)$

queries in a network of n nodes. Since the closeness metric is based on a similarity search rather than a measurement, the closest peer is only closest by the identifier, not by network latency. [33]

The randomness of Kademlia is great at averaging the network hops required to reach a scarce resource. While this is great for network security, the downside is that it also averages latency, reducing overall performance of the network.

Kademlia protocol has four remote procedure calls, or RPCs in short. These are *PING*, *STORE*, *FIND_NODE*, and *FIND_VALUE*. A Kademlia participant's most important operation is node lookup, i.e., locating k closest nodes to a given node ID. It is a recursive operation, which starts by picking α closest nodes from the closest non-empty bucket, and sending them all *FIND_NODE* calls. This is repeated until the initiator has queried and received responses from all k closest nodes it has seen.

Polymorph and Network Topologies

Many DHT networks use what is called a uniring topology. This means that the addressing of the peers is one-dimensional, and do not form logical clusters on the network graph. Unlike Kademlia or other uniring networks like Chord or Tapestry, Polymorph [45] is a DHT network that uses a polyring network topology.

Polyring network topologies can use two or more dimensions for their addressing. For example, first a cluster ID and then the peer's local ID in the cluster. This can be used to group peers in the network by geographical location, latency, or application specific heuristics like subscribed channels in a publish subscribe system, et cetera.

Uniring network topologies can result in worse global performance optimum, but they are less susceptible to congestion than polyring network topologies, which have a small number of connections between clusters. [45]

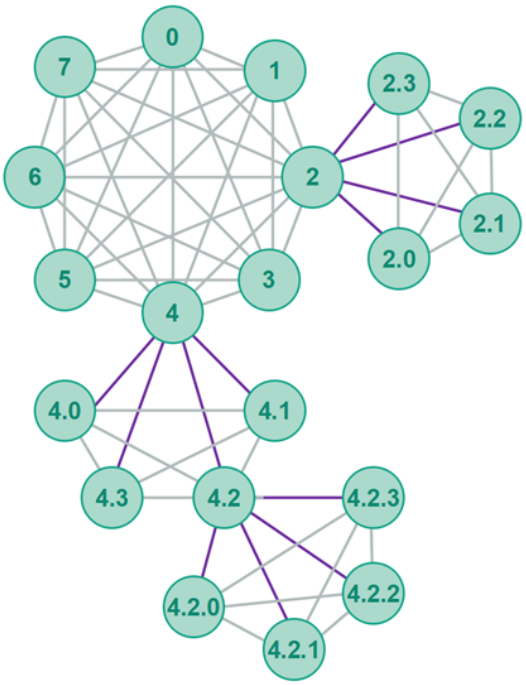

Figure 2.3: Polyring network topology.[45]

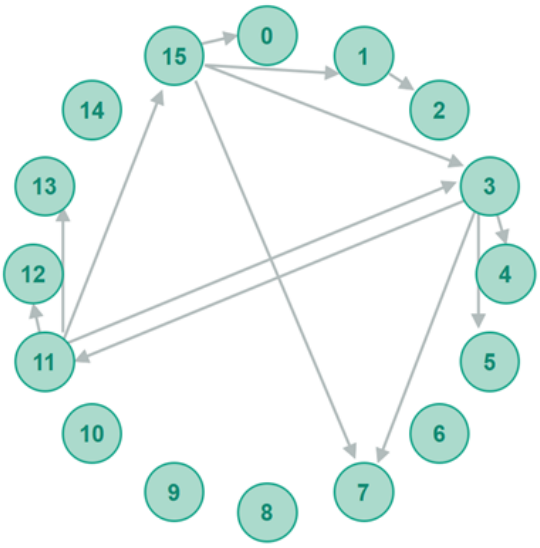

Figure 2.4: Uniring network topology.[45]

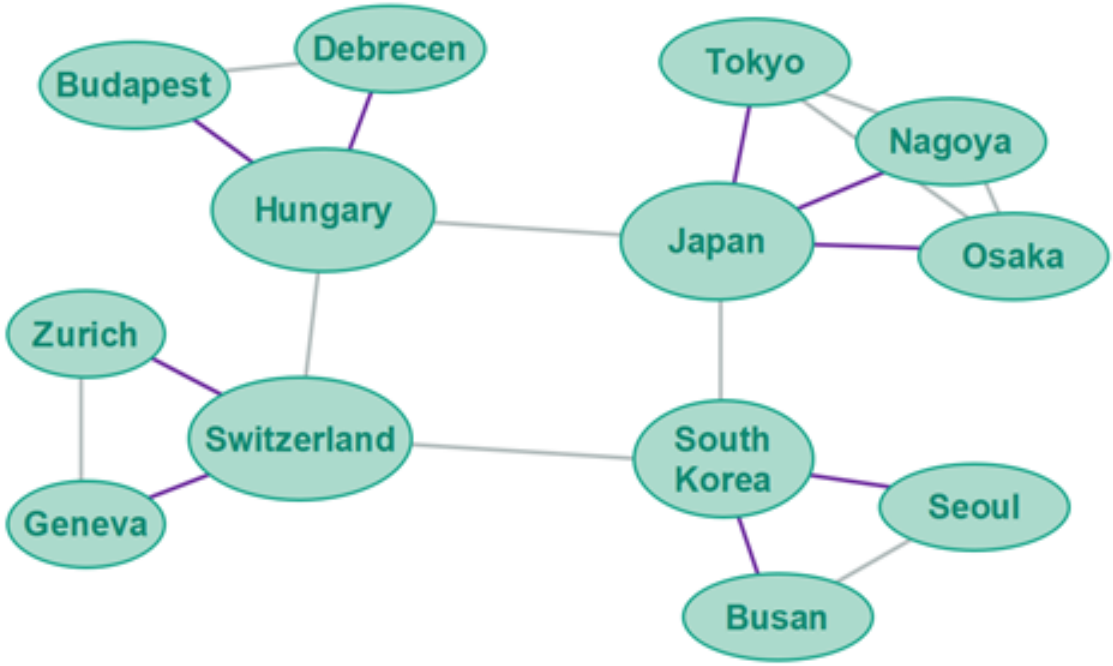

Figure 2.5: Geographically clustered polyring topology.[45]

2.3.3 Eclipse Attacks

Although DHTs are most widely used with random hashed identifiers, the distance metric stays the same. By forging identities that are close-by, one can advertise false friends which take over the search space. For example, in a 2019 paper "Eclipsing Ethereum Peers with False Friends" by S. Henningsen, D. Teunis, M. Florian, and B. Scheuermann, the authors demonstrate that to eclipse a victim in Ethereum P2P network, the attacker needs to fill its eight slots for outbound connections, and fill seventeen slots for inbound connections to completely deny service, without going through the attacker's nodes. [46]

Less structured, random ways of forming connections in an overlay network, like Kademlia, protect against eclipse attacks quite well because of the high connectivity and low locality. Introducing peer scoring and making the network topology more structured opens up possibilities for an attacker to game the scoring system, making sure the target connects to lots of malicious peers eclipsing it. Thus, protecting

against eclipse attacks is a balancing act that requires an overlay network to retain some elements of an unstructured network while improving general performance with a structured group of peers. [47] This balancing act is seen as a multi-armed bandit problem of exploration and exploitation in the 2020 paper ”Perigee: Efficient Peer-to-Peer Network Design for Blockchains” by Y. Mao, S. Deb, S.B. Venkatakrishnan, S. Kannan, and K. Srinivasan.

2.3.4 Sybil Attacks

Sybil attacks mean creating multiple false identities in a P2P network to achieve a botnet-like effect. This could result in gaining disproportionate influence in decentralized governance, block voting, block producing et cetera. An eclipse attack usually involves a sybil attack to create the key pairs that are used to eclipse peers with. Due to peer discovery protocols like the one used in Kademlia being based on public key similarity, creating sybil peers that can eclipse the peer is harder than it sounds like. Although creating new cryptographic identities is not hard, creating identities that are close by is hard, equating to running a Proof of Work algorithm, as outputs in a group of unknown order like RSA are highly random. [48, 49]

2.4 Distance Bounding Protocols

Distance bounding protocols are interactive protocols that aim to measure the physical distance or latency between two participants, the prover and the verifier. They are solutions to problems where a simple ping will not suffice, as either party could quite easily craft a false measurement. Distance bounding is used in applications like IP geolocation, wireless access control, and routing in P2P (ad-hoc) networks.

The first distance bounding protocol was designed by Brands and Chaum in 1993 to counter so-called relay attacks. [51, 5] A commonly used example of a relay

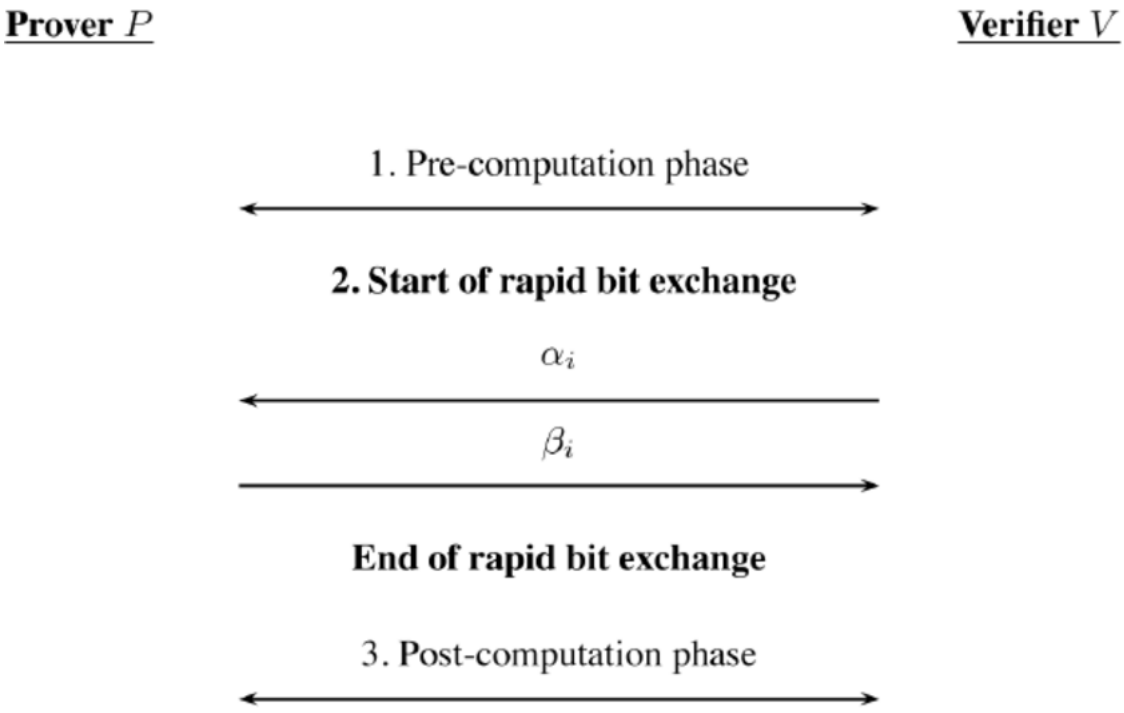

Figure 2.6: A distance bounding protocol [50]

.

attack is that if the signals between a credit or a debit card and a point of sale system were to be relayed over a distance, two attackers could pay with the relayed info in a totally different country, for example. A distance bounding protocol is safe if information never gets passed faster than the speed of light, and if causality holds due to the prover not being able to create a valid response to a challenge before it has received the challenge [51].

The original definition of a distance bounding protocol consisted of three phases: initial phase, critical phase, and a verification phase [5, 52].

1. The Initial Phase — the two peers agree on the parameters used.
2. The Critical Phase — the two peers do multiple challenge/response rounds.
3. The Verification Phase — optional, because the verifier can also verify the proofs during the critical phase.

A way to infer physical distance d from the measured round trip time Δt is to convert the latency to an approximation of the round trip time Δt divided by two times two-thirds the speed of light c [14]:

$$d = \frac{1}{2}\Delta t \frac{2}{3}c$$

Even when not using distance bounding for geolocation one can use the aforementioned method to pick sane parameters for each application, like attack prevention in point of sale systems and RFID lock tags. For the measurement to be as close to ground truth as possible, there needs to be sufficient computing power and a good software implementation to minimize any processing delay introduced between receiving the challenge and sending the response, as this delay lowers the maximum measured resolution achieved by the protocol. For point of sale systems, this delay can be crucial, as we only want the point of sale system to be used by the customer right next to it.

When distance bounding is used in other ways than highly local applications like car keys, e.g. in digital rights management, it can reveal too much of the user's location. Usually distance bounding protocols have assumed that both the prover and the verifier are willing to disclose their locations. Some newer protocols have tried to introduce a cloaking region by reducing measurement resolution by adding a delay to the challenge responses. [54] This is a welcome solution in instances where a verifier only needs to know a rough estimate of the prover's location instead of a tight bound, for example in content delivery networks.

[14] an approximation of network transmission speed in optic fiber widely used in IP geolocation [53]

2.4.1 Attacks

Distance bounding protocols were originally made to solve the problem of relay attacks. These attacks are usually not an issue anymore, and distance bounding is used widely in RFID readers nowadays [55]. There are, however, still potential attacks that could affect distance bounding. Historically, the attacks against distance bounding have been categorized into five different categories [51]:

- Distance fraud, where a far-away prover tries to pass the protocol by itself.
- Mafia fraud, which is a man-in-the-middle attack where a malicious peer in between an honest prover and a verifier tries to get the verifier to accept the far-away prover's proof.
- Terrorist fraud, where with a help of a temporary adversary a malicious far-away prover tries to make the verifier accept its proof without revealing any secrets to the adversary.
- Impersonation fraud, where a close-by adversary impersonates a prover.
- Distance hijacking, where a far-away prover takes advantage of honest provers to make the verifier accept its proof.

Inferring from the categories one can deduce that requiring the participants to digitally sign the passed messages in a cryptographically secure manner will render most of these attacks unfeasible, unless, of course, the prover has been hacked or in some other way it has leaked its secrets to a malicious third party.

2.5 Verifiable Delay Functions

A verifiable delay function is a function that calculates sequential calculations that cannot be skipped or significantly sped up by parallellisation, and creates a proof of

the calculations that is faster to verify than the calculations are to fully re-evaluate. The computation of a VDF can be split into three distinct parts: Evaluation, proof generation, and verification. The evaluation is the hard sequential calculation from which a proof is generated. [17]

The common parameters in VDF calculation are a random input x, the hash of x which is called the *generator*, the group G, and the number of sequential operations, also called the difficulty, T. Every sequential operation during the evaluation consists of an exponentiation and a modulo operation. The exponentiation is always a squaring, so the exponent does not change. This has to do with the proof generation.

Verifiable delay functions are a peculiar bunch of cryptographic proofs, since they are not proving that something has happened or something has been seen, but that a certain amount of time has been spent to compute the result. Like other cryptographic proofs, for the proof to be publicly verifiable it must at least in part be based on public parameters. If a VDF is going to be calculated between two parties, these parties decide the starting parameters between themselves, in a way giving the parameters to each other. This is to prevent possible precomputation — meaning that the peer that provides a VDF proof could have computed it earlier, separately from the current transaction. If a participant or participants can decide the parameters of the computation by themselves, the computation could have occurred far back in the past.

There's no known way to reduce computation costs sublinearly to the bit length of the exponent used. [17] This means that there's no algorithm-based solution to make VDF computation faster, but there is a growing effort in developing faster hardware for VDF evaluation, especially for doing repeated modular squarings, which would speed up both the evaluation and the proof phases of a VDF. [56, 57, 58]

In 2018, two research papers were released independently with descriptions of

VDF candidates. [59, 60] Before that, also in 2018, a paper formalizing VDFs was published by Boneh, Bonneau, Bünz, and Fisch, mentioning the term for the first time. There are multiple formulations of a VDF, and not all use a generated proof, instead using parallel processing with graphics processors to check that the delay function has been calculated correctly. [61] This bars less powerful devices, like embedded devices, from verifying the VDF's result efficiently. Thus, generating a proof that requires little time to verify is more ideal. [62]

2.5.1 Use Cases

VDFs could find uses in many applications not directly related to blockchains or distributed ledger technologies, and my motivation for this thesis is to find such new use cases. I believe I have found one, using parallel calculations between two peers to measure latency with iterations of this sequential computation.

While Proof of Work is the most widely used consensus algorithm in public distributed ledgers, its use can result in wasteful use of computation resources, especially when there are significant latencies between the participating peers, as discussed in section 2.3. While many other consensus algorithms have been proposed and even applied to distributed ledgers, they come with their own set of problems to tackle. In Proof of Stake network nodes participate in the voting of new blocks by staking a part of their assets as a pawn. Concretely this means that a peer hands the control of some of its cryptocurrency to the consensus algorithm. If a voter gets labeled as malicious, faulty, or absent by a certain majority, its stake can get slashed, losing all or a part of the staked asset in the process. This serves as an incentive for honest co-operation, with sufficient computation resources.

A motivation for VDFs came from a problem with Proof of Stake: block generation votes are not done globally, but by a selected group of peers called the validators, which vote for the contents of proposed blocks that are generated by

just one peer at a time, selected as the block generator. The validators are usually selected randomly. This generated a demand for verifiable public randomness that is pre-image resistant, meaning the output of the algorithm generating the randomness cannot be influenced before evaluation by input. This created a need for an algorithm that would prevent multiple malicious actors from being selected to vote at once. Verifiable delay functions can help in creation of public randomness simply because they produce quickly verifiable proofs for results that have high entropy due to repeated calculations with a pre-image resistant hash function.

The hash function inside a VDF can be changed, and VDFs' use in such distributed random beacons is only a small piece of a larger puzzle, as good randomness is hard to achieve using only pseudo-random hash functions. Another problem that served as a motivation for VDFs is scalability and data integrity in distributed ledgers. If the participants of a distributed network of nodes have the same initial input for a high-frequency hash function, and they have similar hardware, it can help putting transactions in order without explicit communication between peers, since one can index each transaction with a hash before communicating it to other peers. The resulting order can be incorrect, but like the case with other forms of synchronized distributed clocks, the main objective is to have an explicit order without ambiguity. This also ties into CRDTs[15], but since that is not the main subject of the thesis, I will not address it.

Time-lock puzzles are precursors to VDFs. Their evaluation can be similar or even identical to VDFs, but they do not contain a proof that is faster to calculate than to re-evaluate the whole puzzle. A way to verify a time-lock puzzle's evaluation would be to link each iteration of the evaluation to the previous iteration, and supply each step of the calculation to the proof's verifier. Then the verifier can use parallel computation to verify the evaluation faster than re-evaluating the whole thing. A

[15] Conflict-free Replicated Data Type

puzzle like this is categorized as a proto-VDF.

Many have shown that there are more use cases for verifiable delay functions than puzzles and random number generators. Some examples include preventing front running in P2P cryptocurrency exchanges, spam prevention and rate limiting. [63] Almost all use cases are based on the property that with a new unique input, and a difficulty requirement, there is no easy way to speed up the calculation of a VDF.

Preventing Front-Running

In the front-running use case, this property can be used to restrict front-running in both centralized and decentralized exchanges. [64] Front-running is a term used in both stock trading and cryptocurrency trading which means using inside knowledge, or just a faster connection, of a future transaction to trade an asset with a better price or deny an other transaction from happening. [65] The issue is pronounced in trustless P2P networks like the Ethereum blockchain, where pending transactions are for all to see and can be overtook by promising the network validators a bigger reward, since there's no synchronization before consensus is achieved. [66] In a centralized exchange, preventing front-running with a VDF means that the exchange waits for a specified time before it starts fulfilling a trade. The VDF serves as a receipt for the order taker that there was no reordering and that take orders were processed in a FIFO[16] fashion. [67] In a decentralized exchange the same holds if there exists only one peer that can fulfill the order. The peer then functions exactly like the centralized exchange mentioned before.

Spam Prevention and Rate Limiting

Spam prevention and rate limiting in the context of VDFs mean roughly the same thing. This is because spam prevention with VDFs is done with rate limiting. If

[16] First in, first out

a network application requires peers to perform some sort of Proof of Work in a form of a VDF, by defining the difficulty parameter T they can set a time boundary for subsequent requests, also called a rate limit. If all the peers need to perform a challenge to request resources from the requestee, a performance-based limit is created for the time between each individual request, thus limiting spam. In a similar fashion, the sequential nature of VDFs can be used to make more power-efficient proof-of-work algorithms, by replacing the requirement from finding a solution to a random challenge to a predetermined challenge with an exact number of steps. This results in better power efficiency, because parallellism will not help in solving the challenge, thus limiting the use of power hungry GPUs or highly parallel ASIC chips for the job.

Proofs of Sequential Work

VDFs can be utilized in proof of work algorithms that are more efficient than trying to find an inverse of a hash as in Bitcoin. [68] This is more energy efficient because the calculations cannot be parallellized, restricting the advantage gained by using multiple processor cores to find the solution. Due to the deterministic nature of VDFs there has been some concern over the smaller variance in the winners of the block race, as the inherently probabilistic nature of guessing hashes does produce useful variance with regards to Proof of Work in current sybil resistance algorithms. Using VDFs as Proofs of Work could result in a winner takes all situation. [69]

2.5.2 Precursors to VDFs

Verifiable delay functions are based on time-lock puzzles. Like VDF's, time-lock puzzles are computational sequential puzzles that require a certain amount of time to solve [70]. Time-lock puzzles were originally created to encrypt information in a manner that could only be opened after a certain amount of computation. Function-

ally, this is a cryptographic time capsule. Ronald L. Rivest, Adi Shamir, and David A. Wagner classify time-lock puzzles generally under the term timed-release crypto in their 1996 paper [70]. They describe the envisioned use cases for timed-release crypto as being the following:

- A bidder in an auction wants to seal his bid so that it can only be opened after the bidding period is closed.
- A homeowner wants to give his mortgage holder a series of encrypted mortgage payments. These might be encrypted digital cash with different decryption dates so that one payment becomes decryptable and thus usable by the bank at the beginning of each successive month.
- An individual wants to encrypt his diaries so that they are only decryptable after fifty years.
- A key escrow scheme can be based on timed-release crypto so that the government can get the message keys but only after a fixed period, say one year.

Verifiable delay functions are time-lock puzzles extended with a publicly verifiable proof that is much faster to verify than the puzzle was to solve. The original motivation for verifiable delay functions was to enable someone to pick off where somebody else had left when calculating a time puzzle — to make it possible to create checkpoints in the calculation that could be trusted. Around the same time, VDFs found use in blockchain applications as a way to achieve unspoofable randomness in elections and smart contract input.

2.5.3 Variations

There are multiple variations on the security principles used within VDFs, since they can use any group of unknown order as a basis for their unpredictability. Known solutions use RSA, elliptic curves and class groups as their cryptographic basis.

Since this work uses the Wesolowski proof [59] in Proof of Latency, I will concentrate on it the most here, but will also explain differences between the most known variations. For the group of unknown order, I will use the RSA group in the examples.

Wesolowski's Efficient Verifiable Delay Function

The "Efficient" in the name of this VDF protocol by Benjamin Wesolowski means that the proof it generates is efficient to verify by a third party, requiring $\log_2 T$ multiplications, with T being the total amount of modular multiplications. The following is a description of the non-interactive[17] variant of the protocol and its phases:

$let\ N = \text{RSA} - 2048, \mathbb{G}\ =$ multiplicative group of integers modulo N

$let\ x\ =$ random input, $H\ =$ hash function, for example $BLAKE3$

$let\ g = H(x) \rightarrow \mathbb{G}, T =$ difficulty

1. Evaluation

   To evaluate the VDF, one must calculate T repeated modular squarings. This means raising the generator g to the power of 2 and taking the remainder of that result divided by N, T times.

   As an equation the evaluation output is the following:

   output $= g^{2^T} \mod N$

2. Proof Generation

   $let\ L =$ random prime number

   $let\ q \in \mathbb{Z}, q = 2^T/L,$ (quotient)

   $let\ r \in \mathbb{Z}, r = 2^T \mod L,$ (remainder)

[17] Requiring no messages to be sent — no interaction

let proof $= g^q$

Now that the proof has been calculated, a verifier needs N, output, g, L, r and proof to verify that the proof is correct and the prover has actually calculated $g^{2^T \mod N}$.

3. Verification

   To be sure that the prover has calculated the output correctly, a verifier must check the following equation:

   output $=$ proof$^L * g^r$, $where\, r = 2^T \mod L$

   If the two sides are equal, the VDF has been verified as being correct.

The following is an example of a VDF evaluation with 3 iterations, using small variables for understanding. The variables are $N = 17$, $g = 11$, and $T = 3$. Keep in mind this is not a working example due to the variables' size, but rather an example to demonstrate what actually happens with the numbers. Also, as programming languages rarely accept algebra as input, the actual implementations of VDF calculations in code are different, if not for the sequential evaluation part.

$$11^2 \equiv 2$$

$$2^2 \equiv 4$$

$$4^2 \equiv 16$$

After the result, 16, has been acquired, the proof generation goes as follows, with the random prime $L = 7$:

$$pi(proof) = (2^3/7)^{2^3} \mod 7 = 1$$

To verify this proof, the verifier calculates the following:

$$16 = 1^7 * 11^{2^3} \mod 7$$

Different VDF constructs, like the most well known ones by Wesolowski [?] and Pietrzak [?] make computational trade offs on different parts of the proof. In a Wesolowski VDF the proof generation is a significant part of the whole VDF calculation, comparable in time to the evaluation unless it is optimized in some way. A Pietrzak VDF is faster to prove, but a lot slower to verify. A Wesolowski VDF produces a proof that can be verified in O(1) time.

2.5.4 Constructs Related to VDFs

Proto-VDF

Time-lock puzzles that do not include a proof but nevertheless can be verified faster than re-evaluated can be categorized as proto-VDFs. A non-verifiable delay function or a proto-VDF can be verified by using multiple CPU cores or highly parallel graphics processing units, checking each iteration separately with parallellism, like in Solana [61]. One could also argue that all blockchains and some random beacons form multi-party calculated VDFs. An algorithm based on sequential hashings, like sha-256 in Solana's Proof of History can be described as a proto-VDF, since it can be verified as correct faster than it can be evaluated, but does not have a separate proof.

Classic Slow Functions

Since a VDF's idea is that a hard and slow calculation can be verified to have been performed correctly quickly, any calculation of whose inverse is harder is a candidate for a VDF-type construct. One example of these asymmetric calculations is Sloth [17]. Sloth does not include a proof and it is not asymptotically verifiable, thus it does not satisfy the definition of a VDF per se.[18] The evaluation of Sloth is calculated with repeated cubic roots, and it is verified with repeated cubings, with cubic roots being harder to compute than the cubings.

2.5.5 Hardware Developments

VDF applications can be made faster with hardware, and it has been estimated that with an ASIC[19] chip a VDF can be evaluated more than ten times faster than with a GPU [71]. The three different parts of a VDF calculation are different in terms of the hardware that can be utilized to make them faster. The evaluation part currently has no de facto hardware to make it faster, but the ASIC and FPGA developments are the ones pushing this part forward. Proving can be made faster with GPUs, whilst verification benefits from fast general computation, thus CPUs[72].

If, or when, hardware specifically optimized for sequential squarings is commercialized, VDFs can become much more mainstream, because they would suffer less from computational differences, thus requiring less trust between the calculating parties. It's a race against time, since if an attacker puts a considerable amount of resources in the development of an optimized chip themselves, they could potentially compromise the security of some blockchains, due to an assumption that is made generally in all blockchains: when competing chains are proposed as the truth, the one with the most work is regarded as the winner. By creating blocks faster than

[18] Asymptotic, meaning that there is a maximum verification time and the verification can't scale beyond that point.

[19] Application-Specific Integrated Circuit. A purpose-built chip for a single algorithm.

the majority of the network, one could attack systems that rely on this assumption with a proprietary chip.

Besides the development of an ASIC driven by vdfresearch.org [63], there's been development on CPU instructions by Intel, aiming to help the specific operation of modular exponentiation, which is used not only in VDFs, but also in classical RSA, DSA[20], and DH[21] algorithms, and homomorphic encryption[22]. [73]

[20] Digital signature algorithm.

[21] Diffie-Hellman key exchange.

[22] Homomorphically encrypted data is data that can be done calculations with without revealing it's contents to the calculator.

Chapter 3

Design and Architecture

Proof of Latency, ”the protocol” or ”PoL”, is a protocol that when used in a P2P context, can offer a way of reducing network latency between peers by connecting with peers that have the lowest latency between each other, thus establishing a more optimal network than achieved simply by peering at random. PoL aims to make network bootstrapping better, making it easier to find performant, physically close-by peers on the first interaction with a PoL-enabled network. When using PoL for peer discovery, it can result in a network in which a peer can at least roughly estimate its latency to another peer before connecting to it.

Distance bounding protocols, as previously described, are a way of measuring latency between two participants in a relatively unspoofable manner. Unfortunately,

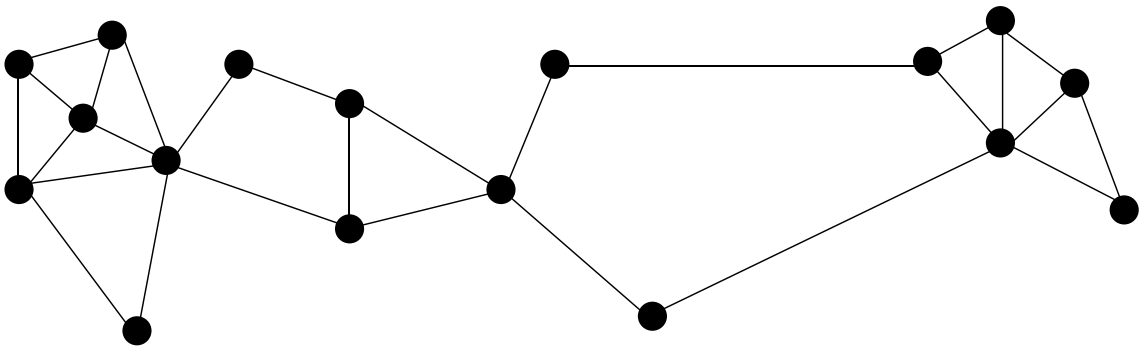

Figure 3.1: Example PoL network topology. Localized and highly connected clusters with high performance bridges, resulting in optimal routing

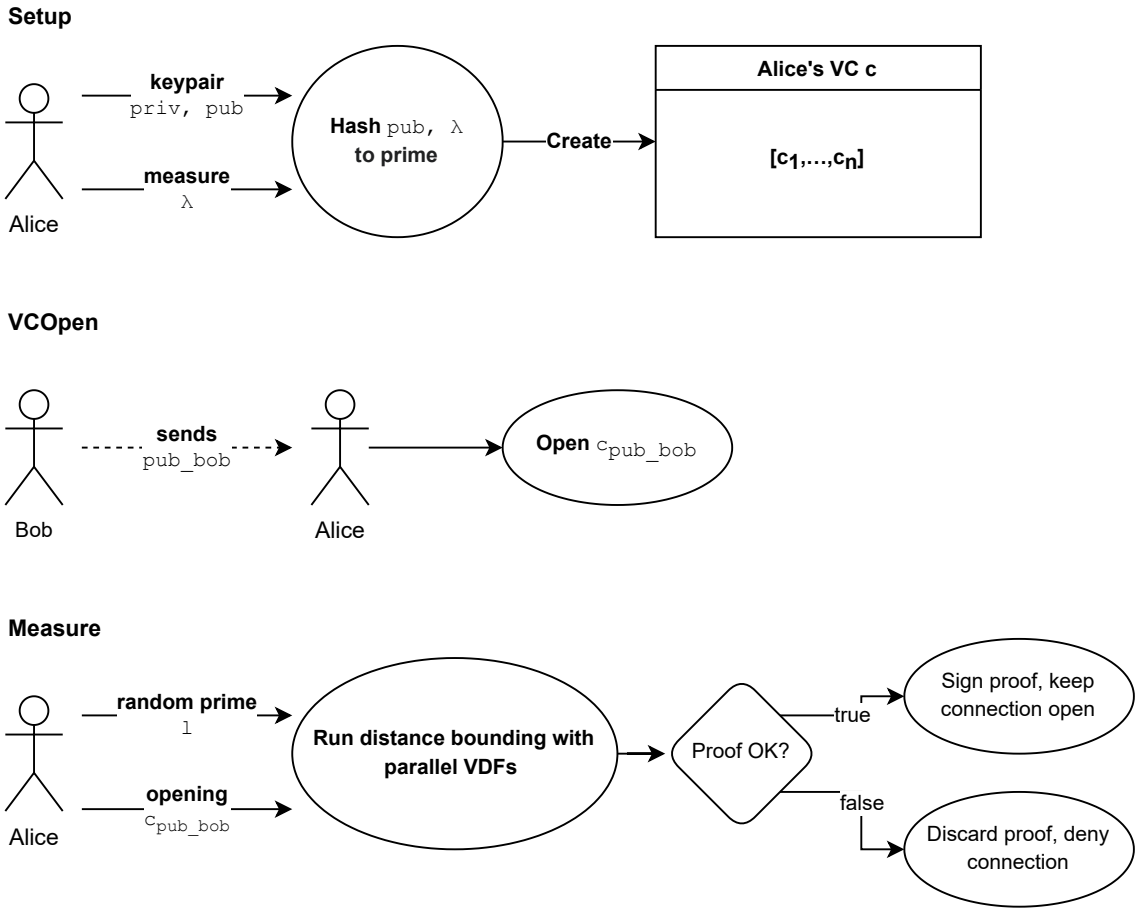

Figure 3.2: A high abstraction view of Proof of Latency.

they can only provide a valid proof between the prover and the verifier, and not a third party. This is because a distance bound could have been calculated at any point in the past, and the prover and the verifier could quite easily craft a proof of 0 latency between them. The motivation for PoL was to try to fix this with its use of verifiable delay functions, but further research revealed that some additional logic was needed to come closer to fulfilling this promise.

The protocol is trustless between the two computing parties and it requires no specific hardware from the participants. This said, special hardware for speeding up its evaluation and proving could make it a better protocol at distance bounding, as difference in the processing powers cause measurement error.

Aside from constructing P2P networks with better routes, the protocol could be used as a benchmark query or a proof of geographical location. A proof of a geographical location could prove useful, since GPS coordinates along with IP addresses can be spoofed.

3.1 Protocol Description

Symbol	Description
p	Prover
v	Verifier
pub	Public key
λ	Processing power
c	Vector commitment
N	The modulus used in a VDF. $RSA-2048$ in this context.
T	The difficulty of a VDF (the number of squarings)
g	The generator of a VDF.
l	A random prime number
VDF	A verifiable delay function
x	The result of a VDF
π	The proof of a VDF
Δ	The measured latency. Formally, $T_{VDF_v} - T_{VDF_p}$

Table 3.1: Symbols used in Proof of Latency

Proof of Latency is an interactive public coin[1] distance bounding protocol that produces a publicly verifiable proof signed by the participants. The protocol cannot be made non-interactive because the setup requires the participants to agree upon the parameters, and distance bounding protocols are inherently interactive.

The protocol is a tuple of three algorithms, $Setup(e, \lambda)$, $VCOpen(c, e)$, and $Measure(g, T, l_p, l_v)$. I've added a vague description on how the proofs would be published and then queried by other peers on the network under "Finalizing", though it is not a part of the protocol per se.

3.1.1 Setup

When a peer wants to use PoL, it must first make a vector commitment. This makes sure that once it has been made, the vector commitment is bound to that cryptographic identity and processing power. Thus, whenever a peer's setup changes, all previously calculated Proofs of Latency are invalidated, as its processing power has changed, resulting in a new vector commitment.

The vector commitment has two inputs: the committer's public key and its processing power measured in squarings achieved during a predefined delay that is defined by a parameter that should be sufficiently large to mitigate any startup lag or other processing performance fluctuations. This means that the committer measures how fast it can run the VDF, committing to this speed before calculating any proofs with other peers. This information is public, as a verifier needs to know the original committed processing power to measure if the prover did performance matching, which would lead to invalidating the proof.

All subsequent Proofs of Latency are based on the same vector commitment, compounding trust.

The setup is done by first measuring the computer's ability in doing modular

[1] A protocol where any random choice by the verifier is made public.

squarings in the RSA group. This measurement is done by doing as many modular squarings as possible in a protocol-defined timeframe, which should be large enough to account for undervolting and other types of performance and battery life optimizations especially in mobile devices. This measurement and the peer's public key then gets hashed to a prime number, which is then used in creating the vector commitment. For additional security, this vector commitment could be submitted to a public blockchain, or duplicated across a P2P network so that it is available even when the peer it belongs to is not connected to the network.

3.1.2 VC Opening

Whenever a new Proof of Latency is calculated, a new opening is done on the individual peers' vector commitment using the other participant's public key as the message. This way, a single peer can be included in the vector commitment only in one index. This opening's output is then used as the generator part advertized to the other peer.

The proof of the opening also gives the peers information about each other, as the vector commitment is required to be created with info of the peer's processing power and public key, the former working as a security parameter in the distance bounding.

3.1.3 Distance Bounding

The distance bound is computed by a race between two VDFs, which reduces the communication cost when compared with regular distance bounding protocols. The two participants are called the prover and the verifier, although they both are actively proving and verifying their VDFs. The difference here is that they both calculate a VDF and the verifier calculates the difference in iterations between the two. Also, the verifier has a head start in the VDF calculation, which results in

the measured latency being the round trip time. Due to this, the verifier ends the protocol, and has more power over the measured latency. This is a bearable consequence, as the prover can dispute the proof and refrain from signing it. Also, if the verifier deliberately was to slow down the proof generation by even a small amount, it would probably work against its incentives of receiving a closer connection.

The most important part of a VDF is the setup. If a prover wants to create a valid proof, it needs a priorly unknown starting point at first, so that it can prove for certain that it cannot have calculated the VDF in advance. Proof of Latency tackles this problem with a two-way generator/seed setup with pseudorandom numbers from which a multiple is taken and hashed to be in the group G integers modulo N.

In the setup, before the distance bounding, the prover and the verifier send their generator parts to themselves to construct a generator, or the starting point, for the VDF. Then, they both start calculating the same VDF independently. The prover then calculates the VDF up to a predefined threshold, and then sends the result at that threshold to the verifier, and starts calculating the proof. Proof is only sent after this, because since the VDFs the two parties calculate have the same input that was defined at roughly the same time, the parties can be sure that the result couldn't have been computed beforehand. The verifier stops its own calculation upon receiving the prover's result, and generates a proof of its own VDF. Then, it waits until it receives the proof from the verifier and then calculates the absolute difference between the amount of iterations between its own VDF and the prover's.

Since calculating a VDF is relatively fast for modern processors, a VDF over as little as a few milliseconds of time can be a valid way of measuring latency. Still, without an ASIC chip for calculating VDFs faster than any other available processor, these protocols are also a measurement of processing performance. This might introduce an unfortunate barrier for entry for mobile and IoT devices. If used to optimize a P2P network, the resulting network topology would result in a

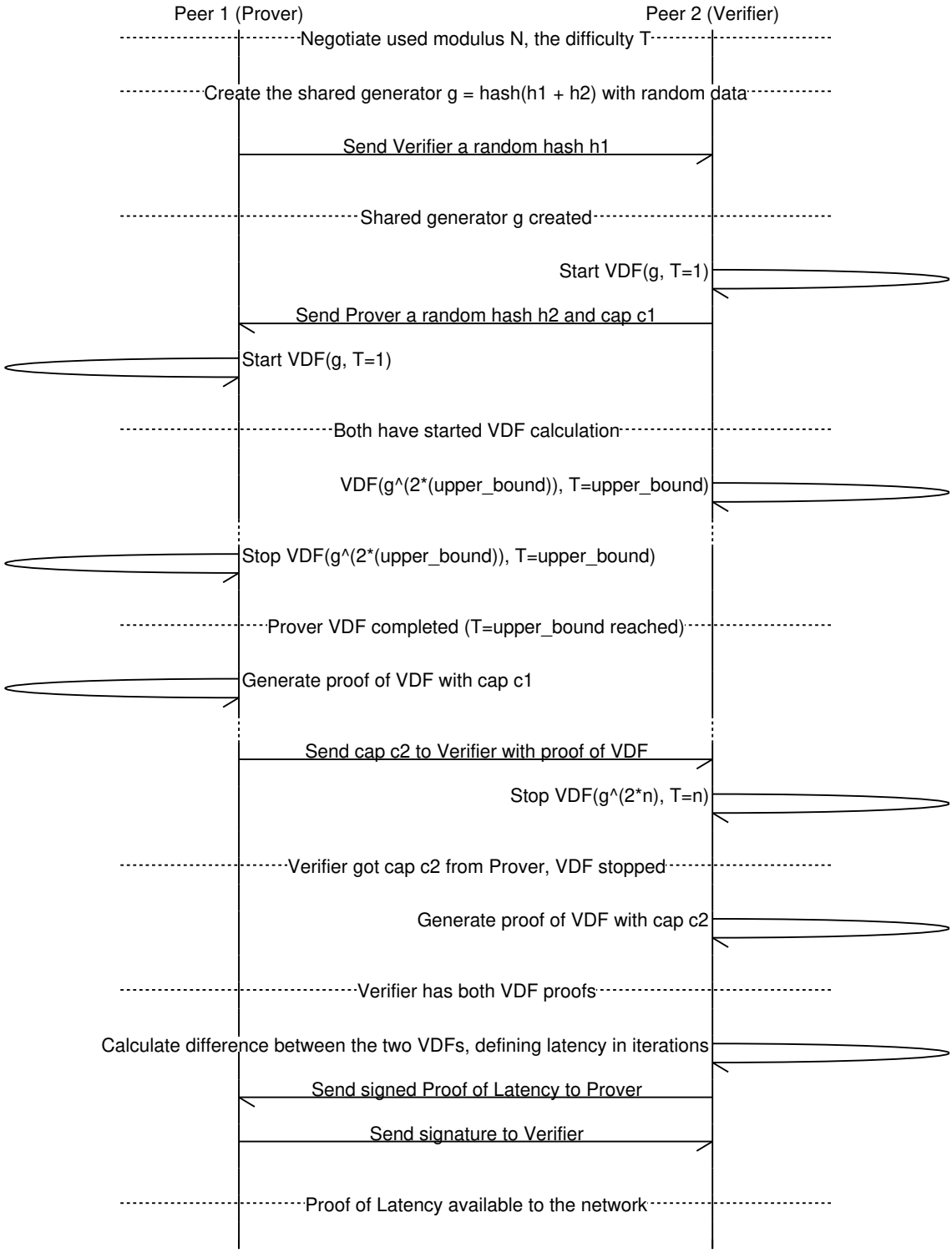

Figure 3.3: Distance bounding, Proof of Latency.

gradient that is defined by geographical location and the similarity in performance. This means that connectedness between mobile and IoT devices is going to be better than between devices that have a huge performance difference. Local performant devices would optimally serve as bridges between the localized clusters.

The distance bounding algorithm can be thought of as a state machine, or in fact, two state machines running in parallel.

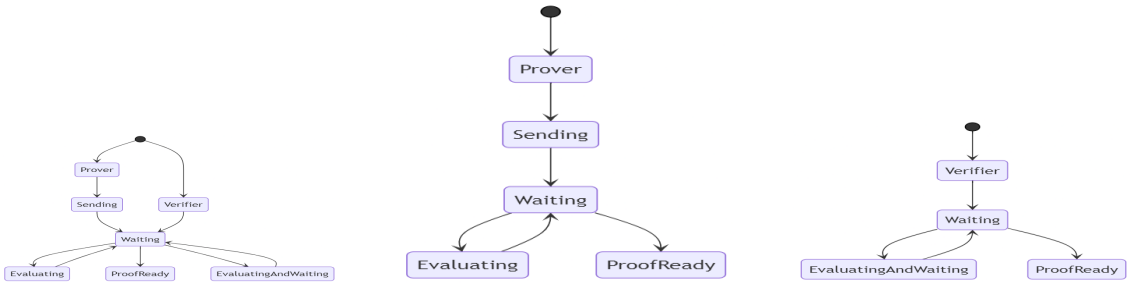

Figure 3.4: Possible states for the state machines.

3.1.4 Finalizing

In an imagined case of using Proof of Latency as a means of optimizing connections between peers in a P2P network, the network would be joined by selecting a pseudorandom peer based on public key string similarity. After joining the network by connecting to a few initial peers, the joining peer would then initiate the generation of PoLs with these new connections. Then, based on the measured latency, the peers would either query for the connected peers' Proofs of Latency, or drop the connection in case of the latency being over a user defined threshold.

Whenever a new Proof of Latency is generated, the proof would then get broadcast to n closest peers to the participants, enabling the n nearest peers to both triangulate their and other peers' positions in the network, or to connect to the participating peers if they have not already. Triangulating with the measured latencies together with IP address geolocation could enable geolocating peers and proving their physical location if that was needed with sufficient accuracy. [74]

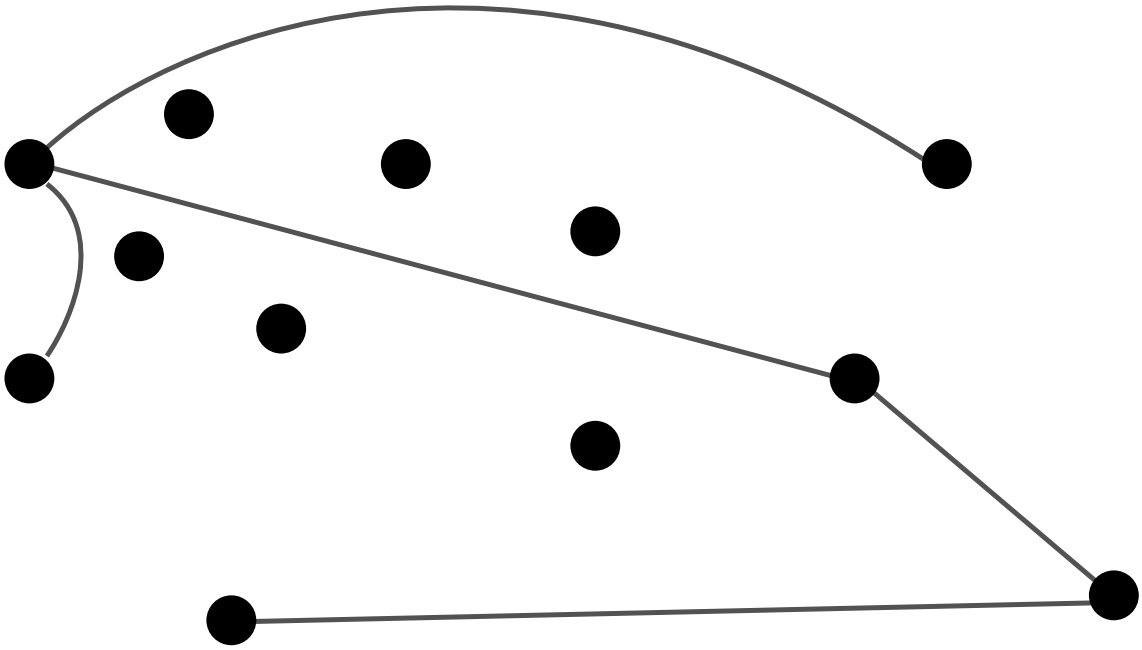

Figure 3.5: Example suboptimal routes achieved by randomly selecting peers.

The data structure that the responses would be served from could be a simple key-value database, indexed by both the public keys and the latencies, enabling for fast queries for exact peers, and queries for the closest peers to the responder.

Anyways, there is a huge number of things that can be done algorithmically after generating the proofs itself to create more performant networks or any of the use cases advertized in the next section.

3.2 Use Cases

Section 2.2 on peer-to-peer networking described some problems Proof of Latency is meant to help with. While peer-to-peer networking is an obvious area for Proof of Latency, the same protocol can be used also for computation performance benchmarking, VDF calculation using the latency proofs themselves, and to prove a geographical location. In the next sections I will describe these use cases more thoroughly.

3.2.1 Dynamic P2P Routing

The use case that drove me to begin working on Proof of Latency is an idea that there could be a trustless way for peers to tell other peers how much latency they have to other peers. In current P2P systems, if one peer were to tell another that it had a 10ms latency to another peer, there would be no quarantees of the 10ms latency holding true. If one could make a proof of that latency using cryptography, one could tell that the reported latency is true, and that some work has been done to calculate it. This kind of proof could also make eclipse attacks harder to accomplish by requiring a closer distance or more resources to reach a lower latency to the targeted peer.

Although DHTs like Kademlia do peer distribution basically at random based on identifier closeness, there are no guarantees that when a peer connects to the peers it has received from an another peer are also random, and thus the promise of random peer walk is lost. With Proof of Latency, new connections are formed in a hybrid manner together with a peer discovery mechanism that can provide protection against sybil attacks by introducing as random peers as possible, which helps in finding an approximate set of connected peers that are close to the possible minimum latency. Stochastic gradient descent works as a nice parallel from the machine learning field to describe randomness' use as an optimization tool.

3.2.2 Benchmark Queries

Like any time-lock puzzle [18], the distance bounding part of Proof of Latency could be used to query for performance. Processor development might render that property less effective in the future, unless one were to measure parallel processing capability with multiple VDFs, for example. Parallel multiple VDFs have been thought of and tested before, but calculating multiple separate VDFs has not been useful in previously imagined use cases, since it doesn't serve as a measure of time,

but performance.

Benchmark queries done with VDFs could serve as a part of a protocol in a distributed computation system. It would be very unfortunate to run large datasets against a data analysis model on a mobile phone, and it could be beneficial to prove that the peer that advertises its services can actually run the computation without hiccups.

There have been proposals for a VDF-as-a-function system [75], in which less performant peers could query for a VDF calculation if they don't have the means to do so in a time frame small enough themselves. The FPGA based system is being tested right now on Amazon Web Services cloud platform. A benchmark query using PoL could also be a part of such a system to verify peers' ability to perform VDF calculations faster than the querying peer.

3.2.3 VDF Calculation with Proved Latencies

Assuming the soundness of the proofs created by Proof of Latency holds, given a VDF challenge with difficulty T the prover could craft a route of n peers that have the needed amount of PoL-measured latency (in VDF iterations) between them. Even a weaker device could then start a game of broken telephone with these peers that would guarantee sequential message passing by repeated hashing with signatures, and give the resulting signed response as a VDF to the verifier. Because of the precomputation that has been done to prove the individual latencies between the peers, one could prove that a received challenge has taken roughly T amount of sequential operations to solve by relaying it through a route of peers on the network. In such a network, VDFs would be first-class citizens and always available, equally for every participant of the network. This means that both computationally powerful and weak peers could equally take part in things that require a delay function to be evaluated and proven without either benefiting from the situation

disproportionately.

3.2.4 Proving Geographical Locations

GPS coordinates or geohashes or any widely used geolocating method do not contain a proof of the reporter's proximity to the reported location. At first, I called Proof of Latency Proof of Proximity, but since it also measures processing power, I decided to go with latency instead, as there's many types of latencies that account for latency in P2P communication. Still, in roughly the same way as in dynamic P2P routing, a proof could be created that the prover is close to the verifier, and give or restrict access to resources based on this proof.

3.3 Security Analysis

Since Proof of Latency removes security quarantees by removing randomness from routing, some new attack vectors are introduced. Proof of Latency is not meant to be a one-stop shop towards a safer network, but rather an add-on to make the aforementioned use cases more efficient while still keeping security in mind. The following imagined attack vectors to create or advertize false proofs are a product of prior knowledge regarding sybil attacks in DHT networks [48, 76, 49, 77], research and the writer's thought process, and none have been tested out in practice, yet.

3.3.1 Advertising Dishonest Peers and Proof Spoofing

A malicious network participant can spawn an arbitrarily large number of new identities and network peers that are close-by in terms of latency, create multiple Proofs of Latency with them, and only advertise these peers to the rest of the network on their DHT. Two peers can also fake the proof if they co-operate by using VDFs they have already calculated and matched their outputs to be as close as possible.

Advertizing dishonest peers can be mitigated somewhat by requiring peers to renew their proofs regularly, using third party validators or a verifiable source of randomness, like a random beacon or a public timestamping service with signed requests. Also, trusted computing modules could be used to verify that the prover's software configuration has not been changed, removing most of the possibility for side channel attacks against using witnesses.

3.3.2 Performance Matching

The Proof of Latency protocol suffers from an issue, let's call it performance matching, which is a timing attack. Timing attacks are side-channel attacks against computer systems. Side-channel attacks utilize information from outer factors affecting the hardware and software the attacked program runs on. Timing attacks rely on gathering of timing data from the target. [78] In performance matching, gathering data could be done by connecting to the targeted peer by another protocol or comparing existing proofs of latencies from all peers that have calculated their latency with the target.

Performance matching enables attackers to perform an eclipse attack on low-performance devices by matching the attacker's performance with the targeted mobile device so that it is as close as possible in the difference between iterations in PoL. This attack could result in a complete network split, as performance differences between devices make peers inaccessible to the other side of the performance spectrum, as network latencies can't compete with the computing power. There is a fundamental barrier for entry to this attack, which is that the attacker must have a more performant processor to calculate the VDF than the target. The malicious peer cannot performance match if its performance is worse or similar to the targeted peer, because of the race between them, without the target co-conspiring with the attacker.

Proof of Latency prevents performance matching with a requirement for peers to advertize their processing power with a vector commitment. With this, any peer can prevent other peers from advertizing false latencies to it by measuring during the distance bounding if the other participant's responses fit its advertized performance.

3.4 Reducing the Attack Vectors

3.4.1 Hiding Information of PoL Results

If the publicized proof did not include both the VDF results and iterations, but just the iteration difference, an attacker would have less info on each peer. This would make attacking more difficult, requiring more queries and PoL runs on average before finding a vulnerable peer. This is more of an exploratory path that would need more research, but if it was possible to create a proof of the latency and only broadcast the iteration difference between two peers, it would be more difficult to gather info about peers and their computing performance.

3.4.2 Web of Trust

There's also a possibility of introducing a web of trust in parallel to PoL to recognize and deny connections to malicious peers more effectively. An example of such a system is SybilLimit, which adds a construction called trusted routes to DHT based routing [77]. I see that trusted routes in conjunction with Proof of Latency would be a good couple, as they tackle different parts of P2P routing, both making it more robust.

The problem of advertising dishonest peers could also be tackled with a trust based system, where suspected foul play could be made into a public responsibility of the peers, checking suspected foul play and broadcasting it to the network.

3.4.3 Peer Scoring and Usage Optimizations

Peer scoring is used regularly in P2P networks, and Proof of Latency serves as a peer scoring metric by itself, which can be used in various ways. To hinder eclipse attacks, connections to peers with the lowest latencies would be kept open for a longer time, even in the case of the peer not being online, and favoring new ephemeral connections that are farther. This would require the attacker to pursue multiple attack vectors at once to totally eclipse a peer.

3.4.4 Using a Third Party Validator

A way to make Proof of Latency more trustless is to introduce a randomly elected third party to the protocol. Since the soundness of the proof depends largely on the generator and its setup, the proof can be improved by requiring it to be salted[2] by a random input from an unbiased party. This could be done by introducing a salt pool to which a group of PoL validators post random salts from which the PoL provers then pick randomly. The validators listen for usage of the salts they have generated, and upon receiving the Proof of Latency inspect if their salt has been used or not, and sign the proof if it is found to be correct.

3.4.5 Using in Conjunction with String Similarity Based Peer Discovery

Proof of Latency is inoptimal for bootstrapping, as finding the physically closest peer from a large network by connecting to peers close to the bootstrap peers would mean having a lot of bootstrap peers. Because of this, it is a good idea to use PoL in conjunction with a string similarity based peer discovery mechanism, like Kademlia. Another way of achieving good breadth is to not discriminate by latency

[2] A value that gets added to data by contract before hashing.

at the start, trying to get as big of a slice of the network at start, optimizing for better connections later on.

Bootstrapping initially to as many peers as one can, the peer also decreases the probability of being eclipsed. As said before, randomized peer discovery is good for security, but bad for performance.

Chapter 4

Proof of Concept

To test out the Proof of Latency, I wrote a software proof of concept in Rust. I picked Rust as a programming language of choice because it has good properties when it comes to cryptography and distributed computing. Since distributed computing, or any server program for that matter, uses some sort of concurrency to get non-blocking responses to requests they receive, the programmer has to worry about race conditions, with most systems programming languages. Rust is different, however, as when a Rust program has been compiled in the default ”safe” compiler setting, a data race where two or more threads try to access a shared memory resource at the same time is simply impossible to achieve [79]. Also, as it is a compiled language without a garbage collector, and with lots of optimization in the compiler, Rust has good baseline performance, on par with C or C++ [80]. Rust has seen a surge of interest in the last few years, and it is used in many projects in production, notably in embedded and distributed computing, because of Rust's modern build tooling and robustness.

William Borgeaud's blog post [81] from November 2019 was the first reference I encountered that described VDFs in terms familiar to software engineers. The blog post and it's accompanying code [81], that also happened to be in Rust, helped me bootstrap the project.

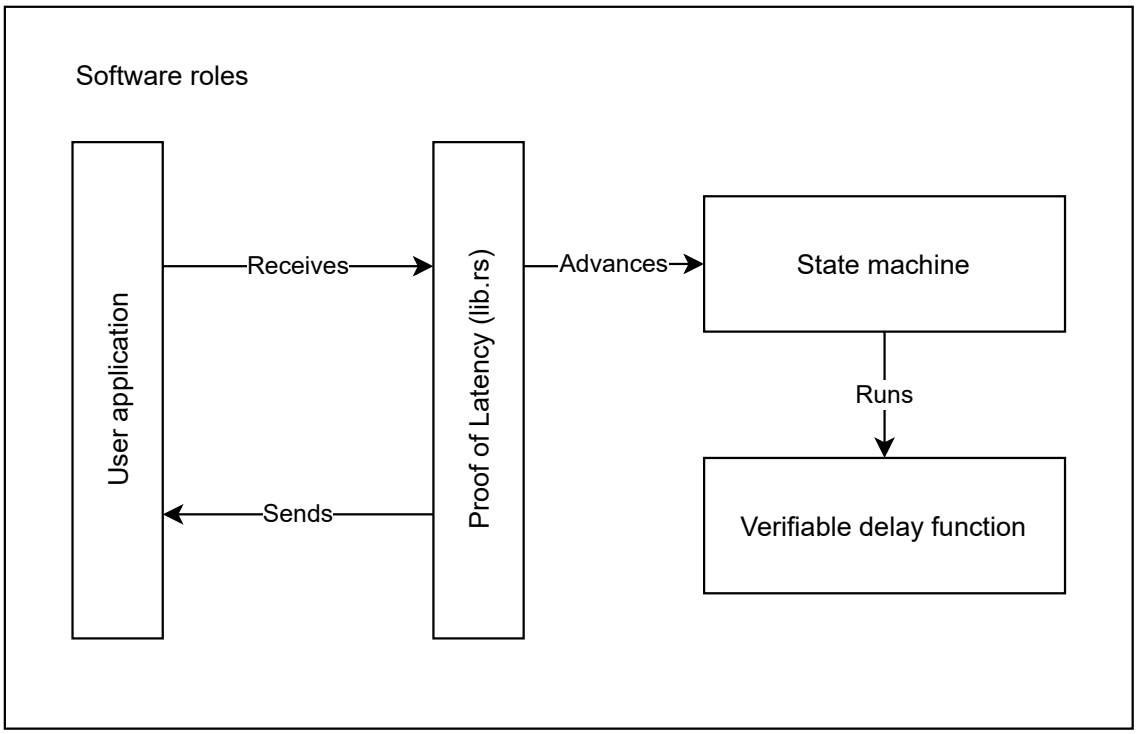

Figure 4.1: Software roles

In the implementation of the VDF protocol, the computations need to run asynchronously, on separate threads. There needs to be a separate task for listening to the other party's input, and preferably, a way to test the protocol locally without networking. The PoL protocol needs a VDF such that the prover's calculation can be ran indefinitely and then ended abruptly by a received "cap", a prime number from the verifier. No previous VDF library seemed to have this option, because the difficulty parameter T is usually predetermined, which is not the case for the prover in Proof of Latency.

The code is structured as follows: Runnable binary demo and the reusable library it uses resides in the top crate. They depend on subcrates that individually provide functionalities like VDF calculation and vector commitments.

The protocol is implemented as a state machine, which helps checking the protocol with model checkers, like TLA+. The protocol implementation uses variable names derived from the protocol description in fig , with slight adaptation from

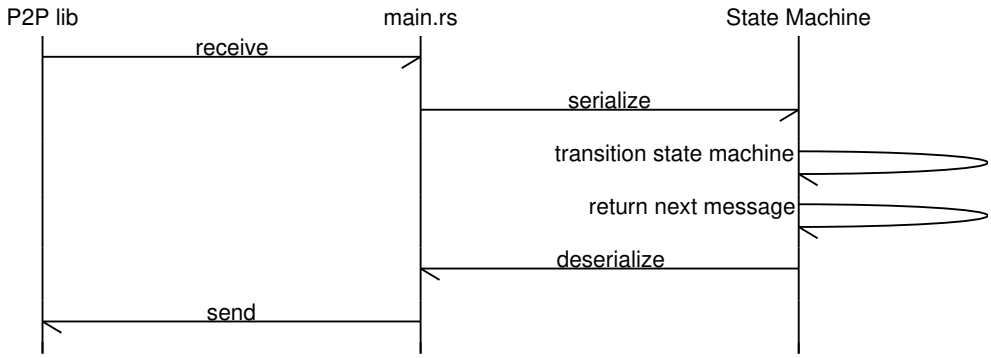

Figure 4.2: Usage of Proof of Latency as a library in a P2P context.

Listing 4.1: VDF iteration logic

```
impl Iterator for VDF {
    type Item = VDFResult;
    fn next(&mut self) -> Option<VDFResult> {
        if self.result.iterations < self.upper_bound {
            self.result.iterations += 1;
            self.result.result = self.result.result.clone().next_square();
            Some(self.result.clone())
        } else {
            None
        }
    }
}
```

mathematical notation to a more semantic approach.

The following is a code example of the threaded valuation part of a VDF inside Proof of Latency, containing evaluation stop logic for both the prover and the verifier: [H] The following is a code example of the proof generation of a VDF: [H]

The following is a code example of the verification of a VDF proof: [H]

The proof of concept is implemented as a library, so that it can be imported and used in other projects.

Listing 4.2: VDF proof generation

```
pub fn new(
  modulus: &Int,
  generator: &Int,
  result: &VDFResult,
  cap: &Int,
) -> Self {
  let mut proof = Int::one();
  let mut r = Int::one();
  let mut b: Int;
  let two: &Int = &Int::from(2);

  for _ in 0..result.iterations {
      b = two * &r / cap;
      r = (two * &r) % cap;
      proof = proof.pow_mod(two, modulus) * generator.pow_mod(&b, modulus);
      proof %= modulus;
  }

  VDFProof {
      modulus: modulus.clone(),
      generator: generator.clone(),
      output: result.clone(),
      cap: cap.clone(),
      proof,
  }
}
```

Listing 4.3: VDF proof verification

```
pub fn verify(&self) -> bool {
  // Check first that the result isn't larger than the RSA modulus
  if self.proof > self.modulus {
      return false;
  }
  let r =
      Int::from(2).pow_mod(&Int::from(self.output.iterations), &self.cap);
  self.output.result
      == (self.proof.pow_mod(&self.cap, &self.modulus)
          * self.generator.pow_mod(&r, &self.modulus))
          % &self.modulus
}
```

4.1 Tests

To gain assurance of the correctness of the VDF calculations I wrote unit tests for the VDF evaluation, proof generation and verification. To further ensure the soundness of the protocol, I added integration tests for Proof of Latency with separate, asynchronous threads for each execution. The tests reside in same files as the program code itself. The Rust standard library has good support for unit testing so that no separate library was needed.

In addition to regular unit tests with predefined input, I experimented with property testing. Property testing is a term that belongs somewhere between fuzzing[1] and unit tests. Otherwise it functions like regular unit tests, but it adds generated input into the equation. By adding generated, sometimes randomized input, one can check more thoroughly for edge cases. While being effective in recognizing some unexpected bugs, it still has a way to go when compared to formal verification, which is a testing method that aims to define a system with mathematical proofs, verifying that the output should always, apart from hardware issues, be correct according to the specification.

[1] Testing systems against a randomized, high volume of input.

Chapter 5

Conclusion

Since calculating a VDF is relatively easy for modern processors, a VDF over as little as a few milliseconds of time can be a valid way of measuring latency. Still, Proof of Latency is also a measurement of processing performance. This might introduce an unfortunate barrier for entry for mobile and IoT devices. When used in P2P routing optimization, PoL should result in a network topology organized by a gradient that is defined by geographical location, network connection speed and the similarity in performance between peers.

This means that closely located and similarly performant devices form strong local topologies that are bridged by their random connections to other peers and also by the connections they have to performant local peers. Highly performant devices form strong connections with each other whether they are located near each other or not, because other devices cannot compete against them in the race that is Proof of Latency, resulting in a network topology that is locally effective but global at the same time. The reality that VDFs can't be sped up by additional processors or cores means that the gradient will not be as drastic as one might imagine.

Proof of Latency as a peer scoring metric not only protects peers from eclipse attacks, but can also function as a way of speeding up the initial bootstrapping process by bringing peers more closely together. This also makes the resulting P2P

network more robust, due to its locality, in instances of internet stoppages between continents, censorship, or high network load. When used to prove a geographical location, Proof of Latency can combat fraud in applications that rely on GPS location.

5.1 Future Considerations

If this system was integrated to a blockchain or a publicly verifiable source of randomness for the initial setup, the proofs could be verified by anyone against consensus. Not only this would add trust to the latency measurements, but also speed up initial bootstrapping of the P2P network. When P2P networks eventually grow larger and larger, the network bootstrapping needs to be rethought to handle more traffic, be more decentralized, and be faster in its initialization. By getting introduced to the closest peers possible right at the start the user can experience a more performant network faster, lowering the barrier for entry and making first impressions better.

I believe the protocol is not cryptographically as secure as it could be, and the field is progressing at a mindnumbing pace. Making sure the protocol is VDF agnostic and quantum resistant would be a logical next step, since the field is still in progress of finding the best possible formulation of a VDF. Quantum computing can render most existing VDF implementations insufficient, and new VDF implementations could change the parameter logic fundamentally.

Chapter 6

Aknowledgements

First of all, this thesis would not have been made unless for my friends and colleagues at Equilibrium who enable me to work on the most interesting stuff on the planet. Thanks to Aurora, my friends, mom and dad, for inspiring me to pursue goals that at first might seem too difficult. This might have not helped with the deadlines at times but it surely helps with the end result.

I would like to thank Justin at VDF Alliance for introing me and my desire for a thesis pertaining to verifiable delay functions to Kelly and Simon at Supranational, who gave me ideas and guided me towards a research subject. Huge thanks for them for the continued interest in my work and efforts in improving the algorithms and hardware around VDFs. Thanks to Samuli for the extracurricular reviews and sparring me on cryptographic proofs and vector commitments. It ultimately was what helped me to transform a distance bounding algorithm to an actual cryptographic proof.

References

[1] Gunnar Kreitz and Fredrik Niemelä. Spotify – large scale, low latency, P2P Music-on-Demand streaming.

[2] David Mazières Petar Maymounkov. Kademlia. 2002.

[3] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Public-Key Cryptography – PKC 2013, Lecture notes in computer science, pages 55–72. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[4] R L Rivest, A Shamir, and L Adleman. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21(2):120–126, February 1978.

[5] Stefan Brands and David Chaum. Distance-Bounding protocols. In Advances in Cryptology — EUROCRYPT '93, pages 344–359. Springer Berlin Heidelberg, 1994.

[6] Foamspace Corp. FOAM_Whitepaper_May2018.pdf. May 2018.

[7] Deepak Maram and Iddo Bentov. GoAT: File geolocation via anchor timestamping. https://eprint.iacr.org/2021/697.pdf. Accessed: 2021-6-5.

[8] P Richard. Public key cryptography with a group of unknown order. undefined, 2000.

[9] Neal Koblitz. Elliptic curve cryptosystems. https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf, January 1987. Accessed: 2022-5-19.

[10] Victor S Miller. Use of elliptic curves in cryptography. In Advances in Cryptology — CRYPTO '85 Proceedings, pages 417–426. Springer Berlin Heidelberg, 1986.

[11] IBM docs. https://www.ibm.com/docs/en/zvse/6.2?topic=SSB27H_6.2.0/fa2ti_openssl_elliptic_curve_cryptography.html. Accessed: 2022-5-19.

[12] Antoine Joux, Andrew Odlyzko, and Cécile Pierrot. The past, evolving present, and future of the discrete logarithm. In Open Problems in Mathematics and Computational Science, pages 5–36. Springer International Publishing, Cham, 2014.

[13] Samuel Dobson, Steven D Galbraith, and Benjamin Smith. Trustless groups of unknown order with hyperelliptic curves.

[14] Nir Bitansky and Alessandro Chiesa. Succinct arguments from Multi-Prover interactive proofs and their efficiency benefits. Technical Report 461, 2012.

[15] Jean-Philippe Aumasson. Serious Cryptography. William Pollock, 2018.

[16] Ronald L Rivest and Burt Kaliski. RSA problem. In Henk C A van Tilborg, editor, Encyclopedia of Cryptography and Security, pages 532–536. Springer US, Boston, MA, 2005.

[17] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, Advances in Cryptology – CRYPTO 2018, Lecture Notes in Computer Science, pages 757–788, Cham, 2018. Springer International Publishing.

[18] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In Proceedings of the 4th conference on Innovations in Theoretical Computer Science - ITCS '13, New York, New York, USA, 2013. ACM Press.

[19] Paul Bottinelli and View All Posts by Bottinelli. Security considerations of zk-SNARK parameter Multi-Party computation. https://research.nccgroup.com/2020/06/24/security-considerations-of-zk-snark-parameter-multi-party-computation/, June 2020. Accessed: 2022-5-19.

[20] RSA Laboratories. The RSA factoring challenge. https://web.archive.org/web/20130507091636/http://www.rsa.com/rsalabs/node.asp?id=2092, May 2013. Accessed: 2020-12-1.

[21] Alin Tomescu. RSA accumulators. https://alinush.github.io/2020/11/24/RSA-accumulators.html, November 2020. Accessed: 2021-11-17.

[22] Georgios Konstantopoulos. A deep dive on RSA accumulators - good audience. https://blog.goodaudience.com/deep-dive-on-rsa-accumulators-230bc84144d9, January 2019. Accessed: 2021-5-3.

[23] Ivan Damgård. Commitment schemes and Zero-Knowledge protocols. Lect. Notes Comput. Sci., 1561:63–86, January 1998.

[24] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Advances in Cryptology – CRYPTO 2019, Lecture notes in computer science, pages 561–586. Springer International Publishing, Cham, 2019.

[25] Andreas Binzenhöfer and Holger Schnabel. Improving the performance and robustness of Kademlia-Based overlay networks. In Kommunikation in Verteilten Systemen (KiVS), pages 15–26. Springer Berlin Heidelberg, 2007.

[26] Ilya Grigorik. Latency: The new web performance bottleneck - igvita.com. https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/. Accessed: 2020-5-13.

[27] Liam Tung. Elon musk: SpaceX's internet from space should be good enough for online gaming. https://www.zdnet.com/article/elon-musk-spacexs-internet-from-space-should-be-good-enough-for-online-gaming/. Accessed: 2020-5-14.

[28] Satellite internet latency - VSAT systems broadband satellite internet latency - broadband satellite internet service latency. https://www.vsat-systems.com/satellite-internet-explained/latency.html. Accessed: 2020-5-14.

[29] Klint Finley, Will Bedingfield, Andy Greenberg, Medea Giordano, Brett Berk, Jess Grey, Matt Jancer, and Evan Ratliff. Brits approach (true) speed of light over fiber cable. Wired, March 2013.

[30] Wei Bi, Huawei Yang, and Maolin Zheng. An accelerated method for message propagation in blockchain networks.

[31] Protocol Labs. IPFS powers the distributed web. https://ipfs.io/. Accessed: 2020-9-24.

[32] Whyrusleeping. Issue #3320, go-ipfs, "writeup of router kill issue", October 2016.

[33] Dean Eigenmann. From kademlia to discv5. https://vac.dev/kademlia-to-discv5, April 2020. Accessed: 2020-5-20.

[34] Xing Shi Cai and Luc Devroye. The analysis of kademlia for random IDs. Internet Math., 11(6):572–587, November 2015.

[35] Harry Halpin. nym-litepaper.pdf. https://nymtech.net/nym-litepaper.pdf. Accessed: 2020-5-14.

[36] Stuart Cheshire and Marc Krochmal. Multicast dns. Technical report, RFC 6762, February, 2013.

[37] pdp. Name (mDNS) poisoning attacks inside the LAN. https://www.gnucitizen.org/blog/name-mdns-poisoning-attacks-inside-the-lan/, January 2008. Accessed: 2020-6-3.

[38] GoTenna. Extend the edge of connectivity. https://gotenna.com/. Accessed: 2020-9-13.

[39] Helium. Helium – introducing the people's network. https://www.helium.com/. Accessed: 2020-9-13.

[40] Mark Milian. Russians are organizing against putin using FireChat messaging app. Bloomberg News, December 2014.

[41] Sam Biddle. The inventors of bluetooth say there could be problems using their tech for coronavirus contact tracing. https://theintercept.com/2020/05/05/coronavirus-bluetooth-contact-tracing/, May 2020. Accessed: 2020-9-24.

[42] Mudhakar Srivatsa, Li Xiong, and Ling Liu. TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks. In Proceedings of the 14th international conference on World Wide Web, WWW '05, pages 422–431, New York, NY, USA, May 2005. Association for Computing Machinery.

[43] Peng Zhong Jeff Foley. Tendermint trust metric, August 2018.

[44] Stefanie Roos, Hani Salah, and Thorsten Strufe. Comprehending kademlia routing - a theoretical framework for the hop count distribution. arXiv:1307.7000 [cs], July 2013.

[45] Jakob Jenkov. Polymorph P2P network algorithm. http://tutorials.jenkov.com/p2p/polymorph.html. Accessed: 2021-9-8.

[46] Sebastian Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann. Eclipsing ethereum peers with false friends. arXiv:1908.10141 [cs], August 2019.

[47] Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. Perigee: Efficient Peer-to-Peer network design for blockchains. June 2020.

[48] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. A survey of DHT security techniques. ACM Comput. Surv., 43(2):8:1–8:49, February 2011.

[49] Thibault Cholez, Isabelle Chrisment, and Olivier Festor. Evaluation of sybil attacks protection schemes in KAD. In Ramin Sadre and Aiko Pras, editors, Scalability of Networks and Services, Lecture Notes in Computer Science, pages 70–82, Berlin, Heidelberg, 2009. Springer.

[50] Roel Peeters, Dave Singelée, and Bart Preneel. Three phases of distance bounding protocols. https://www.researchgate.net/figure/Three-phases-of-distance-bounding-protocols_fig1_220535502, July 2011. Accessed: 2021-11-24.

[51] Ioana Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay. Towards secure distance bounding? https://eprint.iacr.org/2015/208.pdf. Accessed: 2021-6-7.

[52] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Distance-bounding protocols: Verification without time and location. In 2018 IEEE Symposium on Security and Privacy (SP). IEEE, May 2018.

[53] Massimo Candela. Current status of IP geolocation, August 2020.

[54] Cristián Molina-Martínez, Patricio Galdames, and Cristian Duran-Faundez. A distance bounding protocol for Location-Cloaked applications. Sensors, 18(5), April 2018.

[55] Ventzislav Nikov and Marc Vauclair. Yet another secure distance-bounding protocol. https://eprint.iacr.org/2008/319.pdf. Accessed: 2021-6-7.

[56] Chia-Network/vdf-competition. https://github.com/Chia-Network/vdf-competition. Accessed: 2020-3-9.

[57] VDF alliance. https://www.vdfalliance.org. Accessed: 2020-4-29.

[58] supranational/vdf-fpga, April 2020.

[59] Benjamin Wesolowski. Efficient verifiable delay functions. Technical Report 623, École Polytechnique Fédérale de Lausanne, 2018.

[60] Krzysztof Pietrzak. Simple verifiable delay functions. Technical Report 627, IST Austria, 2018.

[61] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain, October 2018.

[62] Dan Boneh, Benedikt Bunz, and Ben Fisch. A survey of two verifiable delay functions. page 13.

[63] VDF research. https://vdfresearch.org/. Accessed: 2020-2-11.

[64] R Khalil, A Gervais, and G Felley. TEX-A securely scalable trustless exchange. IACR Cryptology ePrint Archive, 2019.

[65] Dan Robinson. Ethereum is a dark forest - dan robinson - medium. https://medium.com/@danrobinson/ethereum-is-a-dark-forest-ecc5f0505dff, August 2020. Accessed: 2020-9-21.

[66] Cory Mitchell. What is Front-Running in stocks? https://www.investopedia.com/terms/f/frontrunning.asp, September 2020. Accessed: 2020-9-21.

[67] Dan Cline, Thaddeus Dryja, and Neha Narula. ClockWork: An exchange protocol for proofs of non Front-Running. MIT Digital Currency Initiative, page 13, March 2020.

[68] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. Technical Report 183, 2018.

[69] José I Orlicki. Fair Proof-of-Stake using VDF+VRF consensus. August 2020.

[70] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. page 9.

[71] Stanford Video. Stanford BlockChain - day 2, 2020.

[72] Protocol Labs, Kelly Olson. Hardware acceleration of RSA VDFs - kelly olson, September 2020.

[73] Nir Drucker and Shay Gueron. Fast modular squaring with AVX512IFMA. In 16th International Conference on Information Technology-New Generations (ITNG 2019), pages 3–8. Springer International Publishing, 2019.

[74] Ethan Katz-Bassett, John P John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, and Yatin Chawathe. Towards IP geolocation using delay and topology measurements. In Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, IMC '06, pages 71–84, New York, NY, USA, October 2006. Association for Computing Machinery.

[75] Benjamin Devlin. Really low latency multipliers and cryptographic puzzles. https://blog.janestreet.com/really-low-latency-multipliers-and-cryptographic-puzzles/, June 2020. Accessed: 2020-9-24.

[76] Mahdi Nasrullah Al-Ameen and Matthew Wright. Design and evaluation of persea, a sybil-resistant DHT. In Proceedings of the 9th ACM symposium on Information, computer and communications security, ASIA CCS '14, pages 75–86, Kyoto, Japan, June 2014. Association for Computing Machinery.

[77] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky, and Feng Xiao. SybilLimit: A Near-Optimal social network defense against sybil attacks. In 2008 IEEE Symposium on Security and Privacy (sp 2008), pages 3–17, May 2008.

[78] BearSSL - Constant-Time crypto. https://www.bearssl.org/constanttime.html. Accessed: 2020-5-18.

[79] The Rust Project Developers. Races - the rustonomicon. https://doc.rust-lang.org/nomicon/races.html, November 2018. Accessed: 2020-9-24.

[80] Jesse Howarth. Why discord is switching from go to rust - discord blog. https://blog.discord.com/why-discord-is-switching-from-go-to-rust-a190bbca2b1f, February 2020. Accessed: 2020-9-24.

[81] William Borgeaud. understanding-vdfs. https://wborgeaud.github.io/posts/understanding-vdfs.html, November 2019. Accessed: 2020-3-9.