Proof of Latency Using a Verifiable Delay Function

Master's Thesis
University of Turku
Department of Future Technologies

Department of Future Technologies

2020
Jani Anttonen

UNIVERSITY OF TURKU
Department of Information Technology

Jani Anttonen: Proof of Latency Using a Verifiable Delay Function

Master's Thesis, 31 p., 0 app. p.
Department of Future Technologies
May 2020

Tarkempia ohjeita tiivistelmäsivun laadintaan läytyy opiskelijan yleisoppaasta, josta alla lyhyt katkelma.
Bibliografisten tietojen jälkeen kirjoitetaan varsinainen tiivistelmä. Sen on oletettava, että lukijalla on yleiset tiedot aiheesta. Tiivistelmän tulee olla ymmärrettävissä ilman tarvetta perehtyä koko tutkielmaan. Se on kirjoitettava täydellisinä virkkeinä, väliotsakeluettelona. On käytettävä vakiintuneita termejä. Viittauksia ja lainauksia tiivistelmään ei saa sisällyttää, eikä myäskään tietoja tai väitteitä, jotka eivät sisälly itse tutkimukseen. Tiivistelmän on oltava mahdollisimman ytimekäs n. 120 – 250 sanan pituinen itsenäinen kokonaisuus, joka mahtuu ykkäsvälillä kirjoitettuna vaivatta tiivistelmäsivulle. Tiivistelmässä tulisi ilmetä mm. tutkielman aihe tutkimuksen kohde, populaatio, alue ja tarkoitus käytetyt tutkimusmenetelmät (mikäli tutkimus on luonteeltaan teoreettinen ja tiettyyn kirjalliseen materiaaliin, on mainittava tärkeimmät lähdeteokset; mikäli on luonteeltaan empiirinen, on mainittava käytetyt metodit) keskeiset tutkimustulokset tulosten perusteella tehdyt päätelmät ja toimenpidesuositukset asiasanat

Keywords: list, of, keywords

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer applications, whether they are on the public internet or on a private network, have long preferred a client-server model of communication. In this model, the client application asks things from the server, and the server responds. Applications are increasingly moving towards a more distributed, peer-to-peer (p2p) networked model, where every peer, whether it's a computer as a whole or a single process running on one, serves as the both sides of the client-server model equally. Most uses of p2p do not even get communicated to the end user. For example, the music subscription service Spotify's protocol has been designed to combine server and p2p streaming to improve scalability by decreasing the load on Spotify's servers and bandwith resources.[2]

The main talking points of peer-to-peer networking lately have been cryptocurrency and blockchain technologies, categorized under the roof term of distributed ledger technology. Peer-to-peer has not been really shown in the public light as anything more than a technology to work around regulation and for doing lawless activities, because before blockchain, it was popularized for its use in file sharing applications, like Bittorrent, which it still sees use for.

Public blockchain networks need a way of synchronizing the state globally between a number of peers on a p2p network. Since the data model is sequential and all recorded history must be unchangeable, they need an algorithm to reach this total synchronization between states, a consensus algorithm. This problem is not unique to blockchains, but public blockchains have raised new issues that have sparked an ongoing development effort for new kinds of consensus algorithms.

Proof of Work is the most used consensus algorithm in public blockchains today, including Bitcoin, of in which whitepaper it was first described in 2008.[3] New algorithms have been introduced since to battle its resource intensiveness, including Proof of Stake, which requires network nodes participating in the voting of new blocks to stake a part of their assets as a pawn. Simply this means handing the control of some of the currency owned to the consensus algorithm if the peer wants to participate in the consensus. If a voter gets labeled as malicious, faulty, or absent by a certain majority, it 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.

One problem with Proof of Stake is that the 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 has generated an increasing demand for verifiable public randomness, that is pre-image resistant, meaning the output of the algorithm generating the randomness cannot be influenced beforehand. This created a motivation for an algorithm that would prevent multiple malicious actors from being selected to vote at once. A cure for this problem is called a verifiable delay function.

In 2018, two research papers were released independently with similar formalizations of a VDF.[4][5] By definition, a VDF is an algorithm that requires a specified number of sequential steps to evaluate, but produces a unique output that can be efficiently and publicly verified.[6] To achieve pre-image resistance, a VDF is sequential in nature, and cannot be sped up by parallel processing. There are multiple formulations of a VDF, and not all even have a generated proof, instead using parallel processing with graphics processors to check that the calculation is sequential.[7] This bars less powerful devices, like embedded devices, from verifying the result efficiently. Thus, generating an efficient proof that requires little time to verify is more ideal.[6]

Using a verifiable delay function, I propose a novel algorithm for producing a publicly verifiable proof of network latency and difference in computation resources between two participants in a peer-to-peer network. This proof can be used for dynamic routing to reduce latency between peers, and for making eclipse attacks[1] harder to achieve.

[1] Eclipse attack means polluting the target's routing table restricting the target's access to the rest of the network, which opens up other attack possibilities regarding consensus algorithms.

Chapter 2

Cryptography

Modern cryptography is based on relatively few but robust mathematical principles. The most dominant principles have historically been prime numbers and modular multiplication, but elliptic curve cryptography is increasingly considered to be the most secure method out there to keep things private.

Modular multiplication with large numbers has a useful property that you can exchange encrypted data without the participants knowing each other's private keys, but requiring a computation that is theoretically almost impossible to break. Prime numbers and elliptic curves can be used to create the variables used in modular multiplication, providing the basis for the robustness of the cryptosystem.

2.1 RSA

RSA is named after it's discoverers – Rivest, Shamir, Adleman. It is a public-key cryptosystem, meaning that it is based on a keypair, from which the other is a public key and other is a secret key. The public key is used to encrypt data, and the secret key is used to decrypt data. This means that everyone who has the public key can encrypt data that can't, with a large probability, be decrypted without the secret key. It works due to the fact that it's hard to factor the product of two large prime numbers.

Some institutions also have been publishing these products of two large prime numbers, claiming they have discarded the two prime numbers that were used in the creation. These are published as public puzzles with prizes for correct factorizations as high as 200000 US dollars, but the larger ones can also be used in cryptography to remove a trusted setup. These products are called RSA numbers, of which RSA-1024 and RSA-2048 are widely used. They can be considered relatively safe for production use, since the latest broken RSA number at the time of writing is RSA-250. Still, in practice, the keys could be still stored by the number's creator, requiring trust towards the institutions or individuals who have published them.

2.2 Diffie-Hellman Key Exchange

Diffie-Hellman key exchange is a way of generating a shared cryptographic key between two participants. The working principle is that both participants in the exchange do commutative calculations on the input that result in the same output, using prime numbers that result in good cryptographic primitives because of the prime factorization problem, that quarantees the participants' secret keys staying secret and the cryptography hard to break. Diffie-Hellman is most widely used in TLS. TLS is the protocol that is working behind secure http traffic, and most of us probably use Diffie-Hellman key exchange every day. There's also an elliptic curve version of the Diffie-Hellman key exchange, but let's first touch on an example with primes:

1. Alice and Bob agree on a large prime number $p = 941$ and a primitive root of it, $g = 627$.

2. Alice chooses the secret key $a = 347$ and computes $A = 390 \equiv 627^{347} \pmod{941}$.

3. Bob chooses the secret key $b = 781$ and computes $B = 691 \equiv 627^{781} \pmod{941}$.

4. Alice sends bob the number 390 and Bob sends Alice the number 691.

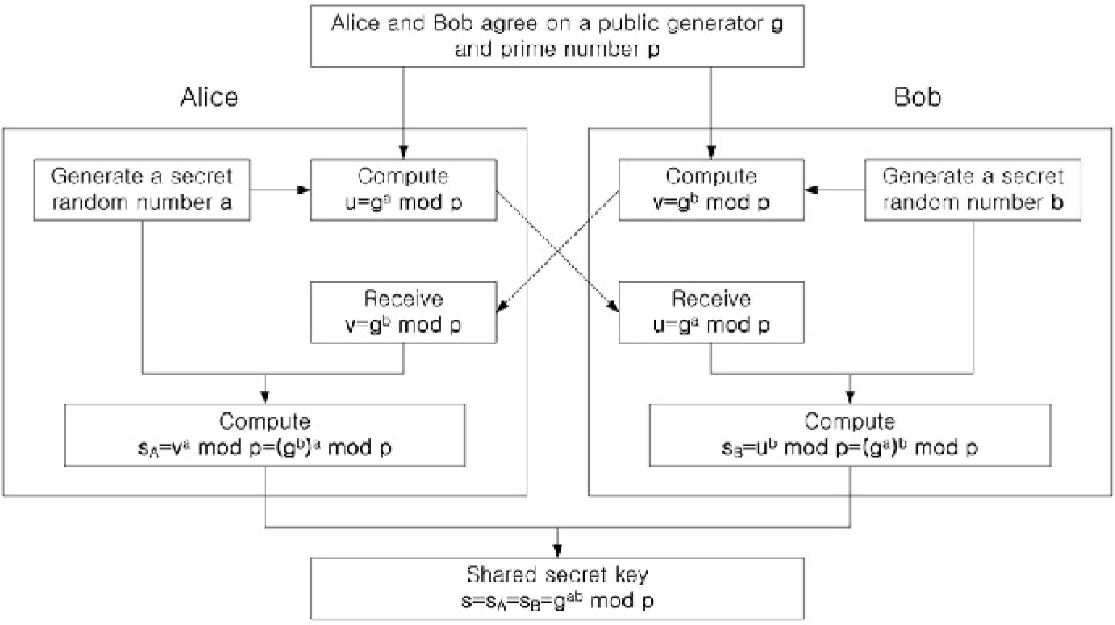
Alice and Bob agree on a public generator g and prime number p
Alice
Bob
Generate a secret random number a
Compute u=g^a mod p
Compute v=g^b mod p
Generate a secret random number b
Receive v=g^b mod p
Receive u=g^a mod p
Compute s_A=v^a mod p=(g^b)^a mod p
Compute s_B=u^b mod p=(g^a)^b mod p
Shared secret key s=s_A=s_B=g^ab mod p

Figure 2.1: Diffie-Hellman key exchange algorithm[1]

5. Since the operations are commutative, Alice and Bob are now able to compute the number $470 \equiv 627^{347 \times 781} \equiv A^b \equiv B^a \pmod{941}$

The number 470 is their shared secret, which they can use for symmetric encryption. Of course, in the real world, the modulus would have to be a large enough prime, because with 941 as the modulus the maximum number of tries to break it would be just 940.

2.3 Elliptic Curve Cryptography

Chapter 3

Peer-to-Peer Networking

Although most p2p networks utilize the internet, and thus familiar things like IP addresses to find other peers through the same routers than anyone else, there needs to be a way of signaling the participation to a particular p2p network. The simplest way to create a p2p network would be to share a single file with anyone who connects to you that has all the IP addresses you've ever connected with. This would be an example of an unstructured p2p network, which could work relatively well in a closed setting, like inside a LAN if there's no support for multicast DNS, or mDNS in short.

3.1 Ad Hoc and Zero Configuration Networking

Multicast DNS was proposed by Apple in 2013[8] as a way of discovering peers in a local area network in a zero-configuration manner, and it is used nowadays for resource sharing, like sharing printers. Multicast DNS doesn't 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 unique, and can be spoofed. If there's two clients with the same name, the first one to respond with its IP address to a query wins.[9] Security of zero-configuration and ad hoc networking must relay on cryptographic identities, so that a peer can verify itself with public-key cryptography, which makes the peers on the network practically unique and hard to spoof.

Zero-configuration networking in an unconstrained, global setting is possible with radios, using either dedicated meshnet radios like GoTenna or Helium, or by using the antennas inside mobile devices to form a network. These types of networks are usually called meshnets or ad hoc networks. Basically, Walkie-Talkies are the simplest form of a p2p network. Problems arise when you want to cross large distances, like country borders or large bodies of water. Mesh networking has been used most famously in protests worldwide by using a smartphone app called Firechat.[10]

Ad hoc mesh networks have a natural metric for latency, signal strength, and thus will not benefit from dynamic routing offered by DHTs as much. They can rely on Bluetooth RSSI[1] as a metric, or triangulate distances by cooperating with multiple peers. This type of system is used to locate emergency calls and contact tracing.[**?**] Mesh networks are relatively easy to route by connecting to the peers that are reachable without hops and have sufficient signal strength. These networks in their simplest form are very localized and performant, but to make them perform well over a distance there needs to be some satellite or internet hops on the way.

3.2 Distributed Hash Tables

Distributed hash tables are a way of pointing content to peers in a distributed network. In addition to indexing content in content-addressed networks like IPFS, they can function as routing tables. A hash table is a key-value mapping from a to b. What makes them distributed is the fact that the data stored is meant to be distributed between peers, with not a single peer keeping all the available data in its hash table, but relaying queries for resources it can't reach to other peers on the network. There are multiple versions of DHTs with different methods for prioritizing certain peers, using tree structures, sorting

[1] Received Signal Strength Indication

by identifiers, using computational trust et cetera.

3.2.1 Kademlia

Kademlia is a DHT designed by Petar Maymounkov and David Mazières in 2002. It is based on a tree of identifiers which 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.[11] These IDs are sorted in a binary search tree, with each node's position determined by the shortest unique prefix of its ID. Kademlia makes sure that any node in the network can locate any other node by its ID by making sure 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 3 network hops, meaning that the query gets relayed through two peers before reaching the requested resource.[12] Network hops are a necessary evil in distributed systems, and Kademlia does well in requiring on average a 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.[13]

The randomness of Kademlia is great at averaging the network hops required to reach a scarce resource. The downside is that it also averages everything else, increasing latency to closest connected peers, and increasing the minimum hops to reach a common resource.

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, which is locating k closest nodes to a given node ID. It is a recursive operation, which starts by picking alpha closest nodes from its closest non-empty bucket, and sending them all FIND_NODE calls. This is repeated until the initiator has queried and gotten responses from all k closest nodes it has seen.

Chapter 4

Verifiable Delay Functions

4.1 History

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.[14] Time-lock puzzles were originally created to send information into the future, which simplifies into 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. They describe the envisioned use cases for timed-release crypto as being the following, although they're very similar with the general concept of sending information into the future:

- 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 fall under the same definition, but introduce 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. At the same time, VDFs found use in blockchain applications as a way to achieve unspoofable randomness in elections and smart contract input.

4.2 Use Cases

Many have shown that there are more use cases for these algoritms than puzzles and random number generators. Some examples include preventing front running in p2p cryptocurrency exchanges, spam prevention and rate limiting[15] 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.

In the front running use case, this property can be used to restrict front-running in both centralized and decentralized exchanges.[16] In a centralized exchange, this means that the exchange waits for a specified time before it starts the fulfilling the trade. The VDF serves as a receipt for the order taker that there was no reordering and the take orders were processed in a FIFO[1] fashion.[17]

[1] First in, first out

4.3 Variations

There are multiple variations on the security principles used, since a VDF can use any group of unknown order as a basis for its unpredictability.

4.4 Similar Constructs

A VDF can only be calculated sequentially, but even without a proof there is a possibility to make the verification faster through parallellism. A non-verifiable delay function, or time-lock puzzle in short, can be still verified faster than the calculation, because there is no sequential requirement after the puzzle has been calculated, enabling to use multiple CPU cores or highly parallel graphics processing units for verifying the puzzle, like in Solana.[7]

4.5 Hardware Developments

All VDF applications can be made faster with hardware, and it has been estimated that with an ASIC chip a VDF can be calculated more than ten times faster than with a GPU.[18] If or when hardware specifically optimized for sequential squarings is commercialized, VDFs can become much more mainstream, and suffer less from computational differences, thus requiring less trust between the calculating parties. It's a race against time, since if an attacker put a considerable amount of resources in the development of such a chip themselves, they could potentially compromise the security of some blockchains, due to an assumption called weak subjectivity.[19] Weak subjectivity means favoring a blockchain with the most blocks in it, rather than focusing on a majority vote.[20] By creating blocks faster than the majority of the network, one could attack proof of stake systems relying on weak subjectivity with a proprietary chip.

Besides the development in the ASIC department driven by vdfresearch.org[15], 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, and DH algorithms, not to mention homomorphic encryption.[21]

Chapter 5

Proof of Latency

Proof of Latency, ”the algorithm” or ”PoL”, is a collection of two algorithms that when used in a P2P context, can offer a robust way of reducing network latency between peers on the network by minimizing the number of hops between peers that are close in terms of performance and geographical location.

The algorithm is relatively trustlessalternative version of the algorithm requires a trusted setting, preferably a trusted computing platform from all participating peers, while the second version is trustless and requires no specific hardware from the participants, while a trusted computing platform would make it fairer and more reliable, by not discriminating based on CPU performance. An ASIC VDF chip would also mitigate any of the later mentioned attack vectors by removing variance in hardware.

5.1 Use Cases

Previously this thesis has described quite thoroughly the problem Proof of Latency is trying to solve. While the use in P2P might be a no-brainer, there is at least one other use case that is somewhat used in the former, but can be thought of as a separate use case, perhaps with a different calculation but the same basic idea.

5.1.1 Dynamic P2P Routing

The use case that drove me to proceed working on this is the idea that there could be a trustless way of telling to other peers how much of a latency you have to another peer. If one peer were to tell you that it had a 10ms ping to another peer, there's no quarantees. Now, if we could make a proof of that ping with cryptography, you could tell by almost certain that it is true, and nobody is trying to advertise you peers that are trying to do an eclipse attack on you.

Although DHTs like Kademlia do peer distribution basically at random based on identifier randomness, there are no guarantees that when a peer connects to 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, you can be sure that the peers advertised are actually close-by. Of course, this is easily avoided by the attacker by spawning multiple peers in the same location.

5.1.2 Benchmark Queries

A protocol like this could be used to query for performance. In fact, Proof of Latency does that already, but 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.

This could serve as a part of a greater 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, 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.

5.2 Role of Latency in Distributed Systems

It's hardly a surprise, but latency is a huge factor in distributed systems, especially trustless, decentralized ones. Latency is mostly constrained by the speed of light, which can not be changed, and thus there are concrete factors that must be taken into account when designing a p2p system with routing. In 2012, the global average round-trip delay time to Google's servers was around 100ms.[22]

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 be 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 provides the network of satellites, claims that there's going to be a round-trip latency of about 20 milliseconds.[23] In legacy satellite internet access, the round-trip time even in perfect conditions is about 550 milliseconds.[24]This difference between legacy and newer satellite internet constellations comes from the difference in their orbits and the sheer amount of satellites involved. Legacy satellite internet uses geostationary orbits, which are very high, beaming 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 you're 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 at first seems like a stretch, but is believable when you take 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.[25] Intercontinental latencies can become much lower because of this. In blockchains, latency plays a role in the efficiency of the power used to achieve consensus.

Miners waste energy on a previous block as long as they don't receive information on the winner of the previous block race. Simulations by Wei Bi, Huawei Yang, and Maolin Zheng in their paper An Accellerated Method for Message Propagation in Blockchain Networks have shown that if you calculate 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, you 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%.[26]

5.3 Network Hops Increase Latency

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 in the internet already. Most of the P2P routing protocols used today are oblivious to the problem of introducing large hops to communications between two peers. These DHT-based protocols, like Kademlia, make the assumption that their users have fast internet access, and minimize the average latency by selecting connected peers basically at random.

While the randomness is great for preventing eclipse attacks, 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 individual connections less efficient. 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.[27]

Proof of Latency is made to improve the performance of current P2P networking solutions and make them future-proof, even for hops between planets, while still being compatible with some privacy-preserving P2P protocols, since it is agnostic of the addressing method used.

5.4 Protocol Description

The most important part when calculating any VDF is the setup. If a prover wants to create a valid proof, it needs a previously 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 RSA Diffie-Hellman key setup with random ephemeral prime numbers, which is an industry standard to achieve a previously unknown key with large certainty.

The proof generation removes trust between the two peers by introducing a race between them. 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.

First the prover and the verifier do a Diffie-Hellman key exchange to construct a previously unknown key. Then, they both start calculating a VDF in parallel. The prover only calculates the VDF up to a predefined threshold, and then sends the proof with a signature back to the verifier. The verifier then stops its own calculation, generates a proof of it, 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 easy 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

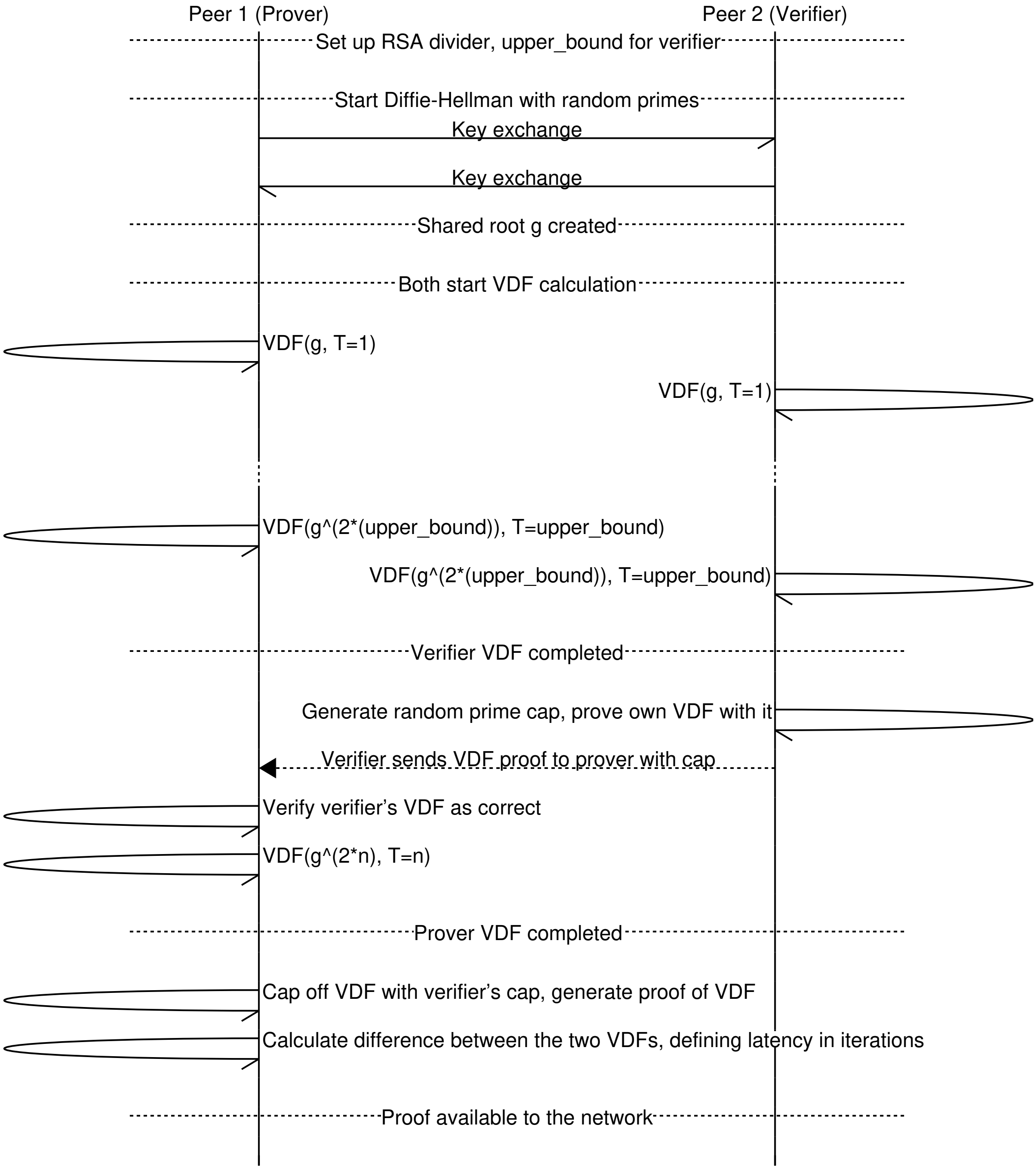
Peer 1 (Prover)
Peer 2 (Verifier)
Set up RSA divider, upper_bound for verifier
Start Diffie-Hellman with random primes
Key exchange
Key exchange
Shared root g created
Both start VDF calculation
VDF(g, T=1)
VDF(g, T=1)
VDF(g^(2*(upper_bound)), T=upper_bound)
VDF(g^(2*(upper_bound)), T=upper_bound)
Verifier VDF completed
Generate random prime cap, prove own VDF with it
Verifier sends VDF proof to prover with cap
Verify verifier's VDF as correct
VDF(g^(2*n), T=n)
Prover VDF completed
Cap off VDF with verifier's cap, generate proof of VDF
Calculate difference between the two VDFs, defining latency in iterations
Proof available to the network

Figure 5.1: Protocol Diagram, Proof of Latency

barrier for entry for mobile and IoT devices. The second version of PoL is meant to tackle this problem by creating a performance and latency gradient to the network. The network topology results in a 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.

5.5 Alternative Version

This alternative version of the algorithm is based on an attempt to simplify the initial Proof of Latency algorithm. It drops the requirement of both parties having to run a VDF, leaving it to the prover. In short, the prover proves the verifier their latency with a regular Wesolowski VDF proof, the difference being that all the input has been negotiated with the verifier. Since the network peers are trying to compete with each other for the least latency, they win the game by calculating the least amount of VDF iterations. In this version of the algorithm, this soon becomes a problem. Another peer can tell you that it only had the time to calculate 1 iteration of its VDF before it received back your signed message of the input it gave you, messing up the measurement metric. A peer would win a place in your DHT by just calculating 1 iteration of a VDF and thus faking low latency. If you flipped the metric around, it would favor large latency over a small one, defeating the purpose of the algorithm.

This version only works if the peers can trust each other, but in that case we could also resort to using ping. Ways of introducing trust would be using a trusted computing platform, but even that could be vulnerable to undervolting or other physical tampering. One way to improve upon this version of the algorithm would be to introduce a VDF handshake between the two peers, with the two flipping their responsibilities in the algoritm at the middle. They would then compare their results, and accept the other peer into their DHT if they were close enough in terms of iterations.

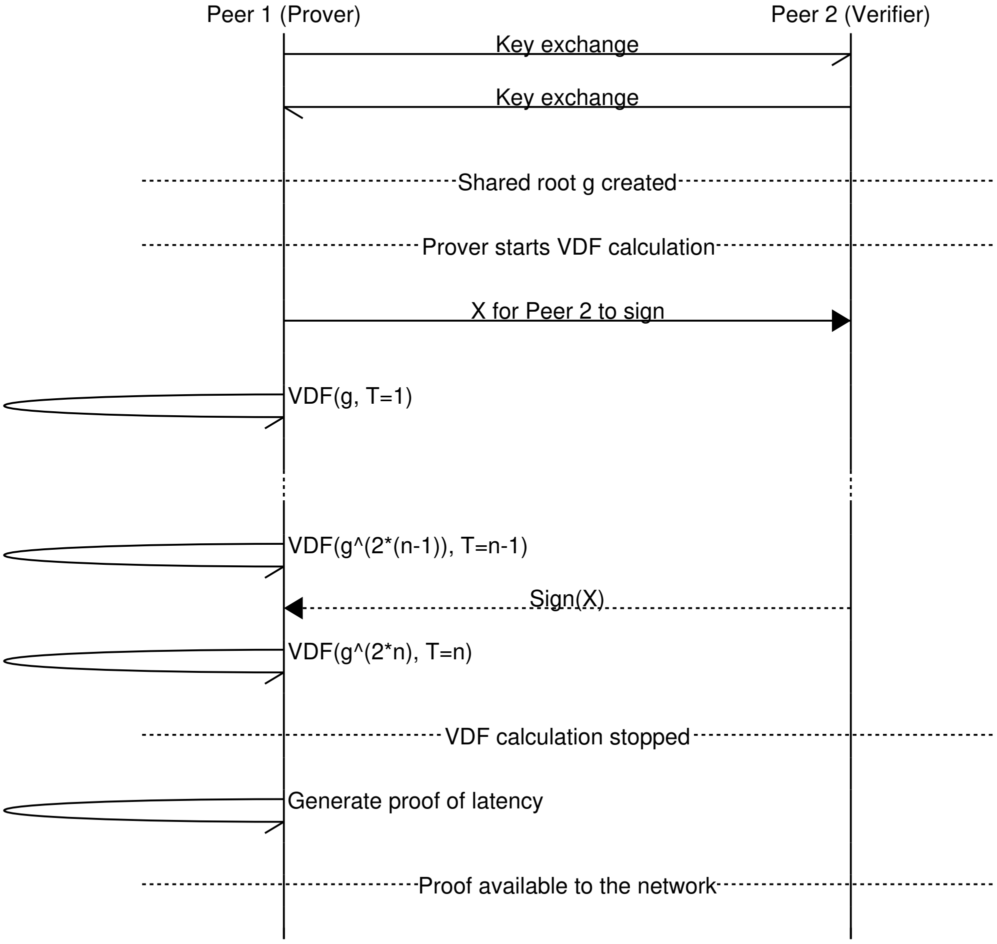

Figure 5.2: Alternative, minimized version of PoL

I largely prefer the original version. It merges the calculation of the two VDFs into the same timeframe, making the total calculation time of the algorithm shorter. I came to the conclusion that the original version is actually the most resistant to attacks and requires less trust from the participants.

5.6 Attack Vectors

Since Proof of Latency removes security quarantees by removing randomness from routing, some new attack vectors are introduced. The two versions of the algorithm differ much in terms of security. Proof of Latency is not meant to be a one-stop shop towards a safer network, but rather an add-on to make things more efficient while still keeping security in mind. The following attack vectors are a product of prior knowledge, research and the writer's thought process, and none have been tested out in practice, yet.

5.6.1 Advertising Dishonest Peers

A sybil network participant can spawn an arbitrarily large number of new identities and network peers that are close-by, 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.

This can be mitigated somewhat by requiring peers to renew their proofs regularly or by using a public source of randomness like Ethereum 2.0's Beacon Chain. It's hard to make systems completely trustless, and this was a good example while doing this research. Unless the proofs are based on a blockchain, there's no ”blind” trust that can be achieved with PoL on how low a latency another peer has to another unless it is calculated yourself by a process you control. Even in this case, the proofs would need constant updating and would at most have a minimum trusted latency roughly equal to the blocktime. This doesn't leave many chains out there at the time of writing that have the capability to do this, and even them would maybe have to introduce some localized consensus to decrease latency even more.

While this sounds bad, if the protocol were to be enforced in a way that it required a proof to be generated between an advertiser and every advertised peer, these sybil, dishonest peers would not propagate falsely to the rest of the network but would remain a separate blob.

5.6.2 Performance Matching

Both versions suffer from the same issue, let's call it performance matching, which is a timing attack. Timing attacks are a family of attacks against computer systems called side-channel attacks, which attack the fact that although software is abstract and can be very well fit against any attacks on the software level, outer factors still affect the hardware it runs on. Timing attacks rely on gathering of timing data from the target.[28] In performance matching, this means 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. Now again, if the algorithm had a trusted platform module requirement, or we had an ASIC for repeated squarings, this wouldn't be an issue on either of the versions of the algorithm. This attack could result in a complete network split.

5.7 Protecting Against Performance Matching

5.7.1 Hiding Information of VDF Results

Zero-knowledge proofs could be used to protect against performance matching. If the publicized proof didn't include both the VDF results and iterations, but just included 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.

5.7.2 Web of Trust

There's also a possibility of introducing a web of trust in parallel to PoL to recognize and shut out malicious peers more effectively. An example of such a system is SybilLimit, which adds a construction called trusted routes to DHT-based routing.[29] I see that trusted routes in conjunction with Proof of Latency would be a great couple. The problem of advertising dishonest peers could also be tackled with a trust-based system. Introducing too much trust into the system could also make PoL useless, since there could be a simple ping protocol with a time-based latency instead.

5.7.3 Switch Responsibilities

This remedy aims to improve PoL version 1 by flipping the responsibilities of the prover and the verifier. The calculating party wants to prove its latency to the verifier by actually calculating the VDF itself, requiring the verifier to respond as fast as it can to the prover's messages. This way, only a proof of one VDF is needed, which the prover then advertises to the network, telling connecting peers exatly the number of calculations it got to calculate before the verifier responded to it.

There's a catch, though. The prover can still lie to other peers by slowing down their VDF calculation. By flipping the responsibilities the protocol becomes trustless for the prover, but not for others. The flip felt promising at first glance, but I soon realized there's no better way to do this than the second version, where trust is shared and not assumed.

5.7.4 Results

Since the network peers are trying to compete with each other for the least latency, they win the game by calculating the least amount of VDF iterations. In the first version of the algorithm, this soon becomes a problem. There's no way to tell how fast another computer is, so in this case spoofed results of 1 iteration would win every time.

This version only works if the peers can trust each other, but in that case we could also resort to using ping. Ways of introducing trust would be using a trusted computing platform, but even that could be vulnerable to undervolting or other physical tampering. This problem drove me to revert back to my original idea, which I'm calling the version two of Proof of Latency.

Chapter 6

Proof of Concept

To test out Proof of Latency, I made a software proof of concept in Rust, using some open source projects, blog posts and previous knowledge as a base for the work. William Borgeaud's blog post[30] from November 2019 in particular was the first reference I encountered that described verifiable delay functions in familiar terms to a software engineer like me. The blog post and it's accompanying code[30] helped me bootstrap the project.

First of all, I knew I had to make the VDF run asynchronically or if possible, in another thread, because it would be one of two VDFs needed in the protocol. For Proof of Latency it was also needed that the prover's calculation could be ran indefinitely, and ended abruptly by a received proof from the verifier.

The code is published as a crate on crates.io under the name proof_of_latency and is meant to be as reusable as possible, able to plug into any P2P connection layer imaginable that uses public key cryptography to identify its peers. To achieve this, I used structs for logical instances of a single VDF, for example. This is mainly to keep program state in instances of these structs, removing possible side effects.

Chapter 7

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, 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. The second version of PoL is meant to tackle this problem by creating a performance and latency gradient to the network. The network topology results in a 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.

References

[1] Seok Hee Jeon and Sang Keun Gil. Optical secret key sharing method based on Diffie-Hellman key exchange algorithm. *J. Opt. Soc. Korea*, 18(5), October 2014.

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

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

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

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

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

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

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

[9] 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.

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

[11] David Mazières Petar Maymounkov. Kademlia. January 2020.

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

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

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

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

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

[17] 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.

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

[19] VDF-based RNG with linear lookahead. https://ethresear.ch/t/vdf-based-rng-with-linear-lookahead/2573, July 2018. Accessed: 2020-2-19.

[20] Vitalik Buterin. Ethereum wiki, weak subjectivity, August 2019.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

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

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

[29] 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.

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