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, 21 p., 0 app. p.
Department of Future Technologies
November 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.[1]

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.[2] 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.[3][4] 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.[5] 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.[6] 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.[5]

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 harder to achieve.

Chapter 2

Cryptography

2.1 RSA

2.2 Asymmetric Cryptography

2.2.1 Diffie-Hellman Key Exchange

Diffie-Hellman key exchange is a way of generating a shared cryptographic key between two participants. The two parties, who are called Alice and Bob, want to generate a shared encryption key to communicate secretly between each other. Alice and Bob first agree on a large prime number p and a nonzero integer $gmodulop$. This info is shared between Alice and Bob, and serves as the starting point for the encryption scheme.

Next up, Alice picks a secret integer that she keeps to herself, a,

2.3 Elliptic Curve Cryptography

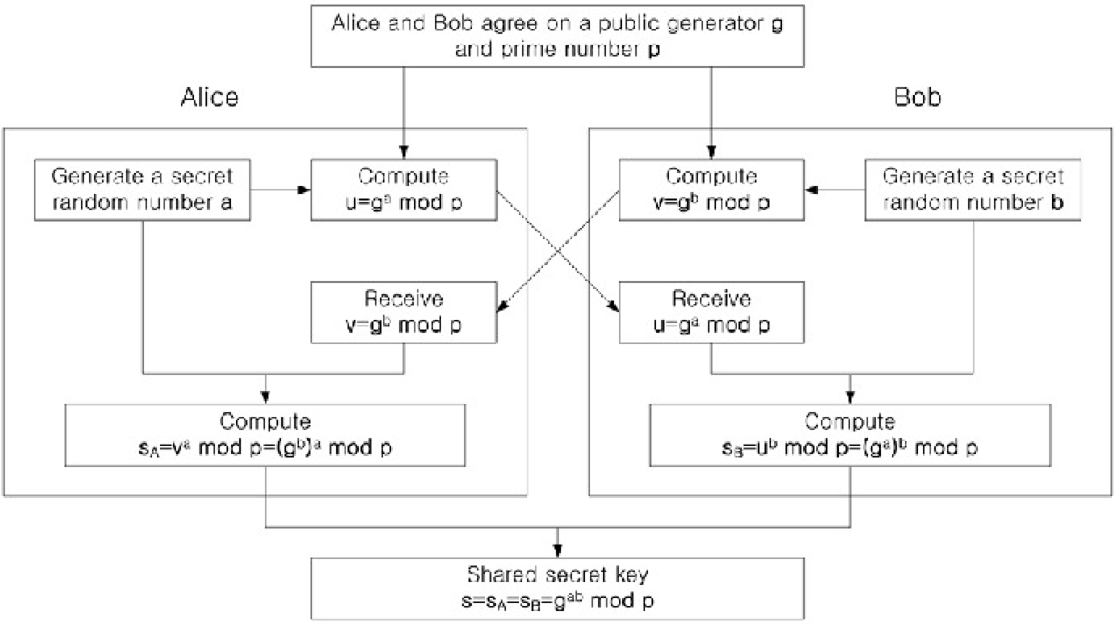

Figure 2.1: Diffie-Hellman key exchange algorithm[7]

Chapter 3

Peer-to-Peer

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 just a regular key-value store, a 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 DHT, but relaying queries that it can't answer to other peers on the network.

3.1 Routing

3.1.1 Distributed Hash Tables

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.

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.[8] 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.

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.

Chapter 4

Verifiable Delay Functions

4.1 History

Verifiable delay functions are based on time-lock puzzles. Time-lock puzzles are computational sequential puzzles that require a certain amount of time to solve.[9] Time-lock puzzles

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.

4.2 Applications

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[10]

All of these 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. If or when hardware specifically optimized for sequential squarings is commercialized, VDFs can become much more mainstream, and suffer less from competition, thus requiring less trust between the calculating parties.

4.3 Variations

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.[6]

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 first 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 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.[11]

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 latency of about 20 milliseconds.[12] But, since speed of light restricts us to about 100ms of minimum global latency[13], those claims cannot be trusted. In legacy satellite internet access, the round-trip time even in perfect conditions is about 550 milliseconds.[13]

5.2 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.[14]

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.

5.3 First Version of PoL

The first iteration of the algorithm is based on an attempt to simplify the initial Proof of Latency algorithm. After some thought, I came to the conclusion that the initial iteration, which ended up as the second and final version of the algorithm, is actually the most resistant to attacks and requires less trust from the participants.

5.3.1 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.

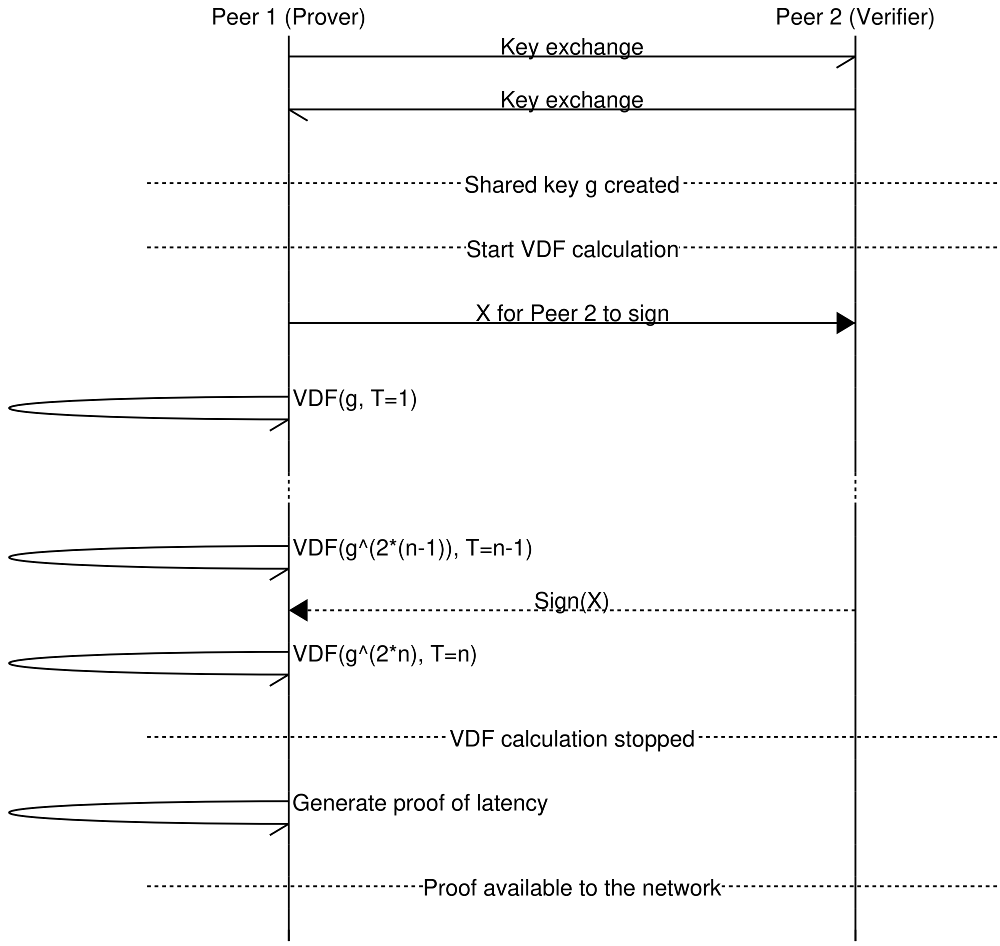

Figure 5.1: Proof of Latency, Version 1

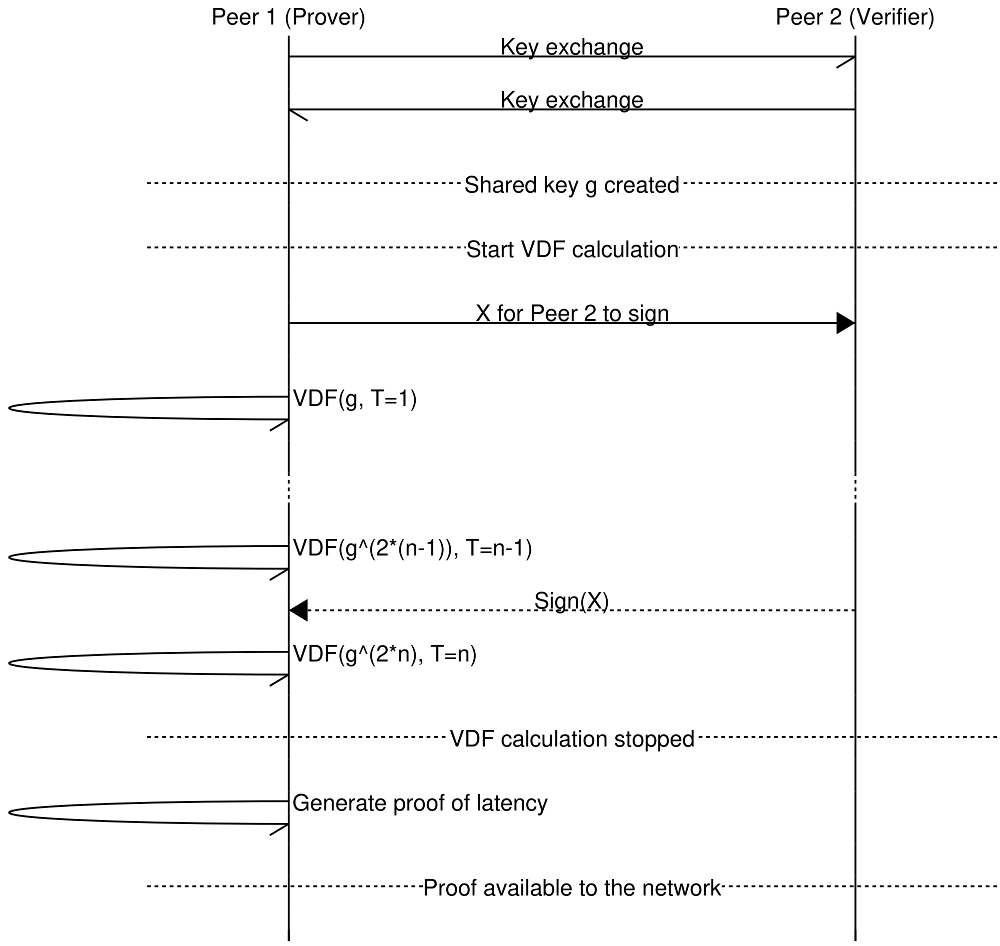

Figure 5.2: Proof of Latency, Version 2

5.4 Second Version of PoL

The second version of PoL introduces a race between the two peers. Like in the first version, there's a prover and a verifier. 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 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 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.

5.5.1 Performance Matching

Both versions suffer from the same issue, let's call it performance matching, which is a timing attack. It 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, this wouldn't be an issue on either of the versions of the algorithm. This attack could result in a complete network split.

5.6 Protecting Against Performance Matching

5.6.1 Zero-Knowledge Proofs

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.

5.6.2 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.[15]

5.6.3 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.

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.

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

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

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

[11] 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.

[12] 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 Accessed: 2020-5-14.

[13] 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.

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

[15] 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.