University of Zurich[UZH]

MASTER PROJECT – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of a Signaling Approach for the Detection of Net Neutrality Breaches using Blockchain-based Smart Contracts

Lawand Muhamad, Daniel Demeter
Zurich, Switzerland
Student ID: 16-729-147, 19-756-451

Supervisor: Eder John Scheid, Muriel Figueredo Franco
Date of Submission: March 5, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Project
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Die Diskussion um Netzneutralitaet (NN) entstand im Laufe der Jahre über das Ausmass, in dem es Netzbetreibern erlaubt sein soll, in die Datenübertragung im Internet einzugreifen und NN-Verletzungen zu verursachen, indem sie Datenverkehr mit höherer Priorität bevorzugt behandeln. Diese Arbeit stellt das Design und die Implementierung einer Blockchain-basierten Smart Contract Lösung vor, um NN-Verletzungen zu erkennen und Anreize für kollaborative Beitraege zu schaffen. Das Tool basiert auf einer Client-Server Architektur, bei der sich eine Client-Anwendung mit dem Anwendungsserver verbindet und Datenströme ausgetauscht, währendessen auf beiden Seiten Messungen durchgefuehrt werden. Die Lösung ist in der Lage, NN-Messanfragen von Clients eines bestimmten Internet Service Providers (ISP) zu signalisieren und Messungen dezentral und unveränderlich zu speichern. Um den Nutzern der Blockchain einen Anreiz zu geben, sich aktiv am System zu beteiligen und Messungen hinzuzufügen, wurde zusaetzlich ein Bounty-System eingeführt, das Nutzer belohnt zum System beizutragen.

The discussion around Net Neutrality (NN) arose throughout the years about the extent to which network operators should be allowed to interfere with the data transfer on the Internet and cause NN breaches giving preferential treatment to higher priority traffic. This work presents the design and implementation of a blockchain-based smart contract solution to detect NN breaches and to incentivize collaborative contribution. The tool is built based on a client-server architecture, where a client application connects to the applications server and data streams are exchanged, while measurements are conducted on both ends. The solution is able to signal NN measurement requests from clients of a given Internet Service Provider (ISP) and store measurements in a decentralized and immutable manner. Additionaly, to incentivize the users of the blockchain to actively participate in the system and to add measurements, a bounty system has been included rewarding users who contribute to the system.

Contents

Chapter 1

Introduction

The Internet is a network composed of many individual networks for transporting data. In each of these individual networks, the network operator has the power to decide whether the transmitted data should be treated equally, or whether they should be treated differently from one another, giving preferential treatment to higher priority traffic [17].

The concept of Net Neutrality (NN) arose in the discussion about the extent to which network operators should be allowed to interfere with the data transfer on the Internet. NN describes the principle that all data is treated the same during its transport through the Internet, regardless of the sender, recipient, service, application or content [21]. Thus, aims to protect against discriminatory interference with data traffic. There are different opinions in the discussion as to what types of interference should be permitted or prohibited and which exceptions should be made in this regard.

Up until now, the Internet has worked according to the principle of "best effort". It means that as long as the network still has free capacity available, all the incoming data will still be transmitted in the same way [1]. Therefore, whoever sends content does not need to conclude any agreements with the various actors involved on the Internet to make sure that the data reaches the recipient. This openness of the Internet has enabled many innovations and opens up new opportunities for opinion forming and information gathering. The Internet has thus developed into an economically and politically central communications infrastructure all around the world.

The public debate on NN began in the USA in 2003. Proponents of a legally established NN fear that the positive characteristics of an open Internet without NN could be lost. Opponents, however, argue that legal regulation could prevent improvements and innovations in the networks. In December 2016, the Federal Communications Commission (FCC) removed the NN Laws in the United States [6]. Since 2009, the EU has had regulations in place to protect NN. In April 2014, the European Parliament has taken a first step towards further regulation of NN. The legislative process is currently underway.

1.1 Motivation

Several countries around the world have different regulations in place in regards to NN [26]. In the European Union (EU), NN is specifically granted, whereas in Switzerland no specific laws exist. Future profit-oriented decisions in which any NN breaches would not result in any consequences for the causer can be a major disadvantage for end users and threaten the role of the internet. Therefore our objective is to develop a system which detects such NN breaches and incentivizes users to participate in detecting such breaches. Detecting NN breaches is a difficult task, but several measurement techniques are available. There are several methods of traffic differentiation detection and the relevant metrics include connectivity, loss rate, delay, throughput, and sent and received bandwidth [21].

In this sense, this work presents the design and implementation of a blockchain-based smart contract solution to detect NN breaches and to incentivize collaborative contribution. The solution is able to signal NN measurement requests from clients of a given Internet Service Provider (ISP) and store measurements in a decentralized and immutable manner. Furthermore, to incentivize the users of the blockchain, we include a native currency rewarding users who participate in this system. However, as the blockchain-based smart contracts is used to share measurements and provide the incentives for measurements and cannot perform the measurements, a monitor application was designed and implemented that performs network measurements, such as jitter, packet loss, throughput and bandwidth.

In the course of this work it will be evident that the proposed solution and the implemented metrics are precise and deliver the expected results. However, it must be noted that a clear signal of a NN breach is difficult to determine, as many factors have influence on those metrics. Therefore, when using the system, it is important to keep in mind that any measurements and the resulting conclusions made, must always be interpreted with care, as it will be shown in this work.

1.2 Thesis Outline

The thesis is structured as follows. Chapter 2 provides an introduction to NN, metrics used in order and the background of blockchain technology. In Chapter 3 previous and related work in regards to NN detecting tools is presented and a comparison is made. In Chapter 4 the design choices and the implementation of the solution during this project is presented. Chapter 5 evaluates the accuracy of the measurements, the interpretation of the metrics measured the viability of the economic incentive model in place, and ability to operate at scale. Chapter 6 concludes this report and provides directions for future works.

Chapter 2

Background

This section provides an introduction to the topic of NN and to the necessary technical details to understand the concepts of blockchain and blockchain-based smart contracts.

2.1 Net Neutrality

The Internet is composed of subnetworks managed by three tiers of ISPs (*cf.* Figures 2.1). In the lowest level of the ISP tiers are the Local ISPs. Customers, such as private households, corporations or public authorities, usually enter a contract with a local ISP. Such contracts specify that the local ISP ensures internet access given a specific bandwidth to the customer. Tier 2 ISPs are the providers on regional to national scale. Their customers are usually local ISPs who have to pay some cost to the regional ISPs based on traffic generated. The Tier 1 ISP are at the top of the hierarchy and they have global reach. Lower-tier ISPs have to pay a fee for passing their traffic from one geolocation to another which is not under the reach of that ISP. Tier 1 ISPs build infrastructure, such as the Atlantic Internet sea cables, to provide traffic to all other Internet providers around the globe. These subnetworks are operated by telecommunications service provider, schools, universities, public authorities or other public and private companies. Generally, ISPs at the same level connect to each other and allow traffic to travel freely amongst each other. Such ISPs are called peers.

The end users of the Internet exchange data with each other via this network of subnetworks. This data can be webpages, emails, or other forms of data. The data is sent back and forth based on the Internet Protocol (IP) between two parties. For example, the data of a single email is split up into smaller data packets, which then travel from one end to the other via different routes. In the systems of the receiving device the data packets are reassembled and put together to form the original email. In the individual networks, routers ensure that the data packets are sent by using the fastest possible way and arrive at the right place.

A significant amount of data packets are sent around the globe every second. Even though an exact number does not exist, one can infer how many data packets each second is transferred through these subnetworks by looking at a Netflix UHD video streaming example.

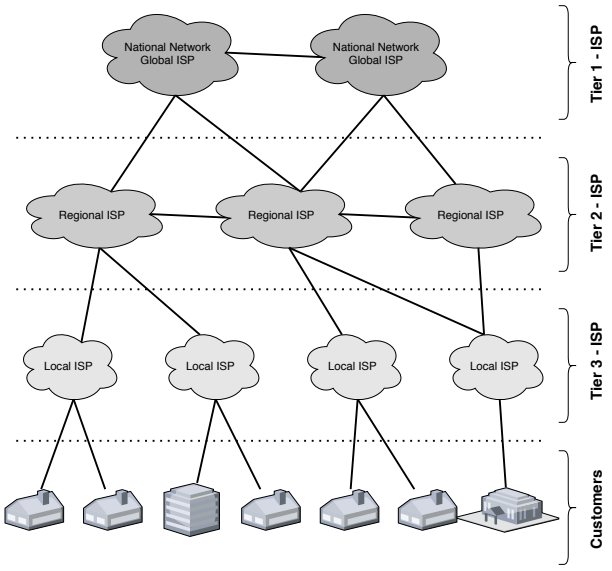

Figure 2.1: Internet Organization and ISP Tiers. Based on [4]

Though this video streaming example does need much more data packet transfers than an average website visit does, given the fact that video streaming makes up a majority of the data packet traffic [3] and also that there are frequently used other examples such as file transfers with potentially higher bandwidth usage, it can be used as a good indicator. Netflix recommends to have at least a bandwidth speed of 25 megabit per second to be able to stream 4k video[13]. Assuming that a one-second-UHD video consumes about 20 megabit of data and given that a data packet is, with today's standards, at most 1500 bytes, the Maximum Transmission Unit (MTU) used in Ethernet LANs, a rough estimation can be given on how many data packets are sent within that second to receive the UHD video on the end users screen. 20 megabit per second divided by 1500 bytes results in approximately 1667 data packets. If we multiply this with the hundreds of millions of devices that are connected on the internet and share data across each other, we can assume how high the true number would be. The global flow of these data packets is what we know as the internet traffic.

The discussion around NN begins with how this traffic that is being generated is being treated. There exist many different definitions of NN around the world, but the one sentiment that is common in each one is that "every type of traffic must be treated equally, regardless of its origin, destination and/or content" [21].

The definition of equal in this context can be quite ambiguous. As data travels through an array of routers to reach its destination, it can encounter many different interpretations of "equal" along the way. Questions often arise around the data transfer of streaming services, VoIP, and advertisements as well. Throttling such high-bandwidth services would degrade the experience of one user, but might improve the experience of many others as a result.

However, even if one arrives at a consistent definition for NN, the next topic of debate is often how it should be applied. A simple example of positive effects of breaching NN are emergency services. It is generally agreed upon that emergency services' data needs should have preferential treatment, thereby breaching NN.

2.1.1 Arguments For and Against Net Neutrality

One of the first points of contention between the proponents and the opponents of NN began with the DARPA project, which developed the TCP/IP protocol suite, the de-facto Internet standard [25].

IPv4, the IP layer, has a field in its header called the Type of Service (ToS), which was subsequently redefined to be Differentiated Service (DiffServ) [14]. The DiffServ field was designed to provide a framework and the building blocks to enable deployment of scalable service discrimination in the Internet. Its main goals was to appropriately label time-sensitive traffic. General user traffic usually consists of real-time voice and video chat, live-streaming or emergency services and non-real-time data. Such differentiation would then allow the prioritization of more important traffic. This, however, lead to a debate in which the opponents of NN argued that the specification of DiffServ documentation also allowed for the possibility of permitting differentiated pricing of internet services.

In Table 2.1, a summary of the arguments of the two opposing sides in regards to NN is presented

Table 2.1: Summary of Arguments in Favor and Against NN

Arguments for NN	Arguments against NN
The advantages of the Internet are being eliminated if the control of the data traffic is not in line with the long-term public interest. The Internet is today the most important socially central communications infrastructure not only for personal communication but also for governments around the world. It is an important driver for the economic, cultural and political development of the society. Network operators with content have the opportunity to give preferential treatment to their services by giving preferential treatment in the transport of such data or by not charging the data retrieved by the customer from their own services against monthly data caps.	Video streaming nowadays takes the majority of bandwidth usage around the world. Owing to this, service providers want to have additional revenue sources to compensate for the additional infrastructure costs that are associated with the additional investments. In addition, the growing demand for more broadband enables network operators to influence their revenues by offering appropriate price plans for their customers. It is contradictory for network operators to compete on the flat rate while at the same time claiming capacity bottlenecks that can hardly be overcome.

2.1.2 Violation/Breaches Metrics

Because the definition of NN is ambiguous and the Internet is a complex set of networks, subnetworks, and devices, there exists a wide variety of practices that could be considered a breach of NN. However, only a few are covered by formal regulations. In the following sections, these practices that are clearly considered NN violations are described.

Discriminating data traffic based on their content, protocol, origin, or destination, will be referred to as Traffic Differentiation (TD). Many different forms of TD exist; however, the most common TD mechanisms are *(a)* traffic shaping and *(b)* traffic policing. Both of these make use of discriminatory routers, which will give lower-priority traffic fewer resources.

Routers can employ a variety of scheduling algorithms for forwarding packets. Because routers only have a limited amount of space in their buffer to store incoming packets before relaying them, when the router's buffer is full, it has to decide how to handle incoming packets. The order in which packets are forwarded is also up to the algorithm.

Scheduling algorithms that do not differentiate between traffic are called neutral schedulers. Non-neutral scheduling algorithms are a tool that can be employed to discriminate between different types of traffic. First-come-first-served schedulers simply forward packets that have arrived first, and Drop-Tail schedulers drop any incoming packets when the buffer is full. As opposed to these neutral schedulers, Weighted Random Early Detection is a non-neutral scheduler, because it drops lower-priority packets with a higher probability.

Traffic policing drops the packets of lower-priority traffic more often, while traffic shaping delays and drops lower-priority packets in favor of higher-priority packets [21]. Many other types of breaches exist, including, but not limited to, routing lower-priority packets through a longer path, port blocking, and the use of discriminatory scheduling algorithms.

Because there are so many ways in which NN can be breached and because data travels through so many networks, it is impossible to provide a single metric or strategy that can accurately measure and locate such instances. Ultimately, the aspects that can be measured are the quality of service at different endpoints. The most common metrics that can be observed to inform such measurements are *(a)* latency, *(b)* packet loss, and *(c)* jitter.

Latency refers to the average length of time it takes for a data packet to travel from the device to its destination and back. Experiencing high latency can be an indicator of a breach of NN, when the packets of a certain service are not relayed immediately or routed on a longer path, in order to give other services preferential treatment.

Packet loss is the percentage of packets that do not arrive in a given time-frame. Experiencing high rates of packet loss may be a consequence of some routers employing non-neutral scheduling algorithms, and will lead to serious deterioration in the quality-of-service.

Finally, jitter is the variance in latency. High jitter may be a symptom of inconsistent routing of data packets, as this would mean the packets have to traverse different paths;

some longer than others. The lower the measurements of each of these metrics, the better the quality of the service.

2.1.3 Regulations

Net Neutrality in Switzerland

In 2013, the Swiss Federal government working group started to develop a report, which was intended to explain and illustrate the core of the debate about the state of NN in Switzerland at that time. It was published in 2014 on the Federal Office of Communications (OFCOM) website [1].

According to the report, at that time, there were no regulations specified in regards to NN in Switzerland. The report listed the following laws which are linked to treatment of data transport in Switzerland:

- If network operators block data instead of transporting it, or if they prefer or disadvantage data over other data, neither consumers nor content providers can, based on the freedom of speech and information in Article 16 of the Swiss constitution, lift the blockade.
- The secrecy of telecommunications in Article 43 of the law on telecommunications does not protect against unequal treatment in data transport. The unequal treatment is, provided that the network operators contractually agree to the possibility of different treatment, also not to be regarded as forgery or suppression of information according to article 49 of the Telecommunications Act.
- Providers of content, services and applications via the Internet are offered through the existing Telecommunications Act no possibilities to take action against possible obstruction of their access to the network operators facilities and customers.

According to these Swiss laws and findings the report concluded that there are no specific laws regarding NN. Further, based on these findings and the increased debate about NN, the Nationalrat Balthasar Glättli filed a motion on NN, which had support from politicians in all parties in the Swiss government. The motion's text translated to English from German reads as follows:

"*In the planned partial revision of the Telecommunications Act, the Federal Council is instructed to legally anchor NN in order to guarantee transparent and non-discriminatory data transfer via the Internet. NN must be explicitly laid down as a basic element of freedom of information and opinion and must apply to both fixed and mobile networks.*"

However, the Bundesrat did not recommend to accept the motion. They recognized that the subject around NN is of controversial international debate. The Federal Council had decided on a compromise. They argued that at present there are no signs that tougher regulation was necessary. However, transparency was important, so informed customers could change providers if they see the need for it.

In the end, the motion was not accepted by the National Council of Switzerland in a vote with 26 votes to 17 rejected. The opponents stressed on the one hand that there were no violations of NN in Switzerland. They based this on the tailor-made NN definition of the Telecom operators [15]. Until today there has not been many advancements made in this regard and the topic around NN in Switzerland is still open.

Net Neutrality in Europe

In the EU, NN has been protected under the ordinance 2015/2120 [23]. The goal of the ordinance is to establish a common set of rules to ensure equal and non-discriminatory handling of data traffic in the provision of Internet access services and related rights of the end user. The Regulation is intended to protect end-users while ensuring that the Internet's "ecosystem" can continue to function as a driver of innovation. Furthermore, reforms in the area of roaming should create confidence among end users, including when travelling in the Union, and lead to prices and other conditions in the Union becoming harmonized.

The ordinance provided for the NN rules to be reassessed by April 30, 2019, some three and a half years after the ordinance came into force. While the European Commission decided not to amend the Regulation, the Body of European Regulators for Electronic Communications (BEREC) decided to revise its implementation guidelines for national regulatory authorities. These updated guidelines were published on June 11, 2020[**?**]. This was BEREC's response to the NN debate, also due to the introduction of next generation 5G technologies which reignited the debate around NN. 5G opens up new technological possibilities for network operators to treat traffic transmitted over their networks in a differentiated manner. In particular, this involves a technology called "network slicing" [2]. This allows services to be given preferential treatment, which BEREC does consider problematic. "Deep Packet Inspection" remains prohibited in the EU, not least for data protection reasons. With the help of this technology, network operators could have qualified and directed traffic in their networks more effectively.

Net Neutrality in the United States

NN became a topic of mainstream conversation in 2017, following FCC Chairman Ajit Pai's proposal to repeal the active NN policies. At the core of the issue was the classification of ISPs, and the FCC's ability to regulate them. ISPs could be classified as either Title I "information services", or Title II "common carrier services." If ISPs were classified as Title II, it would be in the FCC's power to impose significant regulations on them. However, as per the proposal, ISPs would be classified under Title I, thereby precluding FCC regulations.

The proposal sparked massive protests worldwide and over 20 million comments to the FCC's website, of which the overwhelming majority was against the proposal. The proposal was passed, despite the overwhelming public opposition [6]. The Federal Circuit Court of Appeals ruled in 2019 that the reclassification of ISPs was within the power of

the FCC [24]. However, the ruling also stated that the FCC does not have the power to block individual states' or local NN regulations.

2.2 Blockchain

Blockchain is a novel technology with applications in a wide number of fields. It is a decentralized method of storing cryptographically signed records that cannot be altered after their introduction into the blockchain [12]. As such, since its inception in 2008, blockchain has been used to record transactions, electronic votes, and with the Internet-of-Things (IoT), amongst many others.

For all participants in a blockchain network to agree on the state of the chain, a consensus mechanism is employed. There are numerous consensus mechanisms, but they all provide incentives for the creation of correct blocks and harsh consequences for altered blocks [28]. The two biggest blockchains, Bitcoin and Ethereum both use the Proof-of-Work (PoW) consensus mechanism. New blocks are appended by miners, who have to spend computational resources and time to correctly solve a puzzle. The miner who first solves the current block's puzzle gets to append the newest block to the chain, and reap the associated rewards. The solution to this puzzle is hard to find, but easy to verify. As such, incorrect blocks can be weeded out quickly, and correct blocks can be accepted instantaneously. A drawback of PoW is the unnecessary waste of resources: miners have to spend computational resources and time on the puzzle, and all but one miner's efforts go to waste.

Each block includes a number of transactions, chosen by the successful miner. For a transaction to be included in the next block, the parties of the transaction can tag on a tip value. This amount will be transferred to the miner who includes their transaction in a block. As such, transactions with larger tips are prioritized by miners. Usually, the more time-sensitive or important a transaction is, the larger this tip. This tip is referred to as the blockchain fee.

A popular alternative is the proof-of-stake consensus mechanism. Validators, who take the place of miners, deposit some amount of the chain's cryptocurrency into an account; this amount is called the stake. Once deposited, these coins cannot be touched again by the validator. For every block, a validator is chosen from the pool of users who have staked some coins. The probability of getting chosen to be the next block's validator is proportional to the amount staked. If a validator appends an incorrect block, their stake is slashed, and they are disqualified from validating any further blocks. Validators receive the transaction fees, the same as the miners receive in PoW.

The most popular blockchain for building decentralized applications is Ethereum. It is a blockchain-based open-source platform that supports applications through smart contracts. Ethereum uses Ether as a digital currency for payments and transactions on the Ethereum blockchain.

Once a transaction, smart contract, or data is stored on the blockchain, it becomes an immutable and integral part of the chain. As such, the cost of storing, and thus immor-

talizing data on the chain reflects this longevity. Despite being a secure and incorruptible method of storing data, it is expensive to store large amounts of data on blockchains.

One practical way to get around such limitations, while achieving similar levels of security and longevity, is through the use of hashes. Large amounts of data can be stored off-chain, while the hash of it can be stored on the blockchain. If the data is altered or corrupted, the two hashes will not match, warning of the discrepancy.

2.2.1 Blockchain-based Smart Contracts

Some blockchains allow for the storage and execution of programs on the chain: smart contracts. These programs, once deployed to the blockchain, are also immutable, and live for as long as the blockchain does. If any bugs exist in the smart contract at the time of deployment, it cannot be changed, and a new contract needs to be deployed to fix the mistakes [27].

Smart contracts have their own addresses, and are similar to user accounts. As such, smart contracts are able to receive, store, and distribute cryptocurrencies. One key aspect is that smart contracts are unable to take actions on their own. Thus, they require user input and events to perform operations. Every operation that is completed by a smart contract requires computational power. The amount of computational effort that is required to complete operations can be measured in a unit. For example, in Ethereum this unit is measured in *gas*. As such, operations by smart contracts require gas, which is an amount of the appropriate cryptocurrency. The amount of the currency needed depends on the number of operations. When a function is called on a smart contract by a user, the appropriate amount of gas must be sent along with the request. Otherwise, the interaction cannot be processed.

The copies of the blockchain that exist on every client's computing device is, eventually, the same. As such, all smart contracts are visible to the public, as are its contents and bugs. If a bug is noticed before the developers have had time to fix the issue, it may be possible for anyone exploit it. Because smart contracts are able to store cryptocurrencies, it is entirely possible to lose money due to a faulty implementation of a smart contract.

Ethereum supports smart contracts written in Solidity, an object-oriented programming language specifically invented to implement smart contracts on blockchains. Functions on Solidity smart contracts are able to receive funds from accounts simply through the use of the *payable* keyword. If a contract is not designed to receive funds, and does not have a function with this keyword, it will throw an exception upon receiving funds.

Chapter 3

Related Work

Many strategies and tools have been proposed to detect TD and NN. Generally, most retrieve measurements from two or more endpoints, gathering data from a variety of tests, and apply statistical analysis to determine whether a significant difference exists between samples [21]. In this thesis, the role of an external observer who does not have access to the configuration of the ISPs' routers is assumed as the standard. As such, the strategy required needs to determine whether different types of traffic are experiencing different treatment, based only on externally measurable performance metrics. In this section, selected solutions are presented that have been designed and implemented to detect types of TD, by using different techniques.

3.1 Glasnost

Glasnost is a tool that allows Internet end-users to check if their ISPs are applying TD in order to provide more transparency to costumers regarding their ISPs' traffic shaping policies [5]. It was designed as an easy-to-use tool that can be accessed via Web and requires no technical knowledge. Due to its simplicity, it has already been used by thousands of Internet end-users in different parts of the world. Its initial purpose was detecting TD on BitTorrent traffic, but it can also be used to detect differentiation on any traffic of any application.

Figure 3.1 illustrates the architecture of Glasnost. Initially, the end user accesses the Glasnost webpage (1). The user downloads the client application, which is a Java applet, that is executed at the end-user's Web browser and is redirected to a measurement server. A user can be redirected to one of several different measurement servers (2). Because multiple measurement servers are employed, ISPs are unable to prevent the use of the tool through blocking a server. The client connects to the measurement server and then starts a series of tests.

Glasnost sends two flows in sequence between a client and a measurement server, as depicted in Figure 3.2. The first flow, the *baseline flow*, corresponds to the target application, and the second flow, the *control traffic*, generated, is then compared against the

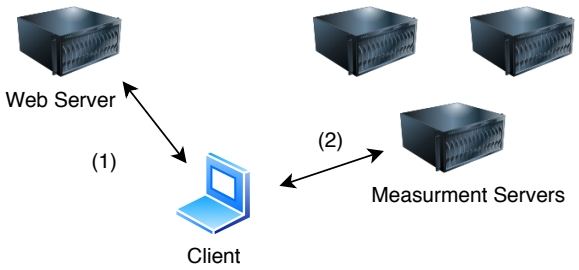

Figure 3.1: Glasnost Architecture

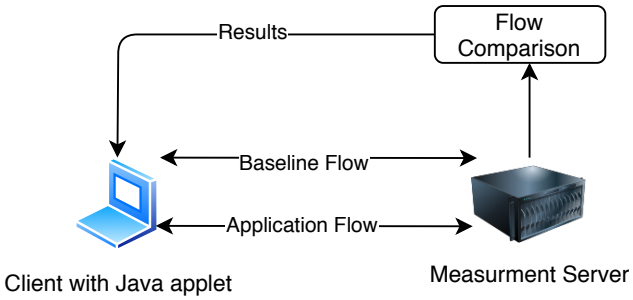

Figure 3.2: Glasnost Metrics Flow

first flow. The *application traffic* consists of messages of the real application. Glasnost works under the assumption that an ISP identifies applications based on destination port or application protocol. The baseline flow is identical to the application flow in terms of the number of messages and message sizes. However, the payload is generated randomly. The measurement server computes the throughput for each flow. The tests are repeated multiple times to reduce the noise in the obtained measurements. At the end, the measurement server processes the obtained data.

Glasnost uses the minimum, maximum, and median of the measured throughput metrics. The maximum throughput observed for each flow are then compared to detect if flows were treated differently. Glasnost assumes if the difference of the flow is larger than a threshold, then TD occurred. In the paper, it is claimed that this threshold represents a trade-off between the ability of the system to detect TD and the generation of false positives. If the threshold is set to a low percentage, *e.g.,* 10%, Glasnost might misclassify the measurement and indicate TD when in fact the difference results for the flows might have been caused by other factors, such as cross-traffic. In contrast, if set to a higher percentage (*e.g.,* 50%), it will only detect TD if the maximum throughput achieved by one of the flows was half the maximum throughput of the other, possibly leading to false-negatives. In the work, it is claimed that 20% is an optimal value. However, it is still important to note that there can still exist false negatives or false positives as the perfect threshold cannot be calculated given the amount of factors.

3.2 VPN-based TD

A solution for detecting TD in mobile networks with a VPN based solution is presented in [9]. The key idea is to use a VPN proxy to record and replay the network traffic generated by arbitrary applications and compare it with the network behavior when replaying this traffic outside of an encrypted tunnel. In order to do so, the authors use a trace record-replay methodology to reliably detect TD for arbitrary applications in the mobile environment.

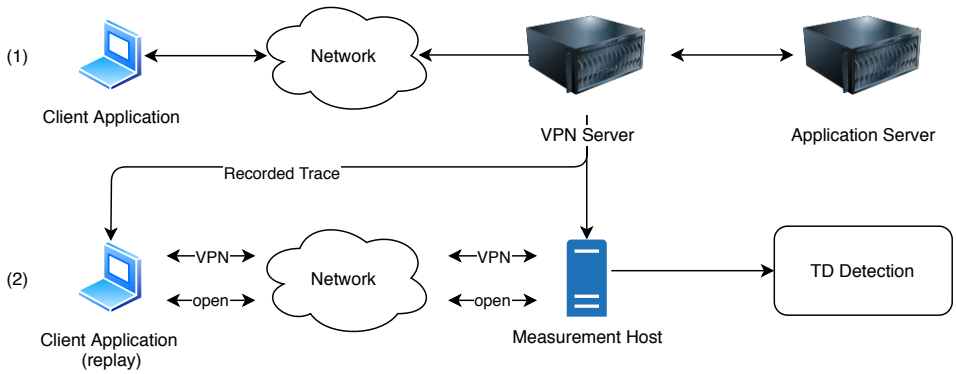

Figure 3.3: VPNbased

Figure 3.3 provides an overview of the methodology they used. The first step is to trace a packet from a target application, extract bidirectional application-layer payloads, and then use this to generate a transcript of messages for the client and server to replay. To test for differentiation, the client and server coordinate to replay these transcripts (Figure 3.3 (2)), both in plaintext and through an encrypted channel by using a VPN tunnel. In a VPN tunnel, payloads are hidden from any shapers on the path. VPN overheads, however, can stem from IPSec encapsulation and latency added by going through the VPN (e.g. if the VPN induces a circuitous route to the replay server), but the authors showed that they were able to minimize the latency and that the overhead represents a reasonable lower bound on the amount of differentiation it can detect. Finally, TD detection is performed based on the collected measurements. The solution employs a statistical test based on KS in order to compare the different distributions and infer the presence of TD. The solution was first evaluated on a local testbed, using two commercial traffic shapers and also evaluated in the wild, through a mobile application made available to the public. On the local testbed, the solution presented good accuracy and showed that three mobile networks in the USA employed TD for services such as Netflix, Youtube and Spotify.

However, there are some potential limitations to the proposed VPN based solution. Detecting TD only when the actual rate is lower than the sending rate of the application may lead to false negatives, especially considering that TD may only take place under congestion, which is not induced by the solution. Furthermore, the solution was designed and validated assuming that TD is implemented by ISPs using traffic shaping middleboxes, which may also lead to false-negatives, since there is several ways to implement TD [25]. Furthermore, cross-traffic may impact both recording and replaying, and thus should be taken into account. The detection may be also hindered if VPN traffic is discriminated by the ISP.

3.3 Gnutella Rogue Super Peer

The destination port of a packet is visible in its header, and as such, ISPs can discriminate packets based on this property. One proposed method to measure such breaches is the Gnutella Rogue Super Peer [21]. This strategy makes use of Gnutella, a peer-to-peer network, and iterates through every port in the range (0 to 2^{16} - 1). If connections cease only while checking a specific port, the most likely culprit is port blocking. Because applications use certain ports by default, ISPs can, through blocking certain ports, block applications from operating.

To set up a Gnutella Rogue Super Peer, one needs to join the Gnutella network with an altered super peer client. The client introduces itself to the Gnutella network as any other super peer would, but reroutes all of its incoming requests to a second, measurement, host. The super peer increments the destination port on the measurement host regularly, such that multiple peers have ample time to connect to it, if they can. If at least one peer successfully connects to a given port on the measurement host, then that port is not blocked by ISPs. If the connection to the measurement host is consistently unsuccessful for a given port, either none of the peers during that timeframe decided to take the rogue super peer's recommendation and connect to the measurement host, or they were blocked from doing so. Empirically, it was determined that peers would ignore a super peer's recommendation 80% of the time, so to achieve a high degree of confidence, at least 50 peers need to be rerouted for any given port.

This measurement strategy detects a very specific case of TD: port blocking. However, it can not determine if the port blocking is being performed by the peer's ISP, or the measurement host's ISP. Furthermore, an ISP may be blocking all Gnutella traffic, based on the application itself, and not the specific ports.

3.4 NANO

A strategy to detect TD through passively obtained performance metrics is NANO. Whereas other strategies detect specific types of TD, such as port blocking or content-based TD, NANO takes a more general approach, and looks at application-based TD [21].

NANO is a generic approach, which applies statistical analysis to passively gathered information [10]. Because of its passive nature, ISPs have a hard time detecting that the test is being run. NANO takes no assumptions in baseline, so it requires other measurements to be effective. However, TD is not the only factor that can cause differing performance measurements of applications. Many factors, such as geographic location, distance to the server, time of day, number of other users on the network, etc., can influence the speed and quality of the traffic at different endpoints. NANO attempts takes these factors into account while performing the statistical analysis on the gathered data. These factors can be categorized into three types of confounding factors that need to be accounted for: (i) client-related confounds, (ii) network-related confounds, and (iii) time-based confounds. The more factors that are accounted for, the more precise its measurement is. Taking these

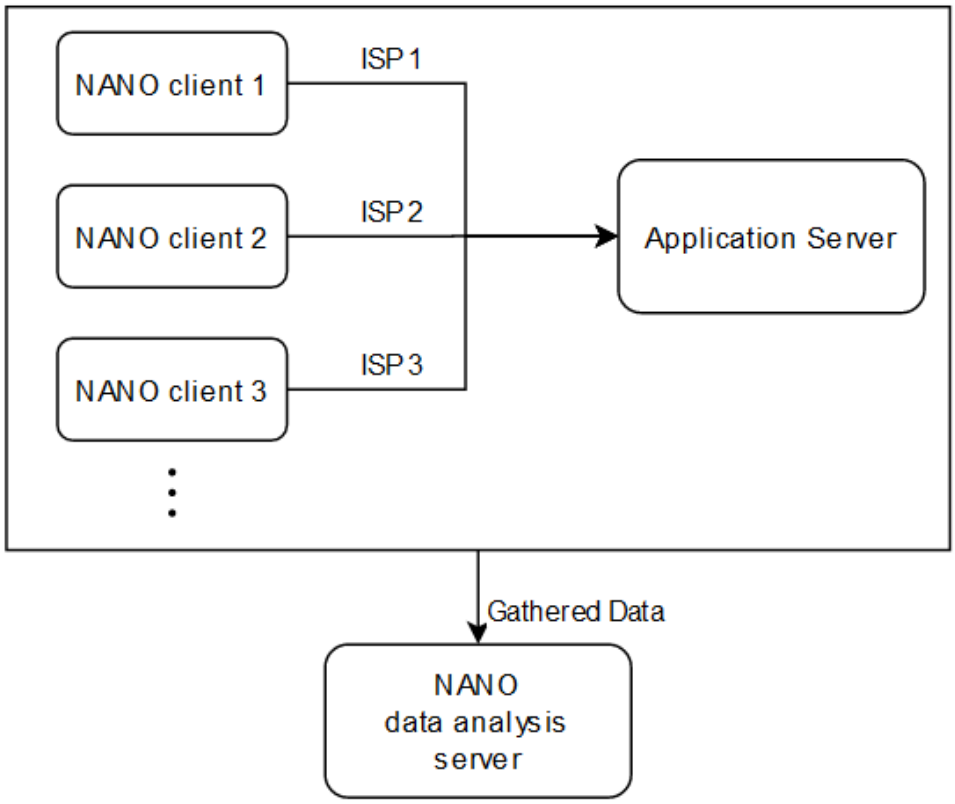

Figure 3.4: NANO

confounds into account, NANO calculates the statistical significance of the differences in a new measurement, as compared to previous measurements.

NANO measures application-based TD, in that, for any given application, multiple measurements need to be taken to attain an adequately accurate comparison basis. As shown in Figure 3.4, several NANO clients passively gather the data from an interaction with an application server. For a simple application that only has one server, we can expect response times to increase as the test is geographically farther from the server. However, for many popular applications, their servers are not located in a single place, but have multiple all over the world – further complicating the analysis. If the differences in the measurements between the clients are large than the expected difference taking into account the confounding factors, then NANO reports TD.

Because this method is a statistical approach, false positives and false negatives may often arise, depending on the amount of confounding factors measured or the amount of comparison measurements attained. NANO uses real traffic, and the more confounding factors it takes into account and the more measurements it has, the more accurate its TD sensing becomes.

3.5 POPI

Another TD Tool is POPI [7], which is based on end-to-end measurements to detect whether non-neutral schedulers are being employed by an ISP. A non-neutral schedulers may be employed to discriminate between different types of traffic classified a slow priority by dropping or delaying. POPI detects whether packets of different types are being forwarded with different priorities. POPI assumes that if only neutral schedulers are employed, then all packets are forwarded according to the arrival order. However, if a non-neutral scheduler is being used, the loss rate will be different for different types of traffic. POPI measures the loss rate for different types of traffic to infer whether different priorities were assigned for the types of traffic measured. POPI works in three steps.

In a first step, measurements are obtained after a series of packet bursts are injected. In the second step the measurements are used to order the types of traffic with larger loss rates for each of the bursts. In the final step, a statistical analysis is then made to detect the prioritization of specific types of traffic. POPI was evaluated in PlanetLab in order to find possible real cases of prioritization. In these tests 162 nodes of the testbed were employed, spread all around the world. POPI was executed on all pairs of nodes and in both directions for each pair. The final results showed that it was effective in large amounts of background traffic and it detected traffic prioritization for 15 node pairs.

3.5.1 Discussion

Different tools that exist to detect NN breaches were presented above. Even though the main goal of these tools is the same, the architecture and the methods vary quite substantially. The table below shortly summarizes what technique it uses, what type of TD it detects and the limitations of each of the presented solution are.

Four different NN breaches solutions were highlighted, but there exist several others that are worth mentioning. NetPolice is a tool for detecting TD in the backbone of the Internet. The solution is able to locate, which ISP is performing TD. The tool measures the loss rate experienced by different types of traffic, sent from multiple sources, as they traverse a target ISP. Through employing TTL-based probes it is able to discover paths traversed by packets in the network, including the internal path of the target ISP [29]. In addition, POPI a tool based on end-to-end measurements in order to detect whether non-neutral schedulers are being employed by ISPs [7]. In particular, POPI detects whether packets of different types are being forwarded with different priorities. Furthermore Packsen, a system designed to detect if an ISP is employing a traffic shaper to assign different priorities to different types of traffic [22]. Additionally, the solution also infers which scheduler is being employed and its properties.

Table 3.1: Comparison of Related Work

	Techniques	**Type of TD**	**Limitations**
NANO [10]	Passive measurements, Client-Server, Measurement Aggregation	Throttling, Longer Delays	False positives and false negatives due to several confounding factors
Glasnost [5]	Active measurements, Client-Server, Traffic Recording and Emulation, Relative Discrimination	Throttling	False positives and false negatives due low threshold or respectively high threshold
VPN based [9]	Active Measurements, Client-Server, Traffic Recording	Throttling, Jitter, Delays	VPN overhead
Gnutella RSP [18]	Active Measurments, Passive Measurements, Client-Server	Port Blocking	It cannot always detect whether a port blocking is being performed by the peer ISP or by the ISP of the measurement host.
POPI [7]	Active Measurements, Path Saturation, Client-Server, Traffic Emulation, Traffic Shuffling	Throttling	False positives due to other factors causing throttling of traffic

Chapter 4

Signaling NN Breaches using Blockchain

This chapter presents the design and architecture of the application solution to detect NN breaches. Furthermore, it details the implementation of the solution, describing the technologies employed and presenting the algorithms.

4.1 Design

The solution's architecture is divided into two parts. The first part is an application which compiles multiple metrics and measurements to detect any NN breaches in its communication. The second part is a blockchain and client application, which serves to store, retrieve, and compare these measurements. Together, these two parts form a versatile and resilient architecture.

4.1.1 Architecture

As shown in the previous chapter, there exists different methods and metrics to detect NN breaches. On the architectural part of the system two main methods exist on how breaches can be identified. On one side, a Client - Server architecture, such as Glasnost [5], where the client and server are both a part of the tool and data is exchanged between these two in order to detect NN breaches. On the other side a system in which a client usually targets a set of specific or random application servers, which are not part of the tool. To detect breaches measurement results between different clients with the same target applications are then compared. The first solution will be referred to as a Client - Server architecture and the second solution as a Peer-to-Peer (P2P) architecture.

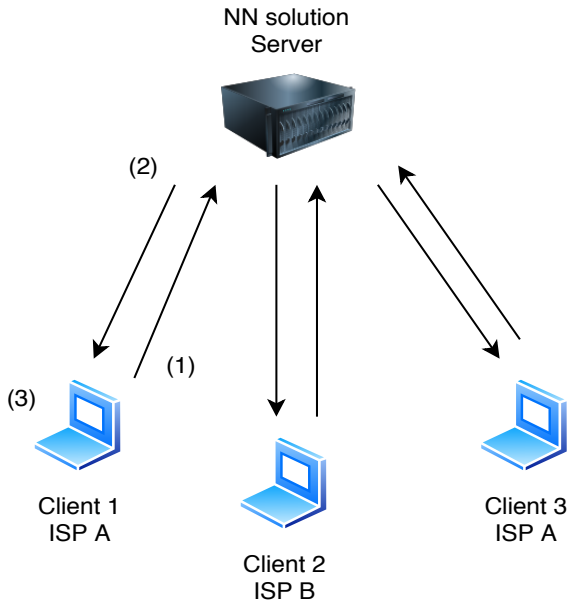

Figure 4.1: Client - Server Architecture

Client-Server Architecture

As shown in Figure 4.1, in an client-server architecture, a client creates a connection to the server to request media and data (1). The server then proceeds to deliver the requested media and in the meanwhile both client and server start measurements. These measurements then are collected by the client and are then evaluated (3). The advantage of such an architecture is that the server and the client are coordinated and the data transfers are clearly defined, so measurements are generally more accurate and precise. Furthermore, comparisons between different clients can be easily performed, since they are based on the same exact tests, and chronicled the same exact metrics.

However, the disadvantages are that the different measurements are limited by the tests that are implemented on the server. This means it is not possible to test NN breaches by consuming data from other application servers, only from servers outfitted with this exact same application. This method therefore assumes that any TD observations also occur with other application server, if the connection and interaction with the application server is the same as with the solution server. Another risk is, that an ISP might identify such an NN breach detection server and whitelist the IP address of the server from any TD methods. In such a scenario the measurements would mislead the observer in believing that the ISP is not applying any TD.

Peer-to-Peer (P2P)

Another alternative for this architecture would be a P2P system, in which multiple different clients are required to probe for NN breaches. As shown in Figure 4.2 the clients in this architecture consume data from one or more different application servers (1). During the consumption the client collects measurements and saves those (2). Clients can then

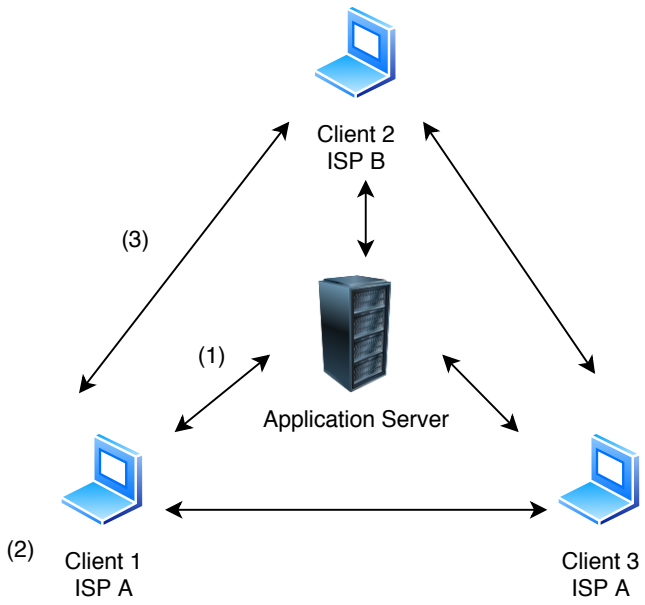

Figure 4.2: P2P Solution Architecture

connect to each other and exchange their results (3). These results are then analyzed and compared to other clients' results and potential NN breaches are then reported. The advantage of such an architecture is that it allows for broader test cases as the measurements are done passively on the client. In addition, the ability to compare measurements from a large amount of clients, allows for broader analysis. Furthermore, for an ISP it is difficult to hide TD from a tool using such an architecture, since the application servers are either randomly chosen or the generated traffic with an application server is indistinguishable, as it represents everyday internet traffic.

The disadvantage, however, is that the measurements are mostly heterogeneous since the client is not targeting specific application server with its probes. Therefore it is quite hard to compare results between other peers. As such, it is generally advised to define specific application servers and specific traffic flows, but also in this case, the tool is dependant on the behavior of an external application server.

Blockchain Component Requirements

The blockchain component of the architecture requires a smart contract and a client application which can interact with it. The smart contract needs to be able to store the results of the clients' tests, and to filter and serve them to the client application. Furthermore, it needs to associate the measurements with other identifying factors, the test descriptors, such as the ISP of the measurement client, the distance of the client to the server, or the measurement device's ping to the server. The smart contract must be able to filter its entries based on any of these factors, so that any new measurement clients can retrieve relevant data quickly from the blockchain records. The storage and retrieval of these records must be relatively cheap, so as to incentivize measurement taking.

The smart contract must also provide a system for requesting tests. These requests are

named bounties. These bounties must have a criteria that the satisfying test must meet. These criteria will be able to constrain the test descriptors' fields, such as ISP, distance to server, or ping. The bounties must further provide a monetary incentive for whoever completes it. Users may submit their tests and claim any outstanding bounty that they qualify for.

Furthermore, the blockchain client must require a user account to function, and every measurement that is stored must be affiliated with the user account. Furthermore, it must be able to conduct a statistical test to compare any relevant previous measurements with its own, and describe the type and severity of any discrepancies.

4.1.2 Proposed Solution

For the proposed solution the Client/Server (CS) approach was chosen, as it provides the most robust measurement data in order to detect NN breaches by ISPs. In the proposed solution, the role of the P2P system where client measurements are compared to different clients' is not done directly between the clients, but through a blockchain.

4.1.3 Protocols

The proposed solution works under the assumption that an ISP identifies applications based on the destination port or application protocol. In order to test for different payloads two flows are compared, a baseline flow and a application flow. The baseline flow is identical to the application flow in terms of the number of messages and message sizes, however the payload itself is different. This assumption is also used by Glasnost. As a result three different protocols were chosen to be tested in the tools. The selected protocols were HTTP, FTP and RTMP. HTTP and FTP were chosen, as they are the two most common used protocols to transfer data, whereas RTMP has been chosen to simulate streaming video files.

HTTP/HTTPS

The Hypertext Transfer Protocol (HTTP) is an application layer protocol for collaborative hypermedia, distributed systems. It is a stateless and generic protocol that can be used in many tasks that go beyond its use for hypertext, such as name servers and distributed object management services Request methods, error codes and headers [19]. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred. HTTP has been in use since 1990 and is one of the most used application layer protocol on the world-wide-web to transport text, multimedia, graphics and in particular HTML.

Hypertext Transfer Protocol Secure (HTTPS) is an Internet communication protocol extension for HTTP that protects the integrity and confidentiality of data between the user's computer and a server. Data sent over HTTPS is secured using the Transport Layer Security Protocol (TLS), which provides three important layers of protection [8]:

- Encryption of the exchanged data to protect it from eavesdropping. This means that while the user is surfing a website, no one can "eavesdrop" on their conversations, track their activities across multiple pages or steal their information.
- Data integrity, data cannot be altered or falsified, either intentionally or unintentionally, during transmission without being detected.
- Authentication proves that your users are communicating with the intended website. It protects against man-in-the-middle attacks.

FTP

FTP stands for File Transfer Protocol and is a standard network protocol used for the transfer of sensitive files between a client and a server on a computer network. FTP itself stands for File Transfer Protocol. It can be used to exchange and manipulate files over a TCP/IP-based network, such as the Internet. FTP is built on a client-server architecture and establishes two separate TCP connections. First a control connection (command port, port 21) to authenticate the user and second a data connection (data port, port 20) to transfer the files. Moreover, FTP requires an authenticated username and password for access With SFTP, data is transferred securely and encrypted, so no file data is transferred as plain text.

RTMP

The RTMP specification is a streaming protocol initially designed for the transmission of audio, video, and other data between a dedicated streaming server and the Adobe Flash Player. Macromedia, today Adobe Systems, developed the RTMP specification for high-performance transmission of audio and video data. RTMP maintains a constant connection between the player client and server, allowing the protocol to act as a pipe and rapidly move video data through to the viewer. While once proprietary, RTMP is now an open specification.

4.2 Implementation

The implementation of the proposed solution features a single client application that will serve as the client for both components of the system's architecture – it will communicate with both the server, as well as with the smart contract as shown in Figure 4.3. A standalone application was chosen, because it can easily receive sufficient permissions to access network traffic. A webpage, which would require the use of the Javascript implementation of *Web3*, would be more constrained by the operating system's permissions. As such, a Python standalone application to act as a client for both parts of the system's architecture was chosen.

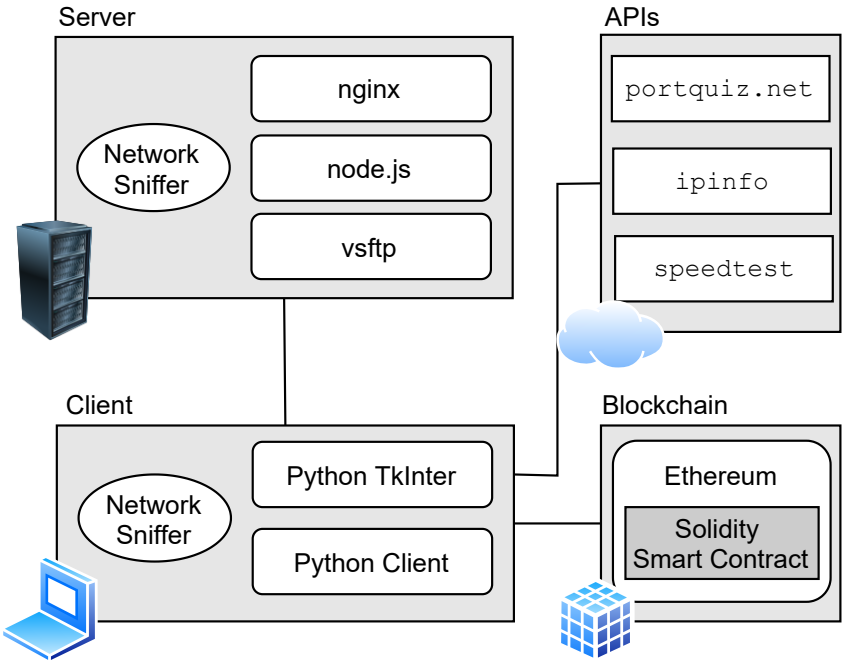

Figure 4.3: Implementation Overview

Lastly, a server is required from which the solution can query data and media for the program's tests and that can be outfitted with the proposed measurement system. The best solution in order to be able to cover all three different protocols was to run three different server applications on a Virtual Machine (VM). In this case for HTTP to run it by *node.js*. *Node* offers many convenient Javascript packages to simplify the server creation process, so the server solution heavily relies on the use of an Express server. Express is an extremely popular framework for HTTP servers. For FTP, vsftpd was chosen, which is the standard FTP linux server. And lastly to run a RTMP server, nginx was selected.

In the implementation, communication between the client and the server is mainly through the use of *get requests*. These requests are used by the client to start the measurement process on the server, retrieve the collected data, amongst other functions. To test NN breaches relating to file types, the server can serve multiple types of files – HTML, video, and audio. To receive any of these files, a client simply needs to make a get request on the server's URL with the desired filetype's name as a parameter. Before any data requests are made, another get request must be made that will start the network sniffing process between the server and the requester client.

To sniff the network traffic, the implementation makes use of the *Scapy* Python library. Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks like scanning, trace-routing, probing, unit tests, attacks or network discovery [20]. Both the client and the server employ the same strategy and library for sniffing the network traffic. The client starts listening to network traffic just before the server sends the requested file, and the server starts listening as soon as the corresponding get request is called. The list of collected packets is saved in a file using python's pickle library, one on the server, and

one on the client. The client, through another get request, can receive the server's list of packets. With network traffic data from the client and also the server, the client can then proceed to calculate jitter and packet loss.

Furthermore in order to measure the bandwidth of a client, to get the ISP data and for testing The implementation uses Ethereum's Solidity-based smart contract environment. The smart contract is able to store a list of objects, each of which contain the results of a single measurement, complete with the associated metrics. Because Solidity offers no support for decimal values, all results are stored as integers.

To communicate with the smart contract, the *Web3* code library was selected. The library works to facilitate the communication, transfer of funds, calling of functions on the smart contract between a client and a blockchain. The library has implementations in Javascript and in Python.

4.2.1 Jitter

Jitter is caused by runtime fluctuation of the time intervals at which data packets arrive at a receiver. As a result of jitter, individual packets may arrive too late to be output in the respective application. Although the data packet was transmitted correctly, it must be discarded in this case. These runtime fluctuations can cause serious problems in data transmission. Time-critical applications are particularly sensitive to excessive jitter values. Other applications that are not time-critical, such as e-mail or file transfer, are generally not bothered by the runtime fluctuations that inevitably occur in the network. However, jitter might be the result of TD and therefore be a signal of a NN breach.

In order to calculate jitter the time intervals between the data packets on the receiving end, the client, and on the transmitting end, the server, are needed. The time interval between the data packets on the server side are needed in order to ensure, that any observation of time interval fluctuation on the client side were not caused by irregular transmissions on the server side. As such a data packet that is sent by the server to the client and was observed by the sniffer on the server is referred to as a server data packet, while the same packet that is received by the client and observed by the sniffer on the client is a client data packet.

To calculate the jitter, first the temporal intervals between the individual data packets on the server side as well as on the client side are totaled. In general, it is assumed that no jitter occurs if the total time intervals between all client data packets are equal to the total time intervals of the server data packets. As shown in Figure 4.4 if the Total Time S is equal to the Total Time C, then no Jitter occured. As such if the Total Time C is higher than Total Time S then the difference represents the jitter.

Listing 4.1 shows how the jitter calculation has been implemented. The arguments serverp and clientp are a list of data packetes. The server packets and client packets are identical in terms of payload, as they represent the same data being transferred from the server to the client. Each single data packet has a timestamp associated to it. The timestamp is in case of a server data packet the time the packet was transmitted and in case of a client

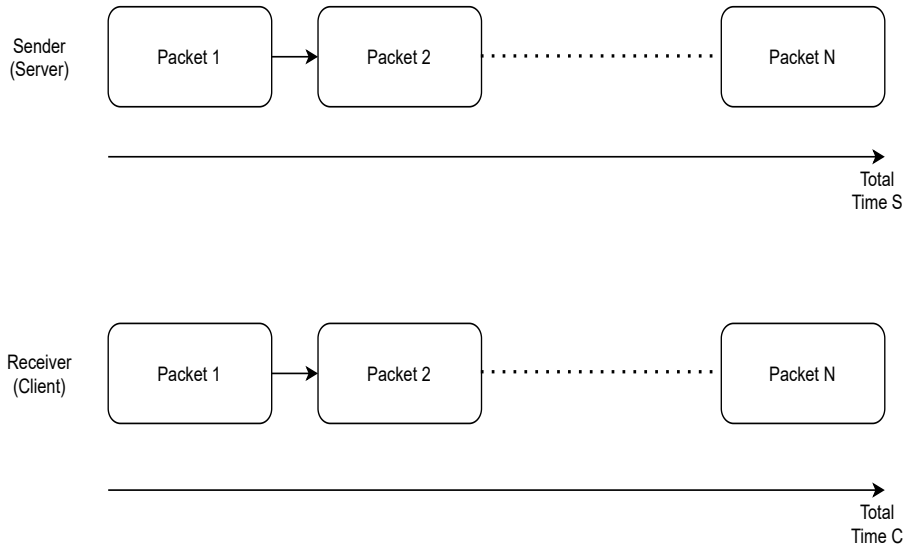

Figure 4.4: Jitter

data packet the time the packet was received. The total time for the server and client is calculated by calculating the difference in time between the last packet and first packet sent or respectively received. The difference between these two results these two results are

```
from scapy.all import *

def calculateJitter(serverp, clientp, latency, treshhold=0):

    lenserver = len(serverp)
    lenclient = len(clientp)
    times = serverp[lenserver-1].time - serverp[0].time
    timec = clientp[lenclient-1].time - clientp[1].time #the first packet is
        the response packet of the server sniffer, therefore not included in
        the calculation
    jitter = timec-times
    print(timec-times)
    return jitter
```

Listing 4.1: Jitter Implementation

4.2.2 Packet Loss

Packet loss is calculated using the retransmission property of TCP, since all three protocols HTTP, FTP and RTMP use TCP on their transport layer. Each byte of data sent in a TCP connection has an associated sequence number.

When the receiving socket, *i.e.,* the client, detects an incoming segment of data, it uses the acknowledgement number in the TCP header to indicate receipt. After sending a packet of data, the sender will start a retransmission timer of variable length. If it does not receive

an acknowledgment before the timer expires, the sender will assume the segment has been lost and will retransmit it. Figure 4.6 illustrates this workflow, where data 2 packet is lost between the client and the server, which therefore leads to a missing acknowledgment by the client and consequently to a retransmission. As packet loss can usually not be identified by the client it has to be done server-side. Iterating through all packets sent by the server and by comparing the sequence number of the packets, packets that were lost can be found at least twice.

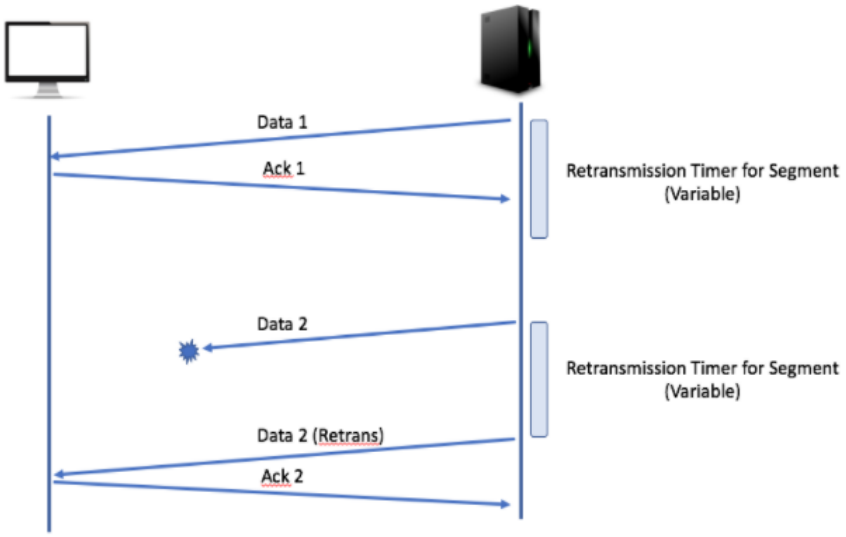

Figure 4.5: TCP Retransmission

4.2.3 Port Blocking

Port Blocking has been implemented with the help of portquiz.net, which itself is a server-application running on a Debian OS [11]. Through iptables rules all incoming requests from all available ports are routed to one server application. Portquiz limits new connections by IP, as the Virtual Machine (VM) running the application is not able to handle large amount of connections. Therefore it is not possible to test all 65535 ports that exist.

However, in the solution a base set of ports consisting of the most important and most used ports were chosen for inspection. These ports are: 20 (FTP), 21 (FTP), 22 (SSH), 23 (Telnet), 25 (SMTP), 53 (DNS), 80 (HTTP), 110 (POP3), 119 (NNTP), 123 (NTP), 143 (IMAP), 161 (SNMP), 194 (IRC), 443 (HTTPS). Inside the client application, a different set of ports can be specified before starting the measurement taking process. The result of the test is a list of working ports and a list of failed ports. The tool does not test incoming ports, since inbound firewalls on the client and on the router protect the network against incoming traffic from the Internet and other network segments by blocking ports in order to prohibit unknown connections, block malware or Denial of Service (DoS) attacks.

4.2.4 Latency

Measuring latency has been implemented by starting a timer when sending a request from the client to the server and stopping the timer as soon as a simple answer has been received by the server. This is repeated several times in order to account for any network noise. The average of those timer observation is used, while deleting any statistical outliers. Since generally the latency a pure number is not , due to the fact that it is dependent on the distance between the server and the client, we additionally measure the latency in relation to the distance to the server, which is currently located in Zurich.

Similar to latency, the calculation of the user's distance to server was also implemented. Using a third-party API of *ip-api.com*, the client application is able to geographically locate the user based on their IP address. Then, because the geographic location of the measurement server is known precisely, a simple mathematical formula can be applied to calculate the distance between the two. The distance to server is a useful measurement to have, because it contextualizes the value of the latency. When the distance to the server is low, one would expect the latency to be low as well. If the latency is high, but the distance to the server is low, then one can already form a suspicion that something is not right with the internet connection.

4.2.5 Throughput

Throughput tells how much data was transferred from a source, in our case the measurement server at any given time. Usually throughput is limited by the bandwidth and the performance of a server. Bandwidth tells you how much data could theoretically be transferred from a source at any given time. Bandwidth is generally limited by the contract a client signed with its ISP provider. So firstly in order to measure the bandwidth the external API of *speedtest* is used [16].

Secondly in order to calculate the throughput the tool divides the total amount of bytes it received by the total amount of time it took to receive them. The total amount of bytes is known by adding the size of all data packets received by the client. The total amount of time is measured by starting a timer when the request to the server is made and stopped when all data packets were received. Generally it is expected, that the throughput is lower than the bandwidth, since the throughput is limited by the bandwidth, the capacity of the server and the users current network usage.

4.2.6 Smart Contract

After collecting all measurements, the user needs to store and share them on the blockchain. To know the address of the smart contract, the user can retrieve the address to the newest smart contract from the server through a get request. Then, the user initiates contact with the contract. The relevant code to store measurements is presented in Listing 4.2, with an in-depth explanation.

```
struct MeasurementResult {
        int64 dist2server;
        int256 ping;
        string ISP;
        mapping(bytes32 => mapping(bytes32 => int256)) collectedMetrics;
}

mapping(address => MeasurementResult[]) measurementResults;
address[] addressesWithMeasurements;
uint256 numberOfMeasurements = 0;

function addNewTestDescriptors(int64 d2s, int256 png, string isp) public
    returns(uint256){
    if(measurementResults[msg.sender].length == 0){
        addressesWithMeasurements.push(msg.sender);
    }
    measurementResults[msg.sender].push(MeasurementResult({dist2server:d2s,
        ping:png, ISP: isp}));
    numberOfMeasurements++;
    return measurementResults[msg.sender].length;
}
```

Listing 4.2: Storing Measurements on the Blockchain

To associate every test measurement with the user's account, the results are stored in a mapping from the domain of account addresses, to an array of all their submitted measurements, in the *measurementResults* variable. If the user has not stored any measurements previously, the smart contract will initialize an empty array for them at this stage, and add the user's address to the list of addresses with measurements, so it can be iterated over later.

Structs are used as a convenient way to group related variables into one object. The *MeasurementResult* struct, for example, is used to group the test descriptors and all the metrics taken during that test into one object.

Because Solidity offers no way to iterate over the keys in a mapping, the addresses that have submitted at least one result are stored in another list. This allows all the measurements previously committed to be checked later for comparisons and bounties.

When initializing a new test result, the smart contract takes in the test's descriptors, the distance to server, the ping, and the ISP of the user. The one who calls the function on the smart contract can be accessed directly by the smart contract, so users are unable to add new measurements in the name of anyone else. The number of measurements is hereby incremented.

```
function addNewMeasurement(string mediaType,
                           string metricType,
                           int256 measurementValue)
                           public payable returns(uint256){
```

```
    require(measurementResults[msg.sender].length > 0);
    require(checkMeasurementTypes(mediaType, metricType));
    measurementResults[msg.sender]
        [measurementResults[msg.sender].length - 1]
        .collectedMetrics[keccak256(abi.encodePacked(mediaType))]
        [keccak256(abi.encodePacked(metricType))] = measurementValue;
    return 0;
}

bytes32[] acceptedMediaTypes = [keccak256(abi.encodePacked("video")),
    keccak256(abi.encodePacked("audio")), keccak256(abi.encodePacked("html"))];

bytes32[] acceptedMetricTypes = [keccak256(abi.encodePacked("avgLatency")),
    keccak256(abi.encodePacked("avgJitter"))];

function checkMeasurementTypes(string mediaType, string metricType)
internal view returns (bool){
    bool mediaTypeChecksOut = false;
    for(uint i = 0; i<acceptedMediaTypes.length; i++){
        if(acceptedMediaTypes[i] == keccak256(abi.encodePacked(mediaType))){
            mediaTypeChecksOut = true;
            break;
        }
    }
    if(!mediaTypeChecksOut) return false;

    bool metricTypeChecksOut = false;
    for(uint j = 0; j<acceptedMetricTypes.length; j++){
        if(acceptedMetricTypes[j] == keccak256(abi.encodePacked(metricType))){
            metricTypeChecksOut = true;
            break;
        }
    }
    return metricTypeChecksOut;
}
```

Listing 4.3: Adding Metrics to a Measurement

Then, one by one, the acquired metrics will be stored in this new array entry. The client will commit each, one by one, and the smart contract will store it if and only if the name of the metric is in its list of approved metrics. The metric must include what media type was being communicated while it was taken.

In Solidity, a variable can exist in either memory, or in storage. Storage is a more permanent location than memory, and as such, costs more gas to use. Measurement results, bounties, and the like are all stored in storage. To compare variables in storage and in memory is not allowed. However, a workaround is to take the hashes of the variables, and compare these to one another. The hashes of a variable in memory and of a variable in storage is allowed. To hash a variable, the *keccak256(abi.encodePacked())* functions are used.

Storing the measurements is one part of the solution. They must be stored in a way so that they can be retrieved easily, and according to certain parameters. Such an example is given below.

```
function filterMeasurementsByPing(int256 testPing)
public view returns (address[], uint256[]){
    address[] memory resAddressesFull = new address[](numberOfMeasurements);
    uint256[] memory resIndexesFull = new uint256[](numberOfMeasurements);
    int256 lowerBound = testPing - 10;
    int256 upperBound = testPing + 10;
    uint256 counter = 0;
    for (uint i=0; i<addressesWithMeasurements.length; i++) {
        for(uint j = 0;
            j<measurementResults[addressesWithMeasurements[i]].length; j++){
            if(lowerBound <=
                measurementResults[addressesWithMeasurements[i]][j].ping &&
                measurementResults[addressesWithMeasurements[i]][j].ping <=
                upperBound){
                resAddressesFull[counter] = addressesWithMeasurements[i];
                resIndexesFull[counter] = j;
                counter += 1;
            }
        }
    }
    address[] memory resAddresses = new address[](counter);
    uint256[] memory resIndexes = new uint256[](counter);
    for(uint k = 0; k<counter; k++){
        resAddresses[k] = resAddressesFull[k];
        resIndexes[k] = resIndexesFull[k];
    }
    return (resAddresses, resIndexes);
}
```

Listing 4.4: Retrieving Measurements

In the case of retrieving measurements similar to a target ping, the smart contract doesn't only look for measurements that match the target ping exactly, but in a range around it. It first creates two arrays in memory, to reference the measurements that fit the search criteria. When creating arrays in memory in Solidity, one must provide the length of the arrays when initializing them. These arrays use the *numberOfMeasurements* variable as an upper bound of how many measurements will fit the search criteria. Then, when all the measurement results are iterated over, new arrays are created in memory, this time with the number of matching measurement results as a length, and populated with the matches earlier.

The measurement results can be filtered by the test descriptors provided above: the ISP, ping, and the distance to the server. The distance filtering also works in a similar way, but uses a larger range around the distance, while the ISP filtering looks only for exact matches. All filters are structured similarly.

4.2.7 Bounty

```
struct Bounty {
    uint16 repeats;
    uint256 bountyValue;
    string bountyType;
    bytes32 bountyReq;
}

mapping(address => mapping(uint => Bounty)) bounties;
mapping(address => mapping(uint => AddressAndIndex[])) bountyMeasurements;
mapping(address => bool) addressHasBounties;
address[] addressesWithBounties;
mapping(address => uint[]) bountyAddressTimestamps;
uint256 numberOfBounties = 0;

function placeBounty( uint256 valuePerBounty,
                      uint16 numRepeats,
                      string bType,
                      bytes32 bReq)
                      external payable returns (bool){
    if(msg.value!=valuePerBounty*numRepeats){
        return false;
    }
    if(!addressHasBounties[msg.sender] ||
        bounties[msg.sender][block.timestamp].repeats==0){
        numberOfBounties += 1;
        addressHasBounties[msg.sender] = true;
        addressesWithBounties.push(msg.sender);
        bountyAddressTimestamps[msg.sender].push(block.timestamp);
    }
    bounties[msg.sender][block.timestamp] = Bounty({repeats:numRepeats,
        bountyType:bType, bountyReq:bReq, bountyValue:msg.value});
    return true;
}
```

Listing 4.5: Placing a New Bounty

Bounties are a system for incentivizing user to participate in the measurement taking process. Understandably, not everyone is willing to set aside a couple minutes to help by running the measurement client, so to make it worthwhile, users can place bounties with monetary incentives. To place a bounty, the client needs to contact the smart contract, and transfer the correct amount of funds to it. The bounty can have one criterion, which needs to be supplied in the shape of a key and a value. The key can be any of the following options: distance to server, ping, or ISP. The value is either an integer or string value representing the constraint of the bounty. Because the criteria would be too constraining if the bounty required a specific value for distance to server and ping, the bounty will instead look for measurements with a values in a range around the supplied criterion.

Once the client has stored all of the metrics it has collected, it can ask the smart contract to check if it qualified for any outstanding bounties. The contract only checks the most recent of the user's measurements.

The bounties are stored in a manner similar to the measurements, with one slight difference. The bounties will be removed from the active bounties list once they have been fulfilled, so the list that an address maps to will not only grow, but can also shrink. Therefore, referencing bounties by an address and by an index is insufficient. Therefore, each account address maps to a mapping from the domain of timestamps to bounties. In this way, bounties can be uniquely identified even if some are deleted along the way. Each bounty will be associated by the address of the one who placed it, as well as the time when it was placed. Using these two values as keys, we are able to uniquely refer to the bounties, even after bounties are deleted.

To create a bounty, one must supply the smart contract with the number of times the bounty can be completed and the amount to be paid. Only when the product of the two are equal to the amount paid to the smart contract will the bounty be placed. Bounties take in a parameter and a value. The parameter is responsible for what criteria must be met, such as the ISP, ping, or distance to server. The value is a byte array that can either store the name of the ISP, or the numerical distance. We use a byte array instead of a string or integer because of the uncertainty of the type that the parameter variable will hold. The name of the ISP would easily be represented as a string, whereas the distance to the server would be a numerical value. To be able to store either of these in the same variable, we convert the given parameter to a byte array. In this way, when a user wants to claim this bounty, their measurements will be converted to a byte array, and the correct property will be matched against the byte array of the required parameter. If the requirement is met, the user is awarded the bounty and the number of times the bounty can be fulfilled is decreased by one. When that counter hits 0, the bounty is removed from the active bounties list.

The creator of the bounty needs to be able to find all the measurements that have completed its bounty, at any point in the bounty's lifecycle. As such, whenever a measurement claims an outstanding bounty, its owner's address and its index are added to the list of measurements that have satisfied the bounty. This list of measurements can be accessed in the same manner as the bounty itself, with the creator's address, and the bounty's timestamp.

4.2.8 Graphical User Interface

The GUI has been implemented with tkinter, the standard Python interface to the Tk GUI toolkit. The GUI is a separate Python script from the scripts that deal with measurement and the scripts that interface with the blockchain. The reason for such an implementation is that this allows for the GUI to launch either script as a subprocess in the background. This allows the GUI to be responsive, even while a script runs in the background. Otherwise, if the GUI and the process which it runs were defined in the same script, the GUI would be unresponsive and freeze until the process finishes. Because our measurement process transfers large files, the duration of these processes would make users question

if the GUI was working, or just froze due to a bug. To circumvent this problem, even with the subprocess way of design, the GUI features a "Progress" module, which shows a percentage-wise representation of how far along the subprocess is. To give users more insight into what the application is actually doing, a "Log" module was added as well. This allows the subprocess scripts to output messages to the GUI window directly.

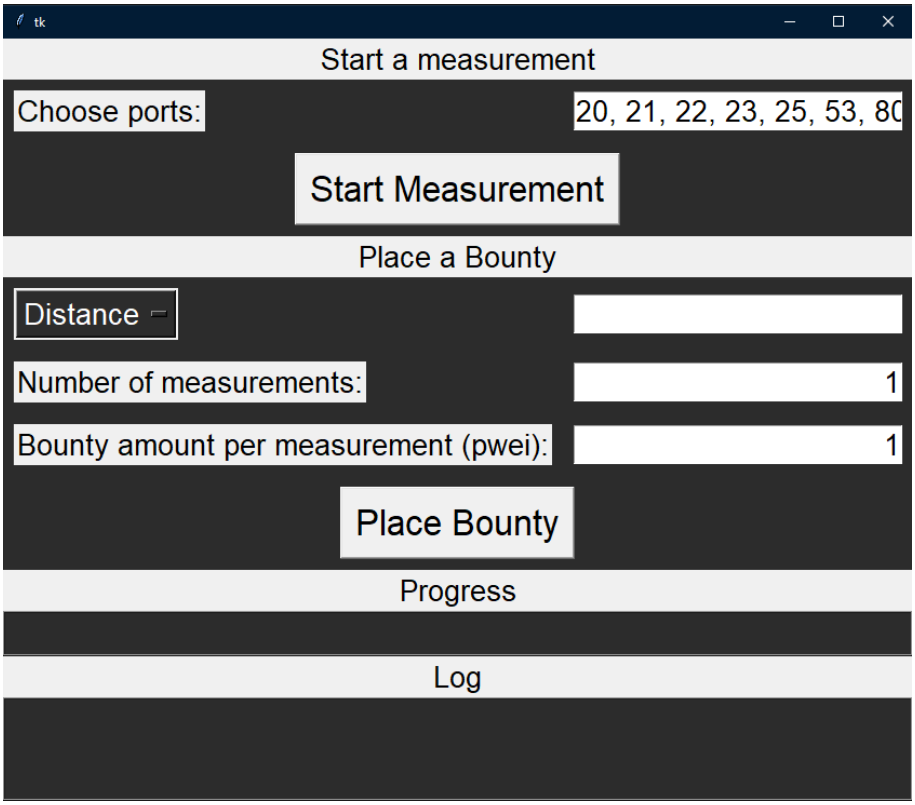

Figure 4.6: The GUI Window

The goal of the GUI is to allow the user to interact with the measurement tool and the smart contract in an easy-to-access way, and abstract away the complexities of the systems. As such, the design of the GUI is very simple; users need only press a single button to start the measurement process. The rest, taking the measurements, analyzing the data, storing the metrics on the blockchain, checking for and collecting any fulfilled bounties happens automatically. The only input a user can tinker with is the list of ports to check. For this, the GUI offers a simple input box where the users can input any number of port numbers to check. The input box is initially filled with common port values, so users can start the process without needing to configure even a single value.

To place a bounty is equally simple. Users only need to select the type of bounty from a dropdown menu, specify the target value for the constrained test descriptor, choose how many times the bounty can be repeated, and how much each bounty completion is worth. Once these fields are properly filled out, a single button abstracts away the rest of the process.

The GUI is built in an entirely modular way. This means that the different sections of the GUI do not interfere with one another, which, in turn, means that the GUI can be extended easily.

4.2.9 Challenges

Given the architecture and design of the tool the amount of test cases in regards to protocols and use cases is limited by the implemented selected test cases in regards to protocols, ports and duration of those tests. While this means that the measurements themselves are comprehensible and robust, it also results in restricted ability for the user to test a wide variety of scenarios. Generally further protocols or can be added relatively easily to the tool, however one must also take into consideration that the metrics and measurements taken and processed are hard-coded into the SC. This results in a relatively limited flexibility and any advancements might increase the complexity of the SC.

To be able to compare different measurements and to be able to place bounties on certain criteria, the test descriptors and collected metrics must be consistent among measurements. The only way to enforce this in the SC was to hard code the accepted variables. This choice, however, results in a rigid structure, where if a new test or metric is added to the measurement suite, a new smart contract needs to be deployed.

Chapter 5

Evaluation and Discussion

The last chapter provided an overview into the design and implementation used in the NN detection application. This chapter evaluates the implemented detection metrics presented in Section 4.2 based on accuracy by comparing results. This chapter is divided into two sections. The first section focuses on the client/server side of the application and the metrics defined and used. Section two focuses on the blockchain and its economical viability as a way to incentivize users to contribute to the blockchain. It is worth nothing that the goal of this section is not to identify ISP, that are using TD, but rather to evaluate the implemented tool in its capability to incentivize and detect NN breaches with a reasonable amount of users participating in such a system.

5.1 Metrics

In this section each metric that is implemented and measured by the tool is evaluated in regards to its accuracy and also its interpretation of NN breaches. Given the limitation of resources and time of this project, metrics are only evaluated based on single tests and not based on a large pool of users.

5.1.1 Port Blocking

After testing the tool with the base set of ports and also several custom sets of ports, the tool returned as expected all ports as open. In order to simulate a scenario, where certain ports are blocked, the windows firewall was configured to block certain ports. The tool successfully identified and returned all ports that were blocked by the firewall.

It is worth noting that a port being returned as blocked, while not being specifically configured in a firewall, is not necessarily a result of a NN breach. Generally, the cause of port blocking might be *(a)* the configuration of a firewall on the client, *(b)* the configuration of the clients router or *(c)* a policy be the ISP. A NN breach would only occur if the third case was the cause. Therefore, one must make sure, that the first two options can be ruled out before concluding a port being blocked to be the result of a NN breach.

5.1.2 Latency

Latency refers to the time it takes for information or a data packet to travel from its source to its destination. The server is currently located in Zurich, Switzerland in hosted in the University of Zurich (UZH). Two measurements from different geographical locations as showed in table 5.1, were done. One within Switzerland and one in Florida, USA. As expected, with higher distance from the client to the server the latency increases. But distance is not the only factor contributing to delays. Moreover, it is composed of various factors contributing to an increase in latency. Those are: *(a) Propagation delay*, the amount of time it requires a message to travel from the sender to receiver, which is the function of distance over speed with which the signal propagates, *(b) Processing delay* Amount of time required to process the packet header and determine the packets destination, and *(c) Queuing delay* Amount of time the packet is waiting in the queue until it can be processed.

In regards to a NN breach, *(c)* is the main factor of interest, since any delay caused by *(a)* and *(b)* would not signal a NN breach. Now, in order for a specific user to detect a NN breach based on the latency, it is required to perform following steps. Firstly, a measurement is started with the tool. Secondly, a bounty on the SC is placed based on distance to server. Now, following several measurements done from other users, an observable is made that latency seems to be significantly higher than other users with the same geographical location. Lastly the user needs to ensure that any other factors can be excluded that might be attributed to the increase of latency. If all other factors could be excluded, a NN breach might be signaled.

Table 5.1: Latency results

Location	**Geograph. Distance to Server**	**Latency**
Solothurn (CH)	79.7 km	9.8 ms
Florida (US)	7783.7 km	112.46 ms

5.1.3 Packet Loss and Throughput

The next metrics, packet loss and throughput are measured during a data flow between the server and client. Several measurements with different protocols, ports and files were made. Table 5.2 shows the number of packets, that were retransmitted and the total throughput that was measured during each of these measurements. The packet losses for all these measurements are in the low single digits. Considering that a single measurements consists of approximately 40'000 packets, the amount of packet losses seem to be low. Noticeable differences between each measurement is not to be made out.

The same also applies to the results for the throughput. Those results were for all measurements around 85 Mbps, where the potential bandwidth measured using the external API of speedtest was 99.6 Mbps. Besides TD techniques used by ISP, that would led to an increase of packet loss or is designed to throttle the throughput and be regarded as a NN breach, other factors such as insufficient signal strength at the destination , natural interference, excessive system noise, or overloaded network nodes can also be responsible

for packet either of these metrics. Therefore, it is necessary to ensure that these factors did not lead to a distortion within the measurements

Table 5.2: Results Packet Loss

Protocol, Port, File type	**Nr. of Packets lost**	**Throughput(Mbps)**
HTTP Port 80, HTML	5	85.4
HTTP Port 3005, HTML	8	84.3
HTTP Port 80, Video MP4	6	86.7
HTTP Port 3005, Video MP4	5	85.2
FTP Port 22, HTML	9	83.3
FTP Port 22, Video MP4	6	84.7

5.1.4 Jitter

As previously mentioned, jitter is the variance in latency. In order to to evaluate the jitter measurement taken by the tool, a transfer of data flow, in particular the transfer of data packets is evaluated in more detail. Since it was not viable to use the complete set of packets consisting of approximately 40'000 packets for a data stream, a subset of the first 100 packets was chosen in order to illustrate the variation of time intervals. As a note, for the complete set of packets the observations are very similar to those visible in the first 100 packets.

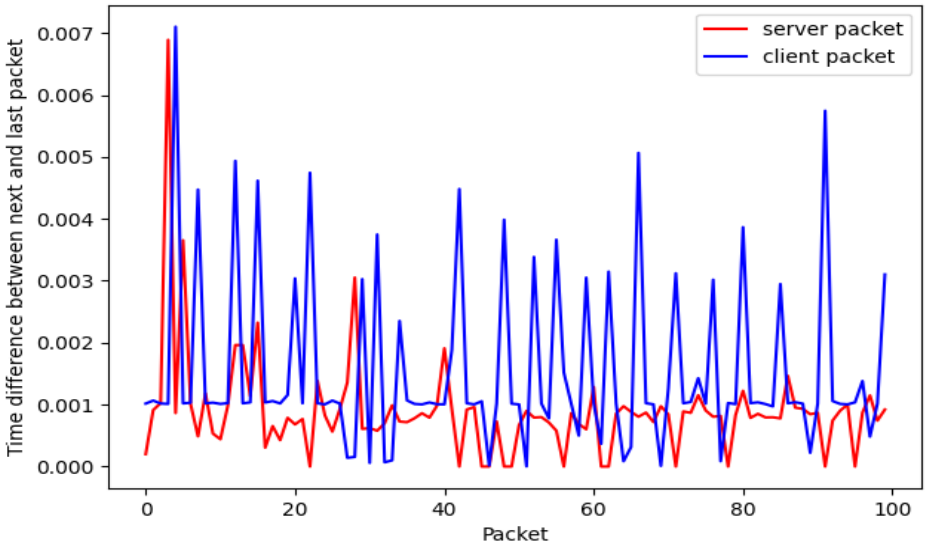

Figure 5.1: Time variation between packets for an HTML, HTTP Port 80

Figure 5.1 shows the time intervals for the first 100 packets transmitted. The blue line is the connected dots for the time interval between the packet previously sent to the next packet to be sent. The red line represents the packets on the client side. Since the client is the receiver of the packet, the red line represents the time interval between the last packet received. Looking at the figure, a general pattern is to be seen. While the time intervals on server side seem to be relative constant between $>$ 0.001 to 0.003 seconds, on

client side the fluctuations are higher with several spikes to 0.006 seconds. The same is observable in Figure 5.2, which shows the time intervals for packets transferred with the same protocol and port, but different in that case a video file. In general, a spike in the time intervals, which can only be observed on client side might be signs of jitter. In this scenario those spikes seem relatively low and might be attributed to other factor such as network congestion. Given a larger set of measurements from different client, it would be possible to detect any anomalies in regards, to the jitter measurements. But as it is also the case for the other metrics, it is necessary to exclude any other factors that may have led to the occurrence of high jitter.

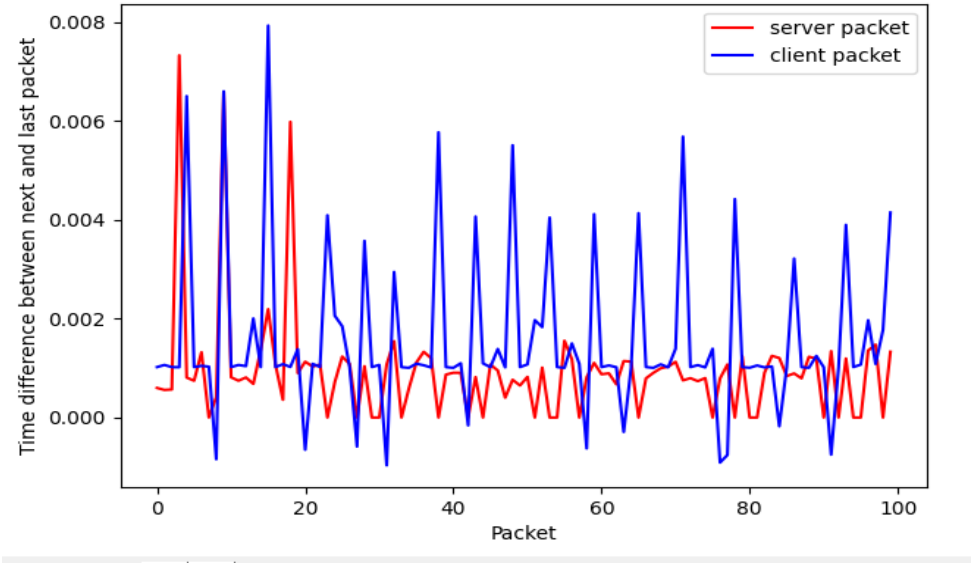

Figure 5.2: Time variation between packets for a MP4 Video file, HTTP Port 80

5.2 Blockchain

As mentioned earlier, every operation in a SC costs gas, which must be supplied by the user calling the function. The higher the cost of using the SC, the less likely users will. As such, it is important to analyze the gas usage of the implementation, and how it would scale.

The *Web3* library provides a function called *estimateGas*, which, given a state of a SC, estimates how much gas would be used to call a function with given parameters. Thus, this function is used to estimate the values for the following analysis.

5.2.1 Economical Aspects

The implementation separates the storing of measurement results into two distinct parts: storing the test descriptors, and storing the metrics. The former occurs only once per measurement, while the latter occurs once per measured metric.

```
function addNewTestDescriptors(int64 d2s, int256 png, string isp) public
    returns(uint256){
        if(measurementResults[msg.sender].length == 0){
            addressesWithMeasurements.push(msg.sender);
        }
        measurementResults[msg.sender].push(MeasurementResult({dist2server:d2s,
            ping:png, ISP: isp}));
        numberOfMeasurements++;
        return measurementResults[msg.sender].length;
}
```

Listing 5.1: Storing a New Test's Descriptors

As seen in Listing 5.1, no loops need to be iterated over to add new test descriptors, so the amount of gas depends solely on the information to be stored.

```
function addNewMeasurement(string mediaType,
                           string metricType,
                           int256 measurementValue)
                           public payable returns(uint256){
        require(measurementResults[msg.sender].length > 0);
        require(checkMeasurementTypes(mediaType, metricType));
        measurementResults[msg.sender]
        [measurementResults[msg.sender].length - 1]
        .collectedMetrics
        [keccak256(abi.encodePacked(mediaType))]
        [keccak256(abi.encodePacked(metricType))] = measurementValue;
        return 0;
}
```

Listing 5.2: Storing a New Test Metric

Similarly, when adding new metrics, only a constant number of loops need to be iterated over (when checking if the given media and metric types are valid). Therefore, the amount of gas required depends only on the data being stored. For generic testing data, the gas cost of adding new test descriptors is around 100,000, while the gas cost of adding metrics is around 50,000 each. While these might seem like large numbers, the going price for a unit of gas at the time of writing is 118 Gwei, so the costs of these transactions are 0.0118 and 0.0059 Ethers respectively.

```
function filterMeasurementsByPing(int256 testPing)
public view returns (address[], uint256[]){
    address[] memory resAddressesFull = new address[](numberOfMeasurements);
    uint256[] memory resIndexesFull = new uint256[](numberOfMeasurements);
    int256 lowerBound = testPing - 10;
    int256 upperBound = testPing + 10;
    uint256 counter = 0;
    for (uint i=0; i<addressesWithMeasurements.length; i++) {
```

```
        for(uint j = 0;
            j<measurementResults[addressesWithMeasurements[i]].length; j++){
            if(lowerBound <=
                measurementResults[addressesWithMeasurements[i]][j].ping &&
                measurementResults[addressesWithMeasurements[i]][j].ping <=
                upperBound){
                resAddressesFull[counter] = addressesWithMeasurements[i];
                resIndexesFull[counter] = j;
                counter += 1;
            }
        }
    }
    address[] memory resAddresses = new address[](counter);
    uint256[] memory resIndexes = new uint256[](counter);
    for(uint k = 0; k<counter; k++){
        resAddresses[k] = resAddressesFull[k];
        resIndexes[k] = resIndexesFull[k];
    }
    return (resAddresses, resIndexes);
}
```

Listing 5.3: Retrieving Measurements

The functions to query similar measurements are all implemented in an identical manner, so the filtering by ISP case will be discussed as an example. As seen in Listing 5.3, the function instantiates two arrays that linearly scale in size with the number of total measurements stored in the SC. Then, the function needs to iterate over every single measurement, then place them in the return value arrays. Because the measurement results are stored in mappings, iterating over every single measurement result cannot be averted when looking for potential matches. Furthermore, it is possible, albeit unlikely, that every single previous measurement will fit the search criteria, so the initial arrays need to accommodate such an eventuality when being initialized.

As such, the use of the SC would become more expensive, the more measurements it already stores. The cost of storing data can be regarded as a constant cost, but the cost of retrieving similar measurements grows linearly.

However, there is one very important caveat that will prevent these costs. In Solidity, a function can be marked with the *view* keyword. *View* functions can read the SC's storage, but cannot modify it. The filtering functions fit this criteria, and as such, have been marked with the keyword. *View* functions, when called externally, from outside the SC, cost no gas. Were they to be called from inside the SC, from a non-*view* function, they would still cost the full amount of gas, proportional to their number of operations.

Because the query costs can in this way be circumvented, the only parts of the SC's use that require spending gas is the addition of new data, and the creation of a new bounty. Both of these costs are constant, so the SC's gas use is constant, even at scale.

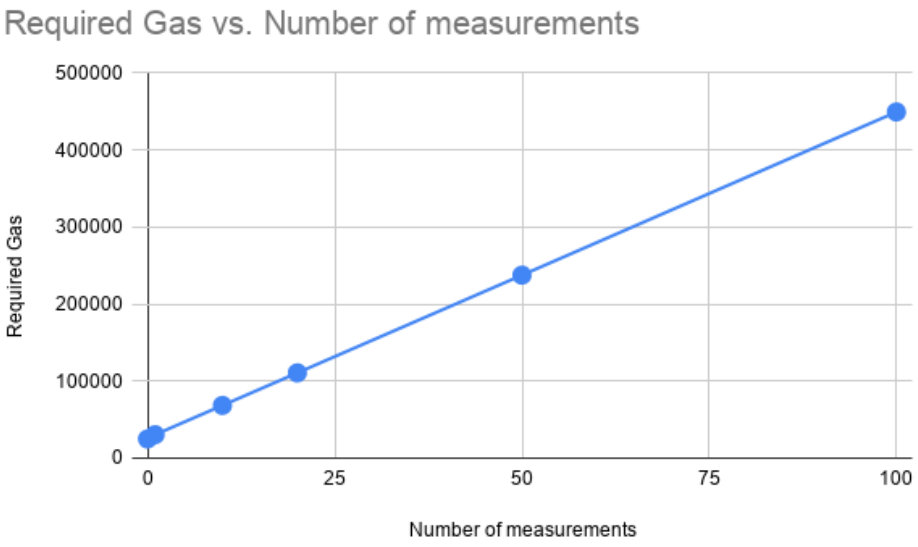

Figure 5.3: Amount of Required Gas of Filtering by ISP at Scale

5.3 Discussion and Feasibility

The evaluation of the implemented metrics has shown that generally, the metrics and measurements implemented are accurate and provide the expected results based on a test from two clients in different geographical locations and ISPs. The results of the measurements indicate, that higher jitter, high latency, increased packet loss or throttling the throughput based on destination port or application protocol would be detected and reported by the tool.

However, for all metrics, results that might indicate a NN breach, such as high latency based on geographical location, high jitter or a higher packet loss compared to results from clients with the same or different ISP, might not always be caused by TD techniques used by an ISP. As presented other factors, such as overloaded networks, firewall or weak signal strength might be a factor contributing for poorer measurements. Therefore, one must always take the results of the measurement as not the single truth and make sure that other factors, which cannot be influenced by the ISP, do not have any influence on the results. In this sense, it can be seen that detecting NN breaches is a complex process, as it depends on a myriad of devices and actors.

Furthermore, with the help of the SC the solution shows, that users are incentivized to use the tool, perform measurements or place bounties to detect NN breaches. While the storing of data and the creation of bounties requires spending gas, the retrieval of measurements is free. As of the time of writing, the current price (1,379.48 USD as of February 28th, 2021) and volatility of the cryptocurrency is of great importance here. According to the calculations above, the addition of a single test's descriptors would cost roughly $16 and ever added metric roughly $8 in gas. As such, if Ethereum were chosen as the blockchain, the transaction costs at the current price would probably be too high for meaningful use. A private blockchain can be seen as a feasible alternative.

Chapter 6

Summary and Future Work

Net neutrality is a much-debated topic all around the world, with many countries enacting policies to safeguard it. Those policies enacted, can differ substantially: while the USA are moving in the direction to repeal NN policies, the EU is adding new policies to protect NN and others such as Switzerland do not yet see the need to react in this topic. Throughout the years, different tools have been proposed and introduced to detect TD used by ISPs, with each using different approaches and techniques, as presented in this thesis. However, it is also true for all of these solutions that detecting NN breaches is a difficult and uncertain task, given the large amount of actors and factors influencing measurements, that may not have been the result of TD. Furthermore for any solution, it is necessary to give users an incentive to discover and report such breaches in order to accumulate a reliable baseline for reference.

To address this issue, this report presented the design and implementation of a blockchain-based NN detection tool. The implementation features a single client application that serves as the client for both components of the architecture. It communicates with both the server, as well as with the smart contract. The solution works under the assumption that an ISP identifies applications based on the destination port or application protocol. Metrics measured by the tool are latency based on geographical location, throughput, jitter and packet loss for a set of different protocols and content files.

For the implementation, Ethereum's Solidity-based smart contract environment was selected. The smart contract is able to store a list of objects, each of which contain the results of a single measurement process, complete with the associated metrics. Because Solidity offers no support for decimal values, all results are stored as integers.

The evaluation of the implemented solution showed, that the metrics used are useful to potentially signal TD techniques used, like the use of throttling, longer delays, blocking of outgoing ports or the increase of jitter. However, it was also shown that for each of these metrics and measurement, which could be indicative of TD, various other factors could also be seen as triggers, given the large amount of devices and actors involved. Furthermore, it showed that the retrieval of such measurements on the blockchain is free, while the storing or the creation of a bounty requires spending gas. Choosing Ethereum as the blockchain, the transaction costs of the tool at the current Ether price would probably

be too high for meaningful use. To avoid the high cost of such transactions, the use of a private blockchain could be a viable solution. In summary, the proposed solution can be a means to get users to participate in such a system and increase its use and acceptance, but a clear signal of a NN breach can only be identified to a limited extent with the help of this tool in its current state.

The possibilities for future work can be broken down into categories: extending the testing suite, extending the GUI, improving on the efficiency of the SC, and creating a comprehensive statistical evaluation to determine the likelihood that a new measurement observed TD. Further work in either of these categories would result in improved usability, and a more reliable tool. However, without a large and diverse list of measurements, the tool is not as effective. The improvements in usability brought about by improvements in the aforementioned categories will enable more users to contribute to the measurements, thereby increasing the reliability.

Bibliography

[1] Bundesamt fuer Kommunikation. Netzneutralitaet, 2014. https://www.bakom.admin.ch/bakom/de/home/das-bakom/medieninformationen/medienmitteilungen.msg-id-54918.html, Last visit February 4, 2021.

[2] S. Christopher and J. Lambert. 5G and net neutrality, 2019, Faculty Scholarship at Penn Law. 2089. https://scholarship.law.upenn.edu/cgi/viewcontent.cgi?article=3091&context=faculty_scholarship Last visit March 4, 2021.

[3] Cisco. Cisco Annual Internet Report (2018-2023) White Paper, 2018. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html, Last visit February 21.

[4] Peter R. Egli. Internet Organization, 2015. https://www.slideshare.net/PeterREgli/internet-organization, Last visit October 5, 2020.

[5] M. Dischinger et al. Glasnost: Enabling end users to detect traffic differentiation. *Proc. USENIX Conf. Netw. Syst. Design Implement.*, page 27, 2010.

[6] Federal Communications Commission. FCC ADOPTS STRONG, SUSTAINABLE RULES TO PROTECT THE OPEN INTERNET, February 2015. https://transition.fcc.gov/Daily_Releases/Daily_Business/2015/db0226/DOC-332260A1.pdf, Last visit September 28, 2020.

[7] S. Birrer F. E. Bustamante G. Lu, Y. Chen and X. Li. Popi: A userlevel tool for inferring router packet forwarding priority. *IEEE/ACM Trans. Netw.*, 18(1):1–13, February 2010.

[8] Google. Secure your site with HTTPS. https://developers.google.com/search/docs/advanced/security/https?hl=en&visit_id=637420211046229548-2794172180&rd=1, Last visit November 26, 2020.

[9] L. Anke K. Arash, R. Abbas, C. David G. Phillipa K. Hyungjoon, G. Rajesh, and M. Alan. Identifying traffic differentiation in mobile networks. 2015.

[10] M. Motiwala M. B. Tariq and N. Feamster. NANO: Network Access Neutrality Observatory. *IEEE Communications Surveys Tutorials*, pages 123–132, 2008.

[11] Marc Maurice. Outgoing Port Tester, 2021. http://portquiz.net/ Last visit February 28, 2021.

[12] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[13] Netflix. Empfehlungen zur Internetgeschwindigkeit . `https://help.netflix.com/de/node/306`, Last visit February 4, 2021.

[14] Network Working Group. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. `https://tools.ietf.org/html/rfc2474#:~:text=This\%20document\%20defines\%20the\%20IP,\%2Dhop\%20behaviors\%2C\%20is\%20defined.`, Last visit February 4, 2021.

[15] Netzpolitik AG. Ständerat lehnt Netzneutralitäts-Motion ab, March 2015. `https://netzpolitik.gruene.ch/staenderat-lehnt-netzneutralitaets-motion-ab/`, Last visit September 18, 2020.

[16] Ookla Speedtest. Speedtest, 2021. `https://www.speedtest.net/` Last visit February 22, 2021.

[17] G. Simon P. Maille and B. Tuffin. Toward a net neutrality debate that conforms to the 2010s. *IEEE Communications Magazine*, 54(3):94–99, 2016.

[18] S. Bauer R. Beverly and A. Berger. The internet is not a big truck: Toward quantifying network neutrality. *IEEE Communications Surveys Tutorials*, pages 135–144, 2007.

[19] J. Mogul H. Frystyk L. Masinter P. Leach R. Fielding, J. Gettys and T. Berners-Lee. Hypertext Transfer Protocol, HTTP/1.1, 1999. `https://www.hjp.at/doc/rfc/rfc2616.html`, Last visit February 17, 2021.

[20] Scapy. Scapy, Packet crafting for Python2 and Python3. `https://scapy.net/`, Last visit March 3, 2021.

[21] L. M. Peres L. C. E. Bona T. Garrett, L. E. Setenareski and E. P. Duarte. Monitoring network neutrality: A survey on traffic differentiation detection. *IEEE Communications Surveys Tutorials*, 20(3):2486–2517, March 2018.

[22] A. Soule U. Weinsberg and L. Massoulie. Inferring traffic shaping and policy parameters using end host measurements. *Proc. IEEE INFOCOM*, pages 151 – 155, April 2011.

[23] European Union. REGULATION (EU) 2015/2120 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL, 2015. `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32015R2120&from=DE`, Last visit October 5, 2020.

[24] United States Court of Appeals. Mozilla v. FCC, February 2019. `https://de.scribd.com/document/428285019/Mozilla-v-FCC-ruling?campaign=SkimbitLtd&ad_group=66960X1514734X678ed29db4193e9e55340ff0b3771559&keyword=660149026&source=hp_affiliate&medium=affiliate`, Last visit September 29, 2020.

[25] M. Omar V. Nguyen, D. Mohammed and P. Dean. Net neutrality around the globe: A survey. In *2020 3rd International Conference on Information and Computer Technologies (ICICT)*, pages 480–488, 2020.

[26] Marwan Van Nguyen. Net Neutrality around the Globe: A Survey. *IEEE Communications Surveys Tutorials*, 20(3):2486–2517, March 2018.

[27] Vitalik Buterin. Ethereum White-Paper, 2015. `https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf`, Last visit September 20, 2020.

[28] P. Hu Z. Xiong D. Niyato P. Wang W. Wang, D. T. Hoang, Y. Wen, and D. I. Kim. A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access*, 7:22328–22370, 2019.

[29] Z. M. Mao Y. Zhang and M. Zhang. Detecting traffic differentiation in backbone isps with netpolice. *IProc. ACM Internet Meas. Conf*, pages 103–115, 2009.

Abbreviations

AAA	Authentication, Authorization, and Accounting
BC	Blockchain
FCC	Federal Communications Commission (FCC)
FTP	File Transfer Protocol
ISP	Internet Service Provider
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
TD	Traffic Differentiation
NN	Net Neutrality
SC	Smart Contract
VPN	Virtual Private Network

List of Figures

List of Tables

Appendix A

Installation Guidelines

Prerequisite to run the application; Wireshark. Available at `https://www.wireshark.org/download.html`

Installation Python Libraries:

- requests
- scapy
- tkinter
- pickle
- speedtest
- urllib
- ftplib
- web3

Environment There must be an env.py file in the Client subfolder. This file must have three variables declared, with these names: serverIP, contractABI, and contractAddress.

To Run:

1. Deploy the smart contract to the blockchain. Recommended: Remix and Ganache. The ABI can be found under the Compile tab in the Remix IDE. Once deployed, the contact's new address (and the ABI if necessary) must be updated in the Client/env.py file.
2. The scripts must be interacted with from a terminal window with administrator privileges. This is to ensure that the measurement process has sufficient privileges.

3. The GUI allows for two use cases: To place bounties To take a new measurement, collect the NN metrics, commit these to the SC, check for any qualified bounties, and collect them

4. Other interactions with the SC must be done through the command line. Possible commands:

   (a) python Client/SmartContractTest.py getBountyTimestamps $address
      - Get a list of timestamps the $address has placed bounties at

   (b) python Client/SmartContractTest.py getBounty $address $timestamp
      - Get the values of the bounty submitted by $address at $timestamp

   (c) python Client/SmartContractTest.py getMeasurementResults *address*index
      - Get the test descriptors and metrics collected by *addressat*index

   (d) python Client/SmartContractTest.py filterMeasurementsByISP $targetISP
      - Get the measurement addresses and indexes with $targetISP

   (e) python Client/SmartContractTest.py filterMeasurementsByPing $targetPing
      - Get the measurement addresses and indexes with ping in a range around $targetPing (in ms)

   (f) python Client/SmartContractTest.py filterMeasurementsByDistance $target-Distance
      - Get the measurement addresses and indexes with distance in a range around $targetDistance (in km)

Appendix B

Contents of the CD

- Project Report PDF file
- Source Code of the Overleaf file
- Source Code of the Tool
- Final Presentation