M.Sc. Thesis
Computer Science and Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Improving anonymity in the use of TURN servers

OnionRTC

Jonas T
Christian M

Kongens Lyngby 2023

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Supervisors:
Birger Andersen (Department of Engineering Technology)
Bhupjit Singh (Department of Engineering Technology)

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

When a client is doing a WebRTC call they will in some cases end up using a TURN server to establish a connection with another client. This means that the TURN service provider will know based on IP who is communicating and can based on metadata of the call, guess if there's audio, screen sharing and or video in the call and potentially in which direction the communication flows.

The thesis project set up a test framework and used different anonymisation networks while using WebRTC through TURN, in order to test different configurations of anonymisation networks. To find if the networks were able to support a WebRTC session with sufficient quality of service. The test was run over a period of a month.

The findings of the experiment successfully showed that some configurations of the TOR network were usable in the context, where only one client was anonymous and the other was a regular client. Thus making one of the clients in the session anonymous. Using an anonymisation network on both sides was also tested, but resulted in unacceptably high latency in the call.

Preface

This thesis "*Improving anonymity in the use of TURN servers*" was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark. The thesis was done in fulfilment of the requirements for acquiring a master degree in computer science.

Kongens Lyngby, February 26, 2023

Acknowledgements

A big thank you to Birger Andersen and Bhupjit Singh for their continuous supervision and guidance throughout the project. All from concept, research, analysis, design and discussions about the project.

A special thank you to Morten H. Gynning for helping with the design and deployment of the experiment setup at DTU Ballerup.

Bifrostconnect for sharing the issue and concern that some of their customers have about the WebRTC part of their product.

Glossary

API Application Programming Interface. 23, 24, 43, 55, 73, 76, 77

CGN Carrier-grade Network Address Translation (NAT). 16, 17

DHT Distributed Hash Table. 35, 36

DNS Domain Name System. 15, 66

DTLS Transport Layer Security. 24, 46

I2P Invisible Internet Project. 34–37, 51–53, 74, 95, 101, 106, 111

ICE Interactive Connectivity Establishment. 1, 2, 15, 21–24, 26, 44, 58–60, 67, 68, 73, 77, 97, 105, 110

IETF Internet Engineering Task Force. 23

IP Internet Protocol. 2, 4, 8, 9, 15, 16, 18, 19, 21, 23, 24, 32–34, 38, 42, 44, 49, 53, 59, 66, 68, 77, 95

ISP Internet Service Provider. 16, 17, 32, 35, 93

ITU International Telecommunication Union. 28, 41, 42

LAN Local Area Network. 16, 18

LLARP Low Latency Anonymous Routing Protocol. 36

MCU Multipoint Conferencing Unit. 27

MOS Mean Opinion Score. 28, 30, 42

NAT Network Address Translation. vii, 1, 15–21, 34, 49

QoE Quality of Experience. 28–30, 41, 43, 50, 55, 56

QoS Quality of Service. 11, 28, 30, 39–42, 49, 55, 58, 89

RTCP Real-time Transport Control Protocol. 11, 12, 24

RTP Real-time Transport Protocol. 9–12, 15, 24, 40, 41, 50, 53, 81, 94

RTT Round Trip Time. 7, 8, 13, 30, 80, 81, 84, 85, 87, 96, 112, 116

SDP Session Description Protocol. 15, 16, 21, 60, 67, 68, 77

SFU Selective Forwarding Unit. 27, 109

SIP Session Initiation Protocol. 15, 16, 41

SRTCP Secure Real-time Transport Control Protocol. 24

SRTP Secure Real-time Transport Protocol. 24

STUN Session Traversal Utilities for NAT. 18–21, 24, 26, 36, 49, 60, 77, 98

TCP Transmission Control Protocol. 8–10, 17, 20, 23, 24, 32–34, 36, 40, 42, 44, 45, 53, 56, 77, 93, 94

TLS Transport Layer Security. 21, 24, 33, 66

TOR The Onion Router. 1, 32–34, 36, 38–42, 44, 47, 51–54, 62, 73, 74, 80, 81, 89, 93, 94, 96

TURN Traversal Using Relays around NAT. 1–5, 7, 18, 20, 21, 24, 26, 36, 44–46, 49, 51, 53, 58, 60, 66–69, 73, 74, 77, 93, 95–98, 105–107, 109

UDP User Datagram Protocol. 9, 15, 17, 20, 24, 34, 36, 40, 42, 44, 45, 53, 77, 94

VoIP Voice over IP. 7, 8, 13, 15, 28, 29, 32, 37–43, 47, 55, 96

VPN Virtual Private Network. 21, 30, 31, 40, 41, 53

WAN Wide Area Network. 16, 18

WebRTC Web Real-Time Communication. 2–4, 7, 20, 23–30, 36–38, 42–47, 49–51, 53, 55–62, 65, 67, 69, 70, 72–74, 77, 79, 80, 93–97, 101, 102, 105–107, 109, 110

Contents

CHAPTER 1
Introduction

The more of the world that gets access to the internet, and the more everyday meetings and social interactions move to online platforms, the more data becomes available for collection. Metadata about people, their households and social interactions and interests are one of the key ways to make money on the internet.

Metadata is most often used to sell ads. But the governments can also use the metadata to make life or death decisions [1]. This is an outcome of the rapid development of the internet, legislation not always keeping up with the technologies and may be due to users' unawareness of the issues leaking metadata can cause. Discussing privacy is often followed by the argument of "I have nothing to hide" but as Edward Snowden has phrased it, it's the same as saying; "you don't care about free speech because you have nothing to say" [2].

One of the best tools to become anonymous on the internet today is using anonymisation technology like network overlays such as The Onion Router (TOR). It's often connected with a taboo, since the media's representation of the "darknet" is that it is used by criminals. But the "darknet" and being anonymous on the internet is also used by freedom journalists and gives access to the "clearnet" for people from countries with heavy censorship.

1.1 Motivation

Within the last decade, companies have adopted video conferencing and screen-sharing applications like Zoom, Microsoft Teams and Slack. Private consumers have started to adopt applications like WhatsApp, Meta Messenger, Signal and Apple FaceTime instead of traditional SMS and phone calls. One thing that all these applications have in common is that for situations where both clients are behind a strict NAT, but still want the fastest connection. Then there may be a need for using a Traversal Using Relays around NAT (TURN) server as an intermediate between the two clients as seen in Figure 1.1.

For video and audio calls, the clients can negotiate how to connect to each other through the Interactive Connectivity Establishment (ICE) protocol, which is used to exchange the types of connections that the given clients can make to connect to each other. The most optimal connection that two peers can make is a direct peer-to-peer connection. The number of connections where this is not possible fluctuates

depending on the source. Callstats.io who makes Web Real-Time Communication (WebRTC) monitoring tools stated in 2017 that "***on average about 30% of the P2P conferences has one endpoint connect via a TURN server.***" [3]. In these 30% of the cases the TURN service provider can gain knowledge of the users who are going through the TURN server.

The knowledge a TURN service provider can gain is however limited. The media data that flows through them is, by today's standard, already end-to-end encrypted. The service provider can, however, by looking at metadata, see who is communicating, their Internet Protocol (IP) address, the amount of traffic and the timing of the interactions. The data can be used to make estimations on what type of call is being made, if it is a voice-, video- and/or desktop streaming call and in what direction data is flowing. Furthermore, the service provider can by gathering information about callers and types of calls, create a social network graph of how different IP addresses communicate with each other.

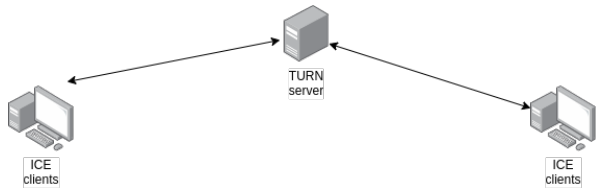
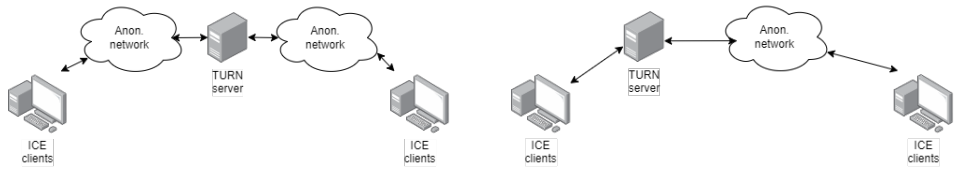


Figure 1.1: Alice and Bob are both connected to the application server and through ICE they find the TURN server that they will use to create a connection through.

A TURN service provider can gather the data by simply having customers using the service, and without the customers being able to know that they use the infrastructure or a specific service provider. This makes it a trust and privacy issue.

The data which a TURN service provider can collect from participating clients in a call, is their public IP addresses. The actual data can be encrypted, but based on the throughput between the two clients can the provider make guesstimates on the type of call i.e. Audio, Video or Screen sharing and know the duration of the call. Further, the provider could create social networks of which IP addresses communicate with whom.

The proposed solution investigated in this thesis is using already implemented internet privacy-enhancing technology, like seen on Figure 1.2. The technology works by introducing pseudonyms, mixing traffic or somehow hiding the origin of where data is from. Privacy networking technologies like proxies, mixing networks and onion routing are used on the internet to hide this information, even when adversaries might be able to look at how the clients' internet traffic flows



(a) Both clients use an anonymisation network.

(b) One client use an anonymisation network.

Figure 1.2: Proposed solutions.

1.2 Objectives

To be able to solve the issue described, the following sub-goals have been identified and will be investigated in the next chapter:

- Validate the privacy problem around using TURN servers.
- Find different anonymisation networks, their advertised network performance and their intended use-case.
- Look at the advantages and disadvantages of anonymisation networks and find if any of them are possible drop-in or configurable solutions to existing applications using TURN.
- Figure out which anonymisation networks offer support for directing WebRTC traffic through the network.
- Understand the theory around media streaming and how to do network performance analysis in regards to WebRTC.
- Find the real-time requirements of the protocols used in WebRTC and how to monitor them.
- Figure out what is required for doing an automated WebRTC session, with statistics gathering.

1.3 Delimitations

This thesis could lead us down many different paths, the majority of which would be both interesting and exciting to investigate further. However, the limited time frame has made it a necessity to limit the scope of the project.

The project will not go into the depth of how and what could be fingerprinted of a user when they are using a TURN service; this includes guessing what type of

traffic and data the client is sending through the TURN server. Neither will we look into matching certain clients using the service, at roughly the same time every day or week.

Within the topic of anonymisation networks, there are many attacks some of which will be described later, but the project will not focus on solving the issues related to the attacks on anonymisation networks or protocols.

The project will focus on TURN servers that are used for relaying WebRTC traffic. WebRTC preferably uses a peer-to-peer connection, due to the real-time requirements related to audio and video communication. WebRTC does, however also support text chat and file transfer, but since the real-time requirements for those functions are not that strict, the focus will only be on the usability of WebRTC video and audio transmission.

To summarise; the thesis work is limited to hiding the real IP address of the client from the TURN service provider. This will be done using anonymisation networks, which hide the origin of IP traffic. Measurements of the usability of the proposed anonymisation network and configurations should be done.

1.4 Structure of thesis

The thesis is divided into 8 chapters. This first chapter is an introduction to the project, which include the motivation for the project, what objectives it is trying to accomplish, scoping of the project and the methodology used. Chapter two covers all the theory and background around the subject of WebRTC automation and anonymisation networks while addressing prior work and possible privacy issues. Chapter three covers the problem statement and in chapter four an analysis and the design of an experiment are covered. In chapter five the implementation of the experiment is explained, covering tools, frameworks and concepts used in setting up and running the experiment. Chapter six presents the results of the experiment, which are followed up with a discussion and future work in chapter seven. Lastly, in chapter eight, the conclusion is presented.

The practical implementation will be documented in the project's GitHub organisation [4].

1.5 Methodology

The process of working on this project is an iterative process with a well-defined core problem from the beginning. The problem arises from a discussion about a potential customer of BifrostConnect Appendix A "*Real-life usecase: BifrostConnect*", which uses WebRTC and TURN as part of their service.

The project starts with a thorough investigation of the problem and a review of academic papers on the subject. Then the problem statement is defined and the project objectives are set up. Next, a testing environment is established to validate

the problem and gain practical knowledge in the field. Afterwards, an analysis and design process is conducted to answer the project objectives, by designing a software solution/framework, for testing and analysing the performance of the proposed solutions. After the tests are finished, the results are analysed, and the project conclusion is reached.

During this process, there are some setbacks or new insights that require the project to go back and set more restrictive requirements or re-evaluate the current design, making the project flow more iterative than a straightforward waterfall process.

1.6 Summary

This chapter has introduced the main topic of the thesis in very broad terms and given some context to where and how the use of a TURN service can leak private metadata. It has also presented objectives to research the problem. Furthermore, delimitations were created in order to help keep the focus of the project. Then an overview is presented of the structure of the report. Lastly, a description of how the project has been run and executed.

CHAPTER 2
Background and related work

This chapter will give a comprehensive introduction to the terms used, as well as a through introduction to the background with references to further reading materials. A walk-through of the related academic work upon which this thesis is based. Lastly two examples of attacks on privacy related to WebRTC and TURN

Due the detailed introduction to the theory in this chapter is it advised to read the Project Definition (chapter 3) first and come back to this chapter.

2.1 Background

The background section will start by covering an overview of different networking measurements that are relevant to the project. Then an introduction to the world of multimedia streaming over the internet will be given, and move on to explain session and connectivity establishment and signalling protocols used. Next, a section on how WebRTC works, how it is used today and how to measure WebRTC streaming quality. Lastly, an overview of anonymisation networks will cover the different strategies/networks that are available.

2.2 Networking measurement overview

When looking at the performance of a network connection, there are multiple things to consider. For Voice over IP (VoIP) traffic specifically, it is important to look at throughput, bandwidth, latency (one-way-transmission), jitter, Round Trip Time (RTT) and packet loss. [5, 6]

For a visual representation of latency, throughput and bandwidth, the pipe on Figure 2.1 can be observed. Bandwidth, which can be thought of as the capacity, or diameter, of the pipe and is usually measured in bits per second (bps) or packets per second. When sending data over the internet via different routers, each link between the routers have its own available bandwidth, which can be different depending on the load and the type of medium/link. Because of this, the bandwidth of a network path

is limited to the link on the path with the lowest bandwidth available. A network link might be a high-bandwidth link, but it can still experience networking errors (like packet loss, jitter or latency), which will impact the throughput. Throughput is the volume of data that can be transferred over a time period, which is also measured in bps.

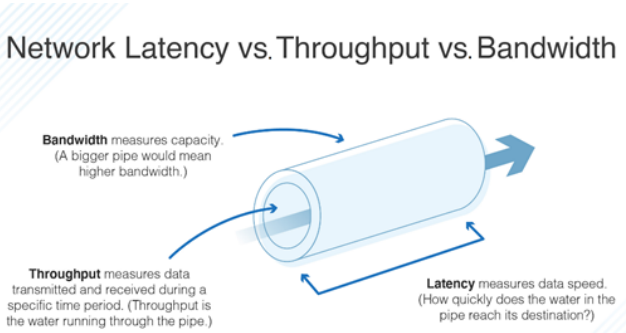


Figure 2.1: Visualisation of latency vs throughput vs bandwidth [7].

Latency is a measurement in milliseconds (ms) of the delay of a "one-way transmission", how long does it take for a data packet to reach the destination. The "delay" is composed of several different types of delays found in traditional packet-switched networks, which happen when a packet is sent over the internet via routers. The first type of delay is the *Propagation delay*, which is influenced by the physical medium, which the bits happen to travel through, whether it is fiber optics or copper wiring. *Transmission delay* is influenced by the size the packets are and the transmission rate of the network link. *Processing delay* is the time it takes for routers to inspect and parse IP headers, do checksum calculations and decide on the routing of the packet. Lastly, the *Queuing delay* is the time a packet is waiting in a buffer to be handled. If the buffer in the router is filled up, the router will start to drop incoming packets, which leads to packet loss. Latency can sometimes be misrepresented and measured as the RTT instead, since Transmission Control Protocol (TCP) and some application-layer protocols require an acknowledgement to be sent. [7, 8, 9]

Jitter is measured in ms and is the variance in latency or the time delay between the packets, which can be seen visualised on Figure 2.2, where X, Y and Z varies in length. The variance can be thought of as the disruption of the normal packet flow, which can lead to artefacts and glitches in the video and audio stream. Jitter can be divided into 3 types: a constant level of jitter, transient jitter, which is caused by a single packet, or short-term jitter, which is due to multiple sequential packets being delayed. This is typically due to congestion control being enforced or route changes on the network. In VoIP applications a small amount of jitter is normally tolerated

and/or accounted for. This can be done with ”error concealment-” or ”recovering-methods” like ”Forward Error Correction”, interleaving or packet loss concealment, at the cost of increased delays [8].

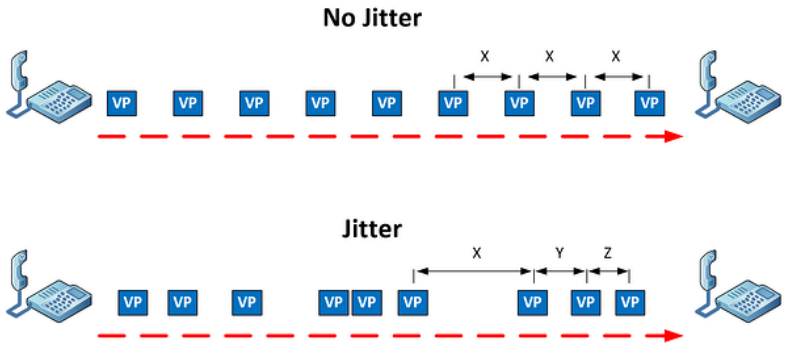


Figure 2.2: Visualisation of jitter [10].

2.3 Media streaming over the internet

Traditionally, TV and telephone conversation was consumed via dedicated cabling that used dedicated slots and resources to provide service. Specifically for telephones a circuit-switching network is used. It works by reserving dedicated resources to a call, which guarantees a stable end-to-end delay, which is important for conversational media that can be highly sensitive to end-to-end delays. But as an increasing amount of modern multimedia consumption is provided over the internet, modern streaming applications need to handle the consequences of switching to the packet-switching networking used today in IP. This includes accepting the ”best-effort” service model, which can not guarantee a stable end-to-end delay or a low jitter.

Generally, multimedia applications over the internet can be roughly categorised into three categories; ”streaming stored audio/video”, ”streaming live audio/video” and ”conversational voice/video”. This project will mostly be focusing on the requirements and design challenges, that come with the ”conversational voice/video”, which has the most strict real-time requirements.

Traditionally internet media streaming has been done using regular HTTP file streaming, Dynamic Adaptive Streaming HTTP (DASH), HTTP Live Streaming (HLS), Real-Time Messaging Protocol (RTMP), or Real-time Transport Protocol (RTP). While the three first are based on HTTP and RTMP is based on TCP, the last protocol is unique by using User Datagram Protocol (UDP) for the transport layer [11]. However as it can be seen on Figure 2.3, the different protocols cater to different use cases and targeted latency.

RTP [12], which, despite having ”Transport” in the name, is actually an ”application layer” protocol and is transport-independent [13], but UDP is usually always

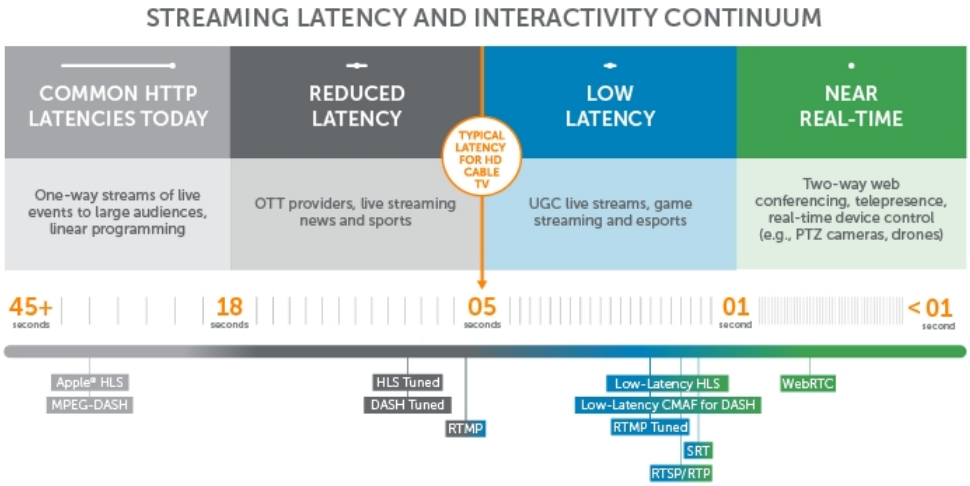


Figure 2.3: Commonly used streaming protocols and the latency, they are targeting [11].

preferred. The RFC3550 [12] defines two protocols, where RTP is used as the data plane which does encapsulation of video frames and audio chunks into streams. It provides the ”missing” sequence numbers that are lost by not using TCP and introduce timestamps for media timing purposes, which can be seen on Figure 2.4. The RTP payload that carries audio and video packets, supports a wide range of formats like MPEG, motion JPEG and the H.26x family for video, and audio formats like PCM, ACC and MP3. These different formats define how the audio and video are encoded and decoded by the sender and receiver. The formats define the following: what sample rate has been used, which compression algorithms has been used, and if the audio and video are bundled into one stream or two streams. The chosen format can be changed during an RTP session and is quite important for the media application, since it allows it to offer different media qualities depending on the current networking conditions or user preferences [8].

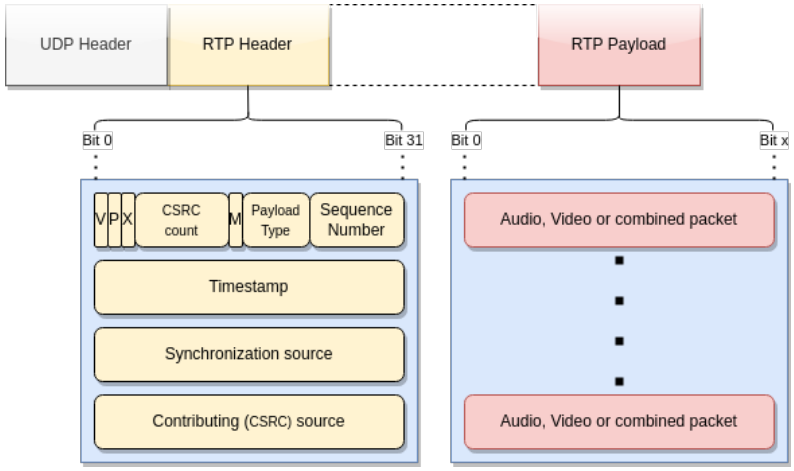


Figure 2.4: Overview of a RTP header and payload [12].

The other protocol Real-time Transport Control Protocol (RTCP), which is defined in RFC3550 [12] is the control plane. It introduces support for multi-party conferencing, audio and video synchronisation, sender and receiver reporting and networking characteristics[1], that the application layer can use for Quality of Service (QoS) feedback. This could be done by adapting the quality of the media that the application tries to send. On Figure 2.5, the fields of an RTCP header and the standard RTCP payload types can be seen. RTCP allows for "compound packets", which means that multiple RTCP payload types (red packets) may be concatenated/encapsulated together and sent in one RTCP packet.

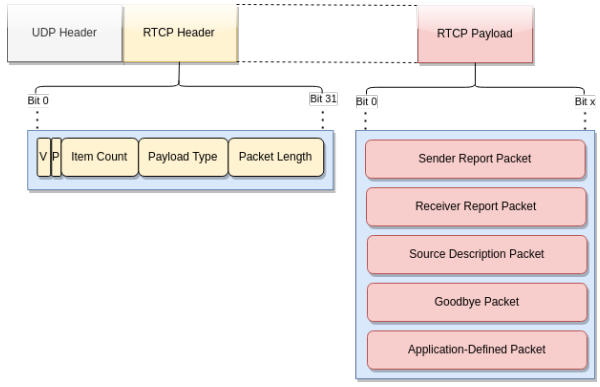


Figure 2.5: Overview of a RTCP packet [12].

[1]Including estimated available bandwidth, that might be asymmetric between peers [14]

The RTCP payload types "Sender Report" (SR) and "Receiver Report" (RR) are respectively sent from the sender to the receiver and then from the receiver to the sender, where the RR is generated as a response to the SR. Both reports include the $timestamp_1$ for when the SR packet was sent, plus the added processing *delay* time on the receiver side. When the original sender receives the RR, the current time is set as the second $timestamp_2$. Then the Round-trip time can be calculated by the sender from the following:

$$rtt = timestamp_2 - timestamp_1 - delay$$

The reports also include data about: packets lost, sequence numbers, jitter and different timestamps for synchronisation purposes [14, 12]. Lastly the (optional) rtsp (Real-time Streaming Protocol) [15], is an application-level protocol that enables control over the data delivery, by providing pause/play, replaying and "jumping" in the media playback, which can be useful in the case of streaming stored/live media [16].

One of the most important differences between streaming conversational video vs stored video is that you rather want *continuous playback* over *lossless playback* (timeliness over reliability). Where *continuous playback* means that when the video starts it should be kept as "live" or "real-time" as possible. To keep this real-time requirement, video techniques and artefacts like "frame freezing" and "frame skipping" might occur in the video [8]. Furthermore, stored and live media can typically use techniques like CDN distribution for broadcasting when the media is multicasted on a global scale and they can use a variety of different buffering delays. Some buffering can also be used for conversational video and audio streaming, but the size of acceptable buffer delays are, however, a lot smaller in conversational video and audio streaming, as the initial buffering nor the end-to-end delay can be that big, without making for a poor conversational experience.

The most important (performance measure/networking condition) for streaming video is the average throughput. The streaming application must be guaranteed by the network an average throughput that at least meets the bit rate of the video itself. But even a fluctuating throughput around the bit rate of the video can be dealt with by adapting the video quality to bandwidth availability and using some of the before-mentioned buffering to smooth out smaller network delays and jitter [8]. One of the techniques to "absorb" some amount of jitter is to use a jitter buffer at the cost of an extra delay. An example of using a jitter buffer for a video frame can be seen on Figure 2.6. RTP will use the encoded media chunks that the chosen compression algorithm created, to be sent as media packets over the wire. The media chunks will be put into the jitter buffer on the receiver side. From the jitter buffer, a timer will start and count how long it takes before enough media packets have been collected. This timer is called the *jitterBufferDelay* and has a great impact on the playback smoothness. If the timer is overrun, the jitter buffer will throw away media packets which are not relevant anymore. If the jitter buffer has a complete set of

media packets, they can then be decoded and reconstructed on the receiver side to a complete frame, which can be displayed to the user [6].

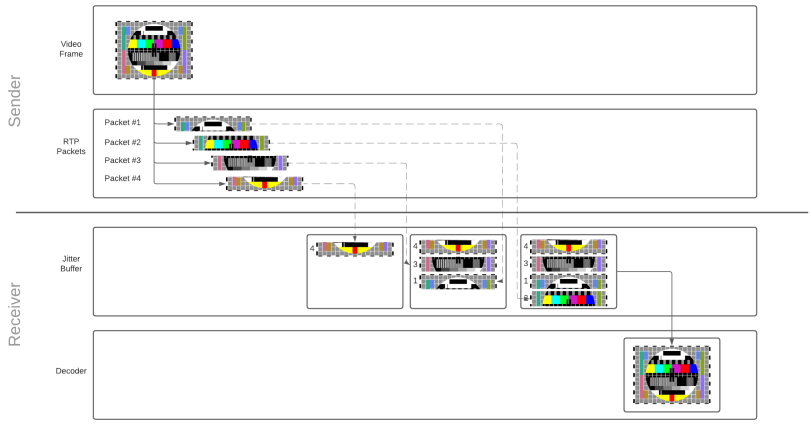


Figure 2.6: Visualisation of jitter buffer [6].

Conversational internet telephony and VoIP are therefore both bandwidth-, delay- and time-sensitive applications. This leads to bandwidth requirements/recommendations for audio around 1-2 kbps to 1 Mbps, and 10 kbps to 5 Mbps for video, which is quite a big range. Delay and time requirements are typically stated as end-to-end RTT delay of 300-400 milliseconds as acceptable, a <2% packet loss and network jitter up to 30-40 milliseconds [8, 16, 17, 18, 19, 20, 21]. Looking at the popular conversational video and audio streaming platform "Microsoft Teams", they have defined similar, but even stricter requirements that can be seen in Table 2.1 below.

Network metric	**Typical value**
Roundtrip time[2]	<200 ms
Received packet loss	<2%
Audio metric	**Typical value**
Sent bitrate	>24 kbps minimum; 36-128 kbps is typical
Received jitter	<30 ms

Table 2.1: Recommended video meeting and call metrics from Microsoft Teams [18]

Zoom has a more clear distinction for their recommended bandwidth requirements. They have divided them into 3 different types of meeting categories: 1:1 video call, group call and "webinar attendees", with video. The video requirement ranges from 600 kbps (up and down, resulting in 1.2 Mbps) for their definition of "high-quality

[2]From source: *"In group calls, it's the response time between your system and the Teams Service. In one-on-one calls, it's the response time between your system and the other participants."* [18]

video" to 3.8 Mbps up and 3 Mbps down. They also state some smaller numbers for video screen sharing 50-150 kbps and for audio calls 60-100 kpbs [20, 21]. The values are summarised in Table 2.2

Network metric	**Typical value**
Roundtrip time	<300 ms
Received packet loss	<2%
Received jitter	<40 ms
bitrate	**Typical range**
Video	600-3800 kbps
Screen sharing	50-150 kbps
Audio	60-80 kbps

Table 2.2: Recommended video meeting and call metrics from Zoom [20, 21].

YouTube as a platform, also supports live streaming. The requirements they have setup for different streaming resolutions is shown in Table 2.3. Since they support different ways of streaming, they also work with different configurations of latency, chosen by the client, "normal", "low" and "ultra-low" where the ultra low is to be used in streams where the entertainer will interact directly with the audience, and low is the most stable and the audience will experience the least amount of buffering [22].

bitrate	**Typical range**
Video @ 240p	300-700 kbps
Video @ 360p	400-1000 kbps
Video @ 480p	500-2000 kbps
Audio	128 kbps

Table 2.3: Recommended bandwidth to stream to YouTube live [22].

2.4 Signalling, connectivity establishment and session management for media streaming

The previous section dealt with encapsulating and transporting media data over the internet. This section will explain how the connection and a session between clients is set up in the first place using Session Initiation Protocol (SIP), Session Description Protocol (SDP), and the ICE protocol.

For clients to be able to establish a voice/video call between a caller and a callee, the caller needs to be able to ”signal” to the callee, that they wish to initiate a call. For this to happen, some informations should be known by the caller and the callee: caller should know how to contact the callee, the callee should know how to contact the caller and which encoding format the callee and caller supports. This information can be transported in a couple of different ways, which is left for the application developer to choose. This is commonly done using Websockets or another protocol that is built on top of HTTP [9, 23]. This allows peers to do discovery and share information related to starting and joining a session. For the actual ”signalling” part, the protocol SIP is popular since it is also widely used in traditional VoIP. It can be used for starting and controlling sessions and supports a wide range of infrastructure. A ”SIP proxy” serves to support traditional phone-like-functionality, ”Session border controller” for NAT Traversal and ”SIP registrar” servers that offer Domain Name System (DNS)-like-functionality for client discovery. The protocol is transactional and uses an ”offer/answer-model” similar to HTTP, where the response includes an acknowledgement and 3-digit HTTP-status like codes. Lastly, SIP also provides general session management, that offers management functionality like adding more participants to a call, changing media encoding used for the call, ending the call etc [8, 24]. An example of a SIP message can be seen on Listing 2.1, where the caller invites the callee ”user” to a session. The callee can use the IP 192.0.2.1 on port 5060 with UDP to connect to the caller.

```
1    Method: INVITE
2    Request URI: sip:user@example.com
3    From: <sip:caller@example.com>
4    To: <sip:user@example.com>
5    Contact: <sip:caller@192.0.2.1>
6    Via: SIP/2.0/UDP 192.0.2.1:5060
7    Max-Forwards: 70
8    Content-Type: application/sdp
9    Content-Length: ...
```

Listing 2.1: Example of a SIP message [25]

As mentioned in the description of RTP, there are a lot of different formats for only audio, only video, and combined audio-video, which are used in media applications. For the best compatibility and to ensure exchangeable media formats, SIP supports clients using different compatible formats in the same session. The chosen media

format and metadata is part of the session descriptors, that can be part of the payload of SIP messages. The session descriptors of an SIP message are usually formatted using SDP, which is a protocol for defining a description of a session. The protocol defines a large number of different fields, some of which are shown in the example Listing 2.2. In this message, 2 audio options and 1 video format is listed as potential media formats for the session.

```
1   v=0 # (protocol version)
2   o=jdoe 372400 373945 IN IP4 192.x.x.1 #(caller and session identifier)
3   s=Call to John Smith #(session name)
4   i=SDP Offer Number 1 # (session information)
5   ...
6   c=IN IP4 192.x.x.1 # (connection information)
7   t=0 0 # (time the session is active)
8   # m= (media type, transport port, -protocol, media format description)
9   m=audio 49170 RTP/AVP 0
10  m=audio 49180 RTP/AVP 0
11  m=video 51372 RTP/AVP 99
12  c=IN IP6 2001:db8::2 # (connection information )
13  a=rtpmap:99 h263-1998/90000 # (zero or more media attribute lines)
```

Listing 2.2: Example of a SDP formatted session descriptor [26] with added comments

SIP and SDP are used for signalling and session management protocols, but unless certain networking conditions are present and SIP infrastructure[3] is setup on behalf of each client to establish connections, there is still the problem of connectivity establishment. This is due to how the modern internet works.

The widespread use of IPv4, limited address space and the increasing demand for internet enabled clients is why NAT often used [27]. NAT enables multiple clients to share a single public IP. NAT can be used in many places, the most common place is in home routers. The router will have on one side a single public Wide Area Network (WAN) IP, which is connected to the Internet Service Provider (ISP) and on the other side of the router, a Local Area Network (LAN) IP address is given to the clients connecting to it. An ISP can also place their customers behind a NAT, which is called Carrier-grade NAT (CGN), visualised in Figure 2.7. In this setup there are multiple customers behind the same singular public IP address [28]. This saves on the amount of public IPv4 addresses that the ISP are required to use in order to run their operation.

[3]SIP proposes to use a "Session border controller" for NAT Traversal, but does not support certain types of NAT configurations and does not see as much usage (https://bloggeek.me/gateway-for-webrtc/) (https://www.thirdlane.com/blog/nat-stun-turn-and-ice)

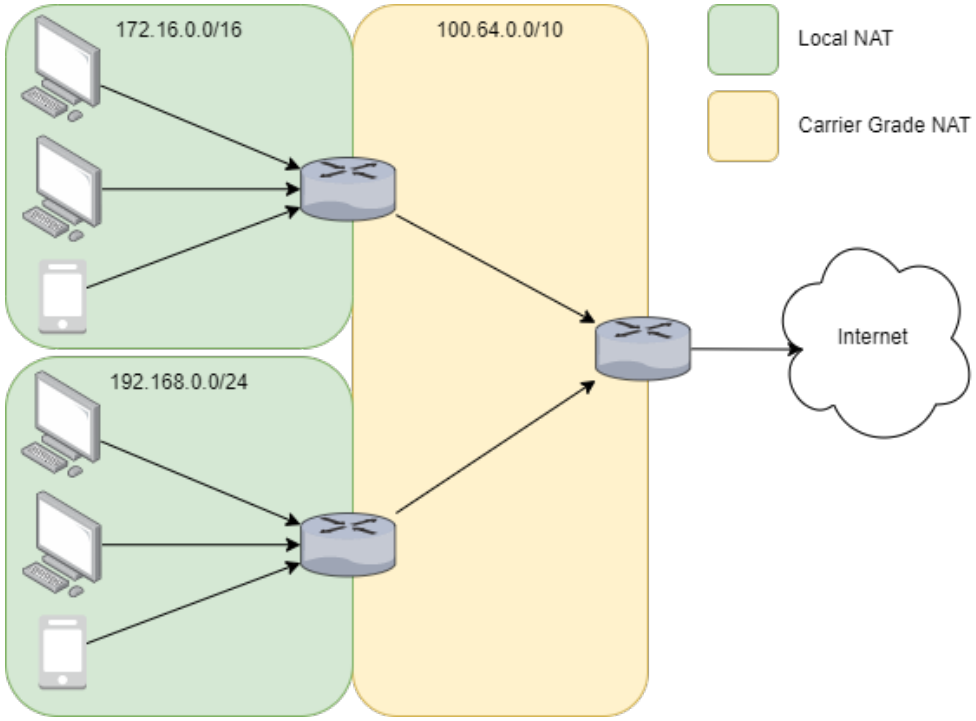


Figure 2.7: A visualisation of how CGN works in combination with local NAT. The figure illustrates two home or office networks where clients connect to their local router which has NAT enabled. The router is then connected to the ISP's router and finally the ISP's router is connected to the internet.

The solution to getting more clients connected to the internet with NAT has been very successful, but it comes with a downside, where it is harder to create a peer-to-peer connection. This is due to when a external host tries to send i.e. a TCP or an UDP package, directly to a host behind a NAT, where no port forwarding or similar is setup. Then the router will drop the packet since it doesn't know to which internal host it should forward the packet to. However, if the internal host, behind the router sends a packet first, then the router will expect a reply from the external host. Therefore, when the external host replies, then the router will know where to send the incoming packet. A visualisation of this can be seen on Figure 2.8.

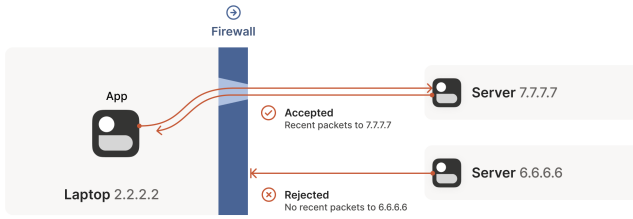


Figure 2.8: A simple view of a NAT firewall in a router. The laptop client creates a connection to the server at 7.7.7.7 and the router receives the reply. But if the server at 6.6.6.6 tries to send an unsolicited message, then the firewall rejects the package [29].

In general a NAT works by maintaining a table of internal hosts who have sent requests to a external host on the routers WAN side. This is done by storing the internal host's IP:PORT, the router's external IP:PORT and the destination IP:PORT in a table. From a top-level perspective there are four NAT configurations: "*Full Cone NAT*", "*Restricted Cone NAT*", "*Port-Restricted Cone NAT*", and "*Symmetric NAT*". The difference between these are the requirements they have in order to forward traffic from the external host's (WAN side of the router) to internal host (LAN side of the router) [30, 31, 29].

- **Full Cone NAT**: A permanent port forward setup. If the router receives data on its external IP:PORT, it knows for this pair, forward the packet to this specific internal host's IP:PORT.
- **Restricted Cone NAT**: Same as Full Cone NAT. But the internal host needs to have started the communication. The external incoming packet needs to come from a host IP which is in the NAT table with the same destination IP.
- **Port-Restricted Cone NAT**: Same as Restricted Cone NAT. But the external incoming packet needs also to come from the same destination PORT.
- **Symmetric NAT**: Same as Port-Restricted Cone NAT. But for each combination of the 3 IP:PORT (Internal, External and Destination) there must be a complete match, and the router's external IP:PORT can only be used once.

The different types of NAT configuration is important. Since the first third are the least restrictive methods; Full Cone NAT, Restricted Cone NAT and Port-Restricted Cone NAT can use Session Traversal Utilities for NAT (STUN) to connect clients directly to each other by doing NAT traversal. With the last NAT type; Symmetric NAT it is not possible to create a direct peer to peer connection and TURN is therefore required to connect with other clients.

Given a network where STUN can be used; then a direct peer to peer connection can work by each client asking a STUN server "*what IP:PORT do you see me coming from?*". Then the STUN server will reply with a STUN answer, which will include where the STUN server sees the original STUN request (the IP) came from. The two clients can now use their IPs which was given by the STUN server in an exchange of IP addresses and use it to start sending packets to each other, see Figure 2.9 for an illustration of this exchange. This method is called NAT traversal.

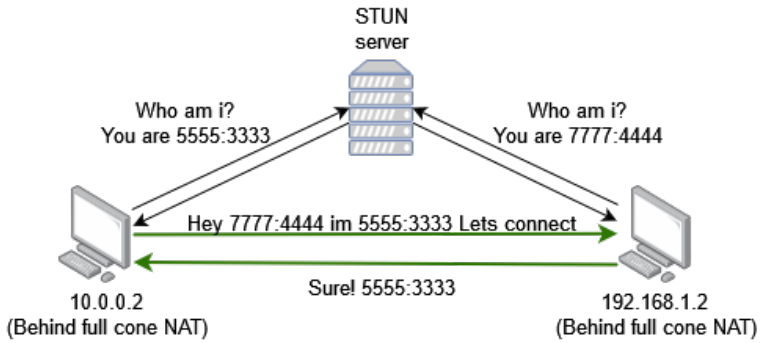
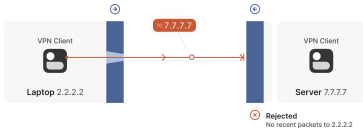


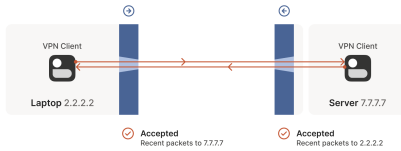
Figure 2.9: Example of the usecase of STUN, where the STUN messaging is hidden. Expect the exchange of the STUN messages to be communicated out of band to each client [32].

NAT traversal works by both clients knowing where to reach the other client, see Figure 2.10. Initially, the connection will one of the routers block the traffic from the other client, seen in 2.10a. Then the other client will also (on its own initiative) start sending packages to the first client and thereby open the NAT table. Since the first client already started sending packages is it's NAT table already opened and the package goes trough, seen in 2.10b. Now a connection between the two clients established as seen in 2.10c. This method works across the different NAT types/methods, except Symmetric NAT.



(a) When the first client receives the second client's STUN message it starts sending packages to the second client.

(b) When the second client receives the first client's STUN message it starts sending packages to the first client and packages goes through since it expects replies from the second client.



(c) A connection between two hosts who are both behind a NAT is now established.

Figure 2.10: Example of NAT traversal to connect two clients, each behind a NAT [29].

In the case of a Symmetric NAT where a TURN server is needed, then the communication pattern work by the clients connecting to a TURN server. The TURN server will then function as a relay/middle man and exchange the messages between the two clients as seen in Figure 2.11.

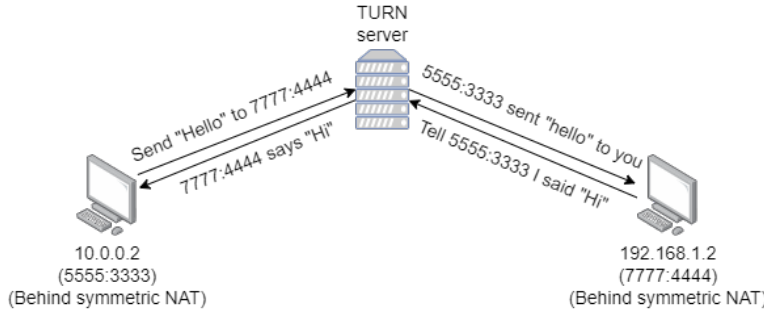


Figure 2.11: Two clients communicating through a TURN server [32].

Application that has real-time requirements usually uses UDP. But to support as many clients and strict firewall setups as possible, these applications usually provide the service with a TCP as a fallback method. This is also the case with WebRTC using TURN, as defined in the RFC6062 which adds TCP to the TURN protocol.

This also allows the media traffic to be encapsulated with optional Transport Layer Security (TLS) to further help bypass strict/cooperate firewalls [33, 23].

Modern devices often have several internet interfaces i.e. Ethernet, WiFi, 4G, virtual interface through a Virtual Private Network (VPN), or similar. Each different interface can give a different route to the internet, use different NAT methods and provide different properties with respect to available bandwidth, throughput and latency. To find the most optimal path between clients the ICE protocol is used.

ICE works by client A starts to collect all the available connection candidates. These can be local IP addresses, reflective addresses through STUN, or relay addresses through TURN. Then the client A create an SDP offer message and send the message through a signalling server to client B. Client B will use the offered SDP message and create an answer SDP message, send the answer to client A and both clients now have all the information needed to create a connection, being either local or publicly peer-to-peer, or through a TURN server [34]. An example of an ICE session can be seen in Figure 2.12

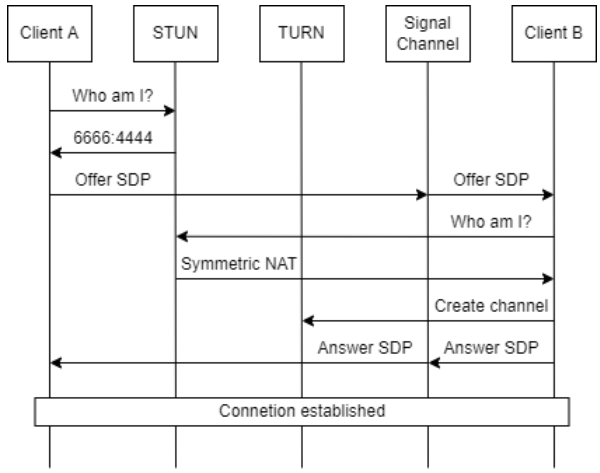


Figure 2.12: Example of an ICE exchange between two clients [35].

The state of ICE can be described with a state diagram shown in Figure 2.13 where *new* is the initial state. *Checking* is a state where the ICE agent has received one or more remote candidates, and is trying to connect to them but the connection have not yet been established. The state *Connected* is when a usable connection has been established, if more candidates are still present it will try to create those connections as well and shift to these if the connection is better. *Completed* is when gathering of candidates, trial of connections and a connection is found. *Failed* when all gathering is done and trial of connections and one or more failed establish a connection. *Disconnected* is when liveness checks to one or more components failed, often seen on flaky networks and can resolve itself over time. *Closed* when the ICE agent is no longer responding to requests [9].

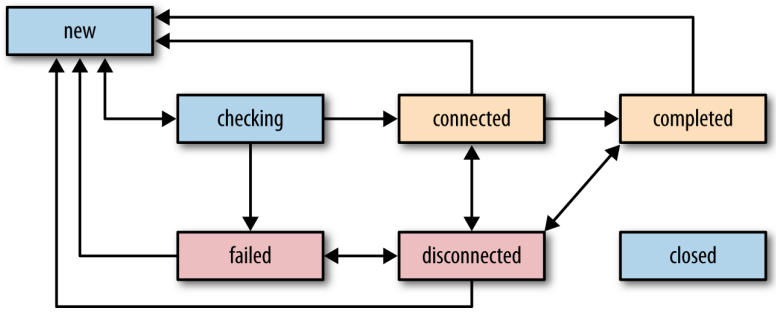


Figure 2.13: ICE protocol state diagram [9].

A problem with ICE is when clients start new internet interfaces or gets a new path through the internet or performing quick connections, the clients can exchange new ICE candidates once the clients gets them, and change the data streams to the other interface. This method is called trickle ICE and an sequence diagram is provided in Figure 2.14 [36].

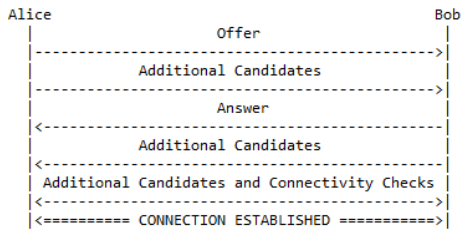


Figure 2.14: Example of a successful trickle ICE exchange [36, Figure 1].

Trickle ICE is typically used when you want to start a session as fast as possible, since gathering different candidates and doing connectivity checks for all possible connections can take time, especially if some of the connectivity checks are timing out [37].

2.5 Theory of WebRTC

WebRTC is a collection of open web standards and protocols, which enables peer-to-peer real-time communication and data sharing. WebRTC can be implemented and used by applications running both in a browser or in a non-browser-based application. The biggest and feature-rich implementations are, however, for modern browsers. This is why W3C has created a simple Javascript Application Programming Interface (API) [38] standard and since WebRTC is still under development a cross-browser interoperability layer has been developed to ensure compatibility between differences in browser WebRTC implementations[4]. The responsibility of defining the relevant protocols, data formats and security considerations of WebRTC is defined by the Internet Engineering Task Force (IETF) [9, 39], who is also responsible for the standardisation of the Internet Protocol Suite (TCP/IP) [40].

The WebRTC API consist of 3 high-level interfaces: ”getUserMedia”, ”RTCPeerConnection” and ”RTCDataChannel” [39]. The ”getUserMedia” handles audio and video capturing and processing. This can be seen on Figure 2.15, where each media type has a dedicated engine with processing steps. The ”RTCPeerConnection” represents a connection between the local host to a remote peer and is in charge of all transport of audio and video data. The last interface is regarding the transport of arbitrary data (like chat messages or file-transfer) over a ”data-channel”.

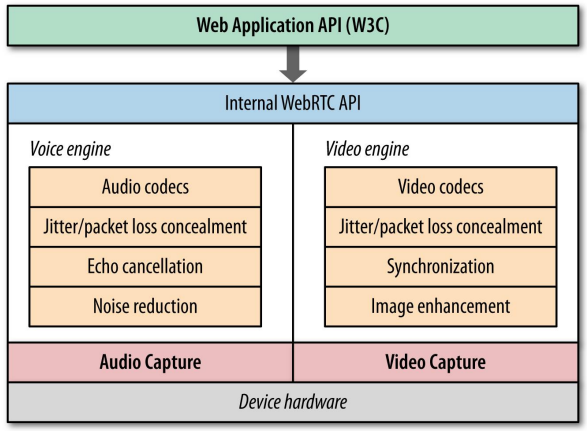


Figure 2.15: WebRTC API and engine overview [9].

For WebRTC to be able to set up and handle real-time audio, video and data-channels connections, the WebRTC stack comprises of different protocols, which can be seen on Figure 2.16. The left side of the WebRTC protocol stack is for signalling/discovery, which is used for exchanging the ICE candidates, so two peers can

[4]Today, most modern browser are chromium-based and therefore uses the chrome specific WebRTC implementation. Firefox does however have its own.

know of each other when trying to set up a WebRTC connection. The signalling uses application layer protocols like WebSocket over HTTPs that use TCP for transport since there is no real-time requirement for the signalling process. On the right side, the transport protocol SCTP (Stream Control Transmission Protocol) is in charge of providing transport for a data-channel, which can be configured to provide reliability and ordered delivery known from TCP in a message-oriented known from UDP.

For the "RTCPeerConnection" box on Figure 2.16, the Secure Real-time Transport Protocol (SRTP) is in charge of sending real-time media. The '-s' is similar to in HTTPS, the "secure" version of RTP. SRTP is defined in RFC3711 as a secure profile for RTP, which provides confidentiality and integrity to the RTP traffic. Just like in the RFC for RTP, the RFC3711 also defines a secure variant of RTCP as Secure Real-time Transport Control Protocol (SRTCP) that has the same purpose and with the same added security as SRTP. The WebRTC standard outright bans the use of plain unencrypted RTP and RTCP, by enforcing the use of encryption. This is typically done by using Transport Layer Security (DTLS) [41], which is the equivalent of TLS for UDP packets. Lastly, the protocols ICE, STUN and TURN are used for establishing and maintaining the connection over UDP on the IP stack.

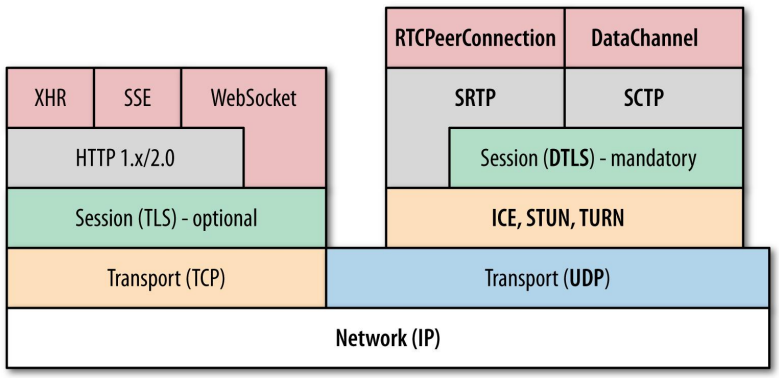


Figure 2.16: WebRTC protocol stack [9].

Taking a closer look at the "RTCPeerConnection" API visualised on Figure 2.17, the different responsibilities and components that the RTCPeeConnection interacts with are shown. This includes connection management and dealing with receiving and sending media streams. Together, the "getUserMedia" and "RTCPeerConnection" both deal with processing media data, the sending of media data, and managing the connections. When they are used by an application, the two APIs start every WebRTC session with video and audio at a lower bitrate (<500 Kbps) [9] and then adapt the bitrate of the media to match the networking conditions.

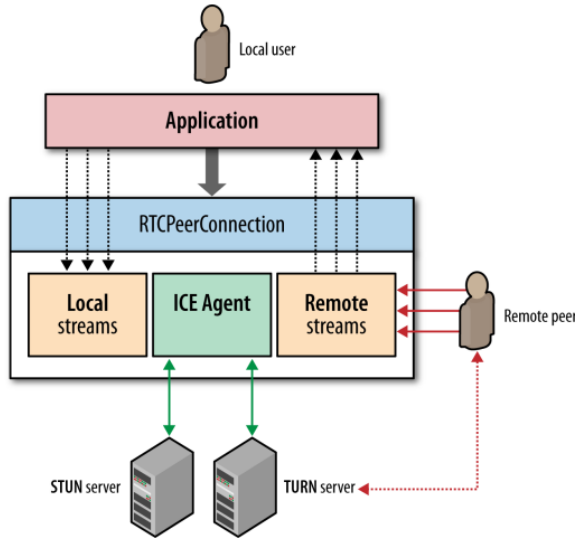


Figure 2.17: The "RTCPeerConnection" components [9].

This is called "Adaptive Bitrate Streaming", and is one of the ways in which WebRTC tries to deal with bandwidth changing, packets loss, and network jitter during a call. This adaptability can be seen in the WebRTC samples demo called "Per connection:adjust bandwidth" [42], which is only supported in Chrome browsers. The demo enables you to adjust the bandwidth available for a WebRTC stream sent from your localhost to your localhost. The quality/compression rate of the received video can be seen drastically impacted by video artefacts like ghosting appearing when limiting the bandwidth down to 75 kbps. From the "Bitrate" graph shown on Figure 2.18 it can be seen that the amount of data that has to be sent fluctuates around the available bandwidth during the call.

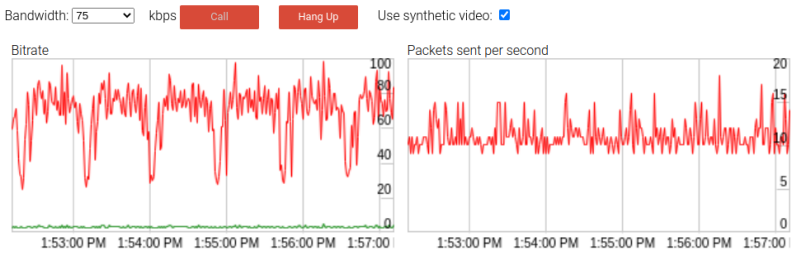


Figure 2.18: Graphs showing "Bitrate" and "Packets sent per second" for a WebRTC session in Chrome, which has been limited to 75 kbps on the demo website [42].

On Figure 2.15 the "Voice" and "Video" engines are shown. They include different media-specific effects and processes, that can be applied to the media. These are known as processing pipelines and can be dynamically changed according to changing user or media parameters as well as networking conditions [9]. The processing pipeline is a part of the "codec", that defines the rules that dictate how a bit stream should be interpreted, which compressing algorithms that should be applied and how the bit stream is encoded/decoded. But different implementations of a codec might strive for different performance characteristics. The most important characteristics when choosing a codec are as follows: kind of encoding (lossless or lossy), the added latency and processing delay (ms), how complex it is (CPU usage), how resilient it is (handling of packet loss), how popular it is (ecosystem and documentation), licensing, and lastly if it has hardware-acceleration support [23].

The WebRTC specification has a baseline of codecs that all WebRTC-complaint browsers must implement, which are described in RFC7742 [43] and RFC7874 [44]. RFC7742 states that the AVC/H.264 and the royalty-free VP8 video codecs must be available [45]. RFC7874 states that the audio codec Opus and the older G.711 "PCMA" and "PCMU" formats, must be available. The older G.711 codec is included for interoperability with legacy phone systems, that might be connected to a WebRTC call [44]. An alliance has been formed called "Alliance for Open Media" [46], which is developing open-source media codecs and members include big video streaming companies like Google, Amazon, Netflix, Apple, Microsoft and Meta. They are mainly working on specifying and developing the video codec "AV1", which is a lossy codec that is supposed to support modern video features like larger video resolutions, HDR and as well as having better data compression performance compared to its predecessors [47]. The codec is still under development, being implemented in browsers and has not seen widespread adoption in WebRTC yet.

As mentioned in Motivation (section 1.1) and "Signalling, connectivity establishment and session management for media streaming (section 2.4)", the ICE protocol, uses the STUN and TURN protocols. They require one type of server infrastructure which is a crucial part of WebRTC deployment. As real-time video streaming has become increasingly popular, the use cases and business cases for video streaming have evolved. This has led to innovations in internet-tv streaming, digital conferencing, real-time live streaming and remote access/desktop streaming. These different use cases can require some sort of pre-processing and can benefit from different communications patterns than the typically one-on-one video conversation. To support these different requirements, WebRTC has another type of infrastructure known as media servers and different communication strategies, that can be used. The different communication strategies can be summarised using Figure 2.19.

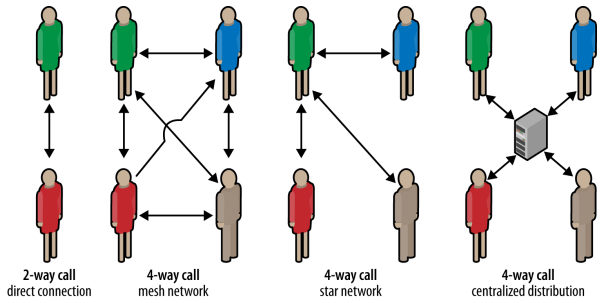


Figure 2.19: Different WebRTC communication architectures [9].

The 2-way call is ideally a P2P connection, which should result in the lowest latency possible. If multi-participant media streaming is required, then there are 3 different techniques which are also known from the field of data communication. The ”mesh” network, connects each participant to everyone else and requires n(n-1)/2 connections, which does not scale well. All nodes also have to pay the bandwidth cost and handle the processing of receiving all connections. Another known communication pattern is the ”star” network, where a node is selected as a ”supernode” and is the connection point for all connections. The selection of the node could be based on the session host or the nodes' processing or networking throughput when network- or resource-constrained nodes are part of the session. The last option is to use ”centralised distribution”, where a self-hosted or 3rd-party service, is deployed that can do different processing tasks. These include tasks like ”Temporal Scalability”, which can dynamically change the bitrate and resolutions of media sent to a participating node that is CPU-bound or add different post-processing effects [23, 9]. In WebRTC the two types of media servers are called Selective Forwarding Unit (SFU) and Multipoint Conferencing Unit (MCU). The SFU is, as the name applies, capable of receiving media from several participants and can ”selectively” choose which media streams to distribute to whom. The MCU is taking on the heavy processing task of being a central point that receives and mixes all media streams into a single media stream, that can be distributed. This can be seen visualised on Figure 2.20, where the MCU has as many connections in as it has out.

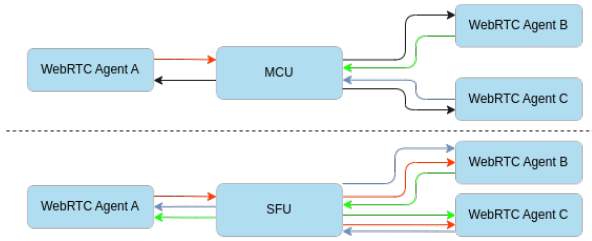


Figure 2.20: MCU and SFU WebRTC infrastructure. Inspired and corrected [48].

2.6 Measuring WebRTC media streaming quality

The networking metrics and their recommended values for VoIP discussed in Networking measurement overview (section 2.2), is a method of measuring whether the networking conditions for a video and audio WebRTC session were present. These metrics are known as important "Quality of Service" (QoS) metrics for VoIP [49]. Literature on evaluating WebRTC calls indicates that these parameters are part of the parameters that are used for assessing the objective quality of a WebRTC call [50, 51, 52]. There is another view, called the "Quality of Experience" (Quality of Experience (QoE)) which is similar to QoS defined by the International Telecommunication Union (ITU) [53] for telecommunications. It defines a collection [54] of measurements that can assess the end users perceived media quality experience for audio and video, which include mainly subjective and some objective measurements that can be conducted. The measurements use the ITU-developed rating/scoring system called Mean Opinion Score (MOS) [54]. The rating can be seen on Table 2.4 which is a rational value, typically in the range of 1 to 5, with 1 being the worst perceived quality and 5 being the best-perceived quality, which can be used in human-involved subjective rating. For non-human involved objective ratings, the ITU has created algorithmic models that are based on an objective "perceived quality" of the media that can be output as MOS. The models should, in a perfect world, match any corresponding human subjective results, which would be seen as the ground truth.

MOS	**Perceived quality**
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

Table 2.4: Scale for quality evaluation of calls used in MOS [54].

The objective models can use different acquisition methods: media packet inspection, looking at raw networking statistics (from the QoS parameters) or doing reference comparisons. The reference comparisons consist of looking at the media and quantifying the degradation compared to the original media. This requires that both the "perfect" sending signal and the lossy received signal are captured. The reference comparison uses classical statistical methods like "Mean Squared Error" (MSE) or "Peak Signal-to-Noise Ratio" (PSNR) [55]. For models based on the actual perceived quality, the "Perceptual Evaluation of Speech Quality" (PESQ) model for audio or "Perceptual Evaluation of Video Quality" (PEVQ) model for video are proposed to be used by ITU [55, 53]. The models are based on how humans experience audio and video, which are two separate research fields in themselves. However, using the subjective human-involved QoE tests and the proposed objective models for QoE proposed by ITU is both time-consuming and a costly affair. They are mainly done

for VoIP companies that look for validation, certification and benchmarking of their solution using standardised tests [56].

Efforts in developing some of the measurements and specifically running the ”reference comparisons” models for automated WebRTC testing have however started in academia. The papers [57, 58] implements an open source framework and a testbed that both uses implementations of the different QoE measurement models for automated WebRTC QoE testing. The framework built in [57] uses containerisation for software and infrastructure for enabling automated WebRTC QoE test, benchmarking and load testing with the architecture shown in Figure 2.21.

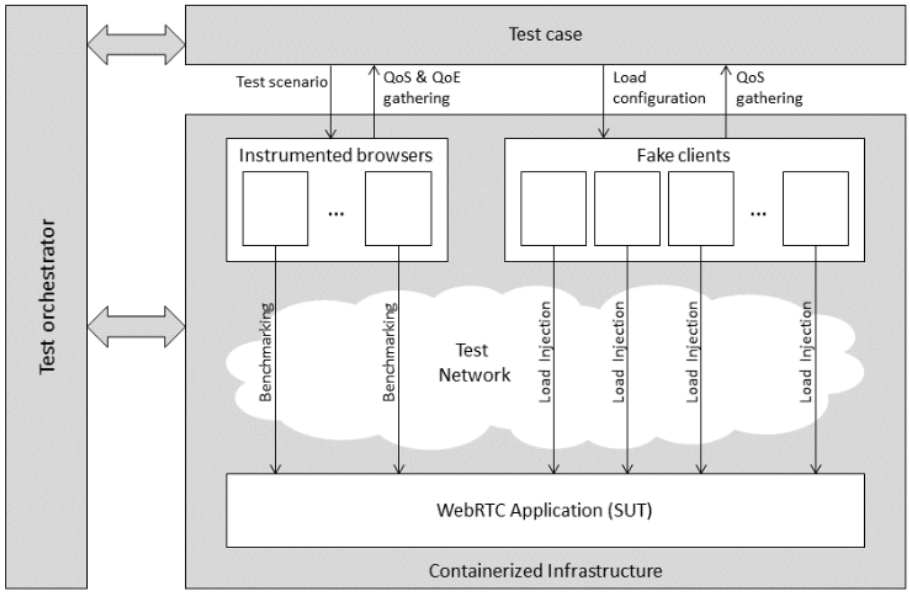


Figure 2.21: One framework architecture for doing automated WebRTC testing [58].

Another example of the pipeline of doing an automated WebRTC QoE test can be seen on Figure 2.22, where PSNR is used as one of the QoE measurement for audio and PESQ is used for video.

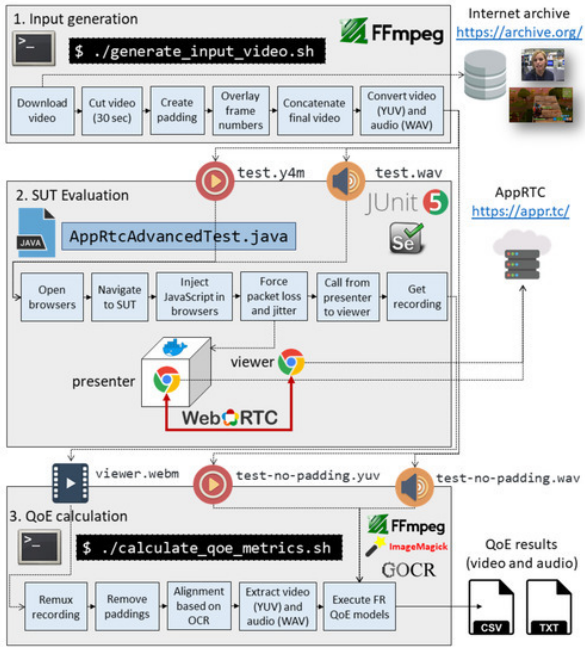


Figure 2.22: Complete ”reference comparison” framework that outputs QoE results [57].

The framework consists of a lot of software components for automation purposes and includes tools for computer vision, media recording, storage and post-processing of the received media. The received media is processed and transformed so it adheres to the evaluation required format, described for each reference comparison model that is done. To the knowledge of the authors of this thesis, there is no standardised/official way of doing QoE on WebRTC according to the WebRTC standard. Commercial WebRTC monitoring companies seem to mainly offer the QoS metrics like jitter, packet loss, and RTT. It is argued that most QoE measurements are not possible to be provided in their ”real-time” monitoring service and therefore not relevant for them to provide [59]. But one company that was identified does try to ”implement” their own QoE metric for audio, by using a combination of the QoS metrics to calculate their own MOS score [60].

2.7 Anonymisation networks

On today’s market there are many solutions which provide a proxy [61, 62, 63] or VPN as a service [64, 65, 66]. The downside of using such technologies is that in most cases they cost money and don’t end up solving the issue, it just moves the problem onto the proxy or VPN provider.

Most of the VPN providers advertise that they don't log users' traffic by having a "zero-logging" policy and that they base their operations in countries without strict data laws about logging. This is generally accepted as the major issue when using centralised privacy services or technology.

There are ,however, other technologies which try to solve this issue. "Mix network" and the specific techniques "Onion Routing" and "Garlic Routing" are some of them, which are a more decentralised way of routing packets through the internet. Mix networks work generally by mixing packets into a proxy/mix network, which is spread across the globe to make it hard to do a traffic analysis attack.

Onion routing is based on the idea of creating a network circuit. Where messages through the circuit will be encrypted and each node in the circuit will only be able to decrypt the part of the message that tells were to forward the data to. The last node will also be able to read the data exchanged between the client and the web service like an ordinary router would be able to [67]. A visualisation of a client using an onion routeing protocol can be seen in Figure 2.23

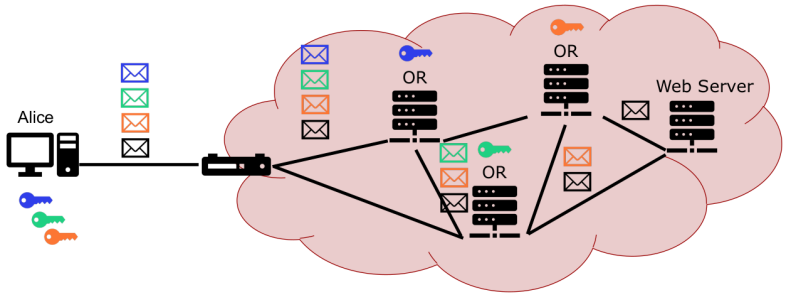


Figure 2.23: Visualisation of packet encapsulation with encryption keys distributed. Diagram from the course "02233 Network Security" at DTU.

Garlic routing is derived from onion routing, using the layered encryption from node to node, but adding mixing/bundling of packets together through the network to their destination as can be seen illustrated on Figure 2.24. This is ultimately to make traffic analysis attacks harder [68]. The red and blue clients on the far left, send packet chunks to a router that bundles several packet chunks into one packet and sends them off to another router. That router splits them up, makes a new bundle of different packet chunks and sends them off again.

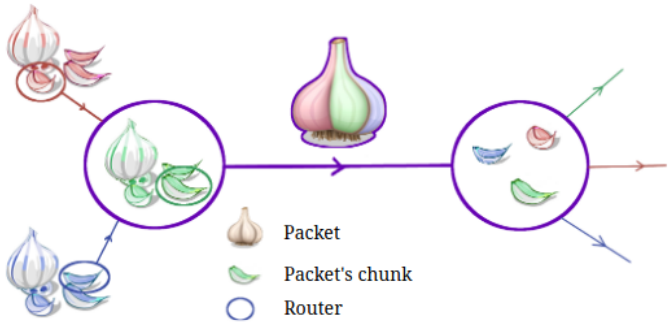


Figure 2.24: Visualisation of Garlic routing [69].

By spreading the packets out, they obtain a stronger defence against a global adversary, but risk introducing a high variation in latency (jitter) to the packet flow. This is due to the way that some of the mixing techniques work, where packets are sent to mix nodes, that wait for enough packets (batching) before the packets are forwarded. This can lead to latencies measured in hundreds of seconds, which is far from what is acceptable for VoIP and different from the ”best-effort” or fast-as-possible known from traditional TCP/ IP.

Some dedicated mix nets for VoIP, which focus on guaranteeing low latency, have been researched and implemented in academia, but none of them has seen real deployment. One of these systems is Herd [70], which uses trusted ”super peers”, that the user has approved based on the super peer's jurisdiction/location.

One of the most commonly used mix networking types is called onion routing [71] which is used by probably the most popular and used anonymisation network implementation, called ”TOR” [72, 73].

2.7.1 Clearnet vs. Darknet

When focusing on anonymisation networks are there two clear distinctions to be made. The ”clearnet” and ”darknet”. Clearnet is defined as the ”regular” internet which can be accessed from a regular browser without using an anonymisation network. Darknet is defined as web services which are only accessible through anonymisation networks.

Clients can often use the anonymisation networks to access both clearnet and darknet web services, both will be as an anonymous client. If the client provides any private information to the site i.e. logging in or similar to the service, the client is no longer anonymous to the service, but the client's ISP can not see which service is being accessed. Web services which are only accessible through the darknet/anonymisation network are often referred to as hidden services.

2.7.2 Tor

TOR is a project based on onion routing which were first described in 1996 [67], further formalised in 1998 [71]. The paper describing the TOR network as it's mostly know today were introduced in 2004 [72]. The TOR project [73] has since continued to evolve and kept updating its service. TOR is the most widespread and popular anonymisation network. The network currently only supports TCP.

The circuits built in TOR typically consists of 3 hops. The messages sent through the circuit is in each layer encrypted with AES [74]. This means that the first two nodes, guard/entry- and middle- node, can't read the content of the message. The exit node can read the data since it needs to know where to send the data. Typically this is encrypted with TLS or similar when browsing the web, in this case, the exit node only sees that a user is communicating with said IP like shown in Figure 2.25.

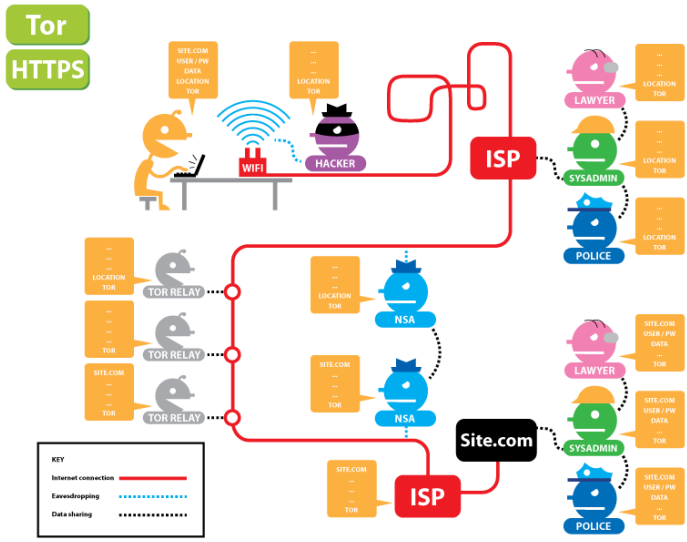


Figure 2.25: Visualisation of a TOR circuit and who can see which data on the route when the client is accessing a clearnet site over https through TOR [75].

TOR will generally recreate circuits every 10 min, but if the connection is long lived it will keep the circuit as long as the life time of the connection [73].

TOR is a partly centralised network. Most of the communication is peer-to-peer but the network relies on directory authorities, which is hosted in a few countries around the world [76]. These servers host a list of currently running relays and periodically publishes a consensus together with the other directory authorities [73].

There are multiple ways to install a TOR client. The most common one is to use the TOR browser or directly run a TOR service on a Linux machine. The service installation allows the user to configure many options in the "torrc" file. One of these options is which countries the client should be allowed to connect through for

both, entry-, middle- and exit-nodes [73]. Another way to interact with the client is a service port which the client can choose to open. This will allow other programs to connect and interact with the TOR service running locally. There are build tools which make integration with the TOR client easy, such as the Stem project [77] that can both be used for controlling and monitoring the state of the service.

2.7.3 I2P

Another low latency mix network option is the open-source project "Invisible Internet Project (I2P)"[5] which has been under constant development since 2003, based on garlic routing instead of onion routing. I2P is a network-layer technology working with IP packets, which supports both UDP and TCP-based applications. Common use cases which are officially supported are web browsing, file sharing, IRC chatting, and instant messaging [78].

Unlike Tor, which uses bi-directional circuits for communication over a circuit-switched network, I2P uses uni-directional tunnels established between clients, who all act as "routing nodes" as seen on Figure 2.26 in a packet-switched network [68, 79]. The default configuration is 6 hops/tunnels between sender and receiver nodes, which is quite a lot, but is configurable by the client. As with any network, the bandwidth of a network path is limited to the link with the lowest bandwidth available. So with 6 different nodes that the packet should traverse, the risk of one hop limiting the overall bandwidth is quite high. As per I2P, nodes are categorised based on how much bandwidth they share. Which is typically between 20-256 KB/s [79, 80] per tunnel.

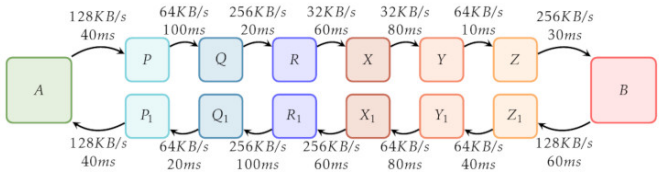


Figure 2.26: Visualisation of the uni-directional tunnels, where client A have the inbound and outbound tunnels between nodes (PQR and $P_1Q_1R_1$) [81].

This is a different strategy compared to the before mentioned anonymisation networks, since using the I2P network requires you to be part of it and contribute. From a user's perspective, this means that if they want usable performance from their I2P connection, they need a dedicated server (potentially do NAT configuration) at home or rent a VPS that is online and connected 24/7 as an I2P router. Then the clients can set up local port-forwarding on their machine to redirect the traffic through their

[5] Several different client implementations of I2P exist, but the first implementation in Java and most popular is from geti2p.net

I2P router and access the I2P network. This is not very user-friendly compared to the TorBrowser bundle, provided by the Tor project, where clients don't have to contribute to the network [68]. Like Tor, I2P has "hidden services" called I2P sites that end with ".i2p". However, unlike Tor, I2P's main focus is on providing and optimising access to these hidden services hosted in I2P [79]. Unofficial "outproxies" that enable accessing the clearnet from I2P, are ,however, offered by 3rd party organisations [82]. According to non-formal initial testing done in this project and from an older academic comparison [83], the connection to the outproxies in I2P, were at the time barely acceptable for web browsing and not as capable as using the Tor Browser. I2P, however, support both HTTP, SOCKS, and SOCKS5 proxies [84, 69] for connecting and using the I2P network.

The I2P project is said to offer a more decentralised way of running a mix network, compared to Tor, since they don't have central directory infrastructure [68, 85, 69]. Instead, I2P is like Bittorrent based on Distributed Hash Table (DHT), which record and share information about the network and routers in a decentralised distributed manner. Like with torrents, some bootstrapping is however required when clients want to join the network. This is for ease of use bundled into the I2P software by default, or can be obtained from volunteers'/different sources distributed on the clearnet [78].

The I2P project is supposed to be designed and built to deal with the privacy attacks and design limitations discovered in Tor by academic work [81]. One of them is against the global adversary doing traffic analysis, who can deploy I2P routers themselves and monitor traffic at ISP's. I2P makes this more difficult by "officially" only keeping traffic inside the network where it is always end-to-end encrypted and mixed with other node's traffic. This technique should, theoretically, become stronger when the amount of nodes and traffic in the I2P network becomes bigger and approaches closer to a "constant rate" of traffic. Another known attack on systems with decentralised components for decision-making is the Sybil attack [85, 86]. The Sybil attack is possible in systems where it is "easy"/non-computationally hard to create multiple identities that can participate in some decision-making process, and change the outcome of the decision. Due to the decentralised nature of I2P, this attack can be highly useful in Denial of Service attacks and generally for advertising wrong network statistics for participating nodes [81, 85].

The I2P project user-base, developer-base, and financial backing are not as big as Tor's. This means that they are not offering many official development or self-monitoring tools like Tor, while also lacking in both developer and user-accessible documentation [68, 81]. Information on the I2P website [81] is also scarce and sometime contradictory, which could be a sign of an immature and under-development project.

2.7.4 Lokinet (LLARP)

Lokinet is an implementation of Low Latency Anonymous Routing Protocol (LLARP) which can be used to build a decentralised anonymisation network inspired by TOR and I2P. The big difference between Lokinet and the two others is in the decentralised part. Where i.e. TOR uses a small range of directory authorities to keep track of the running relays, Lokinet relies on a DHT built and maintained on the blockchain [87].

The purpose of the blockchain (initially called Loki and later moved to OXEN) is firstly to make the DHT publicly available with information on the services in the network in a decentralised manner. A second perk of using blockchain technology is the market-based resistance to Sybil attacks. This is achieved by enforcing that in order to host a service node, like a router or an application, it is required to stake/time-lock 15.000 $ in the cryptocurrency "OXEN", at the time of writing [87, 88, 89].

Another big difference between TOR and Lokinet is that LLARP is designed to work on the networking layer of the osi (Open Systems Interconnection) model and therefor it supports both TCP and UDP [87].

Up until 2020, Lokinet were strictly an anonymisation network which could only access its own hidden services called snapp (Service Node Applcations) [90].

With the introduction of exit, nodes it became possible for Lokinet clients to start browsing the clearnet as an anonymous client. With the concept of onion routing in mind, see figure Figure 2.27 for a visualisation of paths in Lokinet. The client Alice connects to the edge node, and for as many transit nodes as Alice chooses the path continues until it reaches the pivot node which connects to the exit node which then contacts the clearnet web service.

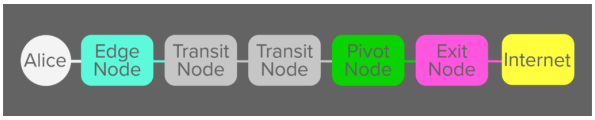


Figure 2.27: A typical "path" and the naming of the different nodes in Lokinet [Link for Lokinet "path" terminology].

One of the first snapp's (Service Node Applcations) which were built for Lokinet were a messaging application initially called Loki Messenger [87]. The application has since evolved/rebranded to Session [91, 92]. The application uses Lokinet to send messages between clients, which does not contain/leak metadata about the clients. The feature to support audio and video calls are currently in beta and in their FAQ, they mention that the calls use WebRTC, STUN, TURN and go through the clearnet. This means that session currently doesn't support anonymous real-time calls through Lokinet.

The Lokinet client supports many different configuration options. In order to exit Lokinet it is required to specify an exit node in the configuration file [93].

It was not possible to find any concrete performance claims about Lokinet, so as part of the research phase of the project, a list of available ”exit node” addresses and their geographical location was found [94][6]. And tested using fast.com for doing an internet speedtest. One of the US exit nodes that were tested, can be seen on Figure 2.28.

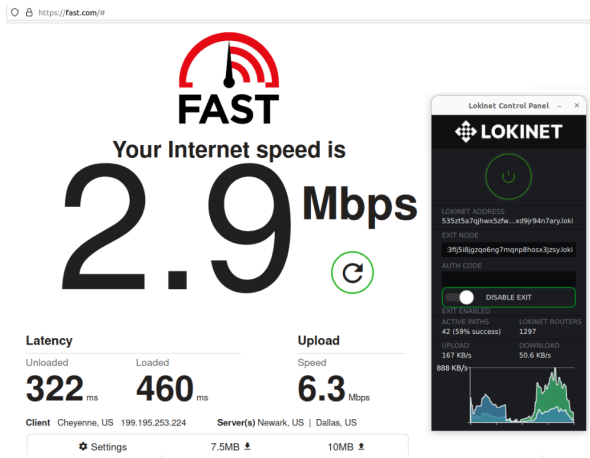


Figure 2.28: Speedtest from Copenhagen using the ”exit.loki” exit node located in USA. Screenshot from the 27/10/2022.

2.7.5 Other honourable mentions

Plenty of different anonymisation networks and protocols have been designed and evaluated over the years [95]. Few of them with a narrow use case with just support for ”file-sharing” or ”instant text messaging” in mind. Tor and I2P, are the biggest deployed networks to the knowledge of the authors. However, the 3 networks shown in Table 2.5 were repeatedly mentioned in the research surrounding anonymisation networks, but are currently not in a state where they can be considered stable or widely used.

2.8 Prior work

The literature study for the project was done to investigate the following three issues/questions: *What have previous studies done, when looking at the feasibility of doing VoIP and WebRTC over anonymisation networks?*, *What suggested solutions*

[6]This is a ”darknet hidden service site” only accessible through the Lokinet network. A screenshot of the site is provided in Appendix B.

Name	Project status	Project purpose	Layer
GNUnet [96]	Under development since 2001	Tries to replace IP and provides hidden services over P2P network.	Network
Freenet [97]	Under development since 2000	Only for anonymous filesharing.	Network
Zeronet [98]	Under development since 2015	Only for anonymous filesharing.	Network

Table 2.5: Anonymisation networks

have been proposed and evaluated in academia? and lastly, *To what degree has WebRTC quality and performance automation testing been done?*

2.8.1 Testing the feasibility of VoIP and WebRTC over anonymisation networks

The Tor project, which is one of the most used and researched anonymisation networks in the world, is made for bi-directional "low-latency" traffic like HTTP traffic and allows for smaller bursts of traffic [72]. It does not claim to support high-bandwidth or throughput applications like media streaming [99]. But because of the user-base and big interest from researchers, most papers that have been identified during the literature review, relate to doing VoIP or other high-bandwidth and high-throughput traffic for the anonymisation network TOR. This is despite the performance claims and the main purpose and design of TOR.

One of the older non-academic projects that were identified was the "TorFone" project [100], which uses TOR for the Transport layer. But it has built its own protocol for routing VoIP traffic and has its own "obfuscation, encryption and authentication" schemes implemented on top of what TOR already provides [100, 101]. The project tried to circumvent the shortcomings of TOR's performance limits, by using selected TOR nodes and duplicate VoIP traffic out on two TOR circuits, instead of just using one circuit. This strategy of selecting and maximising the chance of using high-network performing nodes for circuits and using more "paths" through the network for redundancy, will be seen repeated/popular throughout most "VoIP over Tor" projects. Since the project seems abandoned, information about the performance is scarce, but the owner of the project claimed that a call only required 2 Kbit/s of bandwidth for its low-bitrate audio codec. Furthermore, they state that the system provided up to 2-4 seconds of voice latency, which also seems far from acceptable for today's standard.

The paper "*Enhancing Tor Performance For Bandwidth-Intensive Applications*" [99] propose and evaluate a similar strategy of splitting traffic out on more relays and using the client and an extra "Angel Exit Relay" to gather/merge the traffic and

packets back together. The concept can be seen on Figure 2.29. It is similar to the mix anonymisation network strategy, but instead of being intended for making traffic analysis harder, it is for achieving higher throughput. The Angel exit relays, should be for ”special purpose” traffic like bandwidth-intensive traffic and should be hosted in datacenters with high networking and computing capacity. The proposed design is not supposed to require changes made to the existing TOR relays, but should be implemented on the client side and the Tor relay software running on the specially deployed angel exit relays. The paper shows that the design decreases the variance of the throughput of the created circuits. But leaves on the table the discussion about the performance hit and implication to the TOR network, by using this strategy if it was ever deployed.

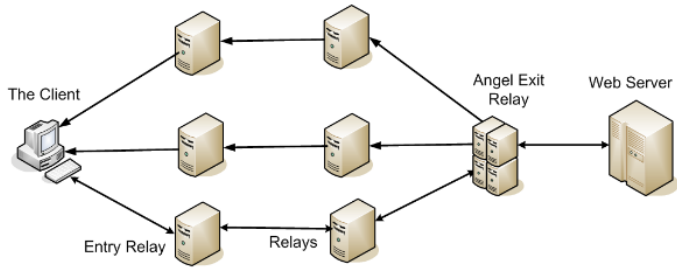


Figure 2.29: Proposed ”Angel relay” TOR circuit redesign [99].

The idea of using the existing Tor network, but splitting up the traffic at the source and merging the traffic at the destination, is also the idea of the paper *”Donar: Anonymous VoIP over Tor”* from 2022 [101]. An overview of the architecture can be seen on Figure 2.30. The project uses TOR specific .onion service addresses to connect the caller and callee, and takes advantage of the security and end-to-end encryption that Onion services provide. The amount of hops in the circuit is configurable from 3 down to 1 hop, which offers a security/performance trade-off. The system, which is open-source, is evaluated under the different QoS parameters with acceptable values, but with no ”future work” or any plan to continue the work described.

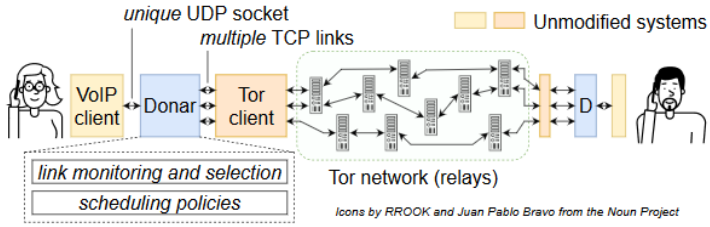


Figure 2.30: Donar allow connecting any VoIP client to a proxy [101].

A less technical solution that is available today could be using applications that provide a proxy interface, which allows VoIP applications to have its TCP traffic tunnelled over TOR. This approach has been tried by a privacy-oriented project called the "Guardian Project". They wrote a blog post back in 2012 [102], which is showing a method of doing an audio call over Skype using TOR. They were using an app called Orbot, which enables Tor to be used as a proxy on Android for any application. They managed to set up an audio call that was described as "surprisingly usable" over Skype using two mobile phone skype clients, where one client was connected over Tor. They argue that when Tor at some point, starts to support UDP (which seems to be ongoing at the moment [103, 104]), then VoIP over TOR will be a very popular future research area, to look into.

Another trend that was found in academic research surrounding VoIP over Tor, was the idea of using a VPN like OpenVPN, to do encapsulation and end-to-end encryption of unencrypted RTP traffic, like it can be seen on Figure 2.31. This was done in the paper "*Empirical Performance Analysis of Anonymizing VoIP over The Onion Router (TOR)Network*" from 2013 [105], where they also tested the impact of only using 2 relays instead of the default 3. The 2 relays were selected from a smaller subset of high-bandwidth European nodes only (177 out of 3003 total), which was gathered based on data from an older, unofficial open-source Tor monitoring service called "TORstatus" [106]. In the experiment, they use an audio codec called "Internet Low Bitrate Codec", which is a narrowband speech codec, with a low bitrate of 13.33-15.2 Kbsp. Their results, which consist of QoS parameters show that for the 3 relay setup, they achieved an acceptable quality of 21% of calls with 1% packet loss and 36.4% calls with 5% packet loss and acceptable jitter below 30 ms. For the 2 relay setup, they achieved an acceptable quality of 36.84% of the calls for 1% packet loss and 54.74% for 5% packet loss. The scenarios had between 93.8-137.5 ms latency for 1% packet loss and 122.9-157 ms of latency for 5% packet loss. They conclude that removing another hop, improves the performance, but at only around 50% acceptable calls, VoIP over TOR leaves a lot of performance to be desired. The paper ends by stating, that as the TOR network grows in bandwidth, users and relays it might be possible at some point.

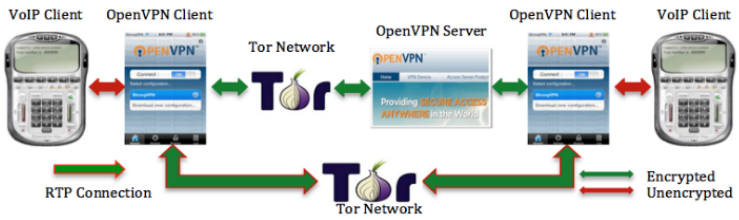


Figure 2.31: Design of "VoIP over OpenVPN over TOR network" experiment [105].

Another project, that takes it a step further and includes both QoS and QoE parameters for performance evaluation of running VoIP over Tor, was the master thesis *"ASTERISK VOIP OVER TOR"* from 2019 [107]. The project also used OpenVPN for encapsulation, but included their own implementation of a "software-phone" that they used for automation of some of their VoIP SIP calls. The calls were captured and analysed using Wireshark for QoS parameters. They also used the industry VoIP load testing tool "StarTrinity SIP Tester" [108], which can automate running VoIP calls and analyse the call quality by using the ITU industry standardised QoS and QoE metrics. An example of quality metrics gathered can be seen on Figure 2.32. The project looked at doing VoIP with RTP directly between two peers and with a VoIP server in-between, which did double duty as a VPN server. The project analysed the security models for the different setups and the difference between doing end-to-end encryption of respectively both SIP, RTP and the combination of both. The project concluded, based on the data from their experiment, that most of the calls were acceptable according to the QoS and QoE parameters and that it is possible to do VoIP over TOR. They propose two future work points: Doing more QoS VoIP performance tests on other anonymisation networks and looking into the possibilities of creating a anonymisation network specifically for VoIP networking.

Measurement started at: 2/28/2015 10:29:55 PM
Measurement duration: 0d 0h 4m 2s

SIP call quality indicators

Quality indicator name	Ncalls	Min	Average	Max	Percentile: 90%	95%	98%	99%	99.5%	99.8%	99.9%	99.95%	99.98%	99.99%
Caller lost packets (%)	24	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Caller G.107 MOS	24	4.41	4.41	4.41	4.41	4.41	4.41	4.41	4.41	4.41	4.41	4.41	4.41	4.41
Caller G.107 R-factor	24	93.20	93.20	93.20	93.20	93.20	93.20	93.20	93.20	93.20	93.20	93.20	93.20	93.20
Caller max delta (ms)	24	21.08	24.77	34.77	33.77	34.36	34.77	34.77	34.77	34.77	34.77	34.77	34.77	34.77
Caller max RFC3550 jitter (ms)	24	0.27	2.16	8.12	7.62	7.72	8.12	8.12	8.12	8.12	8.12	8.12	8.12	8.12
Caller mean RFC3550 jitter (ms)	24	0.09	1.25	6.20	3.99	6.20	6.20	6.20	6.20	6.20	6.20	6.20	6.20	6.20
Called lost packets (%)	24	0.00	0.02	0.20	0.10	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Called G.107 MOS	24	4.39	4.41	4.41	4.40	4.39	4.39	4.39	4.39	4.39	4.39	4.39	4.39	4.39
Called G.107 R-factor	24	92.46	93.12	93.20	92.83	92.46	92.46	92.46	92.46	92.46	92.46	92.46	92.46	92.46
Called max delta (ms)	24	32.99	48.89	78.86	75.81	78.78	78.86	78.86	78.86	78.86	78.86	78.86	78.86	78.86
Called max RFC3550 jitter (ms)	24	6.75	8.38	11.46	10.67	11.14	11.46	11.46	11.46	11.46	11.46	11.46	11.46	11.46
Called mean RFC3550 jitter (ms)	24	5.96	6.12	6.20	6.19	6.19	6.20	6.20	6.20	6.20	6.20	6.20	6.20	6.20
100 response delay (ms)	24	151.00	153.58	187.00	153.00	154.00	187.00	187.00	187.00	187.00	187.00	187.00	187.00	187.00
Answer delay (ms)	24	437.00	453.54	500.00	485.00	487.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00
-24dB delay (ms)	24	959.26	987.14	1063.69	1019.21	1030.06	1063.69	1063.69	1063.69	1063.69	1063.69	1063.69	1063.69	1063.69
RTCP RTT (ms)	24	147.00	153.25	176.00	159.00	165.00	176.00	176.00	176.00	176.00	176.00	176.00	176.00	176.00
RTCP caller lost packets (%)	24	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RTCP caller max jitter (ms)	24	7.00	10.22	14.00	13.00	13.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00	14.00
RTCP called lost packets (%)	24	0.00	0.02	0.20	0.10	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
RTCP called max jitter (ms)	24	6.64	8.14	11.13	10.09	10.65	11.13	11.13	11.13	11.13	11.13	11.13	11.13	11.13
Media threads delay (ms)	24	0.40	4.90	16.96	15.63	15.65	16.96	16.96	16.96	16.96	16.96	16.96	16.96	16.96
Signaling thread delay (ms)	24	0.00	0.00	0.01	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
GUI thread delay (ms)	24	0.06	67.72	490.90	177.50	328.61	490.90	490.90	490.90	490.90	490.90	490.90	490.90	490.90

Figure 2.32: Startrinity SIP Tester Quality report of a test call [108].

All experiments done in the previous papers, are done over a smaller time period, using a specific selection of nodes or are done at a smaller scale, which can raise the question; can the results really be used as empirical evidence or is it just the result of the TOR's networking performance at the time. That is the question that the paper "The Road Not Taken: Re-thinking the Feasibility of Voice Calling Over Tor" from 2020 tried to answer [109]. They conduct both in-lab and real-world experiments on TOR, with over 500.000 voice calls done over a year using different Tor relays, Tor bridges, VoIP applications, audio codecs, and calls hosted in different locations. They recorded both QoS metrics and QoE using the PESQ model, including some

user studies with 20 people, where the users should manually rate recorded calls using the MOS system, with $< 8\%$ of the PESQ scores being rated unacceptable. The results that are presented show an acceptable score from the PESQ model and an acceptable latency. They argue that PESQ model captures the network delays, jitter and packet losses, and is why they don't include those QoS metrics for evaluation. The conclusion that the authors make, is that they are the first "relevant" study that shows TOR to be suitable to make VoIP calls.

Lastly, the paper "*Evaluation of Anonymous Streaming Possibility in Tor Network*" [110] is about trying to determine if a user is video streaming from a hidden service, where all traffic is end-to-end encrypted. The paper is not focused on whether or not the bandwidth requirements are present for streaming video, but rather if the end-to-end encrypted traffic between a client and a hidden service over Tor, hides/obfuscates the signature of streaming traffic. They are not looking at bandwidth, but rather looking at "the length of TCP segments" over time. They conclude that the fingerprint Tor streaming traffic is similar to traffic on a non-anonymous network (as in there is no masking/obfuscation of the traffic pattern).

2.8.2 Papers that try to implement anonymisation networks that can support VoIP

It has been established that academia has proven (with one exception) that no currently deployed and popular anonymisation networks out of the box support satisfy the VoIP requirements, without using different configurations and strategies. This subsection will look into the academic work which has been done, trying to implement anonymisation networks that are designed for supporting real-time requirements of modern VoIP and WebRTC traffic.

The paper "*Empirical tests of anonymous voice over IP*" [111] from 2010 proposes an anonymous VoIP (aVoIP) solution like on Figure 2.33, which is more of a theoretical redesign of a "Tor-like" or Onion routing setup. They propose the following: the call initiator can pick their own hops from a given list, use a lower amount of hops from TOR's default 3 down to 2 and that the hops should be on the same continent. They create an experiment with their own medium-sized Onion-"like" network consisting of application layer proxies, distributed in Asia, America and Europe. In the experiment, they don't send VoIP traffic, but instead gather QoS metrics (packet loss, latency, jitter) and the ITU-defined "R-score". This is done based on traffic generated from a tool that supposedly "mimics" UDP VoIP traffic. Their results are used to conclude that by utilising UDP and lowering their security "level" by using fewer proxies, anonymous VoIP is possible.

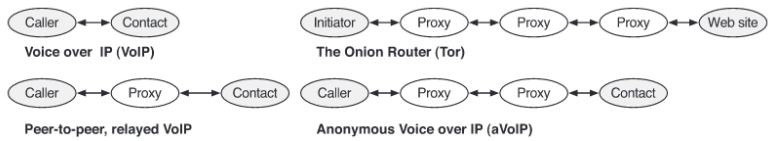


Figure 2.33: The different amount of proxies used in VoIP, relayed VoIP, Tor and aVoIP [111].

There are dedicated aVoIP networks, that have been created and evaluated in academic research, but are of limited use, since deploying a global-scale anonymisation network driven by volunteers is an exceedingly huge task, that only a few anonymisation networks have succeeded in. Mentioned in the other aVoIP papers is the network Herd [70], which is designed from the start to support aVoIP, but is said to not be at a sufficient scale of deployment yet [70, 101]. Herd is a mixing network, that tries to guarantee a low end-to-end delay, latency, and jitter, which is crucial for VoIP networks. Herd, like Tor depends on the notion of trusted infrastructure but uses P2P super-peers for scale-ability that are located in different jurisdictions/countries that the user trust. The authors have evaluated the system in a smaller test experiment and have verified the performance as suitable for VoIP traffic. A newer aVoIP network, that focuses on "metadata-private" voice calls is Yodel [112, 101] which is also a mixnet. It does not achieve acceptable latency for conversational VoIP, but tries instead to design a system that is resilient to both active and passive global adversaries, controlling most of the network.

2.8.3 Automation testing, performance measurements of WebRTC and QoE of media streaming

Both in academia (as covered in "Measuring WebRTC media streaming quality (section 2.6)") and in the industry [113, 59], work has been done in creating automation tools that can be used for WebRTC applications and testing. This includes WebRTC application load testing, monitoring, network testing, and quality analysis, which is important when building a WebRTC streaming application/platform. But when testing actual WebRTC implementations for functionality, API conformance, or low-level components like different codecs or the underlying congestion control algorithm, more specialised automation testing tools are needed. One of the high-level WebRTC benchmarking tools that have been identified is the project "WebRTCBench" [114], which allows for doing a quantitative comparison between WebRTC implementations across browsers, devices and operating systems. An example of a more "low-level testing and evaluation" of different networking impairments effects on QoE parameters, the tool "WebRTC-Analyzer" and the accompanying paper *"Performance Evaluation of WebRTC-based Video Conferencing"* [52] was found. The paper proves the usefulness of the framework by conducting a plethora of different experiments and tests, using the framework. But the most interesting one is looking at how bandwidth,

latency, and packet loss affect the WebRTC congestion control algorithm that is responsible for doing "Adaptive Bitrate Streaming". The experiment setup can be seen on Figure 2.34, where one of the nodes uses a "Network limiter" that enables creating different networking conditions and scenarios. The network limiter is the tool "Dummynet" [115], which can do traffic shaping, do packet scheduling, and emulate any network link.

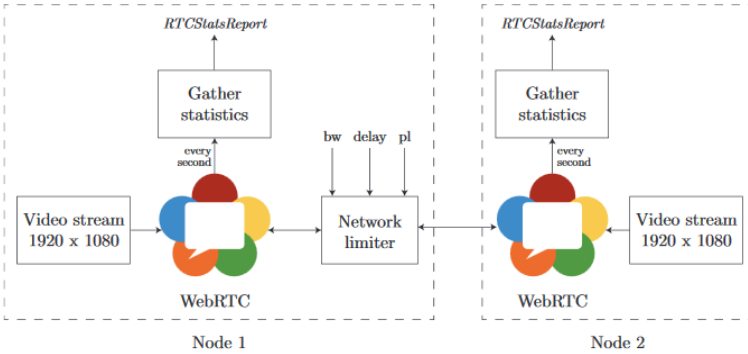


Figure 2.34: Experiment setup of using the "WebRTC-Analyzer" test framework and a network limiter, where they can adjust bandwidth, delays and packet loss [52].

2.8.4 Privacy related problems with TURN protocol/infrastructure

In "RFC 8826 - Security Considerations for WebRTC", the problem of caller and callee IPs being known by the TURN servers is mentioned and a solution is proposed, which is to use "privacy-oriented systems such as TOR" to mask the IP [116, Section 4.2.4].

Looking specifically at WebRTC it is possible to find privacy leaking problems in WebRTC applications like Signal [117] and in-browser specific problems using the ICE protocol. For these problems, solutions have already been proposed [118, 119] and tested. This is, however, not relevant to the privacy problem that this thesis is trying to solve. But are important to mention, since they are partly responsible for the TOR browser, not currently supporting WebRTC [120].

The last "miscellaneous" project that was identifed, is the research project "Using TURN servers as proxies" [121] which investigates an alternative use of TURN servers. Instead of using TURN servers for relaying WebRTC traffic, they instead wanted to relay any TCP/UDP traffic from the Turn server - effectively using available TURN servers as general-purpose proxies. The project's purpose was to build a proof-of-concept Socks proxy server that could connect to a TURN server on the behalf of a client. The TURN server could then connect to a destination host specified by the

proxy server and send UDP and TCP traffic. They did an investigation and found that 5 out of 10 video conferencing companies that hosted TURN servers could be used for proxying UDP traffic. According to the author of the project, TURN TCP Relay (RFC6062 [33]) is seldom used, whereas UDP is normally allowed. The project was inspired by a vulnerability/misconfiguration discovered in Slack's WebRTC cloud infrastructure. The discovery allowed attackers to use Slack's TURN servers as proxy servers to access internal networking addresses and resources on Slack's AWS (Amazon Web Services) infrastructure [122]. This problem is also not directly relevant to the privacy problem of this thesis, but shows another use-case for the TURN protocol.

2.9 Possible attacks

This section will cover two different attacks that are currently possible in regards to how most modern solutions work, where a TURN service provider is used.

2.9.1 Social network graph

The traditional classification of streaming voice and video traffic has previously been done in academia, where established ”simple” recognition methods like looking at packet headers have been used [123]. But with the robust and required encryption method of DTLS used in WebRTC, this seems not to be a trivial task anymore, as less of the information is available for simpler pattern matching. This has led academia to look into the use of machine learning algorithms, where models are trained to do more sophisticated network classification and can tell WebRTC traffic from normal traffic [123]. From the bandwidth used the amount of traffic and the packet sizes, it is possible to determine what kind of media is being exchanged and in which direction it flows [110]. Then in a thought-up scenario, where all popular instant messaging services in the world, that provide audio and video calling, use WebRTC. Then a TURN service provider could start gathering data for a social networking graph like the one shown on Figure 2.35. Here, each node contains information about an identity with multiple online ”profiles” related to a specific service. The edges between the nodes, contain registered communication and information related to the communication between nodes.

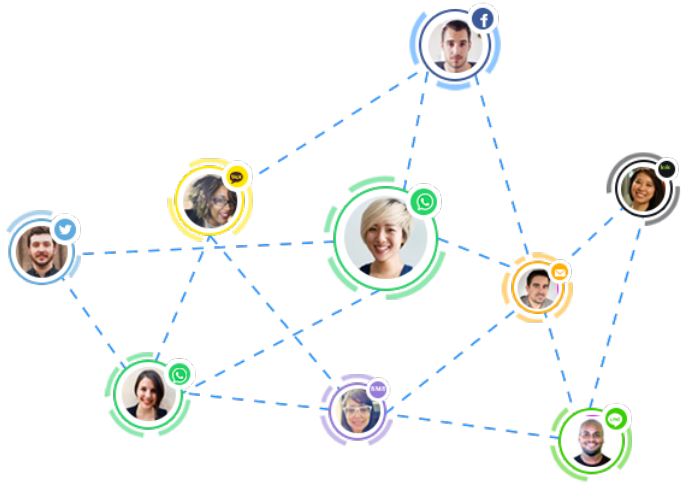


Figure 2.35: Social graph example [124].

Doing this kind of social graph could be a useful resource for people doing forensics and intelligence collection and analysis for an investigation of a group of individuals.

Potentially, the data could end up going into the OSINT (Open-source intelligence) tool Maltego as a paid data source [125].

2.9.2 Supply chain attack

As described in Appendix A "*Real-life usecase: BifrostConnect*", the company BifrostConnect uses WebRTC to provide physical remote access/screen-sharing through their product. They have customers that are asking about the risk of supply chain attacks. The attack has to be considered and included in their security risk assessments, which are required when implementing cybersecurity standards like IEC-62443 [126]. A supply chain attack is done by targeting typically a large enterprise, by locating a supplier or a partner of the company, which has weaker security requirements or is simply an easier target to compromise.

An example of a supply chain attack on BifrostConnect customers could be if the WebRTC solution was deployed at a customer's office, industrial site or production facility, with tight security requirements. Then it could be the case that the company also has chosen to outsource part of its IT service. This could be their IT-support, which is outsourced to a 3rd party IT-support center or they could have a need for enabling remote support from their production-equipment supplier to on-site equipment. If the traffic pattern of the Bifrost device and the WebRTC traffic, could be identified, then attackers could target either BifrostConnect itself, BifrostConnect's suppliers, the IT-support center, or the production equipment supplier using the Bifrost for remote access.

2.10 Summary

This chapter has covered the background knowledge around networking measurements, internet media streaming, WebRTC, measuring WebRTC media quality and given an overview of anonymisation network technologoies. Afterwards, a prior work section, went through both academic and non-academic projects that looked at the feasibility of running aVoIP over Tor. Most papers concluded that to be able to obtain good enough bandwidth, latency and jitter, a redesign or at least configuration changes to Tor was needed. Other researchers decided to design and evaluate dedicated aVoIP networks, that were made for satisfying VoIP requirements, while also trying to increase the security compared to TOR. Then, an academic look at WebRTC automation test/evaluation tools, where two specialised test frameworks were presented from papers. Finally, a short recognition of privacy-related problems that have occurred in using WebRTC. Lastly, two possible high-level attacks were presented, one of which has been brought up in the company case from BifrostConnect.

CHAPTER 3
Project Definition

This chapter will define what the project is about, which research questions the project is trying to answer and at last for scoping the project, some delimitation will be presented.

3.1 Problem Statement

In a modern connected world where real-time communication is running over IP instead of the regular telephone network, requirements for both bandwidth, performance and privacy have increased. This is due to the ever-increasing interest in collecting data and the open nature of the internet, where information flows through a lot more different service providers and infrastructure that can collect different types of metadata.

Specifically, the video and audio chat applications using WebRTC can incorporate specific WebRTC infrastructure that can gather metadata. These services include media servers for additional video processing and media distribution and more crucial servers for NAT traversal using STUN or TURN servers.

This project aims to explore WebRTC and if it is possible to use modern anonymisation networks to hide the origin client IP address from the TURN server, while doing a WebRTC session with video and audio. The anonymisation network should have adequate networking QoS for performing the WebRTC session. With this in mind, the following research questions have been defined:

- *Verify that it's possible to use anonymisation networks to make WebRTC client(s) appear anonymous from the point of view of a TURN server.*
- Formalise a software test framework/suite, that is able to setup a WebRTC session between two clients and collect performance and networking statistics.
- Test, measure, analyse and validate the performance of anonymised real-life client applications using TURN services.

3.2 Delimitations

For the results of this project to be relevant and realistic to a real-life WebRTC video and audio application deployment, the project will not go too deep into different RTP-specific - and WebRTC-specific configurations. This is to make sure that the results are not entirely based on, an anonymisation network ”optimised” configuration for WebRTC. Instead, the project will rely on the browser specific WebRTC implementation and let the audio and video engines choose the right codecs to allow any dynamic configuration changes. This will be valid as long as the changes are within the project's definition of ”usable” for audio and video consumption.

For evaluating the usability of the video and audio quality of a session, the project uses a combination of different recommendations for the typical network statistics like latency, jitter and throughput. This is to keep the testing framework simple with minimal extra client processing for evaluating a client's ”view” of how it ”experienced” the call, also known as QoE.

For the different anonymisation network selection strategies, the ones that require protocol changes or the ones that could lead to compromises in privacy for extra performance gains will not be considered. Instead, the project will navigate within the different configuration spaces, defined and recommended by the anonymisation network projects themself.

CHAPTER 4
Analysis and Design

This chapter will describe the research, analysis, and design of an experiment and a software framework, that will be used to evaluate if anonymisation networks can be used with WebRTC and TURN servers.

4.1 Analysis

As defined in the "Problem Statement (section 3.1)", the project should: "Test, measure, analyse and validate the performance of anonymised real-life client applications using TURN services.". The important keywords are **analyse**, **measure**, and **validate** the performance. The first theoretical analysis part has been done in the theory section and in the related work section, where both the performance metrics and numbers of anonymisation networks and the compatibility for using them with WebRTC have been identified. The last part of the analysis will be presented in this section, where details of the experiment and the purpose-built framework will be considered.

4.1.1 Anonymisation network to be tested

From chapter 2 multiple anonymisation networks were mentioned, some are more theoretical than others, some of them are deployed and others are not. The thesis will use anonymisation networks which are mature enough to be used by clients and has a connection interface that support WebRTC. Despite what other papers have done, this project will not make changes to the protocols or implementations, but instead, use any provided proxy application or any provided instructions for setting up a connection. This narrowed the list down to the three networks; TOR, I2P and Lokinet, which are all different anonymisation networks, with their own purpose as described in the background section.

4.1.2 Anonymisation network configuration strategies

During the related work research, there have not been identified many relevant configuration parameters/options for configuring Lokinet, that should improve the per-

formance. The only parameter that would have an impact, is the selection of the exit node to use. However, there have been proposed many alternatives to improve the performance of the anonymisation network TOR. Some of the changes are, as discussed previously, more theoretical and others are implemented into the TOR clients as configuration options. Amongst these are the amount of hops, which in TOR is defaulted to three hops, which furthermore is limited due to a policy applied in many dedicated TOR exit nodes, that prohibits them from being used in a circuit as an entry node, effectively preventing them to be used for "1-hop" circuits. It is important to note, that using 3 hops is the minimum requirement to achieve anonymisation using TOR, so the usage of fewer hops is at the cost of the possibility of losing some of the anonymisation and is not recommended [127]. A visualisation of the choice between quality and security when using and configuring TOR can be seen on Figure 4.1.

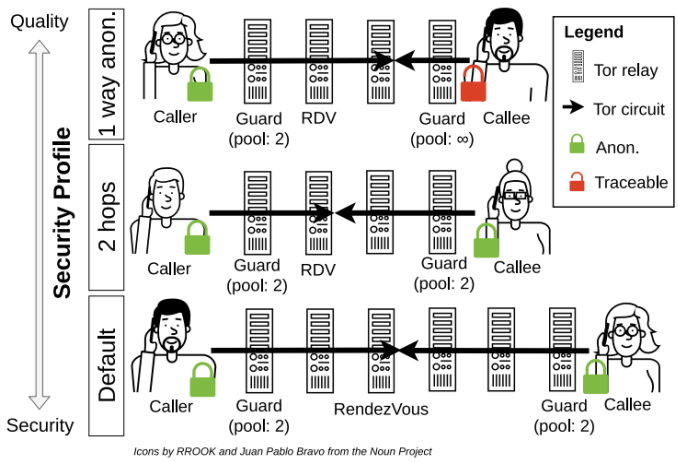


Figure 4.1: 3 different examples of TOR client configurations, having different levels of Quality vs Security [101].

The same strategy of reducing the number of hops can be used in I2P, where the default is 2-3 nodes for a tunnel and four tunnels are required for a full-duplex connection like it can be seen on Figure 4.2. But reducing the amount is also not recommended, since it will increase the risk of an adversary doing a successful traffic analysis attack [128]. Since I2P is based on a P2P network and contribution reputation system, the only valid strategy identified by the authors of this project, is to use and contribute with your own high-bandwidth I2P router, configured to share as much bandwidth as possible. This should let you to be integrated into the network increasing your chance, of creating tunnels using other high performing I2P routers [129].

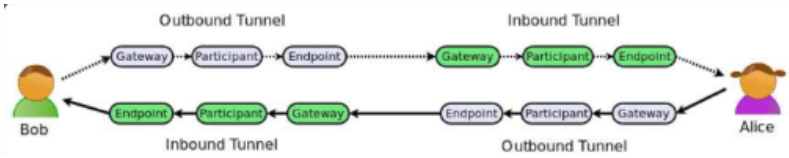


Figure 4.2: Illustration of how I2P tunnels are used for creating connections between peers [69].

Another configuration that could be done in Tor is to configure a list of specific organisation-hosted relays or countries, which the clients will accept. This could be specific countries, that are known for hosting high-performing relays, having good internet infrastructure, being particularly privacy-focused, being geographically close, or having special privacy laws. This method requires a bit of manual work and research for a normal client of TOR, but is more transparent and offers a more intuitive way of adjusting TOR to the user's own threat model.

One last configuration that should be considered is the anonymisation network's support for transporting the WebRTC stream. As mentioned in some of the papers described in "Testing the feasibility of VoIP and WebRTC over anonymisation networks (subsection 2.8.1)" the UDP traffic from RTP needs to be encapsulated before it can be sent over TOR. Instead of encapsulating with a VPN like it was done in the previous papers, this project will be using a "TCP over TURN" connection, which can be seen on Figure 4.3, that encapsulate the RTP packets.

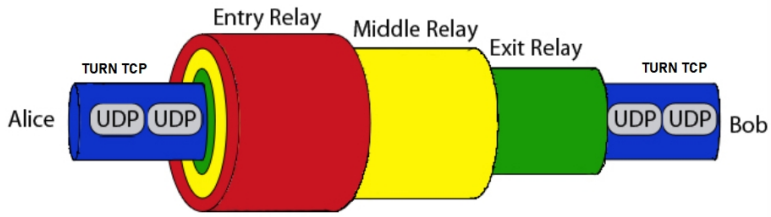


Figure 4.3: The UDP packets from RTP is encapsulated in TURN TCP, so that the traffic can go through TOR that only support TCP [130] (modified).

Lokinet has a VPN-like interface, that can tunnel all kinds of IP traffic through it, so it would be possible for Lokinet to use a "UDP over TURN" connection instead. This would remove the overhead of using TCP and provide the best possible performance for WebRTC. As I2P offers a SOCKS5 proxy interface that also can receive both UDP and TCP traffic, then I2P could also use a "UDP over TURN" connection without the need of encapsulating UDP over TCP. But for keeping the experiment as simple as possible, and the results across the networks comparable, the "TCP over TURN" connection will be used.

4.1.3 Defining high-level testing scenarios

To test the usage of the different anonymisation networks and configurations was a list of different configurations created. The config: *Normal* which does not use any anonymisation network, which will be a baseline. Three different Tor configurations; "*Tor, Normal*" which is the out of the box configuration, "*Tor, Europe*" which accepts nodes in Europe [`DK`, `SE`, `NO`, `FI`, `DE`, `FR`, `BE`, `NL`, `PL`, `CZ`, `LU`, `LV`, `EE`, `CH`, `GB`] and "*Tor, Scandinavia*" which accepts nodes in Scandinavia [`DK`, `SE`, `NO`, `FI`]. An *I2P* client with a standard configuration of 2-hops using the "outproxy" provided by stormycloud. Lastly, a "*Lokinet*" client in a standard configuration where the only non-US exit node in Iceland was chosen.

These different client configurations should be used for defining the different test scenarios, where each scenario uses two unique client configurations. For making it possible to test the test case of both clients using an anonymisation network, each unique client configuration will have to be put on two different clients. This will also make it possible to build redundancy into the test system, by running each test configuration twice, by simply having the clients switch roles. This will make it easier to spot anomalies in the results and reduce the consequences of a specific test client setup crashing, or "under performing". The overall idea and visualisation of this plan can be seen on Figure 4.4, where the scenarios in the "Normal to anonymized" and the "Tor to Tor" boxes have the same arrows from right-to-left and left-to-right.

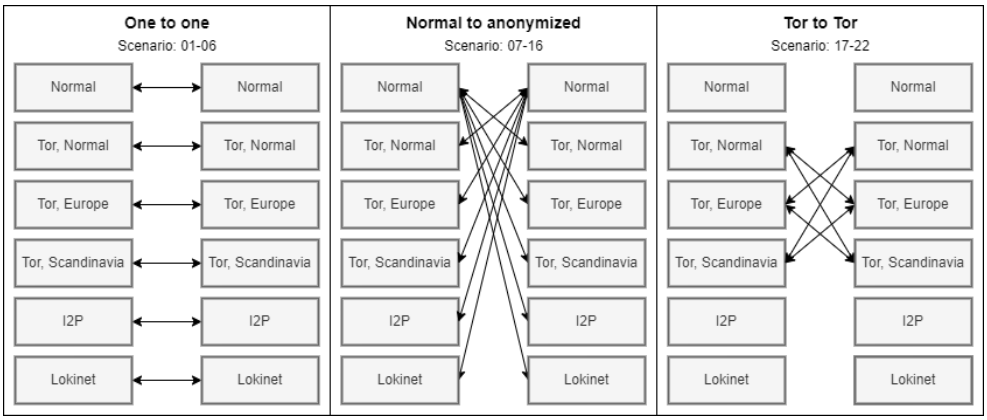


Figure 4.4: High-level visualisation of the test scenarios.

For keeping the number of scenarios within a manageable amount, the structure of the plan does not include cross-network testing. However, since TOR is the only network, which has different configurations, it makes sense that these different configurations are tested "against" each other. This is visualised in the box on the right ("Tor to Tor"), where the rest of the permutations of TOR configurations are tested.

4.1.4 WebRTC statistics and metric gathering

From the papers identified in "Automation testing, performance measurements of WebRTC and QoE of media streaming (subsection 2.8.3)" and from the theory in "Measuring WebRTC media streaming quality (section 2.6)", the following QoS and QoE metrics were identified. These metrics are important for measuring VoIP and WebRTC performance, and are include, but are not limited to:

- QoS
  - Packet loss
  - Jitter
  - Latency
  - Jitter buffer delay
  - Bandwidth/throughput
  - Round Trip Time
  - Sent Bitrate
- QoE
  - PESQ
  - PEVQ

The QoS metrics are crucial and relevant since most of them should be obtainable by using the standard WebRTC available statistics (RTCStatsReport) [131]. The specific metrics and their threshold values that will be used for evaluating the WebRTC sessions are described in Table 4.1.

Network metric;	Accepted value
Roundtrip time	<400
Received packet loss	<2%
Recieved jitter	<30 ms
Audio specific metric:	
Sent bitrate	128 Kbps >value >24 Kbps
Video specific metric:	
Sent bitrate (screen sharing)	150 Kbps >value >50 Kbps
Sent bitrate (240p)	700 Kbps >value >300 Kbps
Sent bitrate (360p)	1000 Kbps >value >400 Kbps

Table 4.1: The QoS metrics and their accepted values for the experiment

Specifically for the "bandwidth" and "throughput" metrics, it is important to define how and where they are measured, since non of them are directly mentioned in the RTCStatsReport from the standard WebRTC statistic API [131]. Measuring the

throughput, is measuring how much data WebRTC is actually sending and receiving. But since WebRTC has it's own congestion control, which acts like TCP's with a slow ramp-up when starting and adjusts the bitrate of the media on the fly, the throughput is not always consuming all of the available bandwidth. Because of this, a tool for measuring both throughput and available bandwidth of the anonymisation networks would be preferred. However, the problem is, that for a tool to actually measuring the available bandwidth it requires sending and generating a lot of traffic. This is the suggested method of one of the previously mentioned papers [109], where they use the tool iperf to measure the bandwidth of the circuit after the test call is done, so that it does not influence the performance of the call. But to do this, you have to be able to control the creation and selection of circuits, to be able to start different TCP-streams over the same circuit. Which was not supported in all three chosen anonymisation networks.

Gathering the QoE metrics have been shown to be somewhat available as open-source software components and offered in a limited form by the industry. Nevertheless considering, that it requires adding extra complexity, processing time and computational resources to the setup, it was deemed not relevant in the scope of the project.

In regards to gathering the metrics during a WebRTC session, the metrics should be sampled with an appropriate[1] frequency over a period of least 1 minute, to gather a couple of data points for each session.

4.1.5 Validating the test setup

To ensure that it is the actual bandwidth and networking limitations of the anonymisation network that the project will be testing, the test setup and environment should be considered.

One way to ensure that no other bottlenecks impact the result is having the clients running on powerful hardware and with a capable internet connection. This should preferably be done in a datacenter, with dedicated hardware resources and with a symmetric networking connection having the same high download and upload speed capabilities. To test and verify if the chosen hardware and the internet connection are good enough, two different approaches will be considered. The first one is identifying, running, and evaluating standardised benchmarks or using load-testing tools mimicking WebRTC traffic, on the test equipment to ensure that it runs reliably and for locating any bottlenecks.

Another way is using infrastructure monitoring and alerting tools that can be used for documenting the load on the test equipment and if any throttling/bottlenecks occurs during the experiment. One could argue that if throttling is detected

[1] The sampling and reporting should not impact the performance of the WebRTC client.

during the experiment over a short time period, the data for the period should be discarded and deemed invalid. But when considering the scope of the project and the time constraint in mind, which already includes deploying and configuration of infrastructure, the latter approach and the risk involved are deemed acceptable. This choice will be discussed later in the implementation as well.

4.1.6 Standardising and reproducibility of the experiment

Another consideration that was analysed, was how to do standardisation and automation of the test clients running scenarios with different configurations. Most of the experiments that have been running in the papers identified, were testing different configurations separately but did not discuss how the software test setup was implemented, orchestrated, or how to reproduce the experiment setup.

One way of ensuring reproducibility is by standardising and documenting the clients, software, and the specific machine configuration used in an open-source manner. A modern DevOps way of doing this is to use "configuration management tools" or containerisation technology like Ansible, Chef, Puppet, Docker, LXC, or Vagrant. These tools can support in doing machine provisioning at different cloud providers, deployment and help with doing machine configuration using "Infrastructure as code". Similarly to virtual machine snapshots, the tools enables one to describe and save a configured machine state using declarative code, which can be versioned as "configuration-as-code". This could enable the experiment to be more easily standardised for being deployed, scaled, configured, and run by others. One thing that was considered during the scoping of the project, was that the experiment should be able to be deployed in datacenters in different geolocations around the world. This was, however, taken out of the scope of the project due to time constraints. Instead, the project was offered the option of getting access to a dedicated server at DTU Ballerup in Denmark.

Another tool that has been seen used in prior work for WebRTC automation, is the usage of browser test automation tools like Selenium and Puppeteer. They provide a way of programmatically configuring, controlling and automating tests in web browsers like Firefox and Chrome, using modern programming languages like Javascript, Java or Python.

4.1.7 Viewpoints

Part of planning the experiment, included defining different questions and viewpoints that would be relevant to answer and consider. Four different viewpoints were identified and discussed:

1. Are any of the anonymisation networks usable to have proxying one or both clients?

2. Can each client use different anonymisation networks and benefit from it?

3. Can the WebRTC infrastructure used for a session be hosted as a hidden service and be accessible both on clearnet and the darknet?

4. Does the anonymisation networks have better coverage/bandwidth in America/EU/Asia and would using any of them lead to a better WebRTC session when testing?

Scoping the experiment lead to answering the first view only. The second view would require a lot more scenarios that should be tested. Furthermore, to isolate each scenario tested, doing parallelised test runs was excluded. This was because it was not deemed possible to ensure that the tests would not impact the performance of each other using the same WebRTC infrastructure. This means, that the scenarios should run sequentially and thus result in fewer total runs of all scenarios, compared to running scenarios in parallel. The third view was tested in the early part of the project, however, it was identified as not being supported "out-of-the-box" by any of the anonymisation networks that were chosen. The fourth view was also deemed beyond the scope of the experiment, since it requires a lot of work in managing and deploying machine configuration in multiple geographical regions as discussed in the previous section.

From the first viewpoint, it is necessary to consider the definition of the anonymisation networks being "usable" for proxying WebRTC data. The specific metrics that have been identified in "WebRTC statistics and metric gathering (subsection 4.1.4)", are the metrics that should be gathered for evaluating the QoS of the call. However, whether or not, a successful WebRTC connection can be established over the given network path created by the anonymisation network, is not a parameter that the experiment can control. The WebRTC connection establishment is used as a black-box, so either the session will be able to start a WebRTC connection or fail within the first couple of seconds during the ICE candidate collection phase. This could be dealt with with the use of a retry mechanism, so that extremely badly created circuits get filtered out. In conclusion, "usable for proxying" should be defined as being able to start, and maintain a WebRTC session over a period of time, where the QoS metric values are at the acceptable threshold values defined.

4.2 Requirements

From the analysis and inspiration from the experiments described in the papers reviewed, the following functional and non-functional requirements were created to support the design decision process for the experiment:

1. WebRTC infrastructure should be setup for doing a 1-to-1 audio and video call using WebRTC over TURN.

2. The chosen infrastructure should have full access to the internet with no restrictions, two dedicated IP addresses, and with a suitable high bandwidth link to the internet.

3. The automation and orchestration of clients should be done from a central control point.

4. The logging, statistics and monitoring data of clients and calls should be gathered at one central point.

5. The logging data should include whether or not a session could be established using a circuit.

6. The statistics gathered during the experiment should at least include the metrics from Table 4.1, which will be used for evaluating if the WebRTC session was usable.

7. The clients should run in the experiment continuously for one month, so they should have high availability and must not be limited on CPU, RAM, disk or networking hardware.

8. The test clients running the WebRTC session should run autonomously.

9. The experiment setup and configurations should be documented and described in such detail, that the experiment can be reproduced and deployed elsewhere.

4.3 Design

To execute the experiment described in the "Analysis (section 4.1)", which follows the requirements described in "Requirements (section 4.2)" an "Experiment protocol" was created. The protocol defines how the experiment was planned, setup, how it should be executed, and includes a suggested plan for data processing of the results. The protocol can be seen in Appendix C "*OnionRTC - Experiment protocol*". The rest of this design section will be describing, the design of the framework and the client configurations used for the experiment.

From the theory section describing WebRTC infrastructure and the ICE protocol, are given the first 5 unique necessary software components, which can be seen on Figure 4.5. This include: the browser clients "Alice and Bob", who are on their own separate network connected to the internet. Each client have the browser automation tool Selenium running, which can be programmed to navigate and operate in a web application context. A web server that is serving a WebRTC application with javascript, html and css to the clients. The clients will upon starting a WebRTC session through the web application using Selenium, make a connection to the signalling server. This server is responsible for managing sessions and connecting people

in rooms. When two clients are in the same room, the signalling server will be used for facilitating the ICE candidate exchange via SDP messages between the clients. The WebRTC application is also responsible for telling each client, if they should use TURN and STUN for their ICE candidate exchange. The application will tell the clients to only use a specific TURN server, which is the last software component.

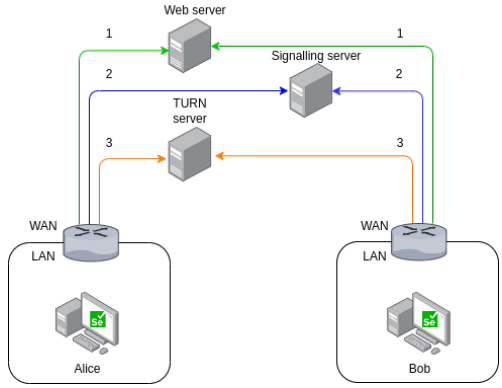


Figure 4.5: Basic WebRTC application and user setup.

Next step, which can be seen on Figure 4.6 is to configure the clients to use the different anonymisation networks for all their traffic. Then create multiple clients, each using one of the anonymisation network configurations like described in Table 4.2.

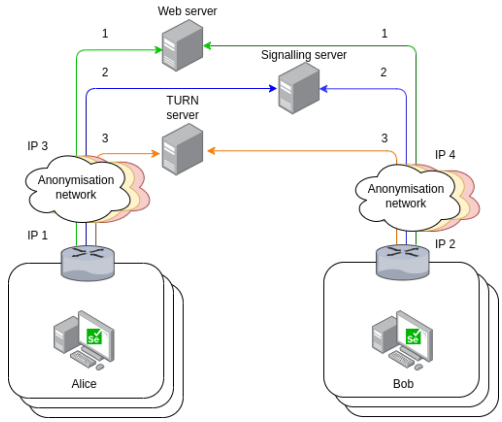


Figure 4.6: Basic WebRTC application with multiple clients using different anonymisation networks.

Networking type	Abbreviation	Client numbering
Normal	`Norm`	`C1, D1`
Tor (Normal)	`TorN`	`C2, D2`
Tor (Europe)	`TorE`	`C3, D3`
Tor (Scandinavia)	`TorS`	`C4, D4`
I2P	`I2P`	`C5, D5`
Lokinet	`Loki`	`C6, D6`

Table 4.2: Table of the different client setups and a description of the abbreviations for scenarios used though out the rest of the report.

The last components that are missing, is the central control of the clients and the logging and system monitoring service which can be seen on Figure 4.7. The central control server is responsible for orchestrating the different clients (step 1) and do logging of the status and results of a scenario run, by sending the results to the logging server (step 2 & 7). The individual clients, are responsible for sampling their own WebRTC statistics and send them to the logging server (Step 6).

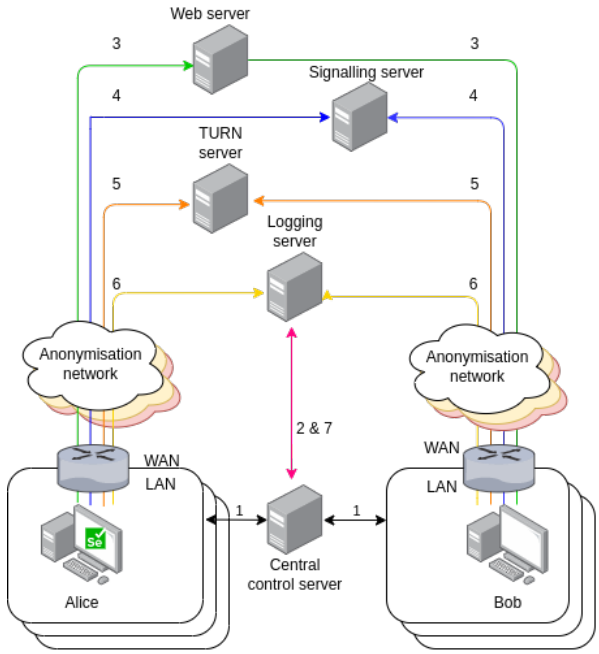


Figure 4.7: System design of the experiment setup and the components of the framework.

An additional extra component, that was originally discussed and planned for, was a dedicated anonymisation monitoring service running locally on the clients. This can be seen on Figure 4.8, where the client has a monitoring service running which probe the anonymisation network for status, performance metrics and other statistics (step 8). Then relevant data about the circuit or path that was used for the WebRTC session could be logged (step 9) and later be used for analysis. The monitoring service, could also have been used for assisting in a selecting strategy for picking the highest performing circuits, based on the self-reporting performance claims returned by some of the anonymisation networks.

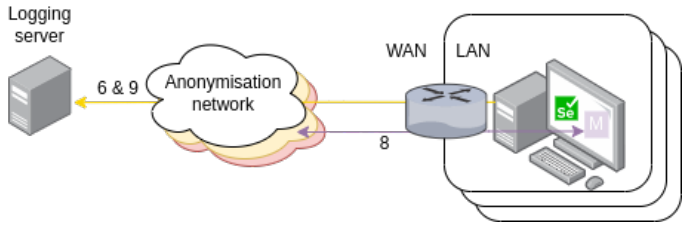


Figure 4.8: Example of anonymisation monitoring running on the client.

In the section ”Defining high-level testing scenarios (subsection 4.1.3)”, the different clients and specifically the different countries used in the TOR client configurations: TorE, TorS and TorN were defined. For a formal definition of the actual test scenarios, using the defined client configurations, the Table C.2 was created.

	Scenario name	Setup (Alice → Turn ← Bob)
One to one	01 Norm-Norm	(c1)Norm → Turn ← Norm(d1)
	02 TorN-TorN	(c2)TorN → Turn ← TorN(d2)
	03 TorE-TorE	(c3)TorE → Turn ← TorE(d3)
	04 TorS-TorS	(c4)TorS → Turn ← TorS(d4)
	05 I2P - I2P	(c5)I2P → Turn ← I2P(d5)
	06 Loki-Loki	(c6)Loki → Turn ← Loki(d6)
Normal to anonymized	07 Norm-TorN	(c1)Norm → Turn ← TorN(d2)
	08 TorN-Norm	(c2)TorN → Turn ← Norm(d1)
	09 Norm-TorE	(c1)Norm → Turn ← TorE(d3)
	10 TorE-Norm	(c3)TorE → Turn ← Norm(d1)
	11 Norm-TorS	(c1)Norm → Turn ← TorS(d4)
	12 TorS-Norm	(c4)TorS → Turn ← Norm(d1)
	13 Norm- I2P	(c1)Norm → Turn ← I2P(d5)
	14 I2P -Norm	(c5)I2P → Turn ← Norm(d1)
	15 Norm-Loki	(c1)Norm → Turn ← Loki(d6)
	16 Loki-Norm	(c6)Loki → Turn ← Norm(d1)
Tor to Tor	17 TorN-TorE	(c2)TorN → Turn ← TorE(d3)
	18 TorE-TorN	(c3)TorE → Turn ← TorN(d2)
	19 TorN-TorS	(c2)TorN → Turn ← TorS(d4)
	20 TorS-TorN	(c4)TorS → Turn ← TorN(d2)
	21 TorE-TorS	(c3)TorE → Turn ← TorS(d4)
	22 TorS-TorE	(c4)TorS → Turn ← TorE(d3)

Table 4.3: Overview of the experiment scenarios.

4.3.1 Summary

In this analysis and design chapter, all high-level details and considerations in regard to the experiment were presented. For the different anonymisation networks, the final configurations were discussed and chosen on the background of not sacrificing too much security for performance gains. The various viewpoints that were identified to be interesting, which define the purpose of the experiment was presented. The corresponding testing scenarios were defined in an attempt to get empirical data related to answering the viewpoint. For setting up the test framework, a list of functional and non-functional requirements was defined. Then a high-level software design and system architecture for the test framework was introduced, which supplement the requirements for the implementation section. Most of the considerations and details discussed in this section were defined quite early in the project in the experiment protocol, which can be found in Appendix C "*OnionRTC - Experiment protocol*".

CHAPTER 5
Implementation

This chapter will cover the implementation details and how to reproduce the setup of the experiment. Everything is also documented in the GitHub organisation repositories [4] created for the experiment.

The structure of the chapter is as follows: start by introducing the physical setup with a description of the hardware, and how the hosts were configured. Then an overview is given of the framework and the deployed applications. Then a description of how the different clients are configured to be used as virtual clients doing a call and how their anonymisation network was configured will also be covered.

5.1 Physical setup

The hosting was provided by DTU. The provided hardware was a server tower with two Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz CPUs which have a total of 16 cores and 32 threads. The server was configured with a total of 256 Gb RAM and disk RAID setup with 2400 GB storage. The server was physically located at DTU Ballerup Campus and connected to Forskningsnettet [132] with a high-speed connection.

To separate concerns on the server, the hardware was divided into a series of virtual hosts following the Table 5.1.

Host(s)	vCPUs (#)	RAM (GB)	Disk (GB)
A	4	32	225
B	2	4	50
C	1	2	50
D	1	2	50
C1-C6, D1-D6	2	4	50
Total: 16	32	86	975

Table 5.1: Specification of all the hosts used in the experiment.

The hosts were configured and connected as shown in Figure 5.1. Where it's visible that host A and B host the services and infrastructure used for WebRTC and

TURN. Host C and D act as a router and gateway for the clients C1-C6 and D1-D6, this was done to limit the amount of required IPv4 addresses needed for the project.

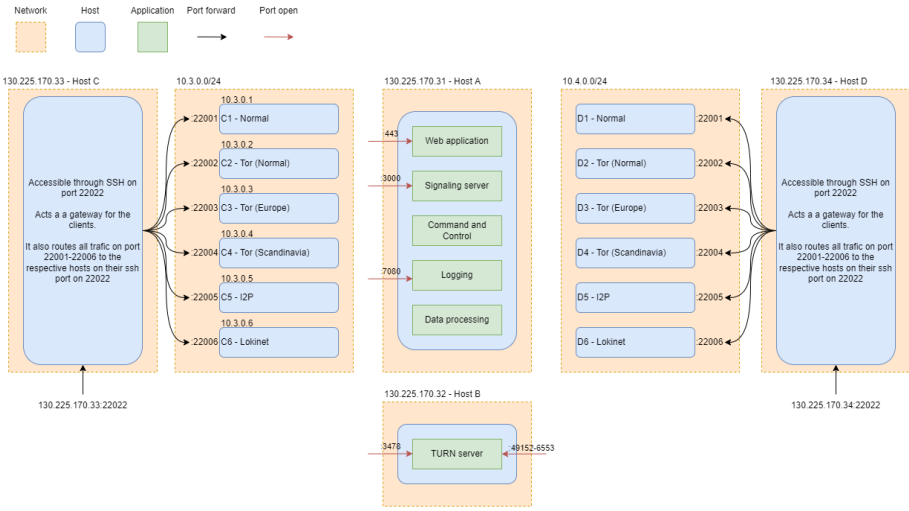


Figure 5.1: Physical deployment diagram with hosts A, B C, the clients C1-C6, D and the clients D1-D6

To enable TLS encryption to the services and allow for easy identification of the services DNS records were set up for the services. The resulting names were configured as shown in Table 5.2.

Domain name	**Host (IP)**
a.thomsen-it.dk db.thomsen-it.dk stage.thomsen-it.dk stage.signal.thomsen-it.dk stage.observertc.thomsen-it.dk	Host A (130.225.170.31)
b.thomsen-it.dk stage.turn.thomsen-it.dk	Host B (130.225.170.32)
c.thomsen-it.dk	Host C (130.225.170.33)
d.thomsen-it.dk	Host D (130.225.170.34)

Table 5.2: DNS configuration used in the experiment.

5.2 Service deployment

The services that ran are described in section 4.3 and the design of how services are going to communicate is shown in Figure 4.7. The Figure 5.1 shows that Host A is used for almost all the services except the TURN server which is the only service running on Host B.

To ease the deployment process, all services are run in docker containers [133] and deployed on the servers with docker-compose files for easy replication, and reproducibility. The services made during the project were also built and released as docker images to docker hub listed below.

- Web application [Docker hub]
- Signal server [Docker hub]
- Command and control [Docker hub]

There are many hosts to configure and set up. In order to eliminate as many human errors as possible and to follow a DevOps mentality, all the clients are configured with [scripts] and Ansible [134] with [playbooks] is used to configure the hosts.

5.2.1 WebRTC infrastructure

This covers the implementation of setting up the WebRTC infrastructure to achieve a 1-to-1 audio and video call. This will be done by a web application using WebRTC, a TURN server and the signalling server to allow clients to exchange ICE candidates as described in the previous section.

A high-level reminder of the process: The web application will be configured to only use the TURN server and send the ice candidate over the signalling server. This happens in practice when the first client creates and joins a room. Then it will wait until the other client joins, in which the signal server will start relaying SDP messages. The web application will then utilise the WebRTC and ICE implementation in the browser to establish a connection through the TURN server between the two clients.

5.2.2 Web Application - [Simple WebRTC]

The web application Simple WebRTC is accessible on https://stage.thomsen-it.dk/ and an image of the application can be seen in Figure 5.2, where the two clients (a) and (b) are having a call.

The application is firstly inspired by an example implementation [135], then modified to only use the TURN server as ICE candidate. Finally, some debugging and scrapable information were added to be displayed as text on the page.

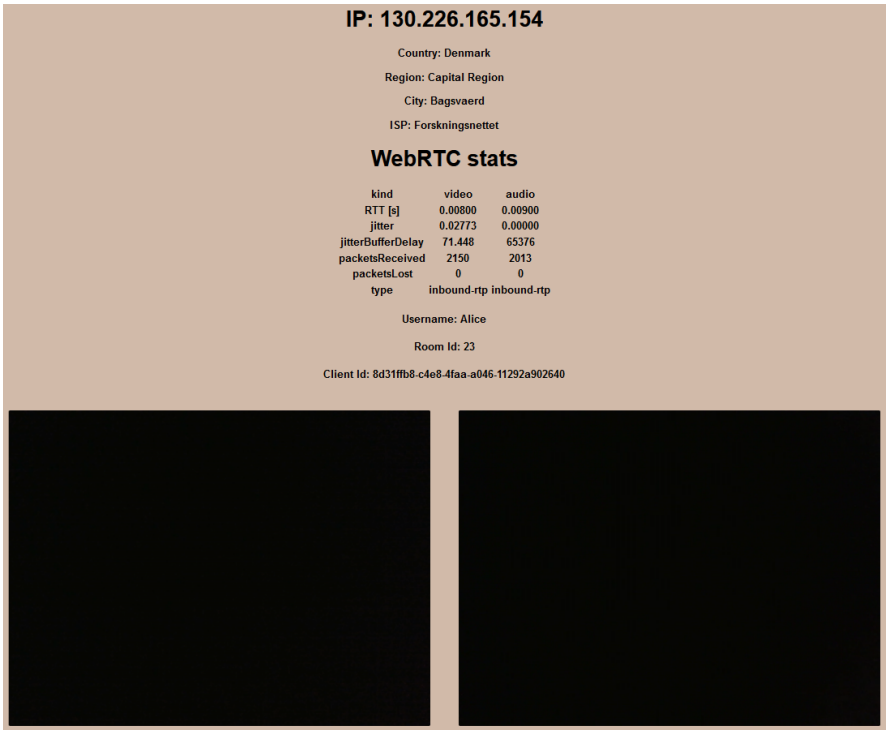


Figure 5.2: Images of two clients doing a call together.

The application uses the browser's implementation of ICE and does an ICE exchange like shown in Figure 2.12. Here, the two clients connect and start both the audio and video streams. The exchange is done by using the Signal server described in subsection 5.2.3, and lastly, the TURN server described in subsection 5.2.4.

5.2.3 Signalling - [Signal Server]

The purpose of the signalling server is to be the "out of band" communication between two clients and exchange their SDP messages between the clients, for them to use ICE and create a connection between them.

The signal server is accessible on https://stage.signal.thomsen-it.dk. The signalling is done through web sockets where the browser joins the requested room, and when the second client joins the same room, and sends the first SDP message the signal server will send the message to everyone who's joined the room, except the sender of the request. The first client will then reply with a SDP answer message, and the clients can connect to each other.

During development and for retrieving information on where the clients are accessing from, a proxy IP location lookup was added to the service. This was used

so that the browser could do a request and then the information would in return be displayed on both the front page before a call and on the call screen. An example of the data which can be gathered can be seen in Figure 5.3.

The two ”ip location lookup” endpoints supported are:

- https://stage.signal.thomsen-it.dk/ip/location, which provides information based on where the application sees the data coming from.
- https://stage.signal.thomsen-it.dk/ip/location/130.226.165.154, which provides information based on the ip at the end of the query (130.226.165.154).

Figure 5.3: Image of the Signal server when queried from which ip it’s accessed from.

5.2.4 Turn Server - [Turn Server]

The TURN server chosen for the project is coturn [136] which did not require much setup or changes. The TURN server is accessible on stage.turn.thomsen-it.dk. But it does not host any websites or does anything else besides acting as a TURN server.

To get the service running a few modifications were, however, needed to be made to the configuration file. One of the changes was to add authentication to the configuration file so that it was only our services, which could use the TURN service. Setting the realm name to the domain of ”thomsen-it.dk” and opening the port ranges which the TURN server required. The technical documentation of the setup can be found in the projects GitHub under [Depolyment/HostB-Turn].

5.2.5 ObserveRTC

To gather statistics of the different WebRTC calls made in a central point during the experiment, the open-source project ”ObserveRTC” [137] was chosen. It provides a simple self-hosted solution and has easy integration with the react application of this project. It also provides a WebRTC statistics collector, which can gather the data from RTCStatsReport and push it to a range of different data sinks.

The ObserveRTC server gets pushed samples directly from the WebRTC clients (monitor) in the browser and sends the samples to the ObserveRTC backend. The backend forwards the information to the database as seen in Figure 5.4.

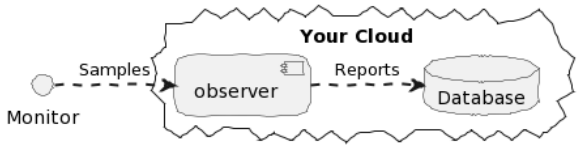


Figure 5.4: ObserveRTC simple design overview [137].

5.2.6 Database

In order to gather all the data in a single location and make data processing easier, the suggested MongoDB [138] setup from the ObserveRTC deployment example [137] was chosen.

This meant that ObserveRTC could send their raw call statistics directly to the database, meanwhile allowing the controller and clients to also be able to send logs directly to the database.

MongoDB is a No-SQL document database, which means that it's a database which uses the term document whereas a relational database would use a table. A document can contain an array of objects which doesn't need to follow a strict column format like in a table, and objects stored in the documents can contain completely different keys than other objects in the same document.

The ObserveRTC service is configured to send logs to the document "reports". The Controller and OnionRTC clients, described later in section 5.3 and section 5.4, send their status messages of the call to the document "calls".

The "calls" document keeps a log of the initiated calls and their outcomes, from both the controller, and the client's perspective. The "reports" are raw WebRTC statistic data from the clients in the calls. Because the controller instructs the clients to which unique room to join, these metrics will be linkable to a specific call between two clients and identifiable to each client.

5.3 Command and control

The experiment was designed to be run from a single location that orchestrated every call made. This is done with the [Command And Control] application.

The command and control service is in charge of running through all the test scenarios as seen in Table 5.3 one by one, checking the configuration of the clients, making initial checks, telling the clients which test_id they should log to and which room_id they should join. The application is also in charge of keeping track of each scenario and parse the response code of both clients and ultimately deciding if the

call in the broadest terms were a successful call. A general overview of the flow can be seen in the flowchart on Figure 5.5.

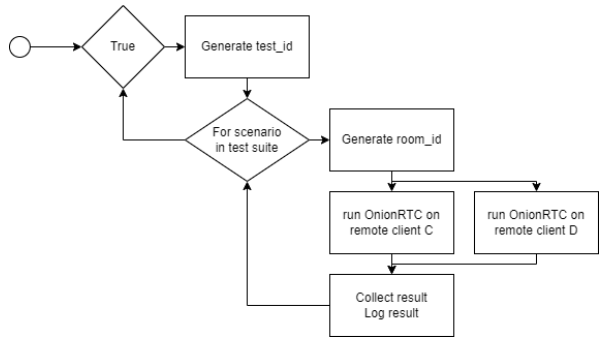


Figure 5.5: Flowchart overview of how the controller orchestrates the tests.

The Controller orchestrates the clients by directly connecting to each client through SSH, using the python library Fabric [139]. The library gives an execution context where the application can execute any Linux command i.e. *python3 OnionRTC.py {args..}* over an SSH connection and retrieve the return code, on the command completing.

	Scenario name	**Setup (Alice → Turn ← Bob)**
One to one	01 Norm-Norm	(c1)Norm → Turn ← Norm(d1)
	02 TorN-TorN	(c2)TorN → Turn ← TorN(d2)
	03 TorE-TorE	(c3)TorE → Turn ← TorE(d3)
	04 TorS-TorS	(c4)TorS → Turn ← TorS(d4)
	06 Loki-Loki	(c6)Loki → Turn ← Loki(d6)
Normal to anonymized	06 Norm-TorN	(c1)Norm → Turn ← TorN(d2)
	07 TorN-Norm	(c2)TorN → Turn ← Norm(d1)
	08 Norm-TorE	(c1)Norm → Turn ← TorE(d3)
	09 TorE-Norm	(c3)TorE → Turn ← Norm(d1)
	10 Norm-TorS	(c1)Norm → Turn ← TorS(d4)
	11 TorS-Norm	(c4)TorS → Turn ← Norm(d1)
	12 Norm-Loki	(c1)Norm → Turn ← Loki(d6)
	13 Loki-Norm	(c6)Loki → Turn ← Norm(d1)
Tor to Tor	14 TorN-TorE	(c2)TorN → Turn ← TorE(d3)
	15 TorE-TorN	(c3)TorE → Turn ← TorN(d2)
	16 TorN-TorS	(c2)TorN → Turn ← TorS(d4)
	17 TorS-TorN	(c4)TorS → Turn ← TorN(d2)
	18 TorE-TorS	(c3)TorE → Turn ← TorS(d4)
	19 TorS-TorE	(c4)TorS → Turn ← TorE(d3)

Table 5.3: Overview of the experiment scenarios.

5.4 OnionRTC

The application [OnionRTC] is the application running on each client, when instructed to by the command and control application. It's built as a command line interface tool in python, to allow for easy development, deployment and testing of different configurations by simply passing arguments to the program.

The client program will upon starting up, parse the arguments given, do various setups, check and validate the connection to the anonymisation network, if any. Then the client starts a WebRTC session using the WebRTC web application with the room id, user id and test id given by the controller. Then as long as the WebRTC session is stable over the duration of 60 seconds, the client will wait. If the session is, however, broken down in the middle of the call, the call ends unsuccessfully. If all 60 seconds have passed, the call is successful and is effectively closed down. The result of whether or not, the call was successful is then reported back to the controller with the status code.

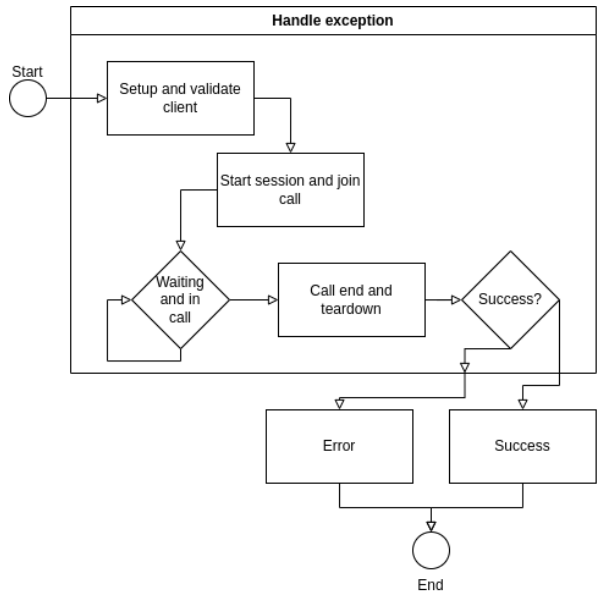


Figure 5.6: High-level flowchart overview of the client.

To have media exchanged during the WebRTC session, a fake webcam and microphone source was created using FFmpeg and Pulseaudio, which is a very simplified way of how it is done in the pipeline of Figure 2.22. A description of how this has been done can be read in the [Selenium folder readme file].

5.4.1 Selenium

For running automated browser tests, the browser automation test tool ”Selenium” was used [140]. The tool offers an API for scripting an end-to-end browser application test, where you script actions such as mouse and keyboard movement/clicks and can interact with the browser rendered html and javascript. This allows you to interact with a webpage like a regular web client, and is ideal for running periodic automated tests.

Multiple programming languages are supported like Python, Java, CSharp, and Javascript, which can interact with the chosen browser driver. In this project, the open-source browser 'Firefox' version 106.0.2 was chosen, but most modern open/closed-source browsers are supported.

Due to the authors' experience, the language support/popularity, and the scripting nature of the language, Python was chosen as the programming language of choice for the browser testing in Selenium.

During the development and testing of the Selenium code that automates Firefox to do a WebRTC session, two configurations of Selenium were needed. The first one, which could not be found as a configurable option in Chrome, was the enabling TURN server relay to be the only ICE candidate available for WebRTC. This can be done in Firefox, by going to the URL ”about:config” and enabling ”media.peerconnection.ice.relay_only”, which can also be programmatically done in Selenium. The other option that was needed, was to tell the browser driver how it should connect through the anonymisation networks. This was done for TOR by navigating to the URL ”about:preferences” and go under ”Networking settings”, where the ”localhost” address of the local TOR SOCKS proxy could be typed in. This was also possible to do programmatically in Selenium, which can be seen in this [code section], where the before mentioned anonymisation connectivity checks also occur.

5.4.2 TOR setup

For the TOR clients, it was decided that it was needed to have 3 different configurations set up, that use nodes from different parts of the world. The three different configurations were: default node selection, use nodes from Scandinavia and lastly use nodes from Europe. In Listing 5.1 an example snippet of the configuration file for TOR can be seen. The *StrictNodes 1* means that it is a strict requirement and any circuits without those nodes should be dropped.

```
1 EntryNodes {dk},{se},{no},{fi} StrictNodes 1
2 MiddleNodes {dk},{se},{no},{fi} StrictNodes 1
3 ExitNodes {dk},{se},{no},{fi} StrictNodes 1
4 ControlPort 9051
```

Listing 5.1: torrc client configuration file using nodes from Scandinavian countries

For running the experiment, a browser is needed which has a WebRTC implementation and has WebRTC enabled. Normally Tor users are advised to use the TOR browser [73], which is preconfigured with privacy settings and plugins. It also has access to interact with the local Tor process, which can be useful when viewing your circuit path, requesting a new circuit, connecting to a Tor Bridge or changing other user settings. But for privacy concerns regarding information leakage, the Tor browser has actively decided to simply disable all WebRTC functionality [120]. So the project instead relies on the Firefox browser[1], which is set up to use the Tor SOCKS proxy on port 9050 locally on the client. The version is shown on Listing 5.2

```
1 agpbruger@d3-tor-eu:~$ tor --version
2 Tor version 0.4.6.10.
3 Tor is running on Linux with Libevent 2.1.12-stable, OpenSSL 3.0.2, Zlib
      1.2.11, Liblzma 5.2.5, Libzstd 1.4.8 and Glibc 2.35 as libc.
4 Tor compiled with GCC version 11.2.0
```

Listing 5.2: TOR version installed and used

For gathering Tor-specific statistics, such as listing the circuit path and bandwidth consumption of the WebRTC session, the Python library Stem was used [77]. This library enables information gathering from the tor process running on port 9015 locally on the client.

5.4.3 I2P setup

During the implementation and actual testing of the I2P setup, it was difficult to find quality documentation and guidance on how to setup I2P in a way, that was compatible with the proposed design. Meanwhile, other papers [80, 83] that were identified during the research and testing phase of the project was using only the default HTTP proxy setup [141]. As the project had an experiment start deadline and considering that I2P had the least promising bandwidth claims, the I2P client configuration was dropped at the last minute.

In the research for anything related to WebRTC over I2P, a project on Github was identified, called ”blind-turn” [142]. The project is made by one of the active developers and community members of I2P and seems to be a work-in-progress attempt at integrating TURN and WebRTC over I2P, using an I2P-specific interface. The project owner was contacted regarding their project and asked for their experience with WebRTC over I2P. A few emails went back and forth, but in the end, this project did not end up pursuing the path of investigating I2P-specific interfaces.

[1]Which Tor has forked and uses for the base of the Tor browser

5.4.4 Lokinet setup

For Lokinet there are only a few different configurations that can be used. The most important one is to choose a specific exit node, which can be chosen from the Listing 5.3 [94][2].

```
exit.loki #(USA)
exit2.loki #(USA, same ip as exit.loki)
xite.loki #(Iceland)
peter.loki #(USA)
door.loki #(USA)
```

Listing 5.3: Lokinet exit nodes that are publicly usable

It made sense that the one in Iceland was picked, for the Lokinet configuration.

5.5 Monitoring

With so many hosts and services deployed it can become hard to maintain an overview of the hosts and services. The duration of the experiment was also quite long and since the project authors were unable to monitor the application logs 24/7, it was deemed necessary to setup monitoring and alerting. The flow of monitoring data can be seen in Figure 5.7.

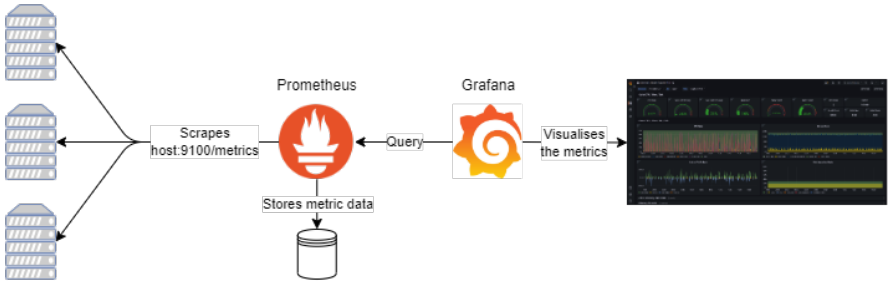


Figure 5.7: The Figure shows the data flow of the metrics scraped. Prometheus scrapes the Node Exporter service on the hosts, when a user access Grafana it will query Prometheus for data in the given time period and visualise the data in the browser.

To keep track of the hosts the application Node Exporter [143] was configured on Host A and all the clients (C1-C6, D1-D6). Node Exporter is an application which works more or less on all operating systems[3]. On Linux all the systems metrics are exposed through files, so the application just needs read access to the host. Then it

[2]This is a "darknet hidden service site" only accessible through the Lokinet network. A screenshot of the site is provided in Appendix B.

[3]Some metrics is not available on all operating systems.

will serve all the metrics through a local web server on the host with default port 9100.

The readings gathered when accessing the Node Exporter is a snapshot of the readings on the host at the time of the request. To make use of the data it needs to be scraped and saved. The application chosen for doing the scraping, was the application Prometheus [144].

Prometheus was configured to scrape all the Node Exporter services on the individual hosts every 15 seconds see the [configuration file]. Notable here, is that it scrapes ”localhost” on a series of ports. This is because a local ssh port forward service was configured to each of the clients, see the [service files running on Host A] which were deployed to the host with the help of an [Ansible playbook].

Now that all the data is collected and stored in Prometheus, then the service Grafana can be used to query Prometheus for the data and visualise the metrics like shown in Figure 5.8.

Figure 5.8: Image of the Grafana visualisation of the metrics gathered from host C1.

Further to get notification when errors occured, a Discord API was configured and used, which enabled instant notifications to be sent from the controller described previously in section 5.3. This API is also used for setting up alerts from Grafana, which are posted on Discord. An image of the Discord API being called can be seen in Figure 5.9, where the controller is reporting that it has started up.

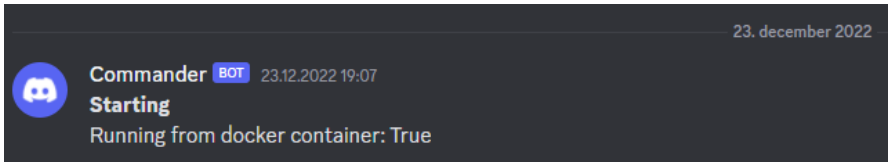


Figure 5.9: Image of the Discord webhook starting the experiment, and showing the connection to the Discord webhook works.

5.6 Consequences or trade-offs caused by chosen design and implementation

A consequence of the chosen deployment strategy shown in Figure 5.1 is that the baseline call, Scenario 01 between `C1` and `D1`, is a bit unrealistic. It has very optimal circumstances since it is routed in the local network, and the same physical machine in a virtual environment and doesn't need to go out on the internet to do its call. More on this, in the discussion.

During the testing phase, it was discovered that one WebRTC implementation difference between Chrome and Firefox, was that it was not possible from a privacy standpoint to force "TURN-only" in Chrome when using WebRTC, which was mentioned shortly in . This was identified as an issue when looking at the WebRTC debugging and statistics page in chrome at the URL "chrome://webrtc-internals". When starting a session using TURN, it was possible to look into the SDP messages which contained STUN ICE candidates, and therefore would reveal the real IP address of the caller to the other peer in a WebRTC call. This was deemed unacceptable and from that point onwards, the Firefox browser with the "TURN-only" configuration was chosen as the browser of choice. However, another WebRTC implementation detail that was not discovered until quite late in the implementation part of the project, was that the WebRTC statistics API in Firefox differs slightly from what is being offered by the Chrome statistics API. This had some influence on the data that was available through ObserveRTC, which mainly targeted Chrome's WebRTC statistics API.

Due to the decision of gathering throughput metrics through the data, which is available from the Node exporter, means that there are a huge amount of samples (1 sample per client per 15 seconds) for the entire period of the experiment. This results in quite a big pre-processing filtering task done for each client, of extracting only the relevant sample from when the client was active in a call.

Lastly, the decision of using "TCP over TURN" with Lokinet, in spite of "UDP over TURN" being a possibility, could have been an interesting reference to have

for comparison. This also has the consequence, that for a successful call the metric ”packet loss” always will remain 0 since all packets in TCP have to be received and acknowledged.

5.7 Summary

This chapter has covered the implementation details of the physical setup used, the components and services of the system, how the services have been configured and lastly, a small recital of some of the trade-offs that have been identified in the design during the implementation. However, the implementation in combination with the design in section 4.3 fulfils the requirements described in section 4.2.

CHAPTER 6

Results

This chapter will include a data analysis and presentation of the results which are available to download. The instructions for this can be found in the Appendix/Result guide. When reading the results and plots there will be mentions of the different scenarios described in Table 5.3 and the abbreviation defined in Table 4.2.

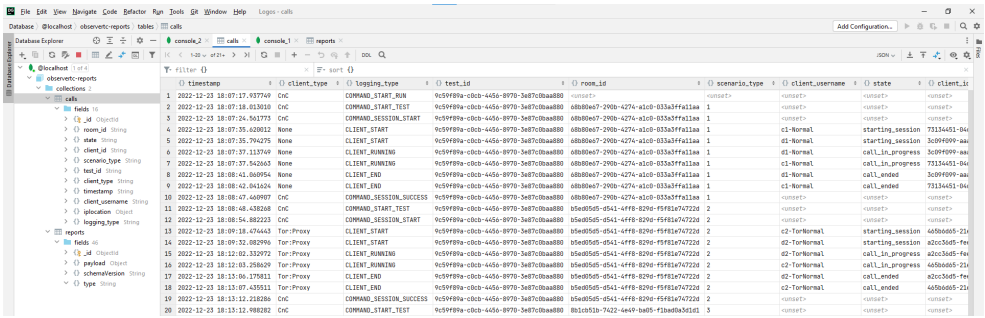
All the data processing is documented in the GitHub OnionRTC-experiment repository [145] under the folder Dataprocess/Readme.md. In the folder all the documentation on how to work with the data locally, what the different Python Jupyter Notebooks do, and how the notebooks are used to generate the graphs for this chapter.

6.1 Data analysis

This section will describe the data sources either generated or gathered from external services. Furthermore, there will be a description of how the data is preprocessed to prepare the data for being used in the plots.

6.1.1 Data sources

From Analysis (section 4.1) the data and metrics that were relevant for doing a WebRTC performance evaluation were decided on. In the Design (section 4.3) it was chosen that there should be data indicating the status of scenario runs. This was reported by the Controller, which started the clients for the given scenario. The clients run a WebRTC session and depending on the outcome, the call is classified as a ”success”, ”failure”, or ”failed setup” call. The clients themselves generate WebRTC performance statistics that are sent to ObserveRTC. Both the controller, the clients, and ObserveRTC would send the data to a Mongo document database store. In the database there are two documents. The first one is ”calls” which contain information on the progress and outcome of all the calls see Figure 6.1. The other document is ”reports” which are filled with data from the ObserveRTC server.



	timestamp	client_type	logging_type	test_id	room_id	scenario_type	client_username	state	client_id
1	2022-12-23 18:07:17.937749	CnC	COMMAND_START_RUN	9c59f89a-c0cb-4456-8970-3e87cdbaa880	<unset>	<unset>	<unset>	<unset>	<unset>
2	2022-12-23 18:07:18.013010	CnC	COMMAND_START_TEST	9c59f89a-c0cb-4456-8970-3e87cdbaa880	68b80e67-290b-4274-a1c0-033a3ffa11aa	1	<unset>	<unset>	<unset>
3	2022-12-23 18:07:24.561773	CnC	COMMAND_SESSION_START	9c59f89a-c0cb-4456-8970-3e87cdbaa880	68b80e67-290b-4274-a1c0-033a3ffa11aa	1	<unset>	<unset>	<unset>
4	2022-12-23 18:07:35.420812	None	CLIENT_START	9c59f89a-c0cb-4456-8970-3e87cdbaa880	68b80e67-290b-4274-a1c0-033a3ffa11aa	1	c1-Normal	starting_session	73134451-04
5	2022-12-23 18:07:35.794275	None	CLIENT_START	9c59f89a-c0cb-4456-8970-3e87cdbaa880	68b80e67-290b-4274-a1c0-033a3ffa11aa	1	d1-Normal	starting_session	3c09f099-aa
6	2022-12-23 18:07:37.113749	None	CLIENT_RUNNING	9c59f89a-c0cb-4456-8970-3e87cdbaa880	68b80e67-290b-4274-a1c0-033a3ffa11aa	1	d1-Normal	call_in_progress	3c09f099-aa
7	2022-12-23 18:07:37.542665	None	CLIENT_RUNNING	9c59f89a-c0cb-4456-8970-3e87cdbaa880	68b80e67-290b-4274-a1c0-033a3ffa11aa	1	c1-Normal	call_in_progress	73134451-04
8	2022-12-23 18:08:41.060954	None	CLIENT_END	9c59f89a-c0cb-4456-8970-3e87cdbaa880	68b80e67-290b-4274-a1c0-033a3ffa11aa	1	d1-Normal	call_ended	3c09f099-aa
9	2022-12-23 18:08:42.041626	None	CLIENT_END	9c59f89a-c0cb-4456-8970-3e87cdbaa880	68b80e67-290b-4274-a1c0-033a3ffa11aa	1	c1-Normal	call_ended	73134451-04
10	2022-12-23 18:08:47.460907	CnC	COMMAND_SESSION_SUCCESS	9c59f89a-c0cb-4456-8970-3e87cdbaa880	68b80e67-290b-4274-a1c0-033a3ffa11aa	1	<unset>	<unset>	<unset>
11	2022-12-23 18:08:48.438268	CnC	COMMAND_START_TEST	9c59f89a-c0cb-4456-8970-3e87cdbaa880	b5ed05d5-d541-4ff8-829d-ff8f81e74722d	2	<unset>	<unset>	<unset>
12	2022-12-23 18:08:54.882223	CnC	COMMAND_SESSION_START	9c59f89a-c0cb-4456-8970-3e87cdbaa880	b5ed05d5-d541-4ff8-829d-ff8f81e74722d	2	<unset>	<unset>	<unset>
13	2022-12-23 18:09:18.474443	Tor:Proxy	CLIENT_START	9c59f89a-c0cb-4456-8970-3e87cdbaa880	b5ed05d5-d541-4ff8-829d-ff8f81e74722d	2	c2-TorNormal	starting_session	465bdd65-21
14	2022-12-23 18:09:32.082996	Tor:Proxy	CLIENT_START	9c59f89a-c0cb-4456-8970-3e87cdbaa880	b5ed05d5-d541-4ff8-829d-ff8f81e74722d	2	d2-TorNormal	starting_session	a2cc36d5-fe
15	2022-12-23 18:12:02.332972	Tor:Proxy	CLIENT_RUNNING	9c59f89a-c0cb-4456-8970-3e87cdbaa880	b5ed05d5-d541-4ff8-829d-ff8f81e74722d	2	d2-TorNormal	call_in_progress	a2cc36d5-fe
16	2022-12-23 18:12:03.258629	Tor:Proxy	CLIENT_RUNNING	9c59f89a-c0cb-4456-8970-3e87cdbaa880	b5ed05d5-d541-4ff8-829d-ff8f81e74722d	2	c2-TorNormal	call_in_progress	465bdd65-21
17	2022-12-23 18:13:06.175811	Tor:Proxy	CLIENT_END	9c59f89a-c0cb-4456-8970-3e87cdbaa880	b5ed05d5-d541-4ff8-829d-ff8f81e74722d	2	d2-TorNormal	call_ended	a2cc36d5-fe
18	2022-12-23 18:13:07.435511	Tor:Proxy	CLIENT_END	9c59f89a-c0cb-4456-8970-3e87cdbaa880	b5ed05d5-d541-4ff8-829d-ff8f81e74722d	2	c2-TorNormal	call_ended	465bdd65-21
19	2022-12-23 18:13:12.218286	CnC	COMMAND_SESSION_SUCCESS	9c59f89a-c0cb-4456-8970-3e87cdbaa880	b5ed05d5-d541-4ff8-829d-ff8f81e74722d	2	<unset>	<unset>	<unset>
20	2022-12-23 18:13:12.908282	CnC	COMMAND_START_TEST	9c59f89a-c0cb-4456-8970-3e87cdbaa880	8b1cb51b-7422-4e49-ba05-f1bad0a5d1d1	3	<unset>	<unset>	<unset>

Figure 6.1: Sample of the data in the *calls* document in the Mongo database.

Another data source that was set up is the service Prometheus, which has been described in Monitoring (section 5.5). The data gathered from the NodeExporter on each client can be queried from Prometheus, among these the different system information like CPU RAM, Disc I/O, and network I/O usage. Note here that the measured values are the total usage of the host and not identifiable to i.e. a specific program. Unfortunately due to a miss configuration in Prometheus which has a default setting of only storing the last two weeks of data, is information older than two weeks from the end of the experiment lost, with a start date of 08/01/2023 at 18:00:00.

The last data source used in the results is an external source from The TOR Project, which gathers performance metrics to enable data driven decision-making in the project. This means that when they update the Tor clients or Tor nodes with i.e. new protocols or routeing algorithms they can compare the different versions to each other and evaluate the performance [146].

The data is gathered by the official Tor monitoring project Onionperf, which is deployed in several places around the world. The project collects different statistics about timeouts, failures, throughput, and RTT by doing a scheduled download of static files over TOR. The files served over HTTP are of different sizes and are hosted from all over the public internet and through onion services [147]. These measurements of the network performance are available as historical data through The TOR Project [148].

6.1.2 Preprocessing

In order to analyse the data, some preprocessing is required. To preprocess the data gathered during the experiment there are four notebooks. The first [Notebook] extracts data from the database ”calls” document and creates a local CSV file with all the calls made during the experiment and summarises the outcome of the call. Next, for all the clients in the database (c1-c6 and d1-d6) it queries the database for the document ”reports” with the gathered WebRTC statistics. Then the data located

in the ”payload” key is extracted and the stats are saved for the individual clients in their own dedicated CSV file.

The second [Notebook] is used to extract the start and end times of the successful calls for the scenarios 1, 8, 9, 10 and 11 from the Mongo database calls document. The result will is saved to a local CSV file.

The third [Notebook] and the fourth [Notebook] are almost the same. They both query throughput data from Prometheus, one for the transmitted data and the other for received data during the successful calls for the scenarios 1, 8, 9, 10 and 11.

During the preprocessing steps, the following data is gathered

- Total success and failed ratios.
- Success and failed ratios over time
- RTT measurements from remote inbound RTP for both audio and video data in successful calls
- Jitter, JitterBufferDelay, and JitterBufferEmittedCount measurements from inbound RTP for both audio and video data in successful calls
- Total throughput on clients during successful calls

6.2 Data presentation

This section will in each subsection show the results of the different performance metrics gathered during the project, each metric will shortly be introduced with how the metric effects the quality or experience of the call, and deeper explanation of the values can be found in chapter 2. The sections will end with a small part conclusions of what knowledge there is to pull from the plots.

6.2.1 The TOR project's metrics

The graphs in Figure 6.2 and Figure 6.3 shows a hybrid between a box plot and a time series, of the TOR projects measurements of throughput and latency in the TOR network [146] for the time of the experiment [149, 150]. On the graphs it shows different nodes names like op-de6a, which is a node located in this example Germany, and the other nodes are located in i.e. Hong Kong, Netherlands and the United States.

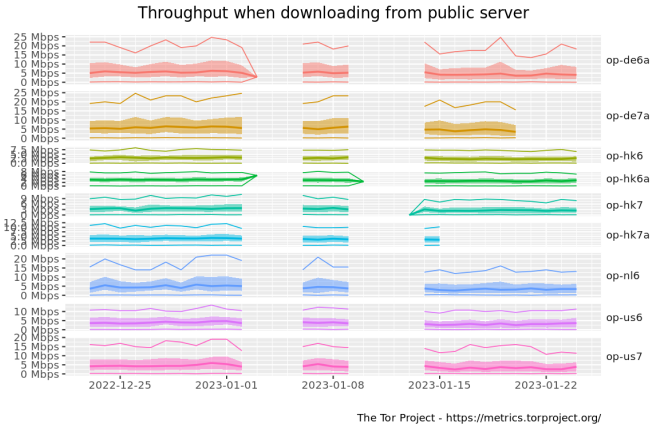


Figure 6.2: The TOR Project's measured throughput for time period of the experiment [149].

The graphs in Figure 6.2 show that the throughput measurements of the network fluctuate a bit but in general it is stable over the entire period and in all regions.

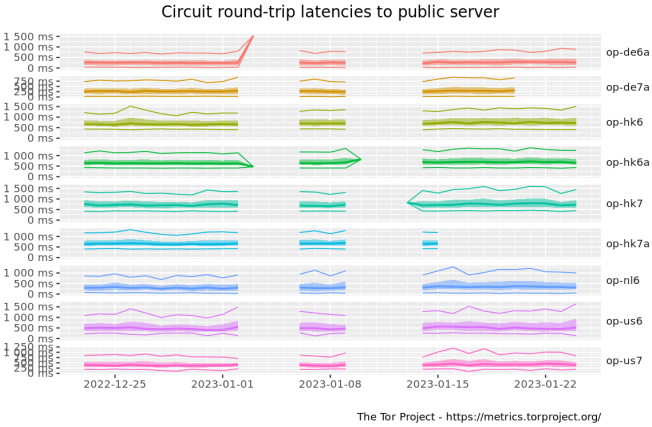


Figure 6.3: The TOR Project's measured latencies for the time period of the experiment [150].

From the Figure 6.3 it is clear that the latency is stable throughout the experiment. Specifically looking at the servers in Germany and the Netherlands (op-de6a, op-de7a and op-nl6) shows that the latency has been stable over the time period of the experiment, with a singular peak on the server op-de6a.

These metrics are used to verify that the network have been available and stable through out the experiment.

6.2.2 Successful calls

A successful call is defined by the Controller after having received feedback from the two clients via exit codes from both OnionRTC client script, and finally decided by the controller.

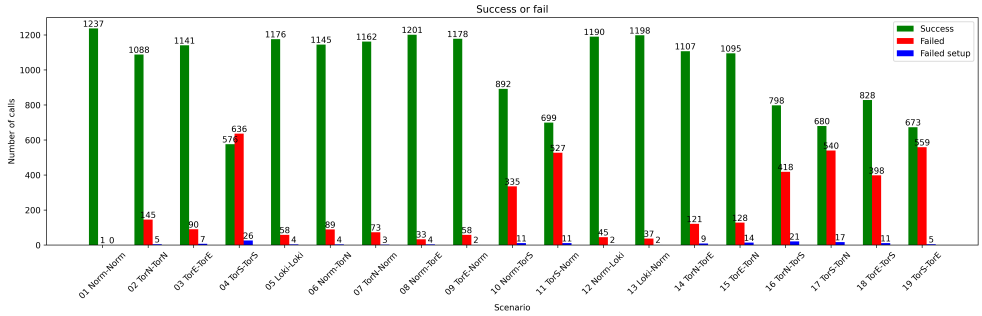


Figure 6.4: The bar chart shows the overall distribution of successful, failed and failed setup runs [Notebook].

In Figure 6.4 it can be seen that scenarios 04, 11, 17 and 19 all have a high failure rate. This is in part because of the client C4 (Alice "TorS" client) that inexplicably started to fail by not being able to start its webcam from the 15/1-2023 and forward. This failure can also be seen in Figure 6.5 and Figure 6.7 where from the 15/1-2023 and on, the scenarios 04, 11, 18 and 19 do not succeed once.

The Figure 6.5 shows the success rate over time for each scenario since the graph is hard to read with so many lines have it been split into two groups based on their success rate, with the scenarios with high success rate in Figure 6.6 and the low success rate in Figure 6.7.

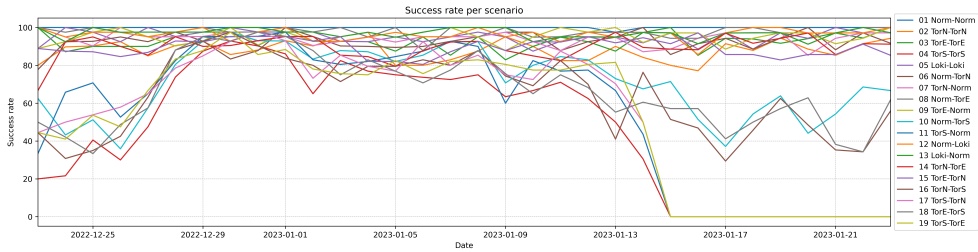


Figure 6.5: The time series shows all the scenarios and for each day the percentage of runs that were completed successfully [Notebook].

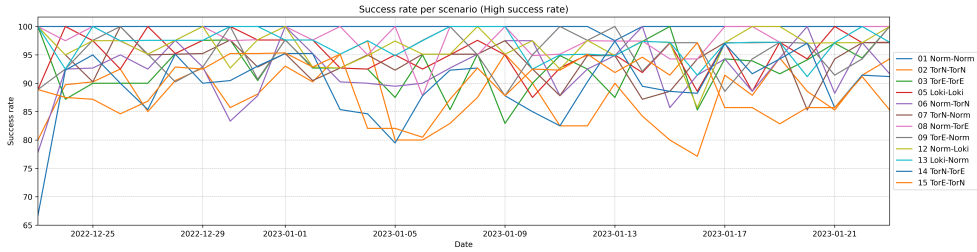


Figure 6.6: The time series shows a sub set and closeup of scenarios with a high success rate Figure 6.5 [Notebook]. An even more divided view can be seen in Appendix D.1 "*Success rate over time*"

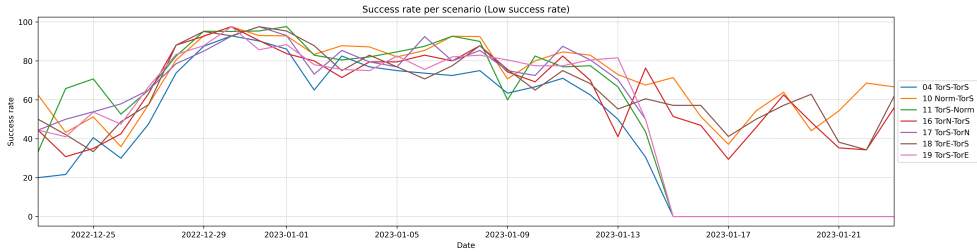


Figure 6.7: The time series shows a sub set of the less performing scenarios from Figure 6.5. Notable here is the scenarios; 04, 11, 17 and 19 starts to fail on the 15/1-2023 due to an error on the hosts C4 [Notebook].

In general, most of the scenarios completed the majority of their calls. The scenarios that failed the most, as can be seen both in Figure 6.4 and Figure 6.7, are the scenarios which include a client with a TorS configuration.

In Appendix D.1 "*Success rate over time*" there is a more detailed view of the scenarios that had a high success rate. The baseline call, scenario 01, performed very well and only failed once. The Lokinet scenarios also had a high success rate. The Tor configurations, with the exception of TorS, also performed well and had in general a high success rate.

6.2.3 Round trip time

RTT, as defined in section 2.2, is the time from the data is sent to the receiver and until the data have been acknowledged. In subsection 4.1.4 the threshold values for video were defined to be optimally 200 ms and acceptable up to 400 ms. Note that the graphs below show all the measured values of RTT during successful calls for each client in each scenario and the y-axis is on a logarithmic scale.

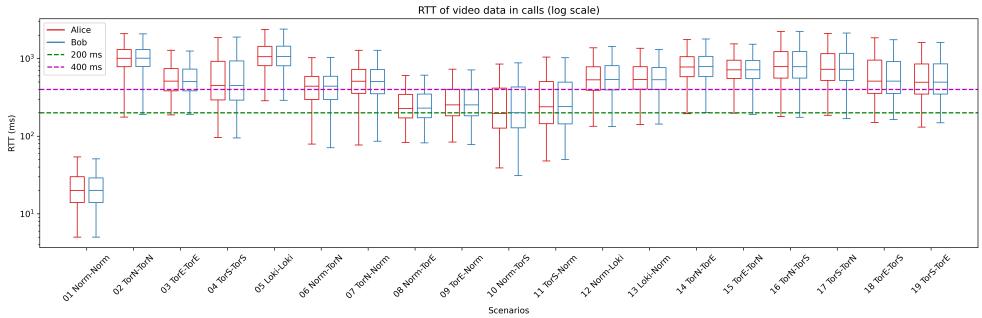


Figure 6.8: The box plot shows the RTT measurements of packets containing the video stream during successful calls on a logarithmic scale [Notebook].

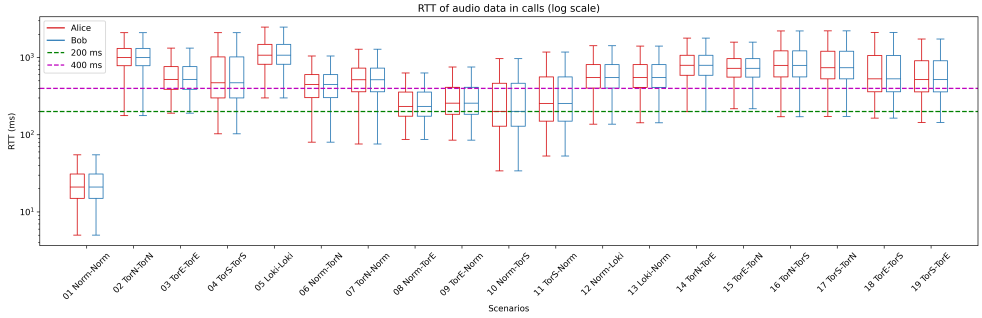


Figure 6.9: The box plot shows the RTT measurements of packets containing the audio stream during successful calls on a logarithmic scale [Notebook].

The results displayed for video in Figure 6.8 and audio in Figure 6.9 shows the baseline scenario 01 as being unrealistically fast, which is also commented in section 5.6. Scenarios 08-11 had the next best performances with more than half for their measured RTT values below 400 ms. The other scenarios had a median RTT value higher than 400 ms, up to 1000 ms.

The impact of a high RTT is that there would be a long time from a clients says or shows something on the video, to the recipient sees or hears the data and acknowledge it. This will result in a bad and unnatural conversation flow. The outcome of the data shown here is that the scenarios 08-11 is considered ”just on the line” of acceptable RTT.

6.2.4 Jitter

Jitter, as defined in section 2.2, is the variance of latency between the reception of packets. The values measured and displayed here are received jitter. From the

subsection 4.1.4 there have not been defined threshold values for the video jitter. However, the threshold value for the audio jitter is 30 ms.

The jitter value is important to the application so that it knows or have historical values of how much the timing between packets can vary, and it will use this to decide when to possibly drop frames, due to, to high delay in delivery.

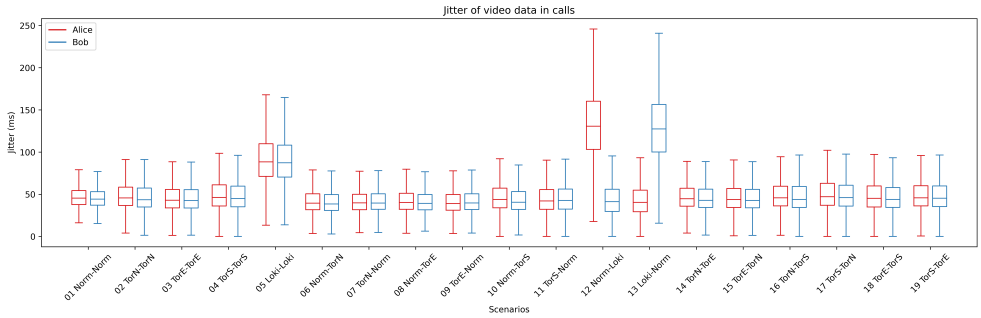


Figure 6.10: The box plot shows the received video jitter for each scenario and client during successful calls [Notebook].

The jitter for video can be seen in Figure 6.10. The box plot shows that most of the scenarios have a video jitter of 50 ms. Except for the scenarios that use Lokinet. The jitter is very high on the clients receiving data from another client which uses Lokinet, and when a client is receiving data through Lokinet it doesn't get high jitter values.

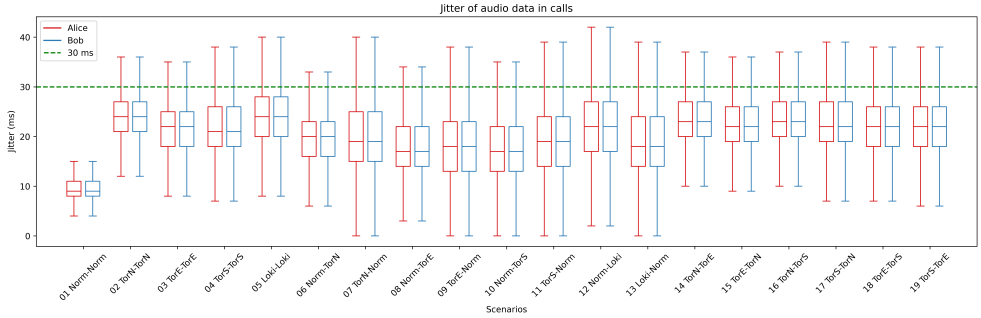


Figure 6.11: The box plot shows the received audio jitter for each scenario and client during successful calls [Notebook].

The audio jitter can be seen in Figure 6.11. The threshold value of 30 ms. Comes from the subsection 4.1.4, and is important so as to not create "holes" in the data stream of the voice during the call. As the graph shows is it not all the measurements that are below the threshold, but for all the scenarios are all the third quantiles lower than the threshold value.

6.2.5 Jitter buffer delay

The jitter buffer delay describes how long time on average an image or sound frame is stored in a buffer when received by the client and until it is released to the application to play the given frame. There have not been defined any threshold values for this metric. Effectively is the jitterbuffer delay additional buffer time on top of the average RTT/2. The jitter buffer delay for video and audio can be seen in respectively Figure 6.12 and Figure 6.13.

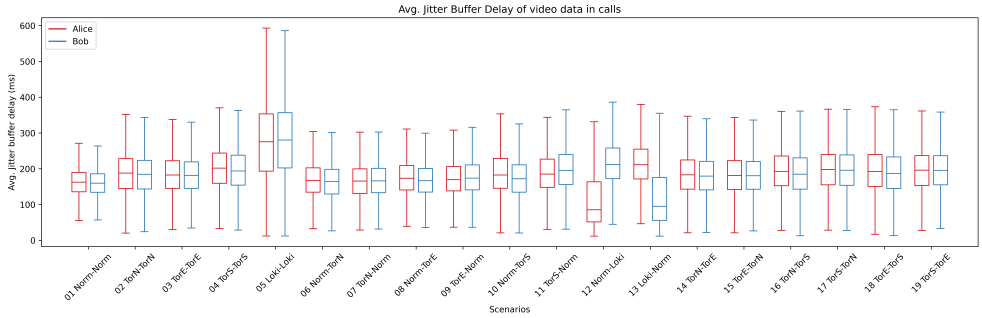


Figure 6.12: The box plot shows the average received video jitter buffer delay for all the scenarios during successful calls [Notebook].

Looking at the video jitter buffer delay in Figure 6.12 it can be seen that the baseline scenario 01 has a median of 175 ms. And that most of the other scenarios are quite close to the same value with the exception of the Lokinet scenarios 05, 12 and 13. Scenario 05, has a higher average and wider distribution. Scenarios 12 and 13 have a lower value on the client who receives data from the Lokinet client but slightly higher on the Lokinet clients.

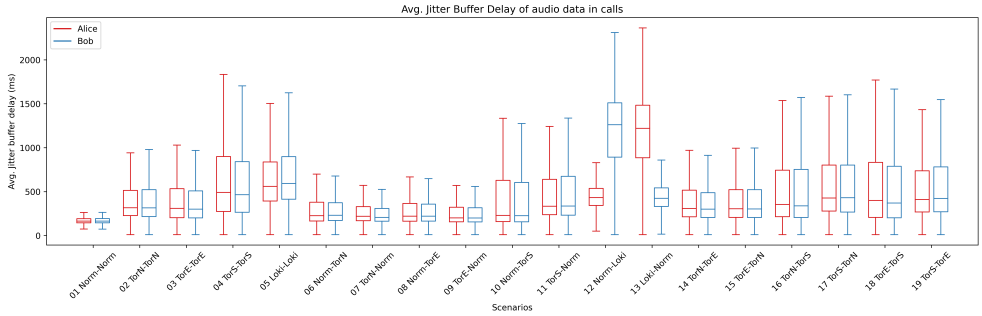


Figure 6.13: The box plot shows the average received audio jitter buffer delay for all the scenarios during successful calls [Notebook].

The jitter buffer delay for audio in scenario 01 has a low median and a comparably

low variance. In general, all the scenarios including a normal client and a TOR client are also quite low. Whereas the rest of the scenarios have a bit higher median and variation. The Lokinet scenarios have in the video data a quite low jitter buffer delay, but in the audio is it very high. The best performing scenarios except the baseline scenarios is 8-11.

6.2.6 Throughput

The last, yet also important, measurement collected is the used throughput for both receive and transmit. In subsection 4.1.4 the threshold values were defined for the required available bandwidth. The range for audio is quite small but for the video range it is very large, therefor the range is quite big in Figure 6.14. A more detailed and specific view can be found in Appendix D.2 "*Bandwidth*". Having a low bandwidth can result in low quality video and or audio.

Due to the strategy used and the way, Prometheus logs the throughput metric, is it very computationally intensive to gather the metric for all the scenarios. Therefore based on the previous results from Section 6.2.3 "*Round trip time*" where scenarios 01, 08-11 are below or close to the threshold. In combination with Section 6.2.5 "*Jitter buffer delay*" where this list of scenarios also has the best performing values are these the scenarios chosen to continue with. A closeup of these scenarios can be seen in Appendix D.3 "*Scenarios 1, 8, 9, 10 and 11*".

Note, as mentioned earlier in the chapter, that these metrics are only available for the last two weeks of the experiment, with data points from 08/01/2023 at 18:00:00.

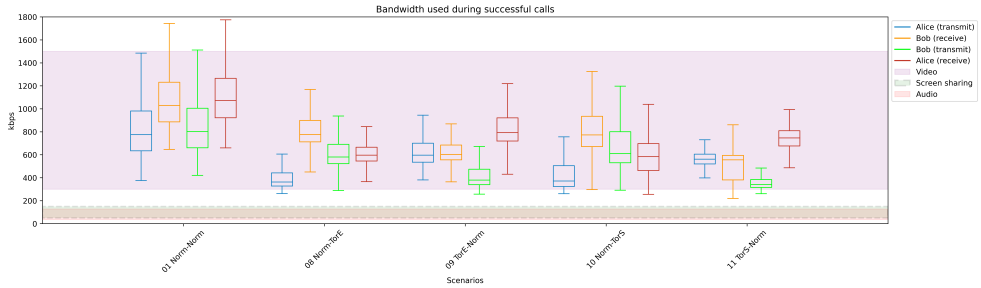


Figure 6.14: The box plot shows the average used to receive and transmit throughput during successful calls, in scenarios 1, 8, 9, 10 and 11. [Notebook].

The box plot shows the measured transmitted and received throughput for successful calls made after 08/01/2023 at 18:00:00. For the audio threshold, the measured throughput is above this requirement. The screen-sharing threshold value is also achieved and the throughput conditions are met for the clients to have been able to do screen-sharing. Looking at the video threshold that have a large range because, the different video resolutions require a different amount of throughput. The throughput used in baseline scenario 01 is about in the middle of the video range. Scenarios 08,

09, 10 and 11 looks to have measured enough throughput to just barely do a video call.

Looking at the Figure 6.5 or more detailed in Figure D.6 is it visible that scenarios 8 and 9 kept up their performance and look pretty stable in success rate. Scenario 10 on the other hand fluctuates a bit in successful calls made. Scenario 11 does not have as many calls since from 15/01/2023, it starts to malfunction with its configuration.

6.3 Summary

From this chapter of results, it was shown that scenarios 8-11, look to have the QoS metrics to do a successful call. The scenarios are the ones where only one client of the clients is using the anonymisation network TOR, where relays from European or Scandinavian countries are used.

Scenarios 08-09 have the same low error rate throughout the whole experiment. Scenarios 10 and 11, however, have about a 1/3 to 1/2 risk of the calls failing. Some of the errors in scenario 11 are due to a faulty configuration in client C4, from the 15/01/2023 causing this scenario to have a higher error rate. The rest of the errors seen could simply be due to bad routes or low-performing relays in the network, from Scandinavia for the scenarios 10 and 11.

CHAPTER 7
Discussion

This chapter will include a discussion regarding the experiment setup, the test parameters and variables and finally the results.

7.1 Results

This section will discuss the results of the experiment; analysed, designed, implemented, and executed, with the results shown in the previous chapters.

The duration of the experiment has been set to one month since that is what the time scope of the project allowed. A continuation of the experiment could be done to ensure that the measured values are representative of the general performance of the networks, and not just unique for the period, where the experiment ran. Since the experiment is using the public internet are there many external factors which could influence the experiment, like how people are using the internet during certain periods. At the beginning of the experiment, there was i.e. the Christmas season which could have affected the experiment's outcome, since some people have holidays and might influence the load on the internet differently (maybe more conversational video or Netflix streaming happening).

The metrics provided by The TOR Project, are shown in subsection 6.2.1. It has not been possible to find explanations for the missing data points in the graphs, but the results from the experiment show that the network has been available and operational throughout the experiment.

7.1.1 Accuracy of the throughput measurement

As touched upon in the previous chapters, the method chosen for this project of measuring throughput, was not satisfactory. This was due to how the sampling were done every 15 seconds and is actually a measurement describing the throughput measured at that exact point of time, instead of representing an accumulated sum over a time period. Furthermore, it is not possible to claim that the data does not exclusively contain WebRTC traffic, since the Linux client machine could be doing other networking tasks at the time. It was discovered after the experiment, that an alternative way of gathering throughput directly from WebRTC itself was available. This was discovered by a combination of looking at the work done in [151] and the

WebRTC example website [42]. The projects, simply use the metrics ”BytesSent” and ”BytesRecieved” that are two accumulated sums of the traffic in bytes WebRTC have processed. Then by sampling every second and keeping track of the previously values of the sums, then the bitrate can be used as a metric for the throughput. The calculations on Equation 7.1 could be used as the throughput metric in bps (Inspired from):

$$\begin{aligned} Throughput = 8 * (bytesSent - last.bytesSent)/(timestamp - last.timestamp) \\ +8 * (bytesRecv - last.bytesReciv)/(timestamp - last.timestamp) \end{aligned} \tag{7.1}$$

7.1.2 Dealing with identified privacy problem in Chrome

As mentioned in Consequences or trade-offs caused by chosen design and implementation (section 5.6), Firefox was chosen over Chrome, even despite the fact that Chrome is the most popular and have the most feature-rich WebRTC implementation. This was due to the privacy problem of Chrome leaking the client's IP address in the SDP messages. One method that was identified, was inspired by the research part of the project when looking at anonymised web browsing in general. As the TOR network only help with anonymising users' IP, the core Onion routing technique can't protect users from themselves. This problem arises typically when talking about fingerprinting and generally trying to identify users, where user accounts, browser agents or browser ”fingerprints” like what browser extensions or settings you have applied, can be used. The proposed solution that the TOR browser has implemented, is to pre-install the extension ”NOscript” [152]. This extension helps by using ”pre-emptive script blocking”, so that cookies, blacklisted URL, javascript, or any other possible privacy-leaking methods are disabled or will prompt a warning like the one shown on Figure 7.1. The filtering happens in the browser on the application level.

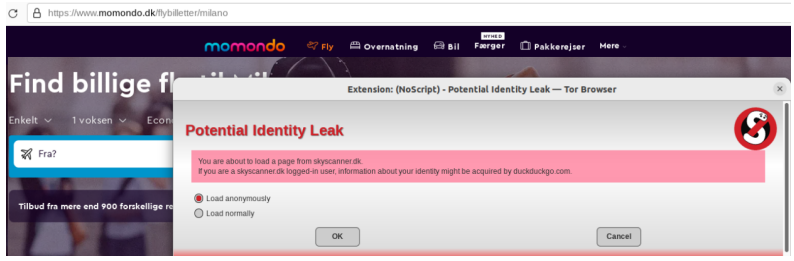


Figure 7.1: Screenshot from a visit to the flight price tracker momondo.com from the TOR browser.

This method of filtering based on application level or ”high-level” information, could be applied to the problem of sensitive IPs being sent in the SDP messages.

The messages are sent to the signalling server and relayed to the other peer, who could then use the IP for malicious purposes. This could either be implemented in the browser as a chrome extension (like this extension) or the filtering could be implemented and offered on the signalling service.

7.1.3 Better performance with TOR

In order to gain better results within the TOR networks, are there a couple of tricks not used in this experiment, one of them is the selective circuit selection strategy. The experiment selected a list of countries in the European and Scandinavian configurations. But if a longer establishment time were acceptable, the client could have created a series of circuits, done a bandwidth performance test, and selected to use the circuit with the highest throughput.

Another way could be to use the performance metrics published by The TOR Project and use these metrics to select high-performing relays to use in the circuit creation.

Lastly, the option of using multiple circuits at the same time could be used, to improve the chance of the traffic reaching the destination in time.

7.2 Ethical considerations

When using the clearnet is it possible for the service providers or ISPs to tell the end user that they feel the end user is abusing the service or is sending too much data to a web server. That is not possible when using an anonymisation network, since the exit node is just forwarding the traffic to the destination host, and doesn't know where it originates from. Therefore the service providers and ISPs can only do a limited set of actions to accommodate such malicious end users. The same thing applies to this experiment. The relays used in the experiment are unable to tell the clients that they are taking too much bandwidth. If a relay thinks that a client is using too much bandwidth the best option it has is to shut down the connection or somehow throttle the connection.

When using the public internet and anonymisation networks to perform tests and experiments such as this thesis has done, is it important to think about not degrading the service for other peers using the services during the experiment. This has been a consideration and parts of a reason to not parallelize testing scenarios and run each node all the time.

7.3 Future work

Using TCP for WebRTC over anonymity networks like TOR using circuits all around the world, is not really efficient in any way, as using TURN relay TCP is a last-resort fallback method for WebRTC connections. This is partly due to the properties of

TCP like retransmission, in-order delivery guarantee and acknowledgement messages. These properties are not well suited for RTP, which for obvious reasons prefers UDP as the transport protocol of choice. At the time of writing TOR does not support UDP, but a proposal has been created [103] and according to Tors Team Lead and Core developer Alexander Færøy, they are working on implementing UDP support, hopefully, to support WebRTC in the TOR Browser [104].

7.3.1 Scaleability

During the experiment, there was enough bandwidth to complete calls. But what would happen if this approach became popular and got implemented in services like Signal, Zoom, MS Teams, etc and the anonymisation networks now needed to support thousands of concurrent calls? The network would be able to support such high demands and what would happen to the available bandwidth in the network? Would that result in a crash of the Tor network? Using simulation tools like Shadow [153] for testing this scenario could be done to further investigate how the network would react or how many concurrent calls it could support.

CHAPTER 8

Conclusion

The structure of the project has been the following: start by doing analyse on the prior work within the relevant fields covered in the project. Then the project went through a design and implementation of a testing framework, that was used to perform an experiment over the duration of a month using the framework. The output of the project aligns with the research questions in the following way:

- *Verify that it's possible to use anonymisation networks to make WebRTC client(s) appear anonymous from the point of view of a TURN server.*

The project has investigated and confirmed that using anonymisation networks can be sufficient for hiding users' IP from the TURN service providers. This was done by looking into the currently available and widespread anonymisation networks and identifying which ones are compatible with the use case of doing WebRTC using TURN servers. The chosen anonymisation networks were the widespread and configurable Tor network, the lesser-known I2P and the new anonymisation network on the block(chain) Lokinet.

- *Formalise a software testing framework/suite, that is able to setup a WebRTC session between two clients and collect performance and networking statistics.*

In order to test the chosen anonymisation network's WebRTC performance, an experiment was designed. This led to the design and implementation of a purpose-built WebRTC test automation framework, that later was used for conducting the experiment. The experiment ran over a period span of one month from 23/12/2022 to 23/01/2023. During the implementation part of the framework, the I2P network was, however, due to time constraints and technical difficulties omitted.

- *Test, measure, analyse and validate the performance of anonymised real-life client applications using Turn services.*

The results of the experiment showed that a useful WebRTC session between a normal client and a Tor client, which uses exclusively Tor nodes from either Europe or Scandinavia, could have been possible to perform during the period of 23/12/2022 to the 23/01/2023. However, the latency between two anonymisation network clients was measured to be too high and thus deemed infeasible to use for a useful WebRTC

session, where both clients needed to be anonymised from the TURN service.

The project's contribution: A study on identifying which anonymisation networks could work as plug-in solutions for WebRTC sessions using TURN. This was done by creating a test WebRTC automation framework, which was tested by running an experiment over the span of a month. Lastly, the results of the experiment can be used for supplementing other empirical data. Specifically, data for validating the claims made by previous papers, that it is possible to use TOR for sending anonymous VoIP one-way over the internet, based solely on the QoS metrics of RTT, jitter and throughput.

This project has successfully investigated, tested, and verified an optional addition to the WebRTC protocol stack which would allow one of the clients in a WebRTC session to become anonymous in cases, where a TURN service provider is used.

APPENDIX A

Real-life usecase: BifrostConnect

The company "BifrostConnect ApS" is a Software-as-a-Service company, that sells a "remote access" solution. The solution which can be seen on Figure A.1 consists of a hardware unit and an online cloud service, that can connect an IT-supporter to the hardware unit through a browser. The IT-supporter can then control, whichever device the hardware unit is connected to via different interface connector types. The typical use case is connecting USB for keyboard and mouse control and HDMI for getting a video of the device's desktop screen (screen-sharing). The service provides similar control and capabilities as the popular Teamviewer application, but does not require any software installations on the device that is being controlled. This is crucial for most industrial equipment, server equipment and other critical infrastructure equipment, that is running a non-supported/deprecated OS and can not be exposed to the internet. The solution uses the industry standard for "real-time video and audio transmission" WebRTC, which in-turn uses the ICE and TURN protocols.

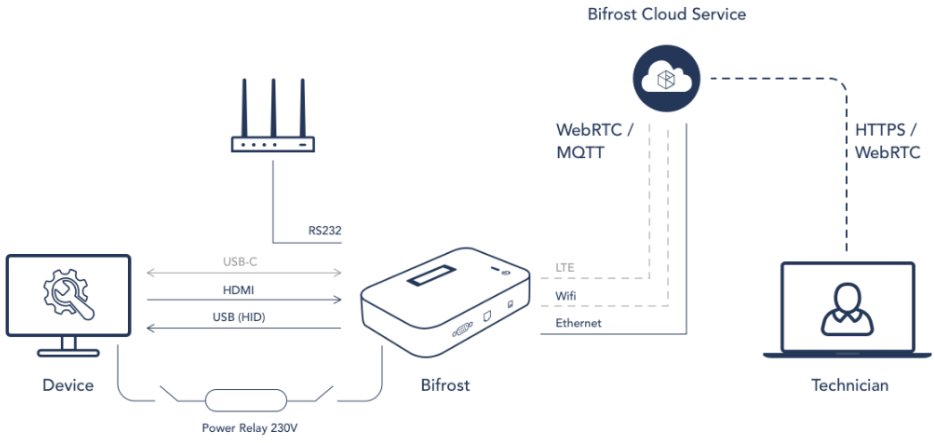


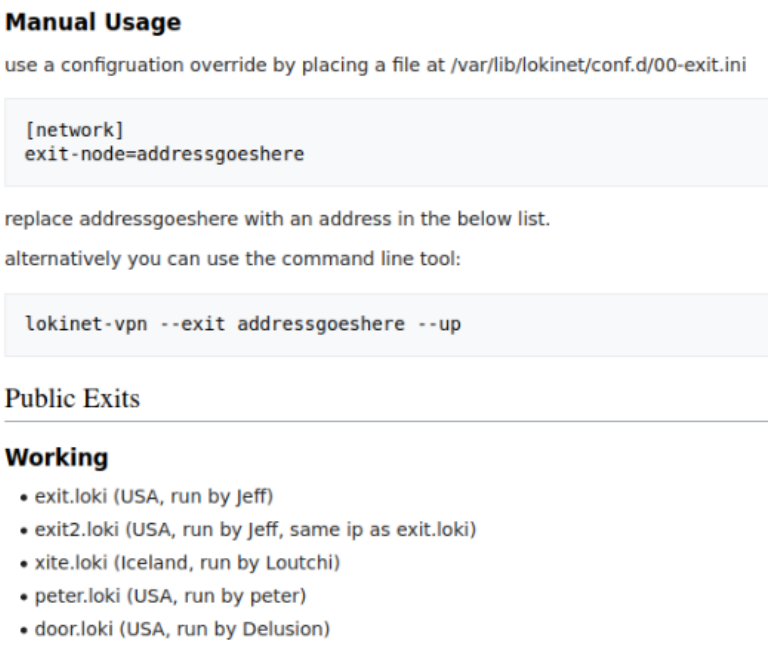
Figure A.1: Overview diagram of the BifrostConnect system [154].

The ”Bifrost Cloud Service” include STUN and TURN servers, which are hosted in different geographical locations by the company using different TURN service providers. These providers are 3rd party companies, which act in their own interest and have a different (maybe non-transparent) business model than BifrostConnect. This can be a cause of concern for BifrostConnect customers, who typically need to perform due diligence when they want to buy/integrate hardware and software products into their company. This typically includes an exhaustive/in-depth risk assessment and technical analysis of the solution and can lead to requirements/demands from the company as for risk mitigation. One risk where the likelihood and consequence are typically unknown or hard to guesstimate for a lot of companies, is the risk of a ”supply chain attack”.

APPENDIX B

Lokinet hidden service citation

The image here is from the citation [94] which is a website hosted on the darknet, using Lokinet. Therefore, a snippet of the site and the data extracted from it is saved here.



Manual Usage

use a configruation override by placing a file at /var/lib/lokinet/conf.d/00-exit.ini

```
[network]
exit-node=addressgoeshere
```

replace addressgoeshere with an address in the below list.

alternatively you can use the command line tool:

```
lokinet-vpn --exit addressgoeshere --up
```

Public Exits

Working

- exit.loki (USA, run by Jeff)
- exit2.loki (USA, run by Jeff, same ip as exit.loki)
- xite.loki (Iceland, run by Loutchi)
- peter.loki (USA, run by peter)
- door.loki (USA, run by Delusion)

Figure B.1: Screenshot of the hidden service site accessed on the 27/10/2022 [94].

APPENDIX C

OnionRTC - Experiment protocol

The following pages are the experiment protocol, which were made in preparation to starting the experiment. It describes the purpose of the experiment, what the perspectives and views of what could be interesting to look into. It goes into depth of how the setup will be configured and how the experiment will run. There are some implementation details of how the clients will do the call, the error states they can end up in and how they will be handled or logged. The metrics which will be gathered during the experiment and how they will be gathered. Lastly, there are a description of how the data should be processed and analysed.

C.1 Introduction

In a highly interconnected world with real-time communication, we have come to expect that privacy is an important part of the technologies we use every day. With dedicated anonymisation networks and technologies like onion routing and proxy services being around for quite some time, they have not yet seen widespread implementation and usage in the everyday applications that we use.

C.2 Experiment purpose

This experiment will answer the question of whether the modern anonymisation networks Tor, Lokinet and I2P are capable of being used in combination with real-time communication video and audio over WebRTC in modern browsers. Capable means providing a usable connection to perform a video and audio call.

The purpose of using these anonymisation networks is not to provide the highest-bandwidth or lowest-latency networking for users. But instead, it tries to provide privacy, which is why the experiment will focus on trying out different configurations and strategies, where configuration of the anonymisation networks are available.

C.3 Viewpoints

We have identified 4 different views/perspectives that could have been interesting to look into, when designing the experiment:

1. Are any of the anonymisation networks usable to have proxying one or both clients?
2. If the clients used different anonymisation networks, could they be used together or provide similar performance?
3. Can the WebRTC infrastructure used for a session, be hosted as a hidden service and be accessible both on clearnet and the darknet?
4. Does the networks have better coverage/bandwidth in America/EU/Asia and would that lead to a better call?

In our experiment, we have decided to try to answer the first view only. The second view would require a lot more scenarios that should be tested. Since the third view did not seem to be supported "out-of-the-box" by any of the anonymisation networks, that were chosen, we did not choose to pursue it. The fourth view was also deemed beyond the scope of the experiment since it requires duplicating and deploying server configuration in other countries through cloud providers.

C.4 Method / Setup

The experiment will run over a span of a month, where dedicated client machines will perform audio and video calls using a WebRTC test application served in firefox, which is connected to the internet via anonymisation networks.

The experiment is planned to run from the 23/12/2022 to the 23/01/2023. During the run of the experiment the infrastructure will be monitored to ensure, that everything looks okay.

What are the steps, and describe the reproducibility of the experiment:
The high-level protocol can be described like this:

1. When the program starts, start to do a "run" in a loop as fast as possible. A "run" is defined as a list of clients in pairs that should do a session.
2. The client pairs are sequentially told to start a WebRTC session with the other client in the pair. During a call statistics and metrics are gathered by WebRTC monitoring tools and clients produce logs that are sent to a central server.
3. When the client is told to start a session they can either succeed, fail before the call started or during the call due to networking problems. If the failure happens before the call started, the client will be told to try again. If a failure

happens during a call it will be logged and classified as a failed session and a successful call will be classified and logged as a successful session.

4. If there are no client pairs left, then go loop back to step 1

The clients (c and d) will have different configurations, which are denoted by their number (1-6), where c1 and d1 are the same configurations but runs on different hosts. The numbers describe the configuration:

- Normal
- Tor
- Tor, Europe
- Tor, Scandinavia
- I2P
- Lokinet

The different configurations we would like to test are the following:

Networking type	**Abbreviation**	**Countries**	**Clients**
Normal	Norm	Direct	C1, D1
Tor (Normal)	TorN	All	C2, D2
Tor (Europe)	TorE	DK, SE, NO, FI, DE, FR, BE, NL, PL, CZ, LU, LV, EE, CH, GB	C3, D3
Tor (Scandinavia)	TorS	DK, SE, NO, FI	C4, D4
I2P	I2P	All	C5, D5
Lokinet	Loki	All	C6, D6

Table C.1: Overview of the different client setups, which anonymisation network they are connected through and their abbreviations.

	Scenario name	Setup (Alice → Turn ← Bob)
One to one	01 Norm-Norm	(c1)Norm → Turn ← Norm(d1)
	02 TorN-TorN	(c2)TorN → Turn ← TorN(d2)
	03 TorE-TorE	(c3)TorE → Turn ← TorE(d3)
	04 TorS-TorS	(c4)TorS → Turn ← TorS(d4)
	05 I2P - I2P	(c5)I2P → Turn ← I2P(d5)
	06 Loki-Loki	(c6)Loki → Turn ← Loki(d6)
Normal to anonymized	07 Norm-TorN	(c1)Norm → Turn ← TorN(d2)
	08 TorN-Norm	(c2)TorN → Turn ← Norm(d1)
	09 Norm-TorE	(c1)Norm → Turn ← TorE(d3)
	10 TorE-Norm	(c3)TorE → Turn ← Norm(d1)
	11 Norm-TorS	(c1)Norm → Turn ← TorS(d4)
	12 TorS-Norm	(c4)TorS → Turn ← Norm(d1)
	13 Norm- I2P	(c1)Norm → Turn ← I2P(d5)
	14 I2P -Norm	(c5)I2P → Turn ← Norm(d1)
	15 Norm-Loki	(c1)Norm → Turn ← Loki(d6)
	16 Loki-Norm	(c6)Loki → Turn ← Norm(d1)
Tor to Tor	17 TorN-TorE	(c2)TorN → Turn ← TorE(d3)
	18 TorE-TorN	(c3)TorE → Turn ← TorN(d2)
	19 TorN-TorS	(c2)TorN → Turn ← TorS(d4)
	20 TorS-TorN	(c4)TorS → Turn ← TorN(d2)
	21 TorE-TorS	(c3)TorE → Turn ← TorS(d4)
	22 TorS-TorE	(c4)TorS → Turn ← TorE(d3)

Table C.2: Overview of the experiment scenarios.

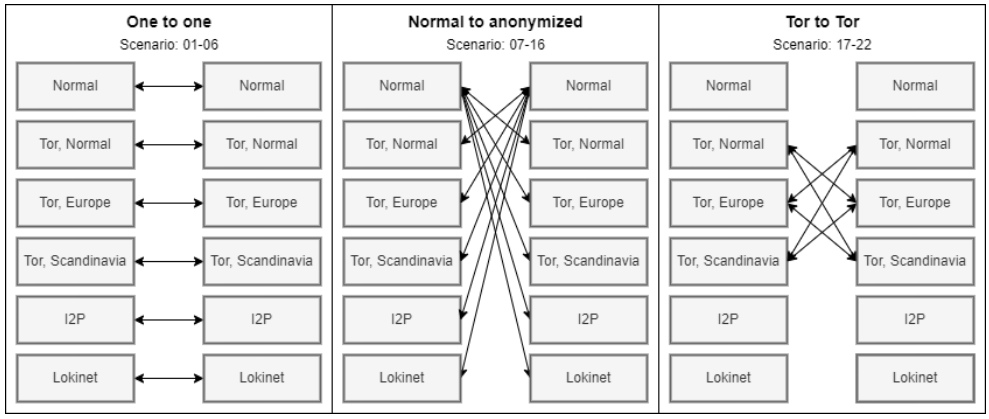


Figure C.1: High level overview of the scenarios and how they are mirrored in the anonymised configurations.

The detailed protocol for how to run the experiment:

1. On the day of the experiment, the C&C server is setup to run a python3.9 script called “controller.py”. The script will keep running for the whole period of the experiment and do as many runs of the scenarios as possible. The script uses a python library called Fabric to remotely execute shell commands over SSH on the clients.

2. The controller.py script is preconfigured with a list of scenarios that describe a pair of clients, that should run. The list is traversed sequentially starting with the first client pair and runs until the list is empty. When there are no client pairs left, the script will start all over again with the same list of scenarios.

3. The controller.py is in charge of the following on the clients: making sure that the webcam is setup correctly, that it can access the logging server and that it runs the “OnionRTC.py” script.

4. The webcam is set up using the kernel module “v4l2loopback” for creating a fake webcam, a combination of the FFmpeg and Pulseaudio programs for handling video and audio streaming to the webcam.

5. The “OnionRTC.py” script takes in as arguments the meeting room id called “room_id” that the clients should join with, what test id that it should use and the client name that it uses for the session. The script then proceeds to verify that the webcam works, selenium works and that the outbound connections to the anonymity network that it is configured to use are working as intended.

6. If the ICE candidate exchange is successful the clients should agree to use the TURN server for the WebRTC connection. Then a session is started and monitored. If not, the clients will try to refresh the procedure up to 4 times.

7. If the session has started and the connection used for the WebRTC connection fails and is not able to reconnect, the session is classified as failed.

8. If the session has started and the session is successful, the session is classified as successful.

9. The client closes down and reports to the logging server, whether the session failed or was successful.

C.4.1 Architecture

To automate the experiment, the following software architecture has been designed:

- Web server - that hosts a WebRTC application over HTTPS.
- Signalling server - that host an event-driven message queue using WebSockets or HTTP long-polling for clients to exchange connection information.

- TURN server - that is a WebRTC infrastructure component. It relays audio and video communication between WebRTC peers that is configured to use it.
- Logging server - that is used to gather logs from clients and C&C server related to a session between two clients.
- Central control server (C&C) - Is in charge of starting scenarios with the different clients, handle client errors and log the status and result of the clients.
- "Onion routing" - is either Lokinet, Tor or I2P that the client uses to access the internet.
- Alice and Bob - are the clients that on command are responsible of running a python script that runs a WebRTC session using the browser automation tool selenium. The clients have different configurations described in the previous section.

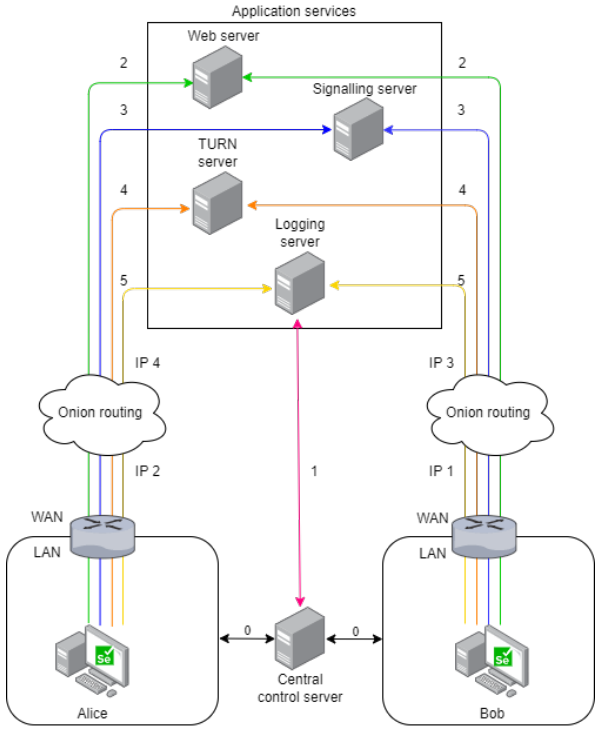


Figure C.2: System design of the experiment setup and the components of the framework.

The communication is described below:

1. A C&C server initiates a run with a pair of clients

2. The clients configure themselves and verify their connection to an anonymisation network. The C&C server sends logs to the logging server, that a run has started with the client pair.

3. The clients connect to a web server, where they will be served a demo page that facilities the html and javascript for a WebRTC session between the clients.

4. The clients exchange their connection details using a signalling server (Standard WebRTC procedure).

5. The clients are configured from step #2 to use a TURN server due to the specific networking circumstances (Standard WebRTC procedure). Video and audio are transmitted through the TURN server, which is delivered to each client's browser.

6. The clients report to the logging server via logs how they experience the call quality and how the session went.

But to simplify the "Onion routing" setup, Alice and Bob will be dedicated machines c1-6 and d1-6, that each run a unique configuration that can be seen in the next section.

C.4.2 Hardware and Software stack

A physical server has been leant by DTU Ballerup, which was used to run the experiment. Below are the specifications of the VM machines, that were set up and running on the server during the experiment:

- The Host A should have 4 cores, 8GB+ RAM and a big disk for the database.
- The Host B machine should have 2 cores and 4 GB ram.
- The Host C and D machines should have 1 core and 2 GB ram.
- The client machines should have 2-3 cores and 4 GB ram.

All the hosts run on the same physical machine with dedicated CPUs

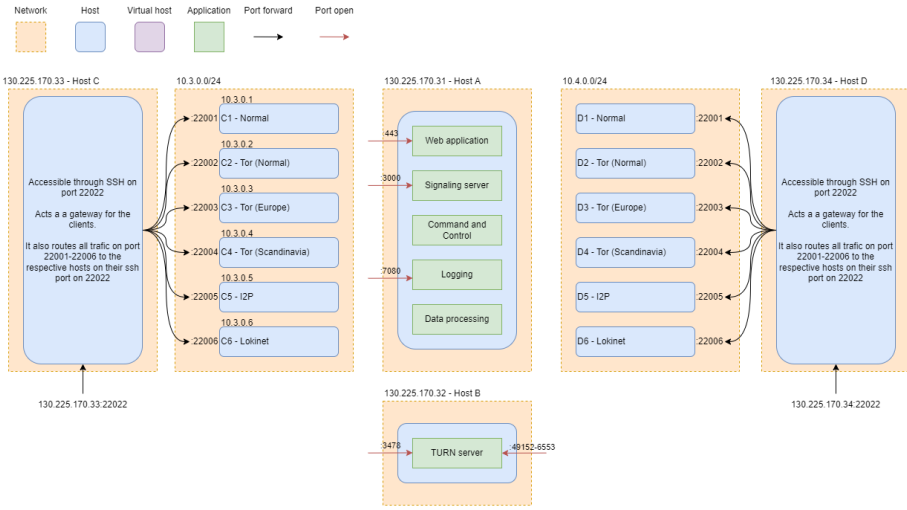


Figure C.3: Physical deployment diagram with hosts A, B C, the clients C1-C6, D and the clients D1-D6.

The Software stack is described on GitHub (Deployment):

C.4.3 Measurements

In the experiment there will be a series of measurements that will describe how a call went.

The C&C application will start the sessions and know if the session succeeded from both clients' point of view and it will save this information.

- Both sessions ended successfully or with an error?

The clients that run the OnionRTC.py script will know if the session executed as expected and in case of an error it will know how and why it failed, and this can be logged.

- The session ended successfully or with an error, what error?

The browser web clients that run will know what data about the session it is sending, and what data it is receiving. Samples are taken every 10 seconds and sent as logs to the logging server. The samples consist of the following measurements:

- Jitter
- Round trip time
- Packet loss

- With more

For measuring the statistics for the WebRTC call, we have been inspired by the work of this article: https://webrtchacks.com/calculate-true-end-to-end-rtt/. They have created an open-source software system called "observer" that can gather WebRTC statistics, in a similar fashion to paid options like: callstat.io and testrtc. The article describes setting up a similar test setup, which uses another type of WebRTC infrastructure called a "Mediasoup SFU", which is another configuration that WebRTC can run in.

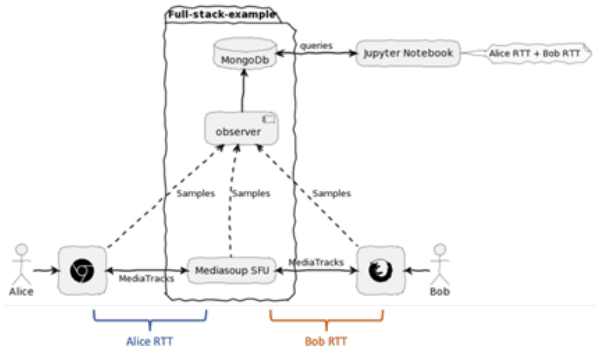


Figure C.4: Example of Observer sampling metrics from clients and an SFU [155].

We used the same setup, but instead of the SFU, we use a TURN server that does not send samples since we have all the data we need from the clients themselves.

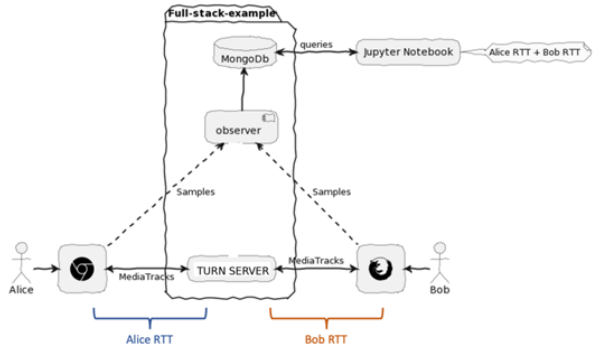


Figure C.5: Modified for the experiment setup.

C.4.4 Protocol and states

The following states have been defined for the C&C and the clients:

Client states:

State	Description
waiting	The clients are waiting for the start signal.
setup_client	The client runs startup and configures the selenium browser, the environment variables and setup logging.
check_media	The client verifies that it can access the video and audio through the browser.
check_webrtc_settings	The client verifies that the specific WebRTC settings are applied correctly in the selenium browser.
starting_session	The client starts by navigating directly into the room that was provided with the start signal using the selenium browser. Then it waits for the "IP lookup" to finish and display in the browser window so it can scrape the data.
waiting_for_call	The client goes into a waiting loop, where it will constantly check whether or not the ICE candidate exchange has failed or succeeded. If it fails, then the client will restart the ICE candidate exchange up to 4 times and if it is successful the call has started.
call_in_progress	The client waits for x amount of time while actively monitoring the liveness of the call. If one of the clients "disconnects" and no reconnect happens then the call ends with an error and goes to the "error" client state. If however, the call continues to work and the x amount of time has passed, the client closes the call.
call_ended	When the call ended without an error, the entire call is categorized as a successful call and gets logged as one.
teardown	The client cleans everything.
done	The client exits.
error	The client sends a log of what went wrong and goes to the "teardown" state.

C&C states:

State	Description
setup	Runs startup: read the environment variables, setup logging, read client inventory/list and logs that it started.
waiting	Wait until it is time to start.
starting	Get two clients that need to run a session. Log that the two clients are about to start a session with a given room_id, test_id and usernames for each of them.
setting_up	Setup and wait for webcams to start.
running	Running a session between the two clients and waiting for them both to return an exit code. If it is zero proceed to "teardown" state. If it is non-zero, go to state "decision".
teardown	Close down the webcam and clean up all processes. Go back to "starting" state or "waiting" state if there are no more clients.
decision	Decide if the clients should retry the session by going to the "setting_up" state or fail by going to the "teardown" state.

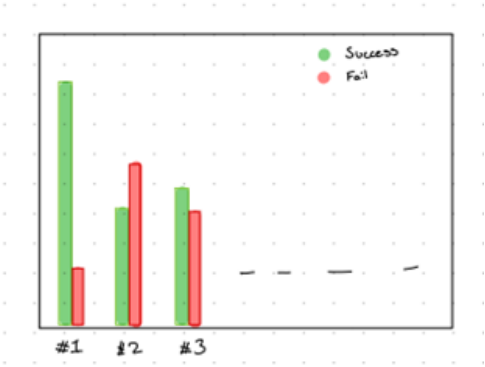
Errors we can foresee:

State	Errors	Retry:
waiting	—	—
setup_client	The anonymity network: Tor might need a restart.	
check_media	Video and the audio script failed.	Yes*
check_webrtc_settings	Nothing, since these settings are applied programmatically and not changed.	—
starting_session	Might lose internet access I2P: loose connection to ssh tunnel to router	Yes*
waiting_for_call	If both clients have waiting time or the number of retries exceeded. If only one client has waiting time or number of retries exceeded.	No, Yes
call_in_progress	The client disconnect and is not able to reconnect before it goes to the failed state.	No
call_ended	—	—
teardown	—	—
done	—	—
error	—	—
*	*	No but log

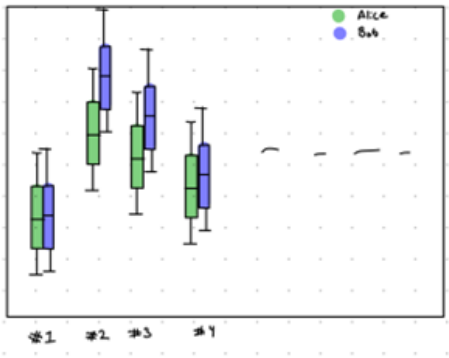
* : Requires action before retry.
If no retry, then the call should be seen as failed for both clients.

C.5 Data interpretation

What will be done with the data once it is collected? Data must be organized and summarized so that the scientist himself, and other researchers can determine if the hypothesis has been supported or negated. Results are usually shown in tables and graphs (figures). Statistic analyses are often made to compare experimented and controlled populations. Once the experiment has run, the first filtration will be done on successful sessions and failed sessions. For each type of scenario, there could be created a bar chart for the distribution. Maybe also with the different errors (Never started, failed during call etc.)



Second for all the calls which were successful, still keeping it within the scenarios. For each call, calculate the average of the jitter and RTT theses numbers can then go into a box plot



The box plot could be made for both Jitter, RTT, and Packet loss.

APPENDIX D

Results

D.1 Success rate over time

In the report Figure 6.6 it can be quite hard to see the lines. The same graph is shown here in Figure D.1 and divided into two separate graphs below in Figure D.2 and Figure D.3

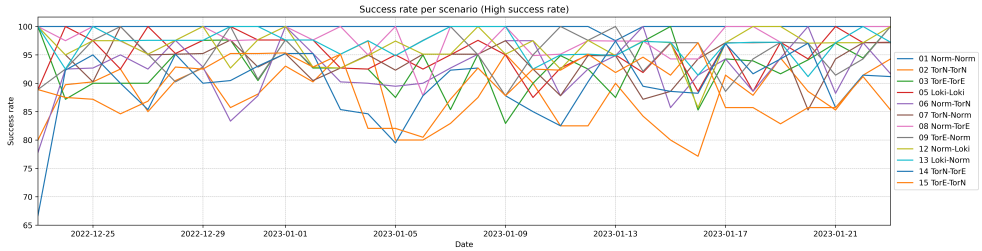


Figure D.1: The time series shows a sub set and closeup of scenarios with a high success rate. Same as Figure 6.6 [Notebook].

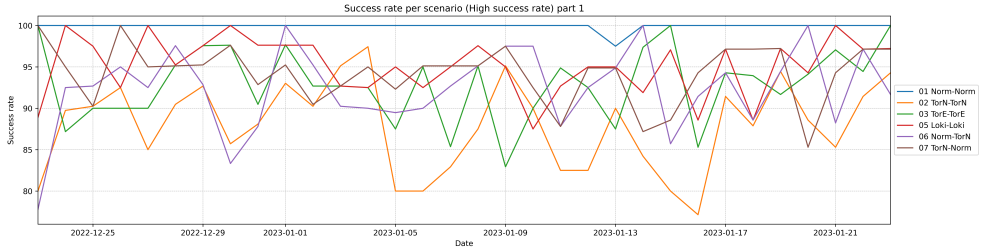


Figure D.2: The time series shows a sub set and closeup of scenarios with a high success rate part 1 of Figure D.1 [Notebook].

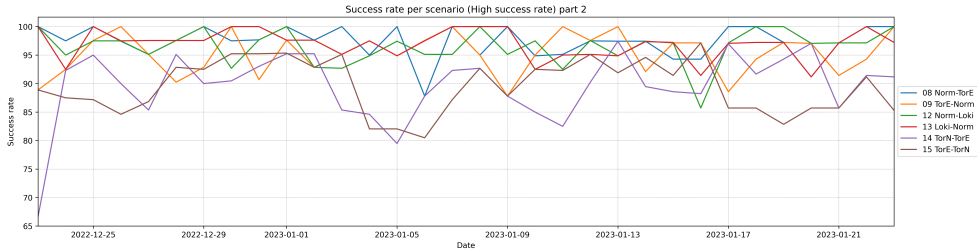


Figure D.3: The time series shows a sub set and closeup of scenarios with a high success rate part 2 of Figure D.1 [Notebook].

D.2 Bandwidth

The plots here show the measured used bandwidth and the different threshold values found from different services. Figure D.4 show the threshold values from Zoom [21] and Figure D.5 show the threshold values from Youtube and Teams [22, 18]

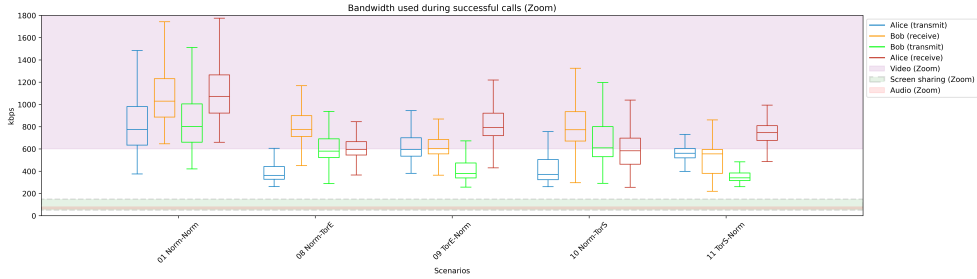


Figure D.4: The box plot shows the average used receive and transmit bandwidth during successful calls, in the scenarios 1, 8, 9, 10 and 11. With the threshold values from Zoom [21] [Notebook].

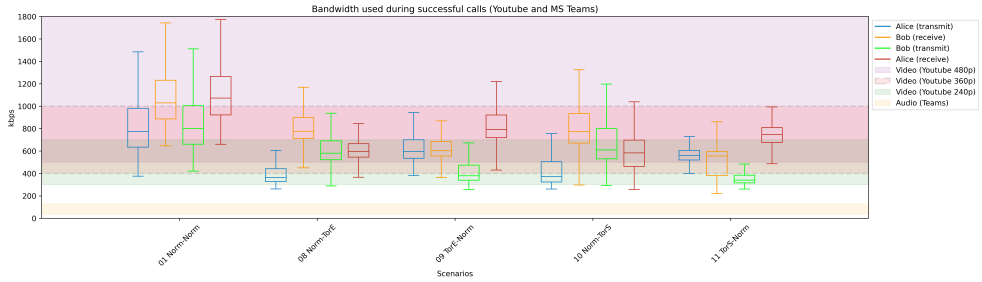


Figure D.5: The box plot shows the average used receive and transmit bandwidth during successful calls, in the scenarios 1, 8, 9, 10 and 11. With the threshold values from YouTube and Teams [22, 18] [Notebook].

D.3 Scenarios 1, 8, 9, 10 and 11

The following graphs are the same as shown in the report, but with the perspective of only scenarios 1, 8, 9, 10 and 11. This is to provide even closer readings of plots and theses scenarios values.

D.3.1 Successful runs

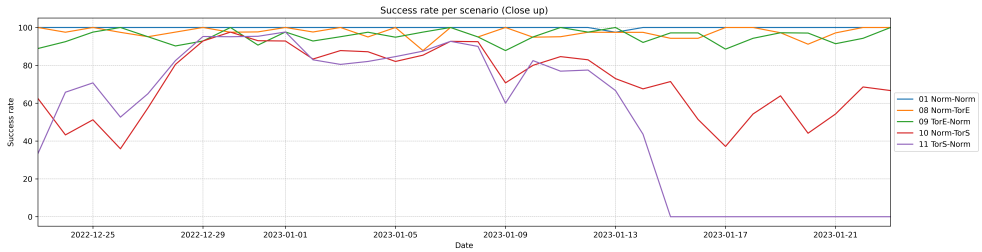


Figure D.6: The time series shows percentage of calls that were successful after having started for the scenarios 1, 8, 9, 10 and 11 for each day the experiment has run [Notebook].

D.3.2 Round trip time

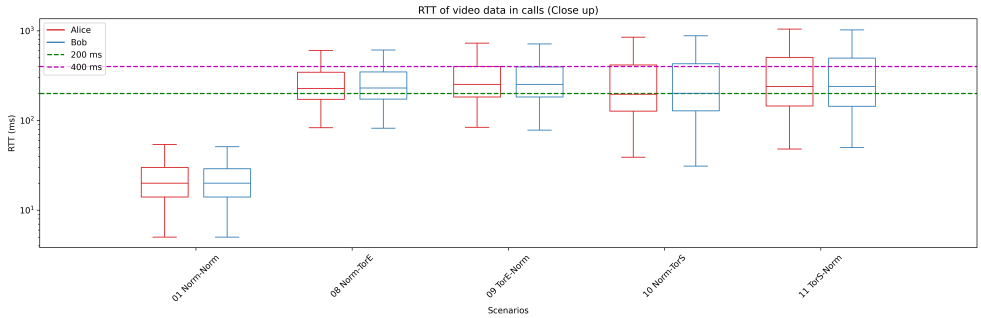


Figure D.7: The box plot shows the RTT measurements for the video stream with a focus on the scenarios 1, 8, 9, 10 and 11 [Notebook].

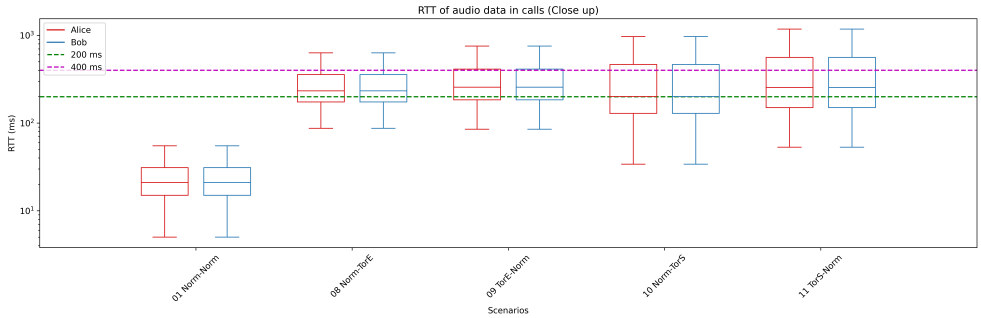


Figure D.8: The box plot shows the RTT measurements for the audio stream with a focus on the scenarios 1, 8, 9, 10 and 11 [Notebook].

D.3.3 Jitter

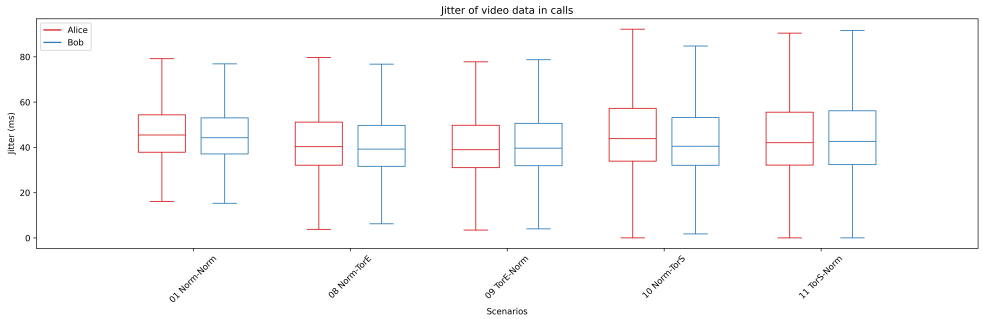


Figure D.9: The box plot shows the received video jitter for each client in the scenarios 1, 8, 9, 10 and 11 [Notebook].

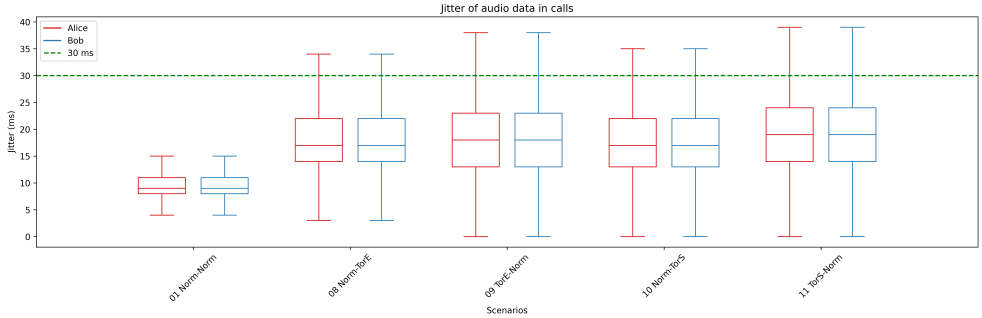


Figure D.10: The box plot shows the received audio jitter for each client in the scenarios 1, 8, 9, 10 and 11 [Notebook].

Bibliography

[1] David Cole. *Michael Hayden: "We Kill People Based on Metadata" - Just Security*. May 2014. URL: https://www.justsecurity.org/10311/michael-hayden-kill-people-based-metadata/.

[2] Edward Snowden. *Just days left to kill mass surveillance under Section 215 of the Patriot Act. We are Edward Snowden and the ACLU's Jameel Jaffer. AUA. : IAmA*. May 2015. URL: https://www.reddit.com/r/IAmA/comments/36ru89/comment/crglgh2/.

[3] Callstats. *Statistics on P2P vs TURN connections*. 2017. URL: https://www.callstats.io/blog/2017/10/26/webrtc-product-turn-server.

[4] Christian M and Jonas T. *Master2022E Github repositories*. URL: https://github.com/orgs/Master2022E.

[5] *Network Latency - Common Causes and Best Solutions | IR*. URL: https://www.ir.com/guides/what-is-network-latency.

[6] webrtcforthecurious. *Real-time Networking | WebRTC for the Curious*. 2016. URL: https://webrtcforthecurious.com/docs/05-real-time-networking/#jitterbuffer-operation.

[7] *Network Latency vs. Throughput vs. Bandwidth Guide - DNSstuff*. URL: https://www.dnsstuff.com/latency-throughput-bandwidth.

[8] James Kurose and Keith Ross. *Computer Networking: a Top-Down Approach*. Global Edition. Pearson Education, Limited, 2016. ISBN: 9781292153599.

[9] Ilya Grigorik. *High-Performance Browser Networking*. O'Reilly Media, Inc., September 2013. ISBN: 9781449344764. URL: https://hpbn.co/.

[10] Daniel Noworatzky. *Gettin' jiggy with jitter*. URL: https://info.teledynamics.com/blog/gettin-jiggy-with-jitter.

[11] Wowza Media Systems. *What Is Low Latency and Who Needs It? | Video | Wowza*. URL: https://www.wowza.com/blog/what-is-low-latency-and-who-needs-it.

[12] H. Schulzrinne et al. "RFC 3550: RTP: A Transport Protocol for Real-Time Applications." In: (July 2003). ISSN: 2070-1721. DOI: 10.17487/RFC3550. URL: https://www.rfc-editor.org/info/rfc3550.

[13] Henning Schulzrinne. *RTP: Overview*. 1997. URL: https://www.cs.columbia.edu/~hgs/rtp/.

[14] webrtcforthecurious. *Media Communication | WebRTC for the Curious*. 2022. URL: https://webrtcforthecurious.com/docs/06-media-communication/#identifying-and-communicating-network-status.

[15] H. Schulzrinne, A. Rao, and R. Lanphier. "RFC 2326: Real Time Streaming Protocol (RTSP)." In: (April 1998). ISSN: 2070-1721. DOI: 10.17487/RFC2326. URL: https://www.rfc-editor.org/info/rfc2326.

[16] Arjan Durresi and Raj Jain. *RTP, RTCP, and RTSP - Internet Protocols for Real-Time Multimedia Communication | Enhanced Reader*. URL: https://www.cse.wustl.edu/%7Ejain/books/ftp/rtp.pdf.

[17] ir.com. *Network Jitter - Common Causes and Best Solutions | IR*. URL: https://www.ir.com/guides/what-is-network-jitter.

[18] Microsoft. *Monitor call and meeting quality in Teams - Microsoft Support*. 2022. URL: https://support.microsoft.com/en-us/office/monitor-call-and-meeting-quality-in-teams-7bb1747c-d91a-4fbb-84f6-ad3f48e73511.

[19] *G.114 : One-way transmission time*. URL: https://www.itu.int/rec/T-REC-G.114-200305-I/en.

[20] Zoom.us. *Accessing meeting and phone statistics – Zoom Support*. September 2022. URL: https://support.zoom.us/hc/en-us/articles/202920719-Accessing-meeting-and-phone-statistics.

[21] *Zoom system requirements: Windows, macOS, Linux – Zoom Support*. URL: https://support.zoom.us/hc/en-us/articles/201362023-System-requirements-for-Windows-macOS-and-Linux#h_d278c327-e03d-4896-b19a-96a8f3c0c69c.

[22] Youtube. *Choose live encoder settings, bitrates, and resolutions - YouTube Help*. URL: https://support.google.com/youtube/answer/2853702?hl=en#zippy=%2Ck-p-fps%2Cp-fps%2Cp.

[23] *WebRTC in a nutshell • BlogGeek.me*. URL: https://bloggeek.me/webrtcglossary/.

[24] J. Rosenberg et al. "RFC 3261: SIP: Session Initiation Protocol." In: (June 2002). ISSN: 2070-1721. DOI: 10.17487/RFC3261. URL: https://www.rfc-editor.org/info/rfc3261.

[25] R. Sparks et al. "RFC 4475: Session Initiation Protocol (SIP) Torture Test Messages." In: (May 2006). Edited by R. Sparks. ISSN: 2070-1721. DOI: 10.17487/RFC4475. URL: https://www.rfc-editor.org/info/rfc4475.

[26] A. Begen et al. "RFC 8866: SDP: Session Description Protocol." In: (January 2021). DOI: 10.17487/RFC8866. URL: https://www.rfc-editor.org/info/rfc8866.

[27] P. Srisuresh and M. Holdrege. "RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations." In: (August 1999). ISSN: 2070-1721. DOI: 10.17487/RFC2663. URL: https://www.rfc-editor.org/info/rfc2663.

[28] J. Weil et al. "RFC 6598: IANA-Reserved IPv4 Prefix for Shared Address Space." In: (April 2012). ISSN: 2070-1721. DOI: 10.17487/RFC6598. URL: https://www.rfc-editor.org/info/rfc6598.

[29] Tailscale. *How NAT traversal works · Tailscale*. URL: https://tailscale.com/blog/how-nat-traversal-works/.

[30] J Rosenberg et al. *RFC 3489: STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. March 2003. URL: https://www.ietf.org/rfc/rfc3489.txt.

[31] Eytan Manor. *An architectural overview for WebRTC — A protocol for implementing video conferencing | by Eytan Manor | Medium*. February 2021. URL: https://eytanmanor.medium.com/an-architectural-overview-for-webrtc-a-protocol-for-implementing-video-conferencing-e2a914628d0e.

[32] Hussein Nasser. *WebRTC Crash Course - YouTube*. November 2022. URL: https://youtu.be/FExZvpVvYxA.

[33] Jonathan Rosenberg and Simon Perreault. *RFC 6062: Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations*. 2010. URL: https://www.rfc-editor.org/rfc/rfc6062.

[34] A. Keranen, C. Holmberg, and J. Rosenberg. "RFC 8445: Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal." In: (July 2018). ISSN: 2070-1721. DOI: 10.17487/RFC8445. URL: https://www.rfc-editor.org/info/rfc8445.

[35] Louis Stowasser and Robert Nyman. *WebRTC and the Ocean of Acronyms - Mozilla Hacks - the Web developer blog*. July 2013. URL: https://hacks.mozilla.org/2013/07/webrtc-and-the-ocean-of-acronyms/.

[36] E Ivov, J Uberti, and P Saint-Andre. *RFC 8838: Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol*. January 2021. URL: https://www.rfc-editor.org/rfc/rfc8838.txt.

[37] Emil Ivov and Victor Pascual. *ICE always tastes better when it trickles! (Emil Ivov) - webrtcHacks*. December 2013. URL: https://webrtchacks.com/trickle-ice/.

[38] W3.org. *WebRTC 1.0: Real-Time Communication Between Browsers*. January 2021. URL: https://www.w3.org/TR/2021/REC-webrtc-20210126/.

[39] mdn. *WebRTC API - Web APIs | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API.

[40] *Internet Engineering Task Force (IETF) — RIPE Network Coordination Centre*. 2012. URL: https://www.ripe.net/participate/internet-governance/internet-technical-community/ietf.

[41] Tsahi Levent-Levi. *DTLS-SRTP • BlogGeek.me*. URL: https://bloggeek.me/webrtcglossary/dtls-srtp/.

[42] WebRTC organization from googlegroups. *Peer connection: adjust bandwidth*. URL: https://webrtc.github.io/samples/src/content/peerconnection/bandwidth/.

[43] A.B. Roach. "RFC 7742: WebRTC Video Processing and Codec Requirements." In: (March 2016). ISSN: 2070-1721. DOI: 10.17487/RFC7742. URL: https://www.rfc-editor.org/info/rfc7742.

[44] T. Melia and S. Gundavelli. "RFC 7847: Logical-Interface Support for IP Hosts with Multi-Access Support." In: (May 2016). Edited by T. Melia and S. Gundavelli. ISSN: 2070-1721. DOI: 10.17487/RFC7847. URL: https://www.rfc-editor.org/info/rfc7847.

[45] *Codecs used by WebRTC - Web media technologies | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/Media/Formats/WebRTC_codecs.

[46] *About | Alliance for Open Media*. URL: https://aomedia.org/about/.

[47] *Web video codec guide - Web media technologies | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Video_codecs#av1.

[48] webrtcforthecurious. *Applied WebRTC | webrtcforthecurious*. 2021. URL: https://webrtcforthecurious.com/docs/08-applied-webrtc/#webrtc-topologies.

[49] *E.800 : Definitions of terms related to quality of service*. URL: https://www.itu.int/rec/T-REC-E.800-200809-I/en.

[50] Boni García et al. "Understanding and estimating quality of experience in WebRTC applications." In: *Springer-Verlag Wien* 101.11 (November 2019), pages 1585–1607. ISSN: 0010-485X. DOI: 10.1007/s00607-018-0669-7. URL: http://link.springer.com/10.1007/s00607-018-0669-7.

[51] Gulnaziye Bingol et al. "The Impact of Network Impairments on the QoE of WebRTC applications: A Subjective study." In: *Institute of Electrical and Electronics Engineers Inc.* (October 2022), pages 1–6. ISSN: 24727814. DOI: 10.1109/QOMEX55416.2022.9900882.

[52] Bart Jansen et al. "Performance Evaluation of WebRTC-based Video Conferencing." In: *ACM SIGMETRICS Performance Evaluation Review* 45.3 (March 2018), pages 56–68. ISSN: 0163-5999. DOI: 10.1145/3199524.3199534. URL: https://dl.acm.org/doi/10.1145/3199524.3199534.

[53] *P.10 : Vocabulary for performance, quality of service and quality of experience*. URL: https://www.itu.int/rec/T-REC-P.10-201711-I/en.

[54] *P.800 : Methods for subjective determination of transmission quality.* URL: https://www.itu.int/rec/T-REC-P.800-199608-I.

[55] *J.247 : Objective perceptual multimedia video quality measurement in the presence of a full reference.* URL: https://www.itu.int/rec/T-REC-J.247-200808-I/en.

[56] Force Technology. *ITU-T P.800 methods Methods for subjective determination of transmission quality.* URL: https://forcetechnology.com/-/media/force-technology-media/pdf-files/unnumbered/senselab/itu-t-p-800-methods.pdf.

[57] Boni García et al. "Assessment of QoE for Video and Audio in WebRTC Applications Using Full-Reference Models." In: *Electronics 2020, Vol. 9, Page 462* 9.3 (March 2020), page 462. ISSN: 2079-9292. DOI: 10.3390/ELECTRONICS9030462. URL: https://www.mdpi.com/2079-9292/9/3/462.

[58] B Garcia et al. "WebRTC Testing: Challenges and Practical Solutions." In: *IEEE Communications Standards Magazine* 1.2 (2017), pages 36–42. DOI: 10.1109/MCOMSTD.2017.1700005.

[59] *WebRTC Quality of Experience: What is the State of the Art?* URL: https://www.callstats.io/blog/webrtc-quality-of-experience-what-is-the-state-of-the-art.

[60] *WebRTC test scoring • testRTC.* URL: https://testrtc.com/docs/webrtc-test-scoring/.

[61] Smartproxy. *The Best Residential Proxy Network | Smartproxy.* 2022. URL: https://smartproxy.com/.

[62] IPROYAL. *IPROYAL TOP Proxy Services Provider, with 2M+ IP's.* 2022. URL: https://iproyal.com/?r=29729.

[63] netnut. *netnut proxy service provider.* 2022. URL: https://netnut.io/?ref=harshakirangudibandi63.

[64] ExpressVPN. *High-Speed, Secure & Anonymous VPN Service | ExpressVPN.* 2022. URL: https://www.expressvpn.com/.

[65] NordVPN. *The best online VPN service for speed and security | NordVPN.* 2022. URL: https://nordvpn.com/.

[66] Proton. *Proton VPN: Secure and Free VPN service for protecting your privacy.* 2022. URL: https://protonvpn.com/.

[67] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. "Hiding routing information." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1174 (1996), pages 137–150. ISSN: 16113349. DOI: 10.1007/3-540-61996-8{_}37/COVER. URL: https://link.springer.com/chapter/10.1007/3-540-61996-8_37.

[68] M Hosseini Shirvani and A Akbarifar. "A Comparative Study on Anonymizing Networks: TOR, I2P, and Riffle Networks Comparison." In: *Journal of Electrical and Computer Engineering Innovations* 10.2 (2022), pages 259–272. DOI: 10.22061/JECEI.2021.8027.466.

[69] Andrew Savchenko. *The Invisible Internet Project powerpoint*. 2018. URL: https://archive.fosdem.org/2018/schedule/event/i2p/attachments/slides/2537/export/events/attachments/i2p/slides/2537/i2p.pdf.

[70] Stevens Le Blond et al. "Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems." In: (). DOI: 10.1145/2785956.2787491. URL: http://dx.doi.org/10.1145/2785956.2787491.

[71] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. *Anonymous Connections And Onion Routing - Selected Areas in Communications, IEEE Journal on | Enhanced Reader*. 1990. URL: http://www.csl.mtu.edu/cs6461/www/Reading/08/Reed-jsac98.pdf.

[72] Roger Dingledine et al. "Tor: The second-generation onion router." In: (2004). URL: https://svn-archive.torproject.org/svn/projects/design-paper/tor-design.pdf.

[73] TorProject.org. *Tor Project | Anonymity Online*. URL: https://www.torproject.org/.

[74] Ciprian Dobre and Fatos Xhafa. *Pervasive Computing*. Edited by Ciprian Dobre and Fatos Xhafa. Elsevier, 2016. ISBN: 9780128036631. DOI: 10.1016/C2015-0-00088-0.

[75] eff.org. *How HTTPS and Tor Work Together to Protect Your Anonymity and Privacy | Electronic Frontier Foundation*. URL: https://www.eff.org/pages/tor-and-https.

[76] The Tor Project. *Relay Search*. URL: https://metrics.torproject.org/rs.html#search/flag:authority.

[77] torproject.org. *Welcome to Stem! — Stem 1.8.1-maint documentation*. URL: https://stem.torproject.org/.

[78] *I2P: A scalable framework for anonymous communication - I2P*. URL: http://i2p2.de/en/docs/how/tech-intro.

[79] *I2P Compared to Tor - I2P*. URL: http://i2p2.de/en/comparison/tor.

[80] Likun Liu et al. "I2P Anonymous Communication Network Measurement and Analysis." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11910 LNCS (2019), pages 105–115. ISSN: 16113349. DOI: 10.1007/978-3-030-34139-8{_}11.

[81] I2P. *I2P Anonymous Network*. URL: https://geti2p.net/en/.

[82] *Meet Your Maintainer: StormyCloud - Blog - I2P*. URL: https://geti2p.net/en/blog/post/2022/09/07/Meet_your_Maintainer_StormyCloud.

[83] Mathias Ehlert. "I2P Usability vs. Tor Usability A Bandwidth and Latency Comparison." In: (2011). URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.476.4614&rep=rep1&type=pdf.

[84] *I2PTunnel - I2P*. URL: https://geti2p.net/en/docs/api/i2ptunnel.

[85] Christoph Egger et al. "Practical Attacks Against The I2P Network." In: (2013). DOI: 10.1007/978-3-642-41284-4{_}22. URL: https://sites.cs.ucsb.edu/~chris/research/doc/raid13_i2p.pdf.

[86] John R Douceur. "The Sybil Attack." In: *Springer Verlag* (2002), pages 251–260. ISSN: 16113349. DOI: 10.1007/3-540-45748-8{_}24. URL: https://www.freehaven.net/anonbib/cache/sybil.pdf.

[87] Kee Jefferys et al. "Loki." In: (2018). URL: https://loki.network/wp-content/uploads/2018/10/LokiWhitepaperV3_1.pdf.

[88] oxen.io. *Session & Lokinet - Oxen | Privacy made simple*. URL: https://oxen.io/session-lokinet.

[89] oxen.gitbook.io. *Oxen Service Nodes - Oxen Docs*. URL: https://oxen.gitbook.io/oxen-docs/about-the-oxen-blockchain/oxen-service-nodes.

[90] Loki.network. *Lokinet exit nodes: Onions gone wild - Loki*. November 2020. URL: https://loki.network/2020/11/20/lokinet-exit-nodes-explained/.

[91] Loki.network. *Rebranding Loki Messenger - Loki .network Blog*. 2019. URL: https://loki.network/2019/12/13/rebranding-loki-messenger/.

[92] getsession.org. *Session | Send Messages, Not Metadata. | Private Messenger*. URL: https://getsession.org/.

[93] oxen-docs. *oxen-docs/LokinetConfig.md at master · oxen-io/oxen-docs*. URL: https://github.com/oxen-io/oxen-docs/blob/master/docs/Lokinet/Guides/LokinetConfig.md.

[94] probably.loki. *Exit Nodes in Lokinet*. URL: http://probably.loki/wiki/index.php?title=Exit_Nodes.

[95] Fatemeh Shirazi et al. "A Survey on Routing in Anonymous Communication Protocols." In: (2016). URL: https://arxiv.org/pdf/1608.05538.pdf.

[96] gnunet.org. *GNUnet*. URL: https://www.gnunet.org/en/.

[97] Ian Clarke et al. "Freenet: A Distributed Anonymous Information Storage and Retrieval System." In: *Springer-Verlag* (2001), pages 46–66. URL: https://www.cs.princeton.edu/courses/archive/fall09/cos518/papers/freenet.pdf.

[98] *ZeroNet: Decentralized websites using Bitcoin cryptography and the BitTorrent network*. URL: https://zeronet.io/.

[99] Raymond Sweha and Azer Bestavros. *Enhancing Tor Performance For Bandwidth-Intensive Applications*. Technical report. 2012. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.348.6965&rep=rep1&type=pdf.

[100] *GitHub - gegel/torfone: TORFone - voice add-on for TorChat: voice-over-Tor and p2p secure and anonymous VoIP tool*. URL: https://github.com/gegel/torfone.

[101] Yérom-David Bromberg et al. *Donar: Anonymous VoIP over Tor*. ISBN: 9781939133274. URL: https://www.usenix.org/conference/nsdi22/presentation/bromberg.

[102] guardianproject.info. *Voice over Tor? - Guardian Project*. 2012. URL: https://guardianproject.info/2012/12/10/voice-over-tor/.

[103] David Goulet. *proposals/339-udp-over-tor.md · main · The Tor Project / Core / Tor Specifications · GitLab*. URL: https://gitlab.torproject.org/tpo/core/torspec/-/blob/main/proposals/339-udp-over-tor.md.

[104] Alexander Faeroey. *Modernizing the Tor Ecosystem*. URL: https://ahf.me/talks/2022/07/24/modernizing-the-tor-ecosystem/.

[105] Maimun Rizal, Somayeh Taheri, and Dieterx Hogrefe. "Empirical performance analysis of anonymizing VoIP over the Onion Router (TOR) network." In: *2013 International Conference on Privacy and Security in Mobile Systems, PRISMS 2013 - co-located with Global Wireless Summit* (October 2014). DOI: 10.1109/PRISMS.2013.6927177.

[106] *TorStatus - Tor Network Status*. URL: https://torstatus.rueckgr.at/.

[107] Supervisor AssocProf Mag Dipl-Ing MICHAEL SONNTAG. "ASTERISK VOIP OVER TOR Computer Science." In: (2019). URL: https://epub.jku.at/obvulihs/download/pdf/4587141.

[108] *StarTrinity SIP Tester™ (call generator) - VoIP monitoring and testing tool*. URL: http://startrinity.com/VoIP/SipTester/SipTester.aspx.

[109] Piyush Kumar Sharma et al. "Proceedings on Privacy Enhancing Technologies ; 2020 (4):69-88 The Road Not Taken: Re-thinking the Feasibility of Voice Calling Over Tor." In: (2020). DOI: 10.2478/popets-2020-0063.

[110] Aleksey M. Klyuchnikov and Igor A. Voronov. "Evaluation of Anonymous Streaming Possibility in Tor Network." In: *Proceedings of the 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, ElConRus 2021* (January 2021), pages 40–43. DOI: 10.1109/ELCONRUS51938.2021.9396141.

[111] Marc Liberatore et al. "Empirical tests of anonymous voice over IP." In: *Journal of Network and Computer Applications* 34.1 (January 2011), pages 341–350. ISSN: 1084-8045. DOI: 10.1016/J.JNCA.2010.06.009.

[112] David Lazar, Yossi Gilad, and Nickolai Zeldovich. "Yodel: Strong metadata security for voice calls." In: *SOSP 2019 - Proceedings of the 27th ACM Symposium on Operating Systems Principles* (October 2019), pages 211–224. DOI: 10.1145/3341301.3359648.

[113] testrtc.com. *testRTC*. URL: https://testrtc.com.

[114] Sajjad Taheri et al. "WebRTCBench: A Benchmark for Performance Assessment of WebRTC Implementations." In: *13th IEEE Symposium on Embedded Systems for Real-Time Multimedia, ESTIMedia 2015*. Institute of Electrical and Electronics Engineers Inc., 2015, page 7351769. ISBN: 1467381640. DOI: 10.1109/ESTIMedia.2015.7351769.

[115] *The dummynet project*. URL: http://info.iet.unipi.it/~luigi/dummynet/.

[116] Eric Rescorla. "RFC 8826: Security Considerations for WebRTC." In: (2021). URL: https://www.rfc-editor.org/info/rfc8826.

[117] David Wells. *Abusing WebRTC to Reveal Coarse Location Data in Signal*. 2020. URL: https://medium.com/tenable-techblog/turning-signal-app-into-a-coarse-tracking-device-643eb4298447.

[118] Alexandros Fakis, Georgios Karopoulos, and Georgios Kambourakis. "Neither Denied nor Exposed: Fixing WebRTC Privacy Leaks." In: *Future Internet* 12.5 (2020). ISSN: 1999-5903. DOI: 10.3390/fi12050092. URL: https://www.mdpi.com/1999-5903/12/5/92.

[119] Nasser Mohammed Al-Fannah. *One Leak Will Sink A Ship: WebRTC IP Address Leaks*. Technical report. 2017. URL: https://webrtc.org.

[120] Richard Pospesel. *Enable webrtc for base-browser (#41021) · Issues · The Tor Project / Applications / Tor Browser · GitLab*. URL: https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/41021.

[121] Sean Liao. "USING TURN SERVERS AS PROXIES." In: (2020). URL: https://rp.os3.nl/2019-2020/p86/report.pdf.

[122] Sandro Gauci. *How we abused Slack's TURN servers to gain access to internal services – Communication Breakdown - Real-Time Communications Security*. URL: https://www.rtcsec.com/article/slack-webrtc-turn-compromise-and-bug-bounty/.

[123] Nadina Ajdinović et al. "Recognition of traffic generated by WebRTC communication." In: *Science, Engineering and Technology* 1.1 (April 2021), pages 15–20. ISSN: 2744-2527. DOI: 10.54327/SET2021/V1.I1.8. URL: https://setjournal.com/SET/article/view/8.

[124] SocialGraph. *social-graph-hero.png (PNG Image, 571 × 405 pixels)*. URL: https://sdk6.getsocial.im/images/social-graph/social-graph-hero.png.

[125] *Data Sources in the Transform Hub - Maltego*. URL: https://www.maltego.com/transform-hub/.

[126] *Understanding IEC 62443 | IEC*. URL: https://www.iec.ch/blog/understanding-iec-62443.

[127] The Tor Project. *Can I change the number of hops Tor uses? | Tor Project | Support*. URL: https://support.torproject.org/misc/misc-11/.

[128] *Tunnel Routing - I2P*. URL: https://geti2p.net/el/docs/how/tunnel-routing.

[129] *Speeding up your I2P network - Blog - I2P*. URL: https://geti2p.net/en/blog/post/2019/07/27/mhatta-post-one.

[130] Maimun Rizal. "A Study of VoIP Performance in Anonymous Network - The Onion Routing (Tor)." In: (). URL: https://d-nb.info/1052682367/34.

[131] w3.org. *Identifiers for WebRTC's Statistics API*. URL: https://www.w3.org/TR/webrtc-stats/.

[132] Deic. *Forskningsnet og tilknyttede tjenester | Danish e-Infrastructure Cooperation*. November 2021. URL: https://www.deic.dk/da/forskningsnet.

[133] Docker. *Docker: Accelerated, Containerized Application Development*. URL: https://www.docker.com/.

[134] Red Hat Ansible. *Ansible is Simple IT Automation*. URL: https://www.ansible.com/.

[135] Karthikeyan S. *Build your first WebRTC app with Python and React*. May 2022. URL: https://www.100ms.live/blog/webrtc-python-react.

[136] coturn. *coturn/coturn: coturn TURN server project*. URL: https://github.com/coturn/coturn.

[137] ObserveRTC. *ObserveRTC/observer: Microservice to monitor and analyze WebRTC stacks*. URL: https://github.com/ObserveRTC/observer.

[138] MongoDB. *MongoDB: The Developer Data Platform | MongoDB*. URL: https://www.mongodb.com/.

[139] Fabric. *Welcome to Fabric! — Fabric documentation*. URL: https://www.fabfile.org/.

[140] https://www.selenium.dev. *Selenium*. URL: https://www.selenium.dev/.

[141] *Basic I2P Tunnels Tutorial with Pictures - Blog - I2P*. URL: https://geti2p.net/el/blog/post/2019/06/02/basic-tunnel-tutorial.

[142] eyedeekay. *eyedeekay/blind-turn*. URL: https://github.com/eyedeekay/blind-turn.

[143] *prometheus/node_exporter: Exporter for machine metrics*. URL: https://github.com/prometheus/node_exporter.

[144] *Prometheus - Monitoring system & time series database*. URL: https://prometheus.io/.

[145] *Master2022E/OnionRTC-experiment: Take your TURN*. URL: https://github.com/Master2022E/OnionRTC-experiment.

[146] The Tor Project. *About – Tor Metrics*. URL: https://metrics.torproject.org/about.html.

[147] The Tor Project. *The Tor Project Onionperf - GitLab*. URL: https://gitlab.torproject.org/tpo/network-health/metrics/onionperf.

[148] The Tor Project. *Performance – Tor Metrics*. URL: https://metrics.torproject.org/torperf.html.

[149] The Tor Project. *Performance (Throughput) – Tor Metrics*. 2023. URL: https://metrics.torproject.org/onionperf-throughput.html?start=2022-12-23&end=2023-01-24&server=public.

[150] The Tor Project. *Performance (Latencies) – Tor Metrics*. 2023. URL: https://metrics.torproject.org/onionperf-latencies.html?start=2022-12-23&end=2023-01-24&server=public.

[151] Adesh Rohan and D ' Silva. "Scaling WebRTC video broadcasting using partial mesh model with location based signalling MSc Research Project Cloud Computing." In: ().

[152] *What is it? - NoScript: block scripts and own your browser!* URL: https://noscript.net/.

[153] *The Shadow Network Simulator*. URL: https://shadow.github.io/.

[154] BifrostConnect. *Technical SPECIFICATIONS of BifrostConnect*. 2021. URL: https://bifrostconnect.com/wp-content/uploads/2021/09/BifrostConnect-Technical-Specifications.pdf.

[155] Balázs Kreith. *Calculating True End-to-End RTT (Balázs Kreith) - webrtcHacks*. July 2022. URL: https://webrtchacks.com/calculate-true-end-to-end-rtt/.

{2.7182818284}ϑφερτυθιοπσδφγηξκλ