UTILIZING IPFS AS A CONTENT PROVIDER FOR DASH VIDEO STREAMING

EVALUATING THE USER EXPERIENCE AND PERFORMANCE OF DASH STREAMING WITH DECENTRALIZED VIDEO CONTENT DELIVERED THROUGH IPFS

ANDREAS ØSTERGAARD NIELSEN, 201303609

ANDREAS MALLING ØSTERGAARD, 201303553

MASTER'S THESIS

June 2018

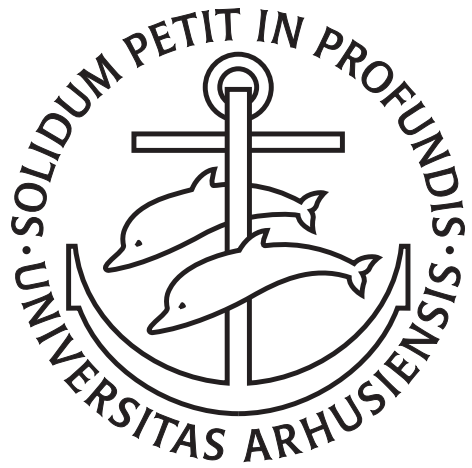
Advisor: Niels Olof Bouvin

AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

UTILIZING IPFS AS A CONTENT PROVIDER FOR DASH VIDEO STREAMING

ANDREAS ØSTERGAARD NIELSEN

ANDREAS MALLING ØSTERGAARD



Evaluating the user experience and performance of DASH streaming with decentralized video content delivered through IPFS

Master's Thesis
Department of Computer Science
Science & Technology
Aarhus University

June 2018

ABSTRACT

The purpose of this study is to determine the suitability of using IPFS as a way to stream videos in a distributed network, with the purpose of offloading high sudden demand that can occur on viral videos.

The study is performed by simulating users with different behaviours and network conditions, in a Docker environment, and having them stream videos in a real DASH video player while using IPFS to retrieve the relevant video files. Results are obtained by having the users report metrics regarding viewing experience and network performance.

A large weakness of IPFS was discovered by the experiments, as the largest amount of data received by each user was duplicate data with a correlation to the number of seeding peers on said data. However the amount of duplicate data fell when sharing larger file segments.

In terms of users, the impact of their behaviour was mostly isolated within themselves, and leeching was punished by a bad user experience. Performance did however decrease for everyone when the amount of leeching users grew large enough.

IPFS was found to have a very high CPU usage, meaning the amount of users that could be simulated was severely limited.

The results show that IPFS is ill-suited for streaming in its current state, as this involves sharing many small files which yields a large overhead of unnecessary data and network traffic. The amount of duplicate data also made IPFS a very poor choice for mobile users as their limited bandwidth would be used on duplicates instead of relevant data.

What one programmer can do in one month,
two programmers can do in two months.

— Fred Brooks

ACKNOWLEDGMENTS

We would like to express our immense appreciation to Niels Olof Bouvin for his support, ideas and great advice throughout this project. His willingness to give his time so generously has been very much appreciated.

We are grateful for the assistance given by Jacob Styrup Bang with providing a *very* powerful virtual machine that made the experiments possible.

We wish to thank Kresten Maigaard Axelsen and Niels Bross for their valuable assistance in reviewing and proof reading this thesis.

CONTENTS

I THESIS 1

1 INTRODUCTION 3

2 RELATED WORK 5
- 2.1 Literature . . . 5
- 2.2 Frameworks and Technologies . . . 7
  - 2.2.1 InterPlanetary File System . . . 7
  - 2.2.2 Dynamic Adaptive Streaming over HTTP . . . 10
- 2.3 Testing Framework . . . 12
  - 2.3.1 Docker . . . 12
  - 2.3.2 Chaos Engineering & Pumba . . . 13
- 2.4 Summary of Related Works . . . 14

3 ANALYSIS 15
- 3.1 Evaluation method . . . 15

4 DESIGN AND METHODOLOGY 17
- 4.1 Frame types and video quality . . . 17
- 4.2 Preparing video for DASH streaming . . . 18
- 4.3 Personas . . . 18
  - 4.3.1 Collective behaviours . . . 19
  - 4.3.2 Individual behaviours . . . 19
  - 4.3.3 Viewing Conditions . . . 19
- 4.4 Evaluation Experiments . . . 20
- 4.5 Summary of Design and Methodology . . . 22

5 IMPLEMENTATION 23
- 5.1 Overview . . . 23
- 5.2 Personas . . . 24
- 5.3 Video Content Tools . . . 25
  - 5.3.1 FFmpeg . . . 26
  - 5.3.2 MP4Box multimedia packager . . . 26
  - 5.3.3 encode.py . . . 26
- 5.4 Experimentation Setup . . . 27
  - 5.4.1 Virtual Machine . . . 27
  - 5.4.2 Docker . . . 27
  - 5.4.3 Pumba . . . 28
  - 5.4.4 IPFS . . . 28
  - 5.4.5 dash.js . . . 29
  - 5.4.6 Python & Splinter . . . 29
- 5.5 Experiment flow . . . 29
  - 5.5.1 Experimental Environments . . . 30
  - 5.5.2 run.py . . . 30
  - 5.5.3 entrypoint . . . 34
- 5.6 Summary of Implementation . . . 36

6 EVALUATION 37
6.1 Baseline experiments . . . 38
6.1.1 Smaller network experiments . . . 39
6.1.2 Duplicate data . . . 40
6.1.3 Seeder Download . . . 41
6.2 Leecher population experiments . . . 42
6.3 Skipper population experiments . . . 47
6.3.1 Skipper configurations . . . 48
6.4 Incognito population experiments . . . 51
6.5 Mobile impact experiments . . . 54
6.5.1 Low bandwidth network . . . 54
6.5.2 High latency network . . . 55
6.5.3 Jittery network conditions . . . 55
6.6 Leaver impact experiments . . . 58
6.7 Socialness impact experiments . . . 59
6.8 Video content influence . . . 61
6.8.1 Increased segment duration experiments . . . 61
6.8.2 Increased bitrate experiment . . . 62
6.9 Public IPFS experiments . . . 64
6.10 Summary of Evaluation . . . 68
7 CONCLUSION 71
7.1 Future Work . . . 72

II APPENDIX 73
A MEDIA PRESENTATION DESCRIPTION 75
B DOCKER COMPOSE FILES 77
C EXPERIMENTAL SETUP OVERVIEWS 81
D ADDITIONAL EXPERIMENTAL PLOTS 85
E SOURCE CODE 87
E.1 Code Repository . . . 87
E.2 Thesis Repository . . . 87
E.3 Docker Images . . . 87
E.4 Getting Started . . . 87
E.5 Video Demonstration . . . 88

BIBLIOGRAPHY 89

LIST OF FIGURES

Figure 1 A Merkle Tree 9
Figure 2 Structure of an Media Presentation Description. 11
Figure 3 Comparison of Virtual Machines and Docker . 12
Figure 4 Frame types used in video compression 17
Figure 5 Diagram of the experimental test setup 24
Figure 6 Sequence diagram for run script pre-processing 31
Figure 7 Sequence diagram for run script post-processing 31
Figure 8 Sequence diagram for run script processing an experiment 32
Figure 9 Sequence diagram for clients 35
Figure 10 Stalls over time from Exp.ID B10 39
Figure 11 Total bandwidth from Exp.ID B10 40
Figure 12 Data recieved from Exp.ID B10 41
Figure 13 Stalls over time from Exp.ID L1B9 43
Figure 14 Total stall time from Exp.ID L1B9 44
Figure 15 Total number stalls from Exp.ID L5B5 45
Figure 16 Stall time per client from Exp.ID L5B5 and Exp.ID L10 45
Figure 17 Mean bandwidth per client comparison between Exp.ID B5 and Exp.ID L5B5 46
Figure 18 Mean stall time for Exp.ID S5B5 and Exp.ID S10 48
Figure 19 Total number stalls from Exp.ID S5B5 49
Figure 20 Total bandwidth from Exp.ID S5B5 50
Figure 21 Total bandwidth from Exp.ID S5B5-c 50
Figure 22 Bandwidth per client from Exp.ID I10 52
Figure 23 Mean stall time from Exp.ID I5B5 52
Figure 24 Mean bandwidth of Binge from Exp.ID B5 and Exp.ID I5B5 53
Figure 25 Total number stalls from Exp.ID B10-m1 55
Figure 26 Total number stalls from Exp.ID B10-m2 56
Figure 27 Stalls over time from Exp.ID B10-m2 57
Figure 28 Received data per client from Exp.ID B10 and Exp.ID B10-m2 57
Figure 29 Mean of received data from Exp.ID B10 and Exp.ID B10-l 59
Figure 30 Mean stalls from socialness experiments 60
Figure 31 Mean of data received from socialness experiments 61
Figure 32 Mean stall time for Exp.ID B10-v9 62
Figure 33 Mean of data received from Exp.ID B10-v9 . . 63
Figure 34 Duplicate data ratio from Exp.ID B10-c18 . . . 64

Figure 35 Total number stalls from Exp.ID B10-p 65
Figure 36 Stalls over time from Exp.ID B10-p 66
Figure 37 Total bandwidth from Exp.ID B10-p 66
Figure 38 Received data per client from Exp.ID B10 and Exp.ID B10-p 67
Figure 39 Total bandwidth from Exp.ID B5 85

LIST OF TABLES

Table 1 InterPlanetary File System Layer Overview. . . 8
Table 2 Experimental Setup of Baseline 38
Table 3 Mean of latencies from Exp.ID B10 42
Table 4 Experimental Setup of Leecher 42
Table 5 Experimental Setup of Skipper 47
Table 6 Experimental Setup of Incognito 51
Table 7 Experimental Setup of Mobile 54
Table 8 Experimental Setup of Leaver 58
Table 9 Experimental Setup of Socialness 59
Table 10 Experimental Setup of Segment size 61
Table 11 Experimental Setup of Public Network 64

LISTINGS

Listing 1 Skipper pseudocode 25
Listing 2 MPD snippet with location addresses 26
Listing 3 MPD snippet with content addresses 27
Listing 4 Docker compose file snippet 28
Listing 5 Steady state env file 30
Listing 6 Experiment env file 30
Listing 7 A full MPD example. 75
Listing 8 Docker compose file for experiment. 77
Listing 9 Docker compose file for plotting. 79

ACRONYMS

API	Application Programming Interface
B-frame	Bi-directional predicted frame
CLI	Command-line Interface
CPU	Central processing unit
CSV	Comma-separated Values
DAG	Directed Acyclic Graph
DASH	Dynamic Adaptive Streaming over HTTP
DDoS	Distributed Denial-of-Service
DHT	Distributed Hash Table
DSHT	Distributed Sloppy Hash Table
FPS	Frames Per Second
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
I-frame	Intra-coded frame
IPFS	InterPlanetary File System
IPLD	InterPlanetary Linked Data
IPNS	InterPlanetary Name Space
kB	Kilobyte
kBps	Kilobytes per Second
kb	Kilobit
kbps	Kilobits per Second
MB	Megabyte
MBps	Megabytes per Second
Mbps	Megabit per Second
MOS	Mean Opinion Score
MPD	Media Presentation Description

ms	Millisecond
NC	Network Coding
OS	Operating System
P-frame	Predicted frame
P2P	Peer-To-Peer
QOE	Quality Of Experience
RNC	Random Network Coding
RTMP	Real-Time Messaging Protocol
RTSP	Real-Time Streaming Protocol
s	Second
SFS	Self-Certified Filesystem
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
UX	User Experience
VM	Virtual Machine

Part I

THESIS

"You live and learn. At any rate, you live."

— Douglas Adams

1 INTRODUCTION

Many videos require a lot of space, and with an evergoing stream of new content, the videos, hosted on sites such as YouTube, only generate a large amount of traffic for a short time, resulting in a potential high amount of access in a short time span, exemplified in viral videos.

This means that a high hosting capacity is needed in both space and bandwidth to meet potential demand and avoid a flooding in page requests that exceeds the resource's available bandwidth or the ability of its servers to respond, and render the resource temporarily unreachable.

This is a somewhat common problem, and is often associated with social platforms such as *Twitter* or *Reddit*, where the latter have named the effect a "hug of death", which is a familiar term on the website.

This report will examine the following hypothesis:

> *By utilizing InterPlanetary File System (IPFS) as a back-end for a Dynamic Adaptive Streaming over HTTP (DASH) video-sharing website, videos are expected to be available in their entirety, viewable without stalling or re-buffering and being persistent, although no adaptive bitrate is utilized. This should be the case for any media and especially apply to circumstances of surging network traffic due to high demand. The increased consumption will allow an increase in availability due to the consumers aiding the sharing by seeding the video.*

A server-less video hosting website will be implemented, by utilizing the IPFS, a content-addressable, distributed file-sharing protocol, for storing and sharing the video content.

The DASH protocol which is used by many major streaming service (e.g., *Netflix* and *YouTube*) to achieve adaptive bitrate streaming, will be tested as the player of the distributed video content. This will be done with simulations of different demands, with different segments of the videos watched and different bitrates for the videos. The simulation will be without multiple bitrate sources of the video content, forcing IPFS to only download from a single stream of video and audio respectively, by relying solely on a single bitrate. The segments should be more available in the Peer-To-Peer (P2P) network due to all clients sharing the same stream, and should also make the results easier to interpret due to the static demand in the network.

The feasibility of the system will be assessed by the User Experience (UX) in the experiments, where as a measurement for UX the focus lies on the viewing experience of the video such as the number of interruptions for needed buffering and the duration of such. This will be done by a number of *Personas* with different use behaviors, to assess the impact a client has on the entire network and the UX of the other clients.

The network setup will be exposed to various network obstacles, such as low bandwidth, high latency and more. This will all be done with inspiration taken from Chaos Engineering, by creating a hypothesis, setting up a *steady state* network and then introducing the variables for the experiment, to either confirm or dismiss said hypothesis.

It is expected that popular content will perform better than lesser known, and thereby lesser watched videos, which should make it suitable to avoid unreachability when a sudden surge of traffic occurs.

2 RELATED WORK

In the following chapter, the inspiration and foundation for this project are presented. Firstly, literature about similar projects is outlined, followed by an introduction to the main technologies examined, and finally a short rundown of the technologies used for experimentation and evaluation is given.

2.1 LITERATURE

Multiple authors have already evaluated DASH video streaming both in relation to other protocols, as seen in Aloman et al. [2], but also in a P2P network as done by Gazdar and Alkwai [7]. In terms of performance of P2P networks, Nguyen et al. [10] have evaluated video acquisition with a Distributed Hash Table (DHT) network and Qiu and Srikant [11] have done a theoretical approach while also listing important aspects to consider when building a P2P network. Since the focus of this thesis is on video viewing performance it is also important to consider how videos are watched, which Broxton et al. [5] have done by defining a socialness factor to videos, based on how viral they are, which affects how a video is watched over time in a P2P network.

Aloman et al. [2] have evaluated DASH (which will be explained in detail in Section 2.2.2) and two other streaming protocols (Real-Time Streaming Protocol (RTSP) and Real-Time Messaging Protocol (RTMP)) for on-demand and live video streaming in both 4G and WiFi conditions. Evaluation is done in terms of Quality Of Experience (QOE) metrics with a large amount of parameters including: Resolution, bitrate, framerate, interval between I-frames (which will be explained in Section 4.1), packet loss rate and delay. QOE was estimated through the quality of service with an algorithmic model, rather than user based scores.

In terms of results, when introducing packet loss and delays DASH sacrifices throughput in order to maintain good QOE whereas the other protocols resort to re-buffering. DASH does however have larger initial buffering time, when starting the video, and as packet loss and delay increases more time is required to fill the buffer initially. They conclude that RTSP is more efficient for starting the video initially but at the expense of packet loss in the video stream played. They claim this is due to RTSP being a User Datagram Protocol (UDP) based protocol where DASH is Transmission Control Protocol (TCP) based. They

claim the extra time buffering is worth it as re-buffering events are almost entirely eliminated. In their results, QOE (measured in Mean Opinion Score (MOS)) is noticeably higher when using DASH in the same scenarios. In their testing, packet loss affected QOE more than delay.

Gazdar and Alkwai [7] have built a streaming system based on the P2P protocol BitTorrent instead of IPFS, while also using DASH as the video player. To accomplish this they needed to make small adjustments to the video's Media Presentation Description (MPD) metadata file by adding information about the tracker. This way the MPD file alone is enough to start the video streaming process.

For tests in the system they focused on segment missing rate and UX, with a variety of quantity of peers, start delay and cache sizes. They observed that the system's performance decreased as the network grew larger, by having more missed segments and more waiting time on playback. The UX also suffered most when viewing at a high quality. These conclusions contradict our hypotheses that a large network should help increase the availability and thereby a client's UX, but this might be due to BitTorrent not being suitable for this purpose, or that their peer selection algorithm is lacking which they themselves point out might be the case.

Nguyen et al. [10] argue that 60% of the commonly used bandwidth cost can be saved using a P2P system as opposed to a client-server model. Their P2P system uses the DHT chord as its network architecture, and the video files are broken into multiple smaller pieces which are then dispersed across peers in the network with redundant copies. To ensure that files are not lost as peers leave and join the network, the new peers share the burden of storing files of the rare pieces, so that it is always possible to reconstruct the video. When streaming a video the system prefers pieces from different peers as this creates multiple internet routes, resulting in increased bitrate. In their testing they store the packets in two ways and compared these. The first is to distribute the packets as is (non-Network Coding (NC)) and the other is having the packets be random linear combinations of the original packets (Random Network Coding (RNC)). The results focused on the probability of being able to retrieve a piece and the latency to do so, and the RNC scheme performed best in both cases.

Qiu and Srikant [11] perform a theoretical evaluation of a BitTorrent network, and list four issues that have to be addressed to understand the system.

Peer evolution is how many peers are in the network and how they act in terms of arrival, departure and bandwidth.

Scalability is how the system's performance should increase as the number of peers grows, which also state that network performance can be measured by average file downloading time.

File sharing efficiency is how the system must be designed such that peers are connected to peers that have the wanted file, and the bandwidth of each peer must be fully utilized.

Finally, the system needs *incentives to prevent free riding*, meaning one should not be able to download files without contributing to the network.

Based on these issues a flow model of a BitTorrent network was constructed, from which they could infer averages of parameters such as seeds, downloaders and download time as functions of arrival rate. It's also shown that BitTorrent had incentives to prevent free riding and a Nash equilibrium existed, meaning it would be optimal for the peers to upload at their maximum speed.

Broxton et al. [5] have analyzed the characteristics of viral videos on YouTube in terms of how such are viewed in respect to a socialness level defined in the article. To qualify as a social video, it has to be viewed through sharing, meaning that access to the video is done through an external link. An unsocial video is instead found through a search or some discovery mechanism within the site itself. The socialness level can then be defined by how many of the videos views come from social sources versus nonsocial. The socialness of a video also evolves over time and the percentage of social views drop over time, e.g. the video is shared more when it is new. It is noted that highly shared videos tend to generate more views than less shared videos. It is also stated that the socialness of a video varies depending on the genre of the video. The behaviour of social views also depend on the source, social views generated through the website *Twitter* drop off harder and peak more in views compared to social views generated through a social network like *Facebook*.

2.2 FRAMEWORKS AND TECHNOLOGIES

In this section frameworks and technologies relevant to or crucial for the project are presented.

2.2.1 *InterPlanetary File System*

The InterPlanetary File System (IPFS) is a distributed P2P file system, as specified in Benet [4] and implemented by Protocol Labs[1].

It consists of several different properties of successful P2P systems such as DHT, BitTorrent, Git and Self-Certified Filesystem (SFS). The IPFS protocol consists of the following sub-protocols, among others.

1 https://ipfs.io/

2.2.1.1 *Identity*

All nodes are identified by a public key of a SHA-256 cryptographic hash. To instantiate a new identity, the node must solve a static crypto puzzle as utilized in S/Kademlia[3].

2.2.1.2 *Routing*

The routing system is capable of finding other peers' addresses in the network and determine which peers serves a wanted object. Using a Distributed Sloppy Hash Table (DSHT) based on S/Kademlia and Coral [3, 6]. A distinction of object's size and usage patterns determines where values are stored. If they are less than 1KB in size they are stored directly on the DHT, otherwise a reference of the identity of peers serving the block is stored instead.

Table 1: InterPlanetary File System Layer Overview.

ROLE	LAYERS	INSTANCES	INSPIRATION
Using the data	applications	dash.js	web
Defining the data	naming	IPNS	SFS
	merkle-dag	IPLD[2]	git
Moving the data	exchange	|	BitTorrent
	routing	libp2p[3]	DHT
	network	|	–

Source: https://youtu.be/HUVmypx9HGI?t=33m12s

2.2.1.3 *Exchange*

Distribution of data happens by exchanging blocks with a protocol called BitSwap, which is inspired by the way BitTorrent exchanges blocks of data. Each peer has a list of blocks they want and a list of what they already have, similar to BitTorrent but instead it is for the entire network.

In BitSwap values are given to each block, meaning to get something from another peer something usually has to be given in exchange. If a peer has little or nothing from the other peer's wantlist it prioritizes blocks wanted by the other over blocks from its own wantlist. This helps reduce the amount of rare blocks in the network.

The protocol also incentivises seeding even when the peer does not currently need any blocks. The idea is that this debt will be repaid in the future, and as such each peer tracks how much they owe, and

2 https://ipld.io/
3 https://libp2p.io/

send more blocks to their debtors. The debt is also used as a measure of trusts, which protects against malicious users.

Nodes also keep ledgers accounting transfers with the other nodes, to track history. When a connection is established Ledger information is exchanged, if it does not match exactly the ledger is recreated from scratch and all accrued trust and debt is reset. The authors claim that malicious users can purposefully lose their debt but this should not be worth the loss in trust.

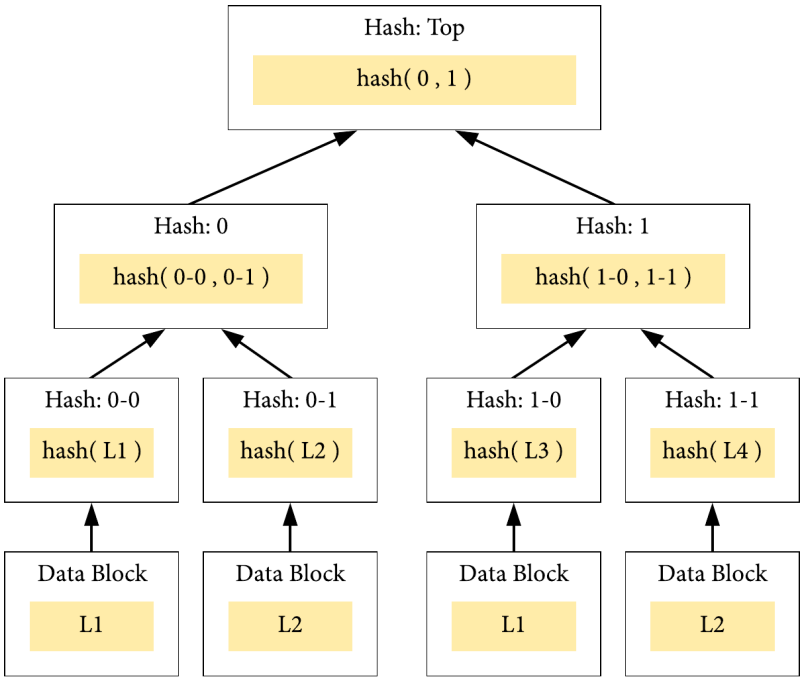


Figure 1: An example of a binary Merkle tree, conceptually like Merkle-DAGs. Hashes 0-0 and 0-1 are the hash values of data blocks L1 and L2, respectively, and hash 0 is the hash of the concatenation of hashes 0-0 and 0-1.

Source: Adaption of `https://commons.wikimedia.org/wiki/File:Hash_Tree.svg`

2.2.1.4 *Objects*

The individual blocks are linked together using Merkle Directed Acyclic Graphs (DAGs), which is a directed acyclic graph where each node has a cryptographic hash, and the hash of the parent verifies the children (See Figure 1). This structure is a generalization of the git data structure and provides the following: Addressing of content using the hashes, tamper resistance since it can be detected and de-duplication as data that holds the same content will also hash to the same thereby only needing to be stored once. From the hash addresses files can also be traversed with a traditional UNIX file system path if the hash is

that of a folder. Since hashes are used to address files, newer versions of a file will hash differently, which means that tracking these is done using additional versioning.

2.2.2 *Dynamic Adaptive Streaming over HTTP*

DASH is an adaptive bitrate streaming technique that enables high quality streaming over Hypertext Transfer Protocol (HTTP). This is done by breaking the content into smaller file segments, each containing a short playback interval. The standard defines that each of these segments can be served at different bitrates, allowing the DASH-client to choose the next segment to download, based on its current network conditions. This makes it possible to serve a stream that adapts seamlessly to deliver streams with high quality and few stalls and re-bufferings, based on the changing conditions of the network.

The segments are described by an MPD file (See Figure 2), in which information about the files, such as timing, Uniform Resource Locators (URLs), bitrates and resolution resides. From the MPD the multimedia selection can then be made based on user preferences (such as specific language audio streams), device capabilities (such as hardware decoding) and the previous mentioned conditions of the network.

The DASH Industry Forum (DASH-IF), consisting of Netflix, Google, Microsoft and more, helps push the implementation of the specification[1]. This is done by creating guidelines for usage and libraries such as `dash.js`[4] that adds DASH compatibility in a Hypertext Markup Language (HTML) video player using JavaScript.

4 https://github.com/Dash-Industry-Forum/dash.js

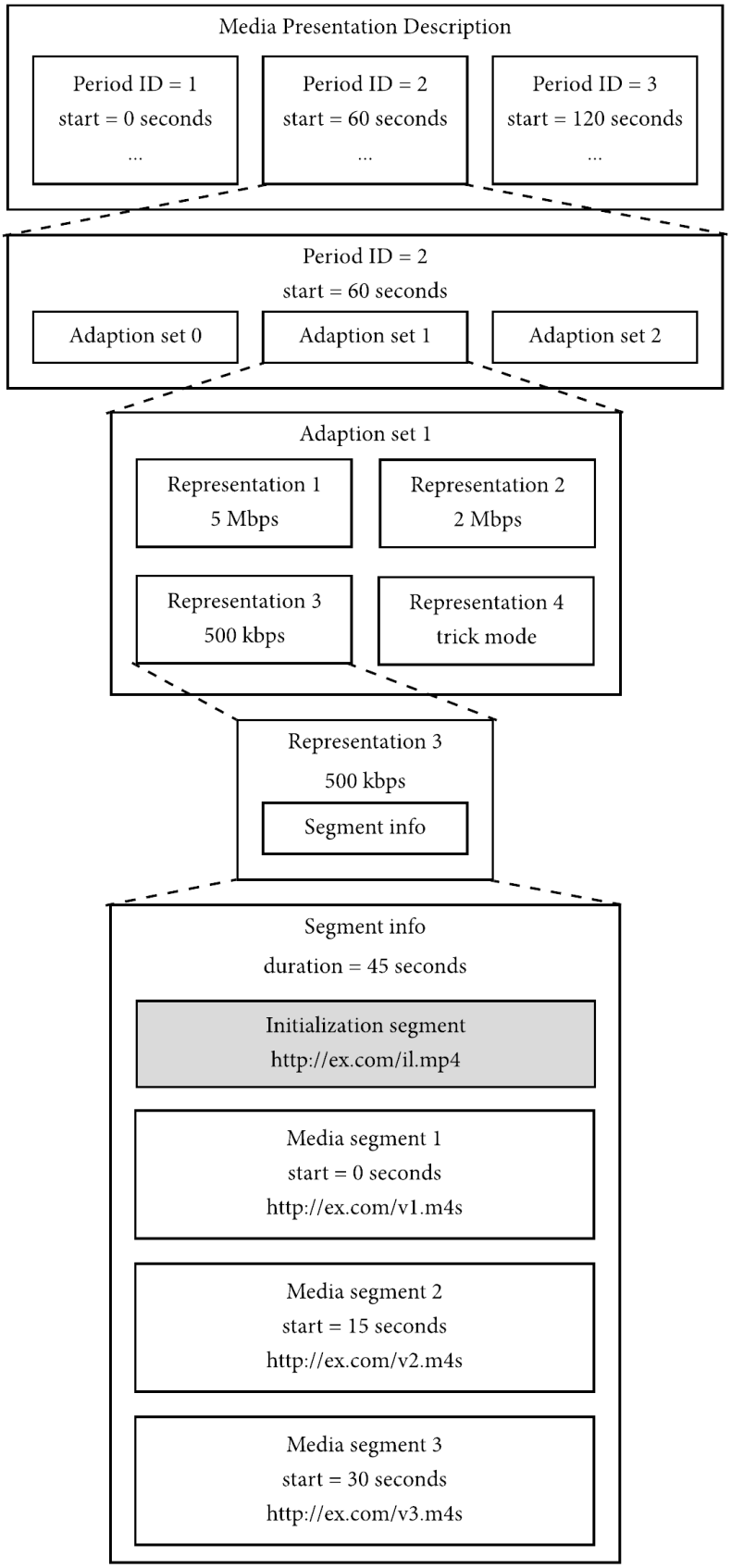


Figure 2: The structure of an MPD.
Source: Adaption of Sodagar [12, Figure 3 on p. 65]

2.3 TESTING FRAMEWORK

In this section the technologies used to create the testing framework are presented.

2.3.1 *Docker*

Docker[5] is a tool that can package an application and its dependencies in a virtual *container* that can run on any Linux server. In contrast to Virtual Machines (VMs), containers share Operating System (OS) and bins/libraries where appropriate, while still being isolated (See Figure 3). This results in lower overhead for each instance running, making it possible to easily scale and run more instances on the infrastructure (See Figure 3).

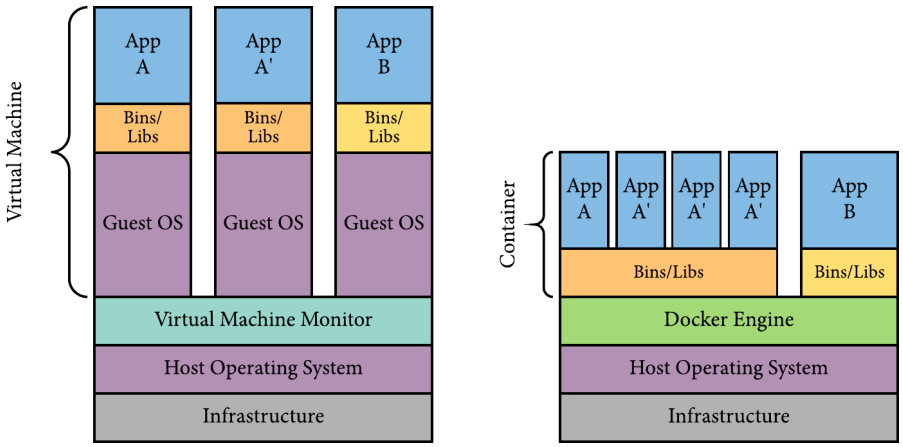


Figure 3: Comparison of the overhead present in virtualization done by a Virtual Machine and by Docker

Individual user spaces are also provided to the containers, meaning that it looks like a real computer from the point of view of the applications running inside. The shared kernel results in less overhead per container when compared to other virtualization approaches, but it is less flexible since it cannot host an OS different from the host machines.

Due to the low resource cost of containers, Docker promotes an application-centric container model[9], so only individual applications or services are run inside, resulting in a loose coupling of the components in the system, as seen with microservices. By running programs in containers hardware independence is achieved due to the abstraction layer of Docker, while also gaining easy access to Central processing unit (CPU) and memory management.

5 https://docs.docker.com/

2.3.1.1 *Glossary*

- *Image:* A Docker image is the basis for the container. It describes changes from a root filesystem and execution parameters to use within a container during runtime. An image can never change as it does not have a state.
- *Container:* A Docker container is the runtime instance of the Docker image. It consists of an image, execution environment and a standard set of instructions.
- *Entrypoint:* An entrypoint of a Docker image is the application to run, when the image is instantiated as a container.
- *Compose:* Docker compose is a tool for defining and running multi-container application, within a single file and command to get it running. Allowing containers to be setup with dependencies of other containers, and to start them in the correct order.
- *Virtual machine:* A Virtual Machine is a program that emulates a complete computer and its hardware, while sharing physical resources with the host computer. Compared to containers a VM is heavier to run as it is more isolated and resource sharing is minimal.
- *Volume:* A volume is a specially-designated directory within the containers or even on the host file system. Volumes are designed to persist data longer than the life cycle of the container using it.

2.3.2 *Chaos Engineering & Pumba*

Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production[6].

The philosophy behind Chaos Engineering is that even though each part in a distributed system works correctly, systemic weaknesses can exist, and to discover these we emulate and manage chaos inherent in the system. This empirical, systems-based approach builds confidence in the ability of these systems to withstand real conditions, through observations of controlled experiments.

Chaos engineering is done by setting up an experiment to uncover systemic weakness. First, a *steady state* is defined by some measureable output, that indicates normal behaviour. The hypothesis is that the *steady state* will continue throughout the experiment. Secondly, variables that reflect real world events are introduced (eg. server crashes,

6 http://principlesofchaos.org/

low bandwidth, spike in traffic, etc.). Thirdly, the hypothesis is tried disproved by examining the *steady state* data. The harder it is to disrupt a *steady state*, the more confidence is gained in the behaviour of the system, and discovered weaknesses can be targeted for improvement, before the behaviour manifests system-wide.

Netflix developed this practice with their program Chaos Monkey which would randomly shut down services, and later expanded to a set of open sourced tools called Simian Army[7] built for different turbulent conditions, but targeted at VM's.

Pumba[8] is a chaos testing tool for containers in Docker. It can manipulate processes, their performance and network.

2.4 SUMMARY OF RELATED WORKS

Multiple authors have already worked with DASH video streaming and P2P networks, with differing kinds of evaluation techniques and focus points.

Aloman et al. [2] evaluated DASH compared to alternative streaming protocols, they evaluated based on on QOE with varying bitrate, I-frame intervals and more. Gazdar and Alkwai [7] built DASH streaming on top of BitTorrent with small modifications to the MPD file format. Their testing focused on segment missing rate and QOE with varying network size, start delay and cache sizes. Nguyen et al. [10] found that bandwidth in the network could be saved by using a DHT. Their testing had files as segments either stored as is, or as a linear combinations of the originals, and they tested the likelihood and latency of getting file pieces. Qiu and Srikant [11] identified issues a P2P system had to handle, based on this they built a flow model for BitTorrent, where they argued that freeriding is discouraged. Broxton et al. [5] analyzed virality of YouTube videos, and defined socialness based on how these videos were shared. They found a correlation between socialness and how viewership evolved over time.

7 https://github.com/Netflix/SimianArmy
8 https://github.com/gaia-adm/pumba

3 ANALYSIS

IPFS is an unexplored technology for the purpose of video streaming. Many have built video streaming services based on P2P technologies such as BitTorrent and DHTs [7].

A common theme is that the videos are split into smaller segments, which can be distributed between peers. An assumption none of the referenced work has made is that some pieces can become rare because they are simply skipped. For example, the last segments of a video could not have been buffered if the user closed the stream prematurely. How the user interacts with the system can vary greatly and is unexplored by other works, therefore it is one of the major focus points of our testing.

3.1 EVALUATION METHOD

Inspired by the following articles from Chapter 2, an evaluation strategy is formed. Aloman et al. [2] had multiple parameters they could adjust on the videos, such as bitrate, interval between key frames and more, and then evaluated different systems based on user experience. But their evaluation only accounted for a single type of user. Their testing was also based on a client-server model rather than P2P, which might impact the performance of the DASH protocol. But according to Nguyen et al. a P2P system would reduce the bandwidth, resulting in better user experience. Their testing focused on probability of retrieving video pieces and latency required to do so. However IPFS does not use a RNC so their results might not be entirely relatable unlike their testing parameters.

Gazdar and Alkwai [7] did make a P2P DASH video streaming service unlike Aloman et al., but rather than making it based on IPFS, they based it on BitTorrent. Their testing was again focused on user experience, while adjusting the network conditions. In their evaluation however they also only had a single type of user. Their results contradicted expectation as they state that they get worse results as the network grew. Nguyen et al. mentions that multiple internet routes should give reduced bandwidth. So it is likely that their peer selection algorithm did not try to use this to their advantage.

These different evaluations were all focused on evaluating the performance and the UXs based on different conditions such as video quality metrics and network conditions. However, none of them went the other way to see how the users behaviours can affect the network and even their own UX.

Broxton et al. [5] defined socialness for videos, but did not build a system to see how socialness of a video can affect the user experience, and neither did any of the above systems. Meaning this is an unexplored territory for testing and is highly relevant as the socialness greatly affects the availability of the video pieces over time.

4 DESIGN AND METHODOLOGY

The goal is to create a web based video player using DASH that retrieves the video files through IPFS. First, the video quality metrics are explained to give an understanding of how videos are constructed (Section 4.1). This is followed by Section 4.2 which describes how the video files should be split into DASH compatible files. Then the intended design used for evaluation is presented in two sections, the first of which is Section 4.3 where user behaviours are presented both in terms of individual users but also collective viewing behaviours. Then in Section 4.4 the intended experiments are presented. The viewing experience is meant to be evaluated in many different network conditions with different viewing behaviours.

4.1 FRAME TYPES AND VIDEO QUALITY

The images of a video is constructed from a series of pictures called frames. Many video codecs (a program or device used for encoding or decoding audio and/or video) use a *group of pictures* frame structure for compression, which consists of 3 different types of frames (See Figure 4).

- Intra-coded frame (I-frame): An independently coded reference frame, containing the entire picture. This frame type can also be referred to as a keyframe.
- Predicted frame (P-frame): Describes motion changes dependent from last reference frame, containing only the differences from the previous frame.
- Bi-directional predicted frame (B-frame): Describes motion changes from last and next reference frame, containing only the differences between these frames.

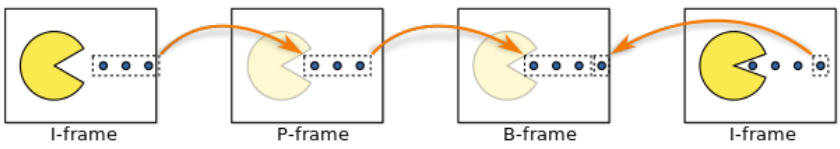


Figure 4: A sequence of compressed video frame types, consisting of two keyframes (I-frames), one predicted frame (P-frame) and one bi-directionally predicted frame (B-frame).

Source: https://commons.wikimedia.org/wiki/File:I_P_and_B_frames.svg

Depending on which type of frame is lost in network traffic, the impact of how many frames are affected varies, i.e. losing an I-frame or a P-frame causes all frames until next I-frame to lose quality, but losing a B-frame only affects that single frame as no other frames are dependent on it.

The size also varies: P-frames are about half the size of I-frames, and B-frames are about a quarter the size of I-frames[1].

These errors are attempted to be concealed during decoding. The portions of the screen where this is done stay in place until the next I-frame or motion changes on that segment of the screen. Videos can also require a re-compression in order to fit into the available bandwidth in which case the image can become blocky or blurry if the compression is too high. Since I-frames contain information on all pixels in the frame, they are always the biggest of the 3 types. Increasing the number of I-frames results in fewer prediction errors, but increases the bitrate due to their size.

4.2 PREPARING VIDEO FOR DASH STREAMING

Preparing a video for DASH streaming (also referred to as dashing) involves creating a MPD file (See Section 2.2.2 and Figure 2) with meta data such as the used DASH profile specification, the duration of the video, the location of the streams, etc..

Splitting up videos allows for better UX, as less of the video has to be loaded before playback can start. The file segmentation is important to enforce the flow of downloaded files through IPFS, due to a request queue of segments generated by the DASH client.

This is possible due to the DASH live profile, which supports multiple files (segments) to form a stream. As a consequence of the segmentation each file should begin with an I-frame, which could mean that a video needs to be re-encoded to have enough "possible split-points".

Though DASH is designed for multiple streams of different bitrates and an adaptive behaviour to maximize the quality of the downloaded content, this aspect will not be examined. Due to the back-end being P2P, a single stream of video and audio is chosen to maximize availability of these files. As part of the hypothesis, it is also expected that a single high quality source on IPFS can be served equally satisfactorily as several streams in a centralized setup.

4.3 PERSONAS

Possible user behaviours can be used in the experiment, varying between collective and individual behaviours.

Collective behaviours affect the entire network and how the video is watched over time.

1 https://en.wikipedia.org/wiki/Inter_frame

4.3.1 *Collective behaviours*

- *Social:* Videos accessed through direct link (shared socially) which are candidates for becoming viral. These videos tend to peak in views and then fall off drastically
- *Non-social:* Videos found through searching or related videos internally on the site. These videos have more stable views over time in comparison to social.

4.3.2 *Individual behaviours*

Behaviours can also vary depending on the individual user. The following are possible kinds of users that could use the system.

- *Seeder:* A user that is sharing content, either being the original poster of said content or re-hosting it after having watched it themselves.
- *Binger:* A user that watches all content in chronological order.
- *Leecher:* A user that tries to exploit the system, by minimizing the content shared with the other peers.
- *Skipper:* A user that watches many small segments of the video, by watching a small segment and then skipping forwards to watch another small segment.
- *Incognito:* A user only available in the network for a short time, watching a single video (or part of video) and deletes all of their content afterwards.
- *Idle:* A user that is inside the network but does not actively interact with video content, and as such instead fulfills the role of padding the other clients DHTs.

4.3.3 *Viewing Conditions*

Behaviours can also vary depending on the network conditions of the user. The following are different conditions that could influence the usage.

- *Normal:* Good bandwidth and low latency.
- *Mobile:* Unreliable due to fluctuating network connections and low bandwidth.
- *Remote:* High latency due to distance.

4.4 EVALUATION EXPERIMENTS

Experiments for measuring QOE, meaning no re-buffering and stalls of streams:

- *Bandwidth:* Both the upload and download speeds of the client can be manipulated, and at which point do these become too low to get a pleasant viewing experience, meaning that the video does not halt after it has initially started. Also how does the low upload affect the other peers, and what influence does fluctuating bandwidth capacity have such as that of a mobile client.

  Given a video with a static bitrate, b, being shared by s seeders in the network, the minimum symmetric bandwidth required for c clients to simultaneously stream the video without stalls is B (See Equation 1).

$$B = \begin{cases} b & c \leqslant s \\ \\ \dfrac{2\,b\,c - b}{s + c - 1} & c > s \end{cases}, \qquad c \geqslant 1, s \geqslant 1 \tag{1}$$

  This is considered the breaking point of the network, where any speeds below B would result in a bottleneck for sharing of the content and thereby worsen the UX.

  Equation 1 states that if the number of clients wanting the resource is less than or equal to the seeders, the minimum bandwidth required must be the bitrate of the resource. If there are more clients, the needed bandwidth per client must be 2 times the bitrate ($2\,b\,c$), due to both upload and download, which is then divided among all peers with the resource, except oneself ($s + c - 1$). Since the data is shared sequentially (starting from a seeder, and then propagating through clients) the last receiving client will not have to share its resource ($-b$).

- *Churn rate:* What effect does clients leaving the network have on the video. Can videos partially or entirely disappear from the network and can a video be popular enough that this cannot reasonably occur. It makes sense to examine the influence of churn rate on IPFS, due to the system's novelty.

- *Segments availability:* How does the socialness and availability of the video impact the user experience and network. The socialness shows the availability over time, while very social videos also have sudden increases in viewership that suddenly drops off. Segments' availability alone can be adjusted by the amount

of seeders in the network and can be considered separate from evaluating the impact of different levels of socialness.

- *Video Quality:* Influence of video resolution, Frames Per Second (FPS), time between I-frames and bitrate as seen in the testing done by Aloman et al.

- *Segment size:* How does different segment sizes affect the performance. It is expected that if it is too large, one does not properly load balance the requests but if it is too small, enough bytes are not given for the amount of traffic trying to locate the segment.

- *Public IPFS:* How does IPFS perform in a small environment dedicated only to running the tests versus running the test in the public IPFS network that is also used for other unrelated services.

- *Population behaviour:* How does the behaviour of the clients in the network affect the network and clients, both themselves and others, with different populations.

These tests are to be performed by setting up a P2P network containing clients with IPFS installed. Each client has a Persona, and tests can be done with different populations of these Personas.

Experiments include:

- Having different percentages of users being Seeders, as large amount of Seeders increases the availability of the video segments, without other clients having to get them themselves before others can.

- Different percentages of users being Leechers. A large amount of Leechers is expected to put a high amount of stress on the network as the amount of peers that request files is disproportionally larger than the amount that can supply them.

- Different percentages of users being idle, this pads the size of the network and can potentially fill up the hash tables of the active peers resulting in slower traffic.

- Different kinds of user behaviours and how they impact the network compared to each other. This can be tested by having similar networks but the active video watching clients vary between them. The users where this would be relevant for are the Binger, Skipper and the Incognito Personas. Between these users stalling is more tolerable on the Skipper and Incognito Persona as they manually jump to a different points of the video and it is expected upon doing so the video will stall while the buffer is being filled with the relevant segments.

4.5 SUMMARY OF DESIGN AND METHODOLOGY

This chapter explained the design of the intended system, where simulated users can view videos through a DASH video player. Video compression consists of three different types of frames, I-frames, P-frames and B-frames. These frame types impact the size and quality of a video. The videos can then be split up into segments using the DASH standard, so clients can retrieve only the wanted segments, thereby potentially reducing the downloaded data size.

The experiments are intended to be performed with a focus of simulating user behaviours, of which there are two types: Collective and individual. The former describes how viral a video is, and the latter how a user interacts with the video player, characterized in the form of a Persona.

The experiments will evaluate the UX gained from the video session by adjusting certain viewing parameters of a user, such as different network conditions, video quality and varying populations of Personas wanting the same video. The experiments are to be done in a simulated P2P network.

5 IMPLEMENTATION

The system is realized through a local network of Docker instances, each running a microservice needed for the purpose of evaluating the system from the point of view of the users. This network has multiple services which are described in Section 5.1 One of these components are the clients watching the video, whose view patterns are determined by their Persona, which are described in Section 5.2. The videos have to be processed into a DASH format suited for streaming, which is presented in Section 5.3. Then the setup for the experiment is presented in two parts. First, the tools that are used (Section 5.4), and second, the sequence of the individual steps that are performed during the experiments (Section 5.5) including how the network is initialized, how data is extracted and more.

5.1 OVERVIEW

The system contains multiple microservices realized in a Docker network:

- `bootstrap` runs an IPFS daemon and is responsible for initializing the IPFS network. All other participants in the IPFS network connect through this node.
- `client` represents the users of the system, and there can be arbitrarily many of them depending on the test. The clients run an IPFS daemon and a browser that is controlled through Splinter. They then play various videos hosted in the network through the IPFS Application Programming Interface (API) from the dash.js video player with different viewing patterns and connection conditions.
- `host` hosts the HTML website that the dash.js player resides on. Each `client` could host this themselves (or even access is over IPFS), but having it a single place makes the system more mutable.
- `metric` is a client to the MongoDB database that is contacted by the client. The clients regularly report data regarding their viewing session to `metric` which then forwards this to the database. This data includes latency for getting a segment, whether the video stalled and more.
- `network` accesses the Docker API to get network stats for all the `clients`. These stats are then stored in the database

- `mongo` is a containerized MongoDB database.
- `plot` is a MongoDB client that processes the data stored in the Mongo database and presents it with various plots. It can also export this to a Comma-separated Values (CSV) format.
- `pumba` is a Chaos Engineering tool that is used to manipulate the `client` instances in terms of their download and upload speed, latency and even shutting them down. While `pumba` could be used as a container, it is opted to use as a binary for ease of automation through scripting.

Relation between these services is also illustrated in Figure 5.

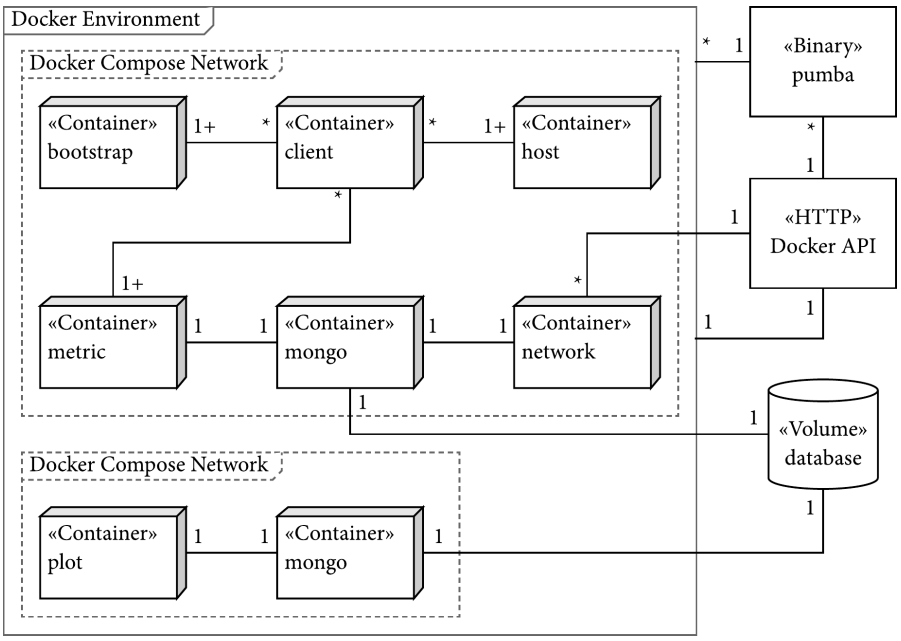


Figure 5: Diagram of the experimental test setup, illustrating the relations between the Docker containers described in Section 5.1.

5.2 PERSONAS

The client viewing patterns are determined by their Persona, with the different types of Personas described in Section 4.3.2. How these Personas are realized will be described in these sections. All Personas function by giving them a hash that acts as a path to an MPD file

- `Idle` simply does nothing actively, but instead just participates in the network, which is implemented by having the client sleep, while the IPFS daemon runs in the background.
- `Seeder` behaves like an Idle Persona, but stores the video that the other peers request. When the Seeder is started it immediately adds the video files to the network.

- `Binger` uses Splinter to view the video for the MPD file. This is done through its local IPFS gateway thereby forcing the IPFS client to retrieve the video and store it in the peer. The data is not pinned, meaning that if the assigned storage runs out the video data will eventually be deleted. After watching the video the user waits until the experiment is over.
- `Leecher` behaves like the Binger Persona but instead of using its own local IPFS gateway it uses one of the seeder's instead. This way it can download the video files without using IPFS to retrieve them and instead does it through the browser
- `Skipper` is like Binger, but instead of viewing the entirety of the video without interacting with it, it watches small segments at a time, by watching a small configurable amount of seconds and then skipping forward another configurable amount of seconds in the video. This pattern is repeated until the end of the video is reached.

  This is also illustrated in the pseudocode (Listing 1), where `skipLength` and `watchTime` are configurable and `timeInVideo` is how far into the video the Persona are (in seconds).

Listing 1: Skipper pseudocode

```
jumpTo = 0
While jumpTo < videoDuration
    skip to jumpTo
    watch video for watchTime seconds
    jumpto <- timeInVideo + skipLength
```

- `Incognito` is very similar to Skipper but instead of watching many segments each a few seconds long, it instead immediately on startup skips to a configurable amount of seconds before the end of the video, that it watches without any further skips.
- `Leaver` behaves like the Binger, but rather than waiting for the experiment to be over when it has finished watching it instead terminates. Resulting in the container hosting the user and its IPFS Daemon being stopped.

The remaining Mobile user Persona is achieved differently as it is not necessarily a different view pattern but instead different network conditions. Mobile user is thus achieved by using one of the above Personas but giving it reduced bandwidth through `pumba`.

5.3 VIDEO CONTENT TOOLS

Using the Python script `encode.py`, video content for testing was generated for sharing and streaming in the experimentation. The following tools were used to generate the content.

5.3.1 *FFmpeg*

The Command-line Interface (CLI) program `ffmpeg`[1] was used for transcoding videos to the proper codecs. The used codecs are H.264 for video and AAC for audio. `ffmpeg` also makes it possible to change the number of I-frames, which could be necessary due to the segmentation of the streams into multiple files in the DASHing process.

5.3.2 *MP4Box multimedia packager*

The tool MP4Box from the GPAC framework[2] was used to split the streams into segment files and generate an MPD.

5.3.3 *encode.py*

The Python script can take a number of options, in which it, among others, is possible to alter the quality (and thereby bitrate) of the produced video, change the number of I-frames and also set the duration of the segments dashed.

The script operates in 4 steps:

1. Encode video and audio to correct media container and codecs using `ffmpeg`.
2. Format the newly encoded media container to the wanted DASH profile of a specified segment length using `MP4Box`.
3. Generate IPFS hash addresses of the formatted media using `ipfs`.
4. Replace location of segments in MPD with hash addresses.

Since the hash addresses always will be the same for the content, it is possible to pre-generate and add the addresses to the MPD before any files are shared in IPFS, no matter which clients adds them later in time (See Section 2.2.1.4 for more information).

Listing 2: Snippet of MPD using the location addresses based on file path. For the MPD in its entirety, see Appendix A

```
25    <SegmentTemplate
26     duration="36864"
27     initialization="Big_Buck_Bunny_video/Segment_0.mp4"
28     media="Big_Buck_Bunny_video/Segment_$Number$.m4s"
29     startNumber="1"
30     timescale="12288"/>
```

1 https://ffmpeg.org
2 https://github.com/gpac/gpac

Listing 3: Snippet of MPD using the content addresses based on IPFS hashing of file

```
25    <SegmentTemplate
26     duration="36864"
27     initialization="
           QmdUhw62spzeGBNFdGRW4UGtLgFCzmhPbzwEXHhYSJJPTR/Segment_0.
           mp4"
28     media="QmdUhw62spzeGBNFdGRW4UGtLgFCzmhPbzwEXHhYSJJPTR/
           Segment_$Number$.m4s"
29     startNumber="1"
30     timescale="12288"/>
```

5.4 EXPERIMENTATION SETUP

The following section describes how the experiments were set up, making it possible to recreate the experiments and generated data.

5.4.1 *Virtual Machine*

Experimentation is performed on a VMware VM with 16 cores[3], 16 GB memory running Ubuntu 16.04[4].

5.4.2 *Docker*

The experimentation is done by emulating clients in a closed network using Docker virtualization. To orchestrate the initialization of the Docker containers, a compose file is used (A Snippet of which can be seen in Listing 4). Within this file the dependencies of the containers are also specified, as some containers cannot work properly without others. For example the Seeder requires the bootstrap, host and the metric services to be running before it starts, which can be seen on line 8-12.

Besides defining dependencies, the compose file is also used for other options. In the case of the Seeder a local directory is mounted, containing the files to be seeded in IPFS (See Listing 4, line 6-7), so if the service `user_seed` was scaled to multiple instances, the video would not need duplication. Finally the command option defines the client behavior, and the Seeder has the default role of a Seeder Persona with 0 seconds start up (See Listing 4, line 13), but with the ability to change the behaviour by utilizing an env file containing the variable `USER_SEED` defined (See Listing 5 or 6 for example). Docker compose checks on run time if a *.env* file is present in the working directory, and if so, substitutes all defined variables in the compose

3 Intel®Xeon®CPU E5-2697A v4 @ 2.60GHz
4 Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-104-generic x86_64)

file. This allows for manipulating the setup at each compose run, by serving Docker compose new environment files.

The VM runs Docker at version 18.03[5] and Docker compose is used on the VM at version 1.20.1[6].

Listing 4: Snippet of a Docker compose file used to define users in the network. Here only the Seeder service (`user_seed`) is shown. For full compose file see Appendix B.

```
version: '3.5'
services:
  user_seed:
    image: andreasmalling/ft_user
    build: ./user
    volumes:
      - ./video_dashed/:/usr/src/app/video_dashed/
    depends_on:
      - bootstrap
      - metric
      - network
      - host
    command: ${USER_SEED:- -r 0 SEEDER}
```

5.4.3 *Pumba*

Pumba is a resilience testing tool used to help with conducting the experiments. It can manipulate the network condition of Docker containers as well as pause and/or stop containers, either chosen randomly or deterministically. Pumba utilizes the Docker API[7] for the resilience testing, tc[8] and iproute2[9] for network manipulation. For the experiments to emulate real world network conditions, the network manipulation is the primary focus of this tool. The VM runs pumba at version 0.4.8[10].

5.4.4 *IPFS*

The implementation go-ipfs[11] is used, though the protocol have been implemented using other languages (Such as JavaScript). This is due to the goal being performance measurements, and the Go implementation is the official reference project of IPFS [12].

---

5 Docker CE version 18.03.0-ce, build 0520e24
6 docker-compose version 1.20.1, build 5d8c71b
7 https://docs.docker.com/engine/api/v1.30/
8 traffic control: https://linux.die.net/man/8/tc
9 https://wiki.linuxfoundation.org/networking/iproute2
10 pumba version 0.4.8, build 537d77d
11 https://github.com/ipfs/go-ipfs
12 https://github.com/ipfs/ipfs/blob/master/README.md#protocol-implementations

The containers run IPFS at version 0.4.14[13] as a daemon in the background. This is tolerable, due to the vision of having IPFS implemented and running natively in the browser.

5.4.5 *dash.js*

The `website` container uses dash.js[14] at version v2.6.8[15], which adds DASH compatibility to the HTML video player using JavaScript, which is needed to accomplish the intended design. The chosen format for the stream is selected from the DASH-AVC/264 standard[8], due to its interoperability and because it is supported by the `dash.js` player. The standard defines the multimedia container as MP4, the video codec as H.264 and the audio codec as AAC. As an addition separate audio and video streams are required, which means that multiplexing is not allowed. The retrieval of video and audio segments is managed by dash.js itself, and no other segment retrieval techniques are implemented.

5.4.6 *Python & Splinter*

The containers run Python at version 3.5.2, for running the entrypoints (See Section 5.4.2) of the containers.

Splinter[16] is a Python library used for emulating user input through a browser. Various Personas (See Section 4.3.2) will be interacting with the website through a Chrome browser (at version 64.0[17]) by utilizing this library, and thereby emulating different behaviours. Splinter is used at version 0.7.7.

5.5 EXPERIMENT FLOW

Experiments are first started by setting up a steady state network of the required services for the experiment. This steady setup is given an initialization time to ensure it is stable, meaning that an IPFS network has formed and seeders have added their files to the network. After a configurable delay the clients are added to the network, and start performing the actions that their Persona and the entrypoint program describe (As outlined in Section 5.5.3 and Figure 9). The experiment then runs for a configurable amount of time that should give enough time for all clients to finish viewing the video. When the time is up every microservice is terminated and the plot service is started to graph the collected data from the experiment (As seen in Figure 7).

13 go-ipfs version 0.4.14, build 5db3846
14 https://github.com/Dash-Industry-Forum/dash.js/
15 dash.all.min.js version 2.6.8, build 888cdcc4
16 https://github.com/cobrateam/splinter
17 Google Chrome version 64.0.3282.140 (Official Build) (64-bit)

The orchestration of this is handled by the run script `run.py`, as described in the following Section 5.5.2 and can be seen in the sequence diagrams in Figure 6, 8 and 7.

5.5.1 *Experimental Environments*

Each experiment is designed by creating two environment files, describing which services should run, which options should they run with and how many instances should be created. The reason two environment files are needed is that one defines the steady state network and the other defines the clients added for the experiment. An example of an environment file can be seen at Listing 5 and 6.

Listing 5: Steady state env file

```
# Steady state scales
SCALE_MONGO=1
SCALE_METRIC=2
SCALE_NETWORK=1
SCALE_HOST=1
SCALE_BOOT=1
SCALE_SEED=5
```

Listing 6: Experiment env file

```
# Client scales
SCALE_USER_1=10
SCALE_USER_3=3
SCALE_USER_6=1

# Client setup
USER_1=-r 60 -v QmXM... INCOGNITO
USER_3=-r 40 -v QmXM... -l BINGE
USER_6=-r 10 -l IDLE
```

The experiments are run with a default steady state network which includes one instance of every role listed in Section 5.1 except `client` and `plot`, which is handled differently. The `clients` are initialized based on their Persona, and only the Seeder Personas are added to the steady part of the network. The Seeder holds the MPD file as well as the video segments in the IPFS network, for the other users to retrieve.

5.5.2 *run.py*

The experiments are executed using the the script `run.py`, which takes the two environment files (See Section 5.5.1) as arguments. The run script consists of three overall stages.

The first stage consists of the database being dropped, should there be any data left from a previous or failed experiment (See Figure 6). This is done by running the `mongo` service solely, followed by a docker-compose exec command which runs a mongo shell command, dropping the database. This has to be done, due to the data being stored on the host machine of Docker, to ease access and ensure persistence.

Since the `mongo` service needs time to start before the command can be executed, we have to try multiple times until we receive a correct return code.

Figure 6: Sequence diagram of run script pre-processing, illustrating the clean up run at the beginning and end of each experiment

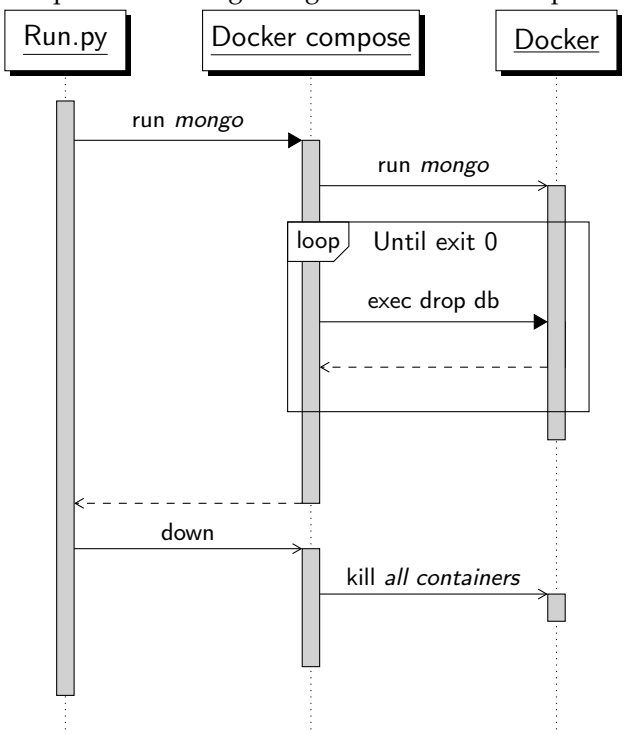


Figure 7: Sequence diagram of run script post-processing, illustrating the plotting and export of data at the end of each experiment

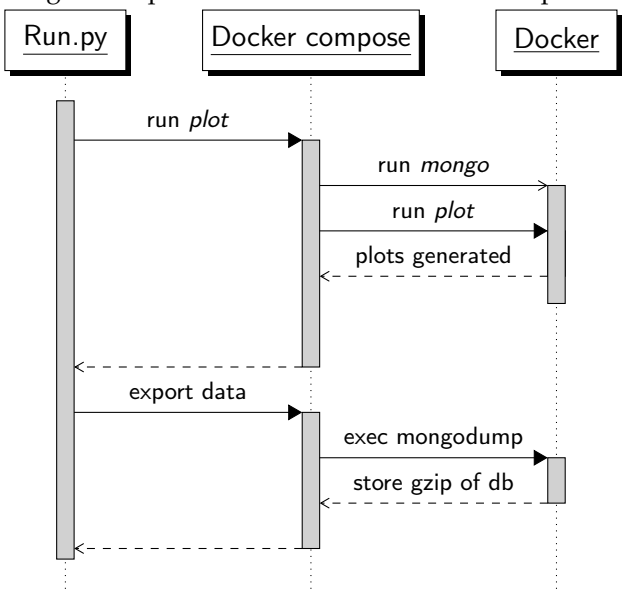
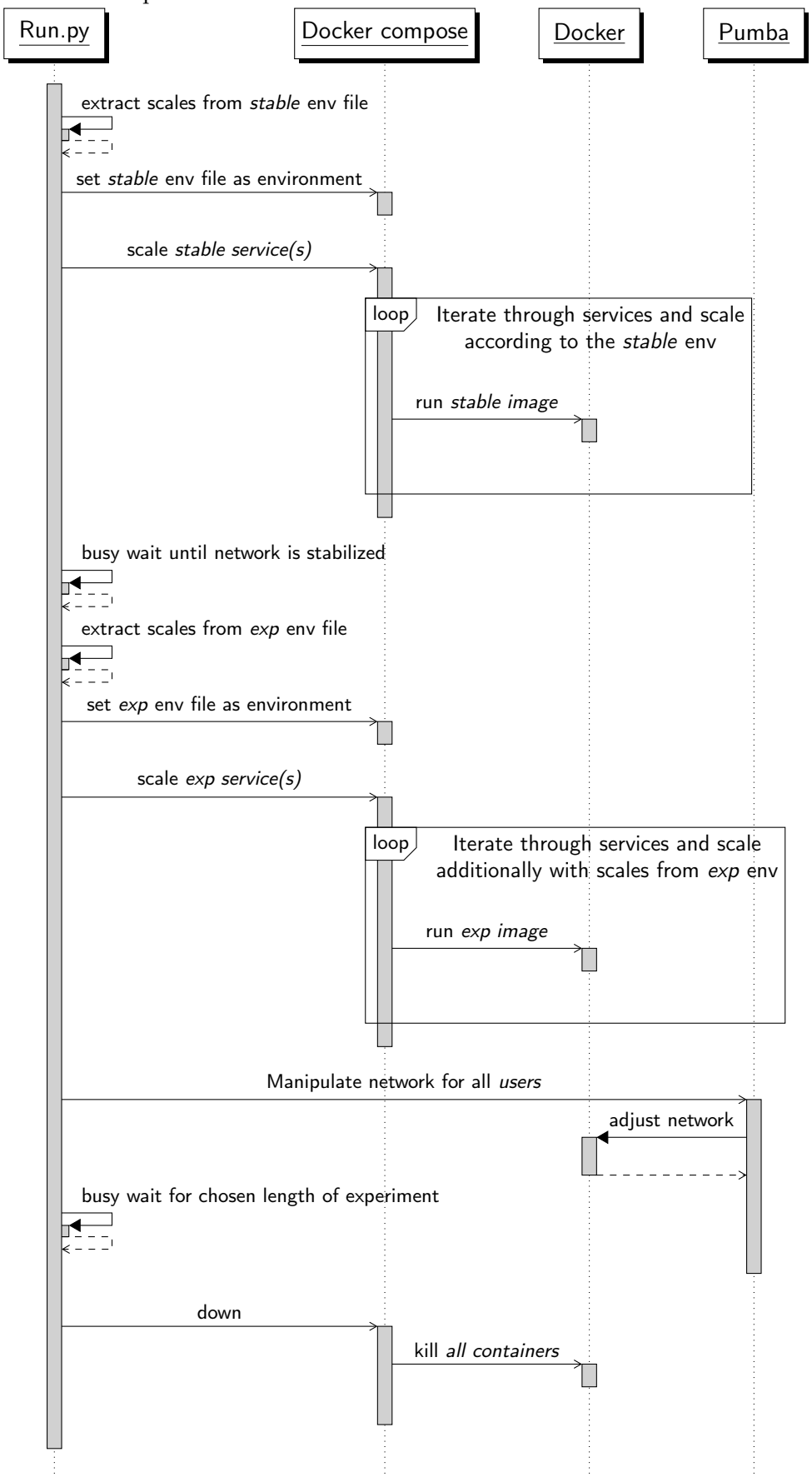


Figure 8: Sequence diagram of run script processing an experiment, illustrating the setup of a steady state network and the introduction of the experimental variables.



Afterwards all services are killed, to stop the mongo instance, avoiding other hanging services interference and ensure a fresh start for the new experiment. This stage is also run at the end of each experiment.

The second stage consists of the set up of the steady state network, followed by the experiment after its stabilization (See Figure 8). First the scales defined in the environment file for the steady network are extracted. The scales are needed by `run.py`, since the scaling is done through the CLI of docker-compose's up command, which has a scale option [18]. As default *run.py* sets the scale of all services to 0, which is then overwritten by environmental files in two sittings.

After importing the scales, the environment file is soft linked to `.env`, which is as an environment for the compose file used by docker-compose[19]. The user behaviours from the environment file is then used along with the compose file by docker-compose when calling the run command on Docker.

When the scaling of the steady network has begun, *run.py* busy waits for a configurable amount of time until all services are running and are stabilized, which is based on empirical data from the VM.

After the stabilization the *clients* are added to measure their influence on the network and their individual UX. This is mainly done with the same procedure as the steady state network, but from another environment file. The only difference in the procedure is pumba (See Section 2.3.2 and 5.4.3) is run to influence the network for all clients, before busy waiting for the length of the experiment.

Finally all services are killed, to stop the data collection of the experiment.

The third stage consists of plotting the results and exporting the data to an archive (See Figure 7). Since `plot` is not a part of the experiment it is defined in a separate compose file (*plot-compose.yml*), which is used with docker-compose to generate the plots. Because the data is kept in a mongoDB, the `plot` service is dependent on the `mongo` service. This effectively means that when docker-compose is asked to run the `plot` service, the dependencies are also started. Since `plot` terminates when all plots are done, we wait for it to exit before continuing.

After the termination of `plot`, the data from the still present `mongo` service is exported to a gzip[20] archive using the utility mongodump[21].

18 https://docs.docker.com/compose/reference/up/
19 https://docs.docker.com/compose/env-file/
20 https://www.gnu.org/software/gzip/
21 https://docs.mongodb.com/manual/reference/program/mongodump/

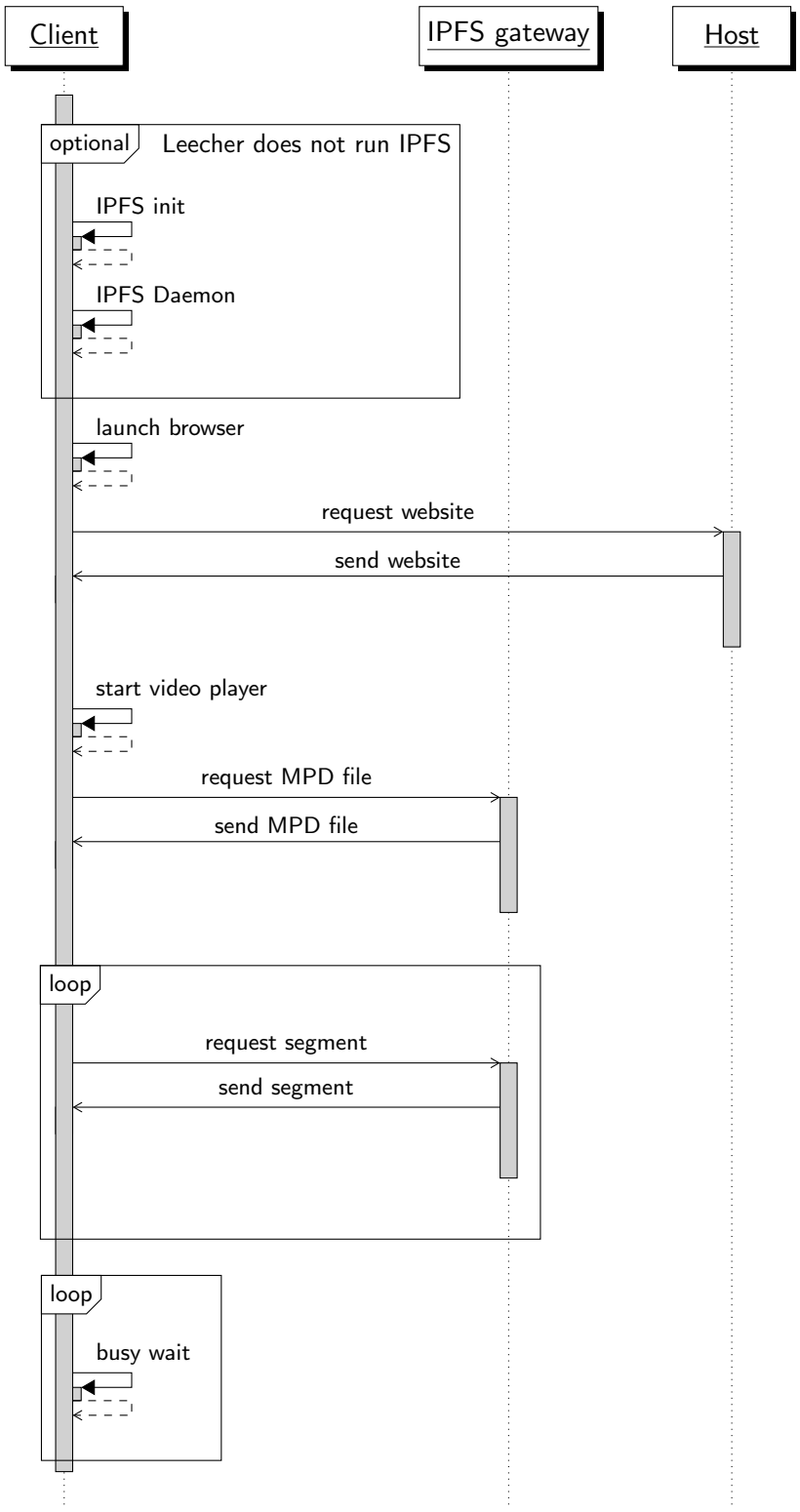
5.5.3 *entrypoint*

The entrypoint is illustrated in Figure 9. The client starts by performing the initialization of the IPFS peer. Here the peer solves a crypto-puzzle in order to generate a peer id, and key-pairs used for secure communication. Next the client starts a local daemon and connects to the peers listed in its list of bootstrap peers, which can either be ones limited to a small network entirely for the experiment or the bootstrap peers that one would use to connect to the public IPFS network. Starting a daemon also starts a new thread that handles IPFS related requests, these are not illustrated in the figure. In order to preserve clarity. These two IPFS API calls are not performed by the clients with the entrypoint Persona, as they do not need it to retrieve the video segments.

The client then launches a Google Chrome[22] browser that is controlled through Splinter and requests the website containing the video player from the Host container. The website is stored in a separate container to ease modifications during development. After receiving the website, Splinter starts the video player, which requests the MPD file from other peers in the IPFS network, through its own gateway, or the Seeder's in the case of the Leecher. After receiving the MPD the video can start and Persona behaviour is then performed. While watching the video the client requests the necessary segments as it needs them until all segments have been received. When the video is finished the browser is then closed and the client either starts busy waiting or terminates. If the client busy waits, the IPFS daemon thread is kept alive, meaning other peers can retrieve data from the client.

22 https://www.google.com/chrome/

Figure 9: Sequence diagram of client requesting video pieces in the IPFS network



5.6 SUMMARY OF IMPLEMENTATION

The realized system contains multiple components, first of these are the Personas that specify how a client acts after video playback has started. These come in multiple varieties:

- Seeder shares files.
- Binger watches video, without any interaction.
- Leecher watches video through the Seeders IPFS gateway.
- Skipper watches the video but skips forward every couple of seconds.
- Incognito user skips to the end of the video.
- Leaver watches the video and then terminates.
- Mobile user has lower bandwidth, which is realized with Pumba.

For the client to be able to watch the videos they need to be encoded. This is done with a combination of `FFmpeg` and `MP4Box`, so that the video is in a format suitable for DASH streaming.

The testing setup consists of a local network of `Docker` containers, where network conditions are throttled by `Pumba`. Each client hosts IPFS and watches the video using the dash.js library, where the user actions can be managed through the `Splinter` library in the Google Chrome browser.

Experiments are run by setting up a steady state network first which consists of the necessary microservices needed for the experiment to work, such as metric loggers, a database for storing the collected metrics and file hosting of the website and video content. Afterwards the clients are added and testing begins for a chosen duration. When the time is up, all clients are shut down and various plots are generated from the experiments metrics, which finally get exported for possible future inspection.

6 EVALUATION

This chapter evaluates the results of experiments based on the implementation described in Chapter 5.

All referenced experiments in the following sections are outlined in a table by category. All tables can be found at the beginning of the section and in Appendix C. Unless else is stated in the tables, a default steady state network is used for the introduction of the experiment (See Section 5.5.1). The default network consists of numerous microservices: `bootstrap` used for a local IPFS network, `host` for serving the website, `network` for measuring transfer rates, `mongo` for storing collected data and `seeder` for seeding the MPD as well as the video segments in the IPFS network (See Section 5.1 for all roles).

All experimental clients introduced to the steady state network, are introduced in a uniformly randomized way, with a default delay ranging from 0 and 60 Seconds (s) before participating actively in the network. The bandwidth rate for the clients is standard at 20 Megabit per Second (Mbps) (unless other bandwidth is stated) and with no artificial latency added on a closed IPFS network, meaning that the only peers present are set up by the experiment.

All clients (except Leechers) use their local HTTP gateway to send requests for retrieving the video segments via their own IPFS daemon. This is considered the least complex and most desired usage of the system, and is therefore used as a baseline for the other experiments.

The video used in all experiments is Big Buck Bunny[1], shortened to show only the first 90 s of playback and thereby making the experiments faster to run. The video is by default divided in a segment size of approximately 3000 ms resulting in 30 segments of video data plus an initial container file. All segments amount to the size of 55.99 Megabytes (MB), yielding an average bitrate of 4919 Kilobits per Second (kbps), with individual segment sizes ranging from below 450 Kilobytes (kB) to as high as 4.1 MB.

Scalabilty is limited to 10 clients due to high CPU usage, even though the experiments are run on a powerful VM (See Section 5.4.1). This is done to avoid *false* stalls, with thread scheduling as the cause. During testing each user used around 100 percent CPU (out of 1600 percent). 40 to 60 percent of this was due to the Chrome browser viewing the video, and 30 to 40 percent was because of the IPFS daemon. The seeders daemon used less CPU at around 10 to 20 percent. Having a substitute to the DASH player that only retrieved the pieces without viewing them could potentially double the number of users,

1 https://peach.blender.org/download/

but this has been omitted, since this would no longer evaluate the DASH component, and the evaluation would be reduced to just how fast the system could retrieve the segments. This feature could be added as future work.

6.1 BASELINE EXPERIMENTS

A Binger client is a user that watches a video and does not interact with the player after the video is started.

Table 2: Experimental Setup of Baseline experiments

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B1	1 Binger is introduced to the default stable network. Network bandwidth is at 20 MBps.
B5	5 Bingers are introduced to the default stable network. Network bandwidth is at 20 MBps.
B10	10 Bingers are introduced to the default stable network. Network bandwidth is at 20 MBps.

The first experiment consists of the steady state network, which is introduced for 10 Binger clients after stabilization (Exp.ID B10). It is expected to perform well with almost no stalls or failures, and segments are expected to be retrieved with low latency. Clients are expected to download about 56 MB which is the size of the video, plus a small overhead of DHT communication and duplicate data.

In the experiment (Exp.ID B10) every client stalled exactly once (See Figure 10), which is expected when the video was initially started and the buffer was getting filled. This can therefore not be considered a failure since an initially empty buffer is unavoidable in the system. From this it is concluded the Bingers had a good QOE, due to a session with the entire video and without interruptions for re-buffering.

In terms of the amount of data received by the clients, only a single peer downloaded about the expected amount, while every other client downloaded increasingly more. In the worst case about 7 times more (See Figure 11). This is very unexpected and undesirable, due to the expended bandwidth used on redundant data.

Since the video segments are the only files being distributed in the entire IPFS network, the extra data downloaded must be duplicate data, which can be confirmed from the IPFS CLI[2], in which a metric for duplicated data download through the block exchange can be accessed. It is also observed that the ones downloading the least are also the clients that get to watch the video first, where the files are

2 `$ipfs stats bitswap`

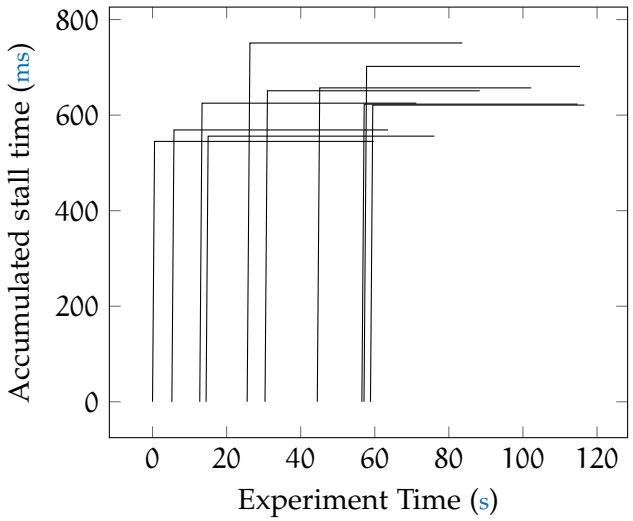


Figure 10: Accumulated stalls during playback of the video over time, from the 10 Binger baseline experiment (Exp.ID B10).
The clients stall at the beginning and afterwards not at all which signifies a good UX.

least distributed since initially only the Seeder is seeding the content, whereas the number of seeding clients increase with the run time of the experiment (See Figure 12). This indicates a correlation between the number of seeding clients and the overhead of duplicate data, which is very problematic.

As a consequence of this, the overhead seemingly increases with the number of seeders, potentially making the consumption of bandwidth by redundant traffic in the network throttle relevant and time critical exchanges of needed segments and thereby making the system unscalable due the narrowing bottleneck as a result of high availability.

In terms of latency in acquiring the files, it varies greatly between each segment and no pattern can be observed, but for the average latency of every client the best performing client has an average of 1053 ms and the worst has an average of 912 ms (See Table 3), making it seem evenly distributed.

In spite of the worst performing Binger client downloaded the data 7 times, it does not seem to impact the clients UX negatively. This might however change, as it is expected that the network could get swamped if this factor increases conjointly with a resource's seeders.

6.1.1 *Smaller network experiments*

The baseline experiment is also repeated with smaller amounts of Binger users in the network, respectively from 1 to 10 Bingers introduced to the default network. Those experiments are later referenced for comparison as baseline can be found in Table 2.

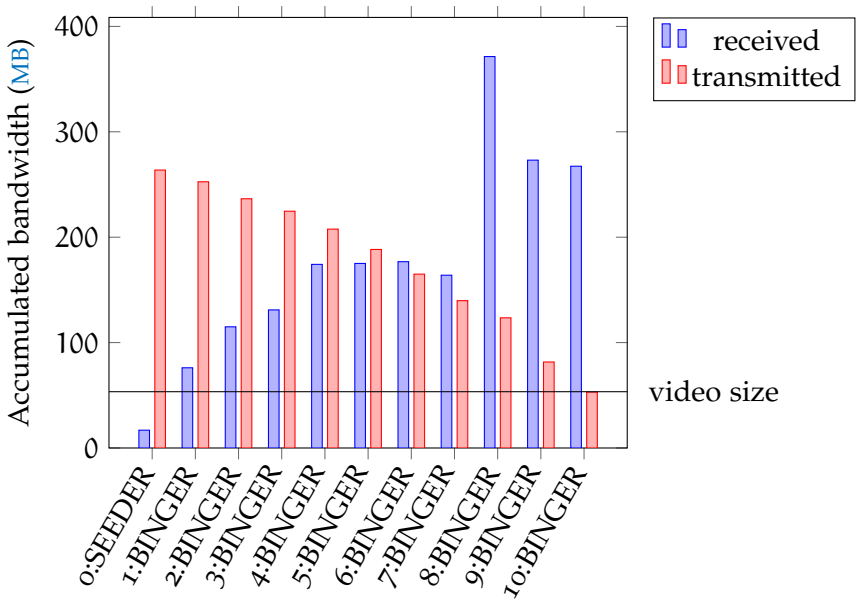


Figure 11: Recieved and transmitted bytes per client, from the baseline experiment (Exp.ID B10).
Download overhead is as large as factor 1:7 as exemplified by *8:BINGER* receiving 389 MB while asking for a 56 MB video.

While these experiments primarily were done for comparison of future setups, they support the theory of a correlation between duplicate data and seeding clients, since same behaviour was observed, but with a smaller maximum overhead depending on the number of peers (See Figure 39 in Appendix D for example).

6.1.2 *Duplicate data*

Through the baseline experiments, a correlation between number of peers seeding a resource and the size of the duplicates received when requesting said resource. This is a reported bug on the go-ipfs github[3], where several others experience same behaviour as observed in the baseline experiments.

This bug makes IPFS ill-suited for the proposed purpose of effectively serving videos with high demand. In a typical P2P network, resources being downloaded, distributes them and makes it more widely available to the rest of the network. On the other hand IPFS also distributes the resource, but this currently results in a penalty of redundant data, which increases with the number of seeders of the resource in the network, thereby cancelling the positive effect of the increased throughput.

Since the UX in spite of the the bug did not suffer, the planned experiments will still be evaluated. It is also possible that the bug will

3 https://github.com/ipfs/go-ipfs/issues/4588

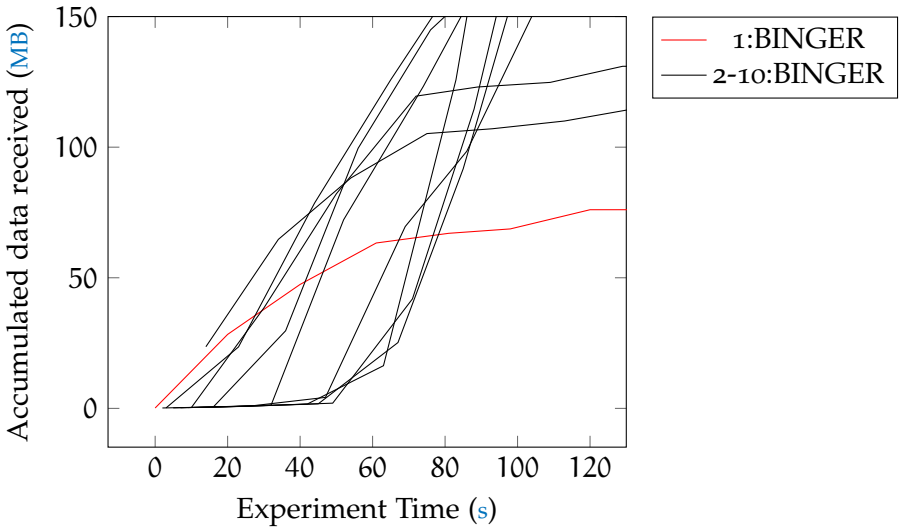


Figure 12: Accumulated data received over time from the baseline experiment (Exp.ID B10).
The Binger that started first (*1:BINGER*) had to download less data compared to the other Bingers.

not be persistent and an improvement in performance is observed, since the baseline experiment suddenly can be seen as a worst case scenario, as it contains most well behaving and seeding clients. Despite the expectation of the setup being ideal for performance.

Due to this bug no experiments, where the amount of seeders are scaled, are performed, as this would simply increase the amount of duplicates the users have to receive from, resulting in worse performance.

6.1.3 *Seeder Download*

In the baseline experiment (Exp.ID B10) the Seeder client downloads over 16 MB (See Figure 11), which ideally should be close to 0 as it should not request any data. The downloaded data is communication between peers over the duration of the experiment, but the download primarily happens over the 2 minutes in which the Binger clients requests the video, yielding an average download rate of ≈124 Kilobytes per Second (kBps). Using the built in metric tool from IPFS' CLI[4], there are no reports of the Seeder downloading any blocks of data.

Since the downloaded data reported both by the CLI and externally from the `network` service confirms this surprisingly high estimate. The downloaded data must be communication in the DHT considering nothing else is running in the container.

4 `$ipfs stats bw` for bandwidth and `$ipfs stats bitswap` for block exchange

Table 3: Mean of latencies of segment downloads per client, from the baseline experiment (Exp.ID B10). Measured across 30 segments. Clients that started later have lower latency, although not by much.

Clients	Latency (ms)
1:BINGER	1,053
2:BINGER	1,285
3:BINGER	1,084
4:BINGER	1,115
5:BINGER	1,110
6:BINGER	996
7:BINGER	1,005
8:BINGER	912
9:BINGER	756
10:BINGER	745

6.2 LEECHER POPULATION EXPERIMENTS

The Leecher acts just like a Binger user, but rather than using their own IPFS gateway, they retrieve segments directly from the gateway of the seeder.

Table 4: Experimental Setup of Leecher population experiments

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
L1B9	1 Leecher and 9 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.
L5B5	5 Leechers and 5 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.
L10	10 Leechers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.

In this experiment we alter the percentage of Leechers versus Bingers that the network contains, otherwise the conditions are the same as the baseline experiment (See Table 4 for overview). Based on the first experiment adding Leechers to the network should decrease the amount of duplicate data being sent, but it also means more requests to the holders of the video segments potentially as there are fewer

sources to get them from. Therefore the network is expected to perform better as long as the Seeder can keep up.

The results however were different than expected. Having a single Leecher in the network, had no real impact on the Binger clients, but the Leecher clients had very high stall time (See Figure 14), with over 17 s stalls spread across the video rather than just at the beginning (See Figure 13).

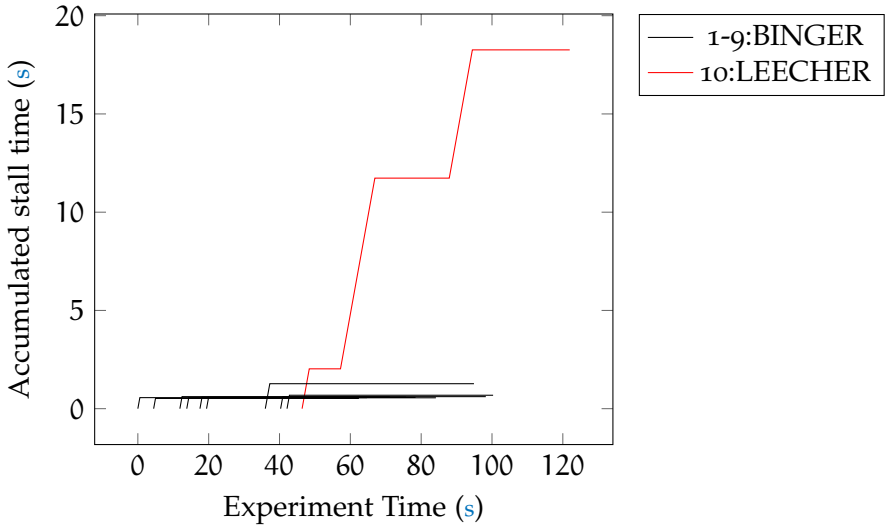


Figure 13: Stalls over time from the 1 Leecher and 9 Bingers experiment (Exp.ID L1B9).
The Leecher stalls multiple times during the experiment and not only at the beginning like the Bingers.

Scaling the network to having half of the users being Leechers and the other half Bingers (Exp.ID L5B5), negatively impacted all the clients, as the 5 Bingers now stalled multiple times (See Figure 15), and had an average stall time of about 13 s while the stall time for the Leechers became much worse with an average of about 53 s. For both types of clients the standard deviation of stall time was around 10 s (See Figure 16), meaning the users got vastly different performances, and even two of the Bingers only stalled once at the beginning of the video, meaning they got a good quality of experience.

Scaling the amount of Leechers further, to all the clients being Leechers, increased the bad quality of experience further, and the total mean stall time became 100 s.

These results show that leeching off of a public IPFS gateway is punished compared to the users retrieving the files themselves and using their own gateway and that the tit-for-tat principle is enforced. Unfortunately Leechers still impacted the well behaving Binger users's quality of experience as they experienced more stalling as the percentage of Leechers in the network grew.

In terms of data downloaded Leecher users were much more efficient, in the half Binger half Leecher experiment. The Leechers only

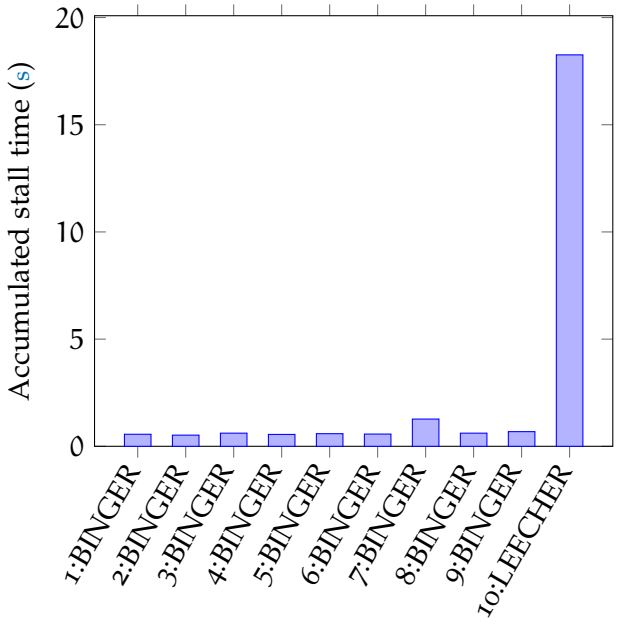


Figure 14: Accumulated stall time per client, from the 1 Leecher and 9 Bingers experiment (Exp.ID L1B9).
Only the Leecher stalls for a long time, while Bingers barely stall at all.

downloaded on average of about 60 MB which is only slightly higher than the file size of the video files. While the Bingers on average downloaded about 140 MB, which is more than double of what the Leechers did (See Figure 17).

Having a network of only 5 Binger users in comparison gave an average of about 120 MB of downloaded data, as both cases had fairly high standard deviation this is within the margin of error. But adding Leechers to the network does not seem to harm the Binger users in terms of the amount of data they have to download but instead it might lower it as they get less access to the Seeder, meaning they have to rely on the other users more.

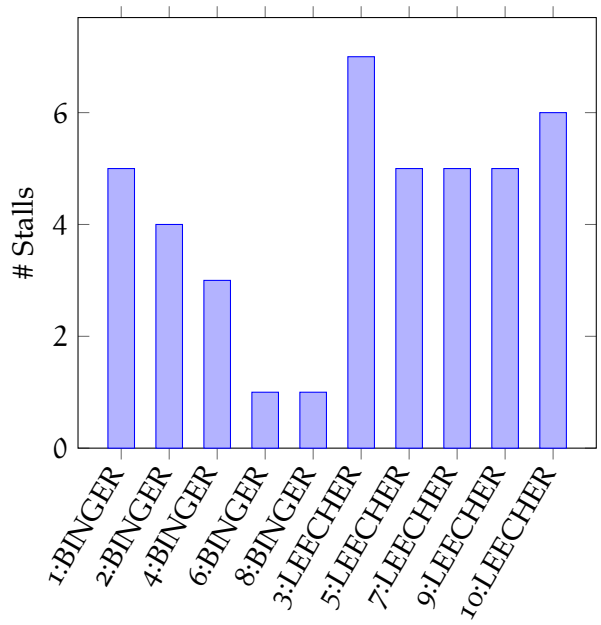


Figure 15: Total number of stalls per client, from the 5 Leechers experiment (Exp.ID L5B5).
Leechers stall more, but also makes some of the Binger clients stall, which they did not do when Leechers were not present.

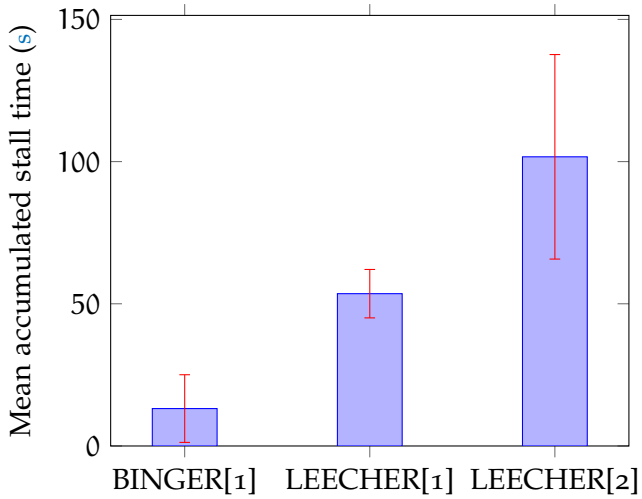


Figure 16: Mean of the accumulated stall time for each type of client for:
[1] is a network with 5 Leechers and 5 Bingers (Exp.ID L5B5,
[2] is a network with 10 Leechers (Exp.ID L10).
Adding more Leechers gives more stall time for each Leecher, while Bingers are less affected.

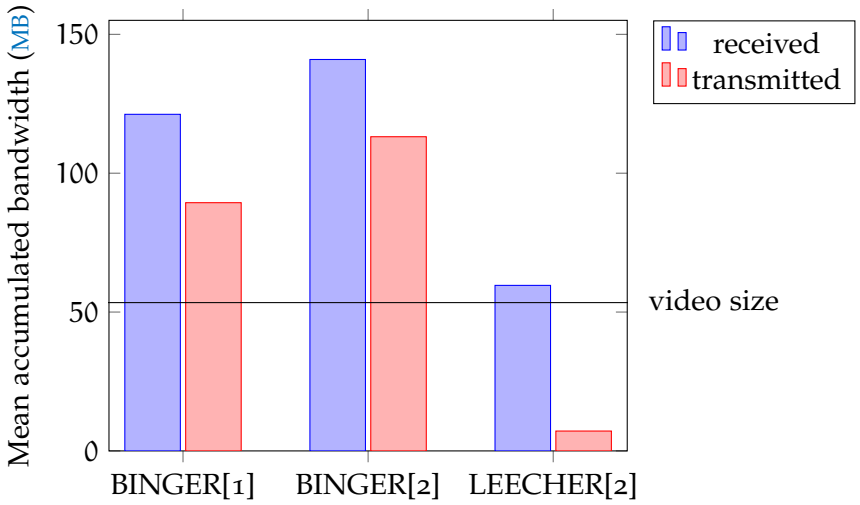


Figure 17: Comparison of the mean of accumulated bandwidth for each type of client, for:
[1] a network with 5 Bingers (Exp.ID B5),
[2] a network with 5 Leechers and 5 Bingers (Exp.ID L5B5).
Leechers are more efficient in the amount of data they download while Bingers downloads duplicate data. Bingers had very similar amount of received data regardless if there are Leechers present or not.

6.3 SKIPPER POPULATION EXPERIMENTS

This experiment has another client behaviour in the Skipper Persona. The Skipper Persona user acts just like the Binger user, but they interact with the video player while the video is playing. They repeat the following behaviour, watch the video for a set amount of time (referred to as watch time), and then skip forward in the video a set amount of time (referred to as skip time).

Table 5: Experimental Setup of Skipper population experiments

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
S5B5	5 Skippers and 5 Bingers are introduced to the default stable network over 60 s. Skipper watch time is 3 s and skip time 10 s. Network bandwidth is at 20 MBps.
S5B5-c	5 Skippers and 5 Bingers are introduced to the default stable network over 60 s. Skipper watch time is 1 s and skip time 25 s. Network bandwidth is at 20 MBps.
S10	10 Skippers are introduced to the default stable network over 60 s. Skipper watch time is 3 s and skip time 10 s. Network bandwidth is at 20 MBps.

The goal is to see what kind of impact the Skipper has on the network as a whole and if it can negatively impact a Binger. We also want to see what kind of performance a Skipper gets when watching a video, the video should stall every time the client skips in the video. But the amount of time spent in the stalling state should ideally be small, but based on the latency seen in the baseline experiment, it is likely that it stalls for more than a second after every skip as this is the latency seen when acquiring segments. The configuration of the Skipper is to have a watch time of 3 s and a skip time of 10 s. If the only stalls are because of skips, they should have exactly 7 stalls each. In the experiments Skipper users had around 7 stalls (See Figure 18), meaning they did not stall more than they were required to do by the Persona behaviour, the amount of skippers in the network did not impact the amount of stalls. The time spent stalling is also unaffected as can be seen from the experiment with 5 Skippers versus the one with 10. The average time spent stalling for Skipper Personas were about the same.

The quality of experience for the Bingers also seemed unaffected, as they like the baseline experiment still only stalled at the start, and infrequently once more after that (See Figure 19).

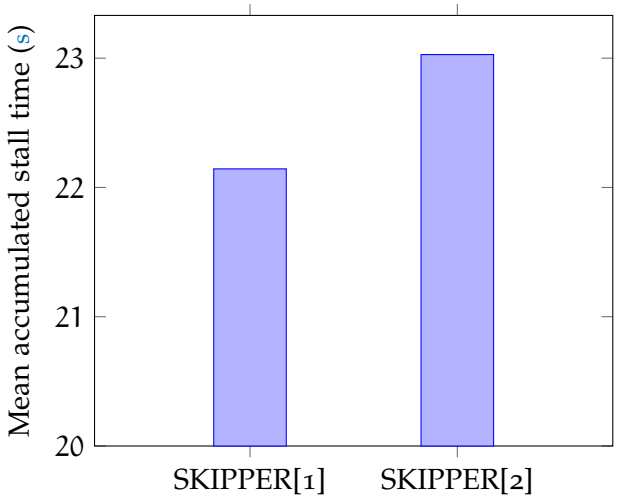


Figure 18: Mean of accumulated stall time for:
[1] a network with 5 Skippers and 5 Bingers (Exp.ID S5B5),
[2] a network with 10 Skippers (Exp.ID S10).
Stall time is almost identical regardless of the amount of Skippers.

Although the Bingers did tend to download more data than the Skippers (See Figure 20), this can be excused to the Skippers going ahead in progress in the video faster than the Bingers. Meaning they will have the segments before and as the Bingers reach that point in the video and requests segments relevant to it. Those segments are more duplicated in the network resulting in downloading more duplicates.

In conclusion Skippers have very little impact in relation to the network and the worsened performance is isolated within themselves with little to no negative impact on the other users in the network.

6.3.1 *Skipper configurations*

The Skipper can be configured in both the amount of time it watches videos in between skips, and how far forward it skips, which is tested in a single network configuration in this experiment.

The watch time is expected to have little impact as the dash.js player buffer should be about full at all times. But adjusting Skipper length means that potentially more of the buffer has to be replaced and if the Skipper length is set high enough, the entirety of the buffer is replaced.

Increasing the skip length and decreasing watch time (Exp.ID S5B5-c) had very little impact. Bingers still only stalled around a single time, and skippers still only stalled when they were performing the skip. The only notable change is that a Skipper managed to only download 36 MB (See 2:SKIPPER in Figure 21) when watching for 1 second and skipping 25. Meaning they skipped enough to avoid

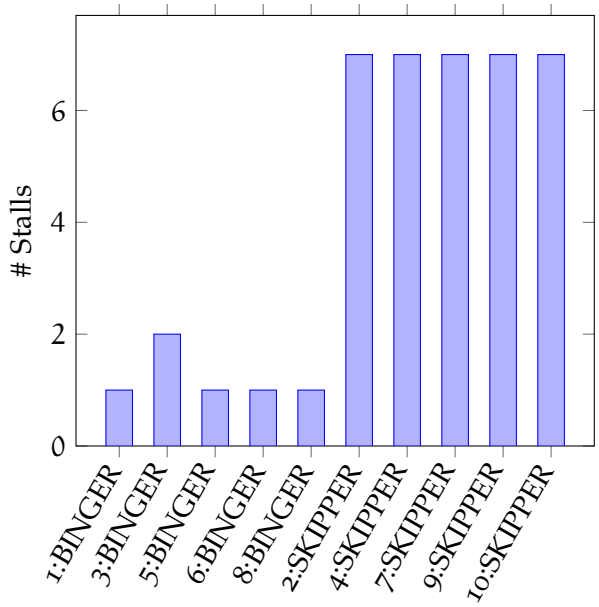


Figure 19: Total number of stalls per client, from the 5 Skippers experiment (Exp.ID S5B5).
Skippers only stalled when they were forced by a skip, meaning they should have 7 skips, if no other stalls occur. The only client deviating from the optimal number of stalls is *3:BINGER*, which encounters a single re-buffering event after the initial buffering.

downloading some segments, which also means that the previous buffer was unusable after the skip.

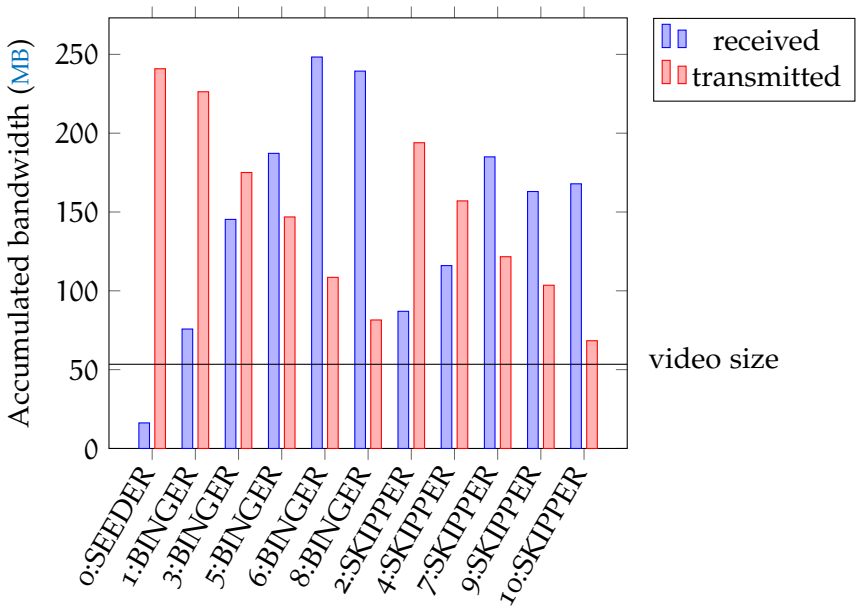


Figure 20: Received and transmitted bytes per client, from the 5 Skipper experiment (Exp.ID S5B5).
Bingers tended to download more than Skippers.

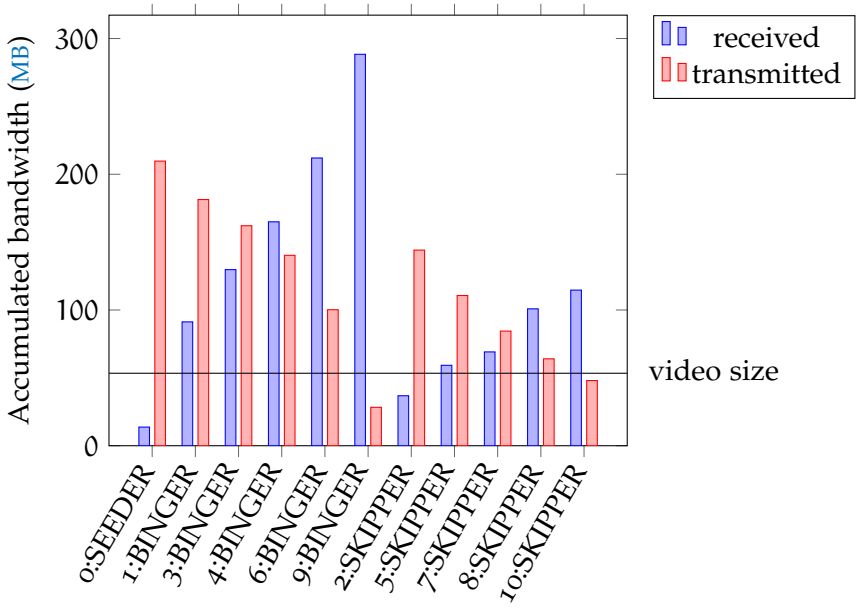


Figure 21: Received and transmitted data per client, from the 5 Skipper experiment with the configuration 1 second watch time and 25 seconds skips (Exp.ID S5B5-c).
The client *2:SKIPPER* downloaded less than the video size, meaning that some video segments were not downloaded.

6.4 INCOGNITO POPULATION EXPERIMENTS

This experiment has the Incognito behaviour in the network. The Incognito Persona user acts just like the Binger, but it immediately skips to the point in the video that is 10 s before the end, and continues watching normally from there.

Table 6: Experimental Setup of Incognito population experiments

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
I5B5	5 Incognito users and 5 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.
I10	10 Incognito users are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.

This experiment is expected to only impact a small amount of segments of the video, which are the segments that are watched by the Incognito users. For the rest of the segments we should see similar results to the baseline experiment with equal amount of Bingers. Based on the experiments with the Skipper user, the Bingers are not expected to be affected by the Incognito user in terms of quality of experience.

In the experiment Incognito users performed like Bingers that just watched a shorter video. They stalled exactly once like Bingers, and users that watched segments late had to download more like Binger users (See Figure 22).

Unlike the Binger users the initial stall was much longer, in the 5 Bingers and 5 Incognito users experiment. Bingers stalled for an average of about 0.4 s while Incognito was at 2 s (See Figure 23). This delay might be because of the Incognito user initially downloads the first segments of the video and then skips to the end of the video and starts downloading new ones. The dash.js player also has to determine which segments correspond to the specific timestamp, which can add to the stalling time.

In terms of downloaded data the Binger users had to download slightly more compared to the baseline experiments, for instance 5 Bingers with 5 Incognito users meant that the Bingers downloaded on average 145 MB, compared to having 5 Bingers alone downloading 84 MB (See Figure 24). This can be explained by the later segments of the video being more available, leading to an increase in duplicate data, which is in line with the other results.

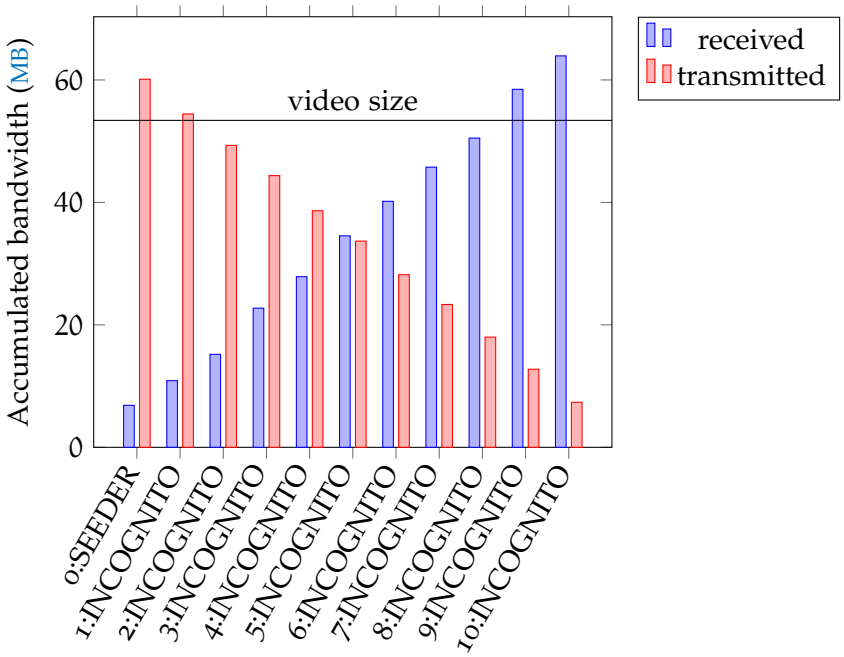


Figure 22: Recieved and transmitted data or each client, from the 10 Incognito users experiment (Exp.ID I10).
Like the baseline the first users download significantly less than the latter. Most download less than the video size, which is expected due to only 10 % being wanted by the Incognito users.

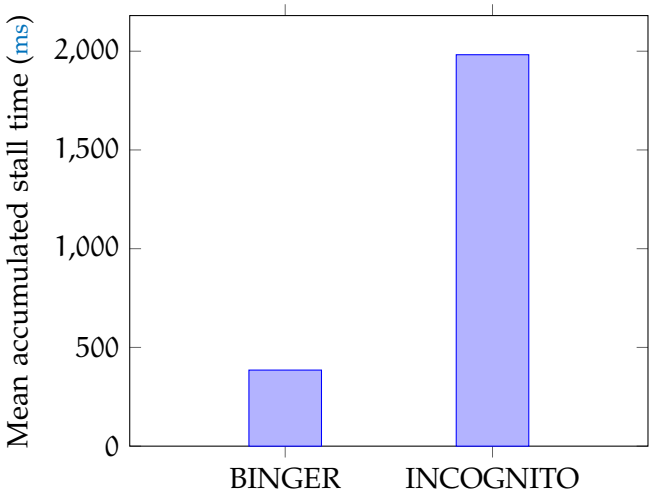


Figure 23: Mean of accumulated stall time for Binger clients compared to Incognito clients, from the 5 Incognito experiment (Exp.ID I5B5).
Incognito users stall much longer before they start playing the video.

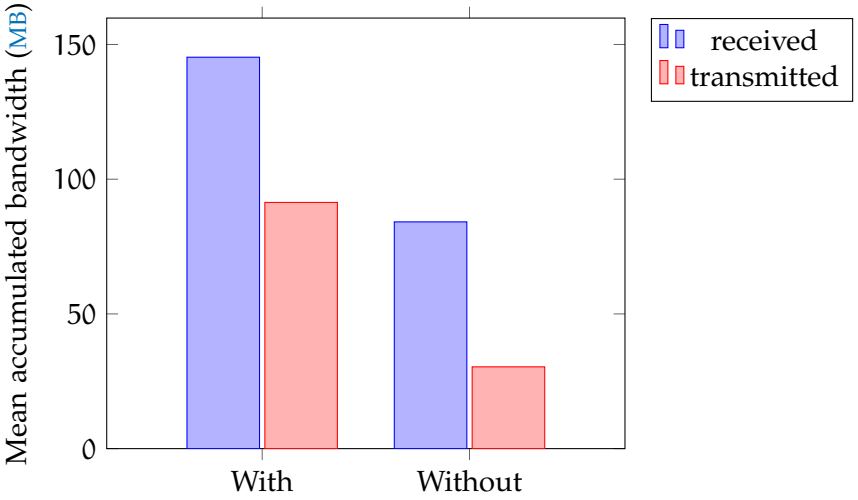


Figure 24: Mean of the accumulated data received and transmitted for 5 Bingers with and without 5 Incognito users in the network (Exp.ID I5B5 and Exp.ID B5).
With Incognito users present the amount of downloaded data is increased by around 50%.

6.5 MOBILE IMPACT EXPERIMENTS

This experiment in terms of behaviour is the same as the baseline, but the network conditions are worsened in terms of the bandwidth available and/or latency.

Table 7: Experimental Setup of Mobile impact experiments

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-m1	10 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 10 MBps.
B10-m2	10 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps with an added latency of 100±10 ms.

6.5.1 *Low bandwidth network*

In these experiments the amount of bandwidth available to the clients is decreased, which should punish IPFS more for its excessive amount of data being transmitted between peers.

By applying Equation 1 on a network with 1 Seeder and 10 Bingers, sharing a video with the bitrate of ~ 5 Mbps, the calculated lowest possible bandwidth to avoid stalls would be ≈ 9.5 Mbps.

Decreasing the bandwidth below the videos average bitrate, would make it impossible for a client to watch the video without stalls, even in an ideal configuration with a direct connection to a host with no other network communication present.

In the experiment (Exp.ID B10-m1) lowering the bandwidth to 10 Mbps was enough to achieve a very poor viewing experience. Each user got 9 or more stalls throughout the video (see Figure 25), with a mean accumulated stall time of 77 s, which approximates the duration of the desired video.

This high amount of stalling is likely due to the large amount of duplicate data in the network (see Section 6.1.2), as the scarce bandwidth is wasted on getting duplicate segments rather than retrieving future segments relevant to the user. This results in the DASH buffer running out of segments due to IPFS downloading the same segment multiple times.

Due to the performance of this experiment, there was no need to perform further experimentation with lower download rates.

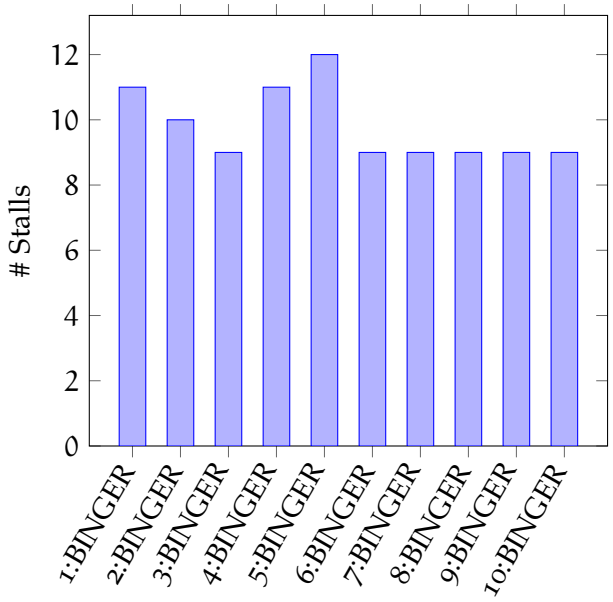


Figure 25: Total number of stalls for each user in Exp.ID B10-m1.
Half of the users stalled once more after the initial required stall.

6.5.2 *High latency network*

High latency is expected to increase the amount of duplicate data the clients receive, as changes in the peers want list take longer to propagate to peers that have segments that are no longer wanted. It could potentially also increase the amount of stalls, since it takes too long for the relevant segments to arrive to the client that needs them. An added latency of 100 ms is chosen (based on a ping to Google's data center in Sydney, Australia). In the experiments the increased latency significantly reduced the quality of experience, as half of the users now had an extra stall (see Figure 26), which were about halfway into the video and lasting multiple seconds (see Figure 27). This is in line with the hypothesis that it takes too long to get relevant pieces.

But the amount of duplicate data did not increase as much as expected as the average amount of received bandwidth only increased by 12 MB (see Figure 28). The standard deviation also decreased from increasing the latency, meaning the amount of downloaded data for each user was more consistent with each other. And there are less users that have to download significantly more. This however comes at the cost of each user having to download more on average.

6.5.3 *Jittery network conditions*

In this experiment the conditions of mobile users were to be emulated, where the connection fluactuates. The conditions should vary between a good connection (high bandwidth such as 4G) and low

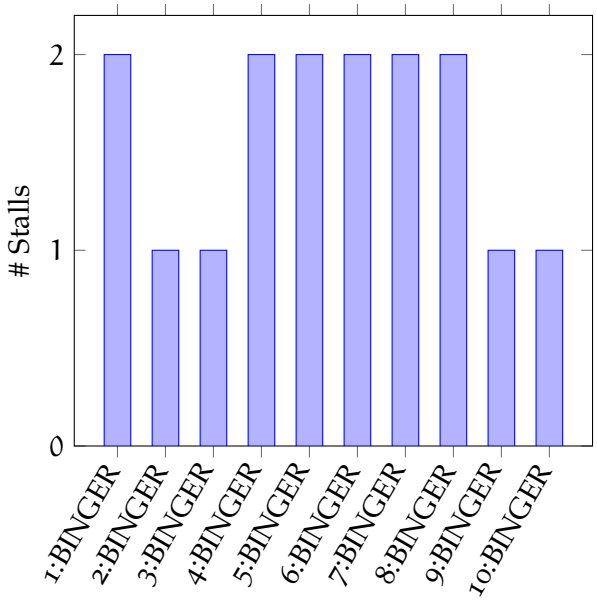


Figure 26: Total number of stalls for each user in Exp.ID B10-m2.
Half of the users stalled once more after the initial required stall.

bandwidth or no connection at all for short amounts of time. The results of the experiment were expected to be a good UX as long as the conditions are not bad for too long, due to the buffer being filled, which should be able to keep the video running smoothly.

This experiment was not conducted, though highly relevant, due to limitations in the pumba tool, which only enabled jitter for latency and not bandwidth rate. This could have been possible given more time, e.g., by modifying the run script to accept series of individual bandwidth rates to enforce sequentially upon each particular mobile client.

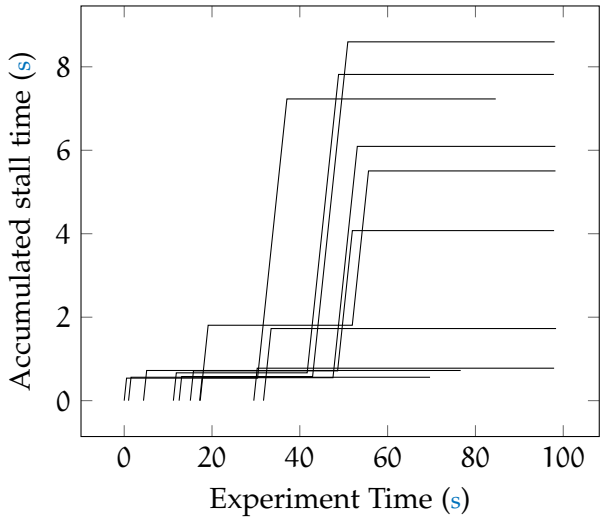


Figure 27: Stalls over time from Exp.ID B10-m2.
Even though there was a small amount of stalls, the stalls that occured lasted several seconds during playback.

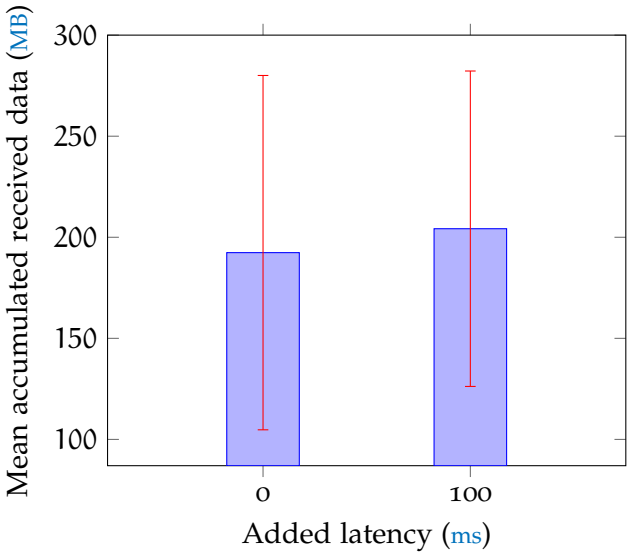


Figure 28: Mean of received data with std. dev. from the Exp.ID B10 compared to Exp.ID B10-m2.
The mean of received data increases slightly while the standard deviation lowers.

6.6 LEAVER IMPACT EXPERIMENTS

In this experiment the clients leave the IPFS network after they have finished watching the video, meaning they would no longer provide the video to other peers, when they themselves have finished.

Table 8: Experimental Setup of Leaver impact experiments

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-l	10 Bingers are introduced to a stable default network over 60 s. Each client will leave the network upon finishing the video. Network bandwidth is at 20 MBps.

The experiment is based on the baseline (Exp.ID B10), but where clients leave when they are done. Based on the results of the baseline this should result in reduced duplicate data as the clients that join the network early, leave before the remaining clients are finished. And when they do that is one less source of duplicate data that the remaining clients receive from. In terms of quality of experience there should be no major difference. The experiment went very much like expected, the amount of stalls was the same as the baseline, but the mean amount of downloaded data was decreased by 17 MB (See Figure 29). The difference in received data between the peers also lowered drastically, as can be seen by the almost halved standard deviation. This is likely due to fewer sources for retrieving data being available to the users that play the video later, meaning these users receive less duplicates.

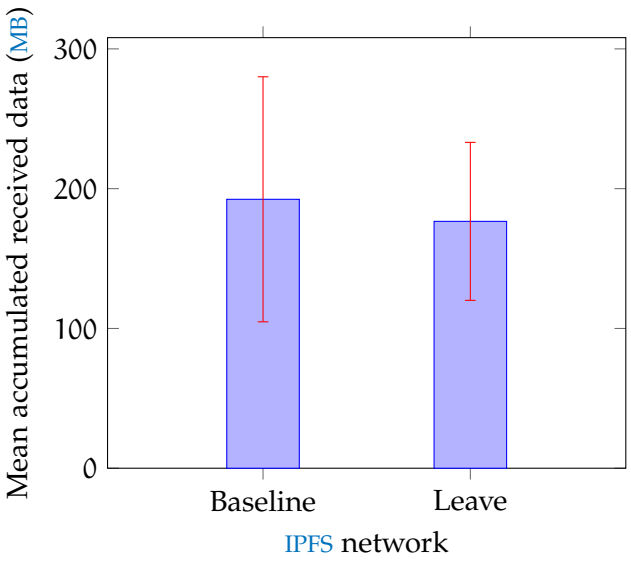


Figure 29: Mean of received data with std. dev. from Exp.ID B10 compared to Exp.ID B10-l.
The mean amount of received data is increased slightly, while the standard deviation almost halves.

6.7 SOCIALNESS IMPACT EXPERIMENTS

The other experiments focused on having all clients watch the video at the same time within a 60 s startup interval. This experiment changes the time between startup, effectively changing the socialness of the video.

Table 9: Experimental Setup of Socialness impact experiments

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-s1	10 Bingers are introduced to a stable default network over 10 s. Network bandwidth is at 20 MBps.
B10-s2	10 Bingers are introduced to a stable default network over 180 s. Network bandwidth is at 20 MBps.

The experiment Exp.ID B10-s1 lowers the startup delay to 10 s, effectively increasing the socialness due to the same demand but in a narrower time span. On the other hand, Exp.ID B10-s2 lowers the socialness as the startup delay is 180 s.

Although to properly test the impact of socialness the experiments have to last much longer, multiple days, which is omitted and replaced with the smaller time frame experiment instead.

In the experiments increasing the maximum startup delay gave a universally better performance. With the longer startup delay from Exp.ID B10-s2 there were only the initial stalls just like the baseline, but lowering it, as done in Exp.ID B10-s2, meant stalling became frequent (See Figure 30). The high amount of stalling is likely due to pressure on the Seeder as more users now want segments only available at the Seeder, as they have not been distributed yet.

Increasing the startup delay, also decreases the amount of duplicate data the users receives (See Figure 31), which is unexpected as adding more users with the data available, usually meant more duplicate data.

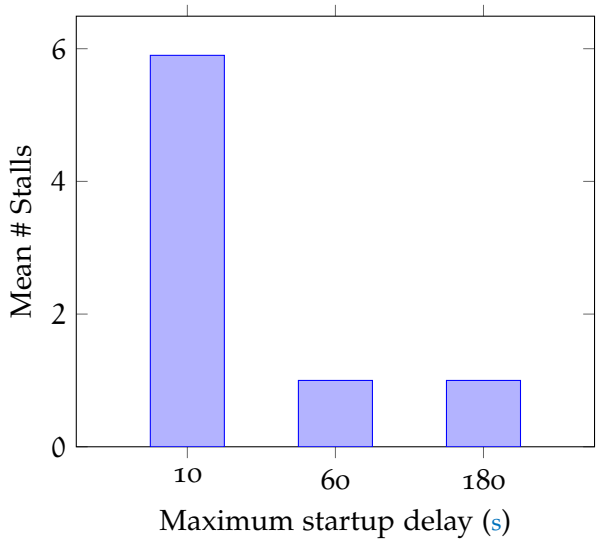


Figure 30: Mean number of stalls from Exp.ID B10-s1, Exp.ID B10, Exp.ID B10-s2.
Decreasing socialness gives better UX as the number of stalls decreases to the single required one during startup of the video.

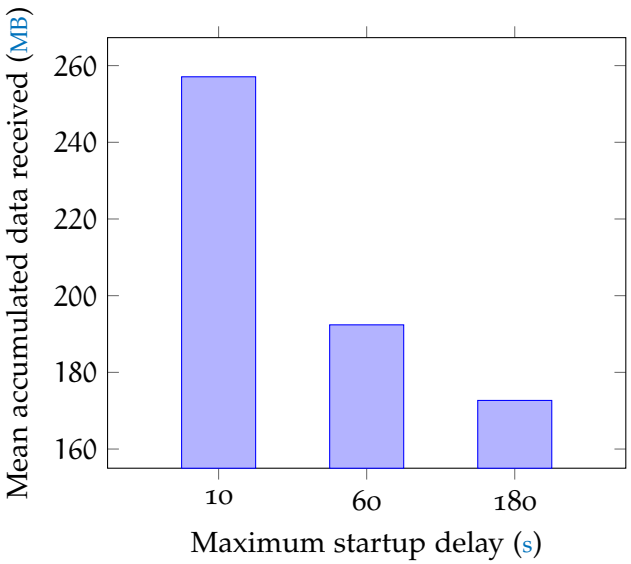


Figure 31: Mean of accumulated data received from Exp.ID B10-s1, Exp.ID B10, Exp.ID B10-s2.
Decreasing socialness gives reduces the amount of duplicate data the users receive.

6.8 VIDEO CONTENT INFLUENCE

In these experiments the video files are altered, to observe which influence the content being shared have on the network.

Table 10: Experimental Setup of Video content influence

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-v9	10 Bingers are introduced to the default stable network. The desired video is divided into segments of approximately 9000 ms duration. Network bandwidth is at 20 MBps.
B10-c18	10 Bingers are introduced to the default stable network. The desired video has a approximately doubled bitrate of 10.4 MBps. Network bandwidth is also doubled at 40 MBps.

6.8.1 *Increased segment duration experiments*

In this experiment each segment is around 9000 ms long rather than the default 3000 ms. This is to try to capitalize on the way IPFS separates files into blocks itself rather than doing it manually, as having many small segments means that each segment is only a few blocks.

This can however impact the user experience, as users now have to download a larger segment before they can play anything from that segment's time frame.

In the experiments all the clients stalled exactly twice, although the second of these stalls can not be considered bad UX because it happens on average 30 ms after the video starts. A frame lasts 33 ms for the video used in the experiment, so this stall only lasts around a single frame. The length of the second stall is also very small at an average 32 ms, which should not be noticeable when it happens a single frame into the video. The time spent stalling is however very significant. In the experiment the initial stall lasted 6.9 s longer than the baseline (see Figure 32). In terms of downloaded data the experiment performed better than the baseline The mean amount of received bandwidth was about two thirds of the baseline, with much lower standard deviation too signifying that IPFS is more efficient at handling larger files where it can utilize the way the files are split into blocks in the Bitswap. This however comes at the cost of the initial stall time.

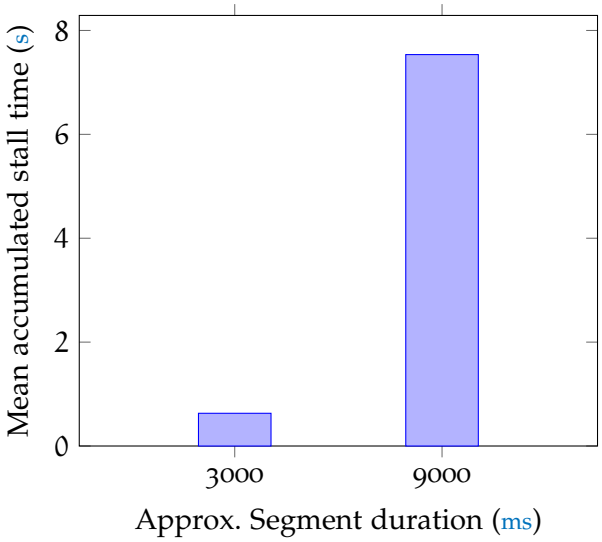


Figure 32: Mean of accumulated stall time from the baseline experiment (Exp.ID B10) and the segment experiment (Exp.ID B10-v9). Increasing the segment duration results in longer wait time spent on the initial stall.

6.8.2 *Increased bitrate experiment*

In this experiment the bitrate of the video is increased so that the total file size is 117 MB, which is approximately double of the video file size in the baseline experiment. To accommodate this increase in bitrate, the network bandwidth is also doubled to prevent the same amount of stalls as seen in Exp.ID B10-m1. Based on Exp.ID B10-v9 the amount of duplicate data relative to the filesize should decrease,

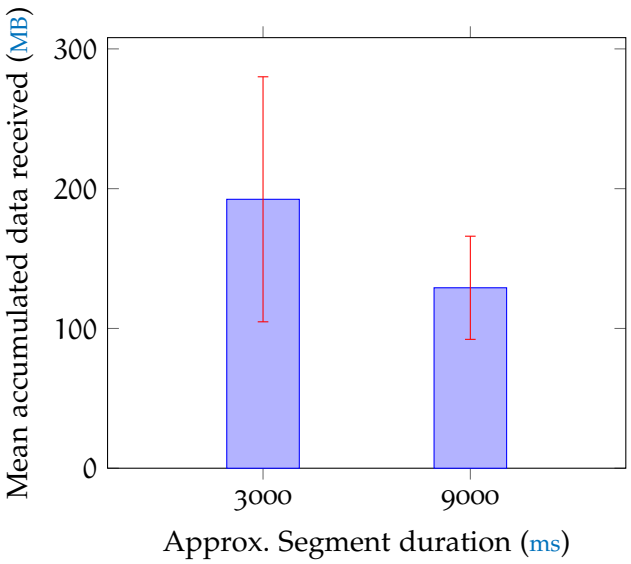


Figure 33: Mean of accumulated data received with std. dev. from the baseline experiment with 3 s segments (Exp.ID B10) and the segment experiment with 9 s (Exp.ID B10-v9).
A larger segment size results in a significant reduction of received duplicate data.

as IPFS is expected to perform better on larger files. In the experiment a few of the users stalled a single time in the middle of the video as the only UX penalty. In terms of duplicate data increasing the bitrate resulted in less as expected. (See Figure 34) In the baseline the amount of data was 3.6 times the file size, but with increased bitrate this fell to 3.2 times. This further cements the theory that IPFS is better suited for larger files rather than small ones.

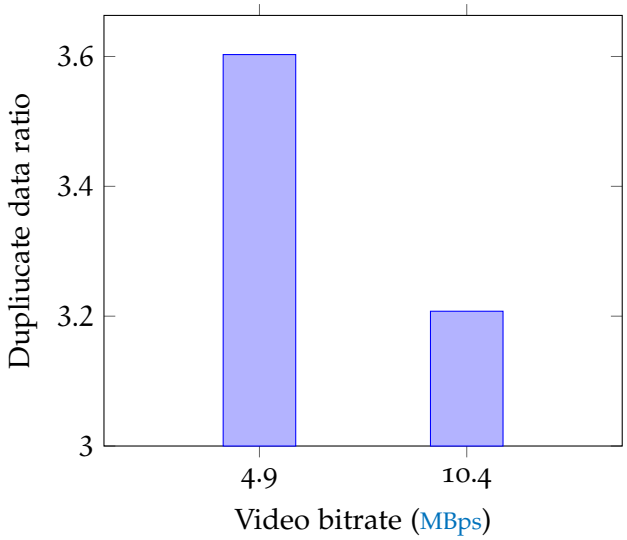


Figure 34: Mean of accumulated received data from Exp.ID B10 and Exp.ID B10-c18.
As the bitrate (and thereby the separate segment size) increases, the amount of duplicate data seems to decrease.

6.9 PUBLIC IPFS EXPERIMENTS

In the other experiments all clients in the IPFS network were relevant to the video files, contrary to the intended design of IPFS where all users of IPFS are in a shared network regardless of the content they have.

Table 11: Experimental Setup of Public IPFS experiments

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-p	10 Bingers having public IPFS access are introduced to a stable *near* default network over 60 s. The stable network differs by the 1 Seeder having public IPFS access and no bootstrap being present. Network bandwidth is at 20 MBps.

This experiment runs the same configuration as the baseline but operates in the public IPFS instead, meaning it is using the default bootstrap addresses to connect to IPFS network. It is expected to perform similar to the high latency experiment, as having larger network means larger hash tables, which results in having longer look up times for the video segments, due to the DHT's worst case lookup time being $O(\log n)$, where n is the number of peers in the network [4, p.2].

During the experiments some clients experienced multiple stalls (see Figure 35). These stalls where also fairly lengthy, up to 7 s, and

around the middle of the video (see Figure 36). Over half of the population suffered from additional stalls and encountered a longer rebuffering event during playback than the initial buffering, which signifies a poor UX of the viewing experience.

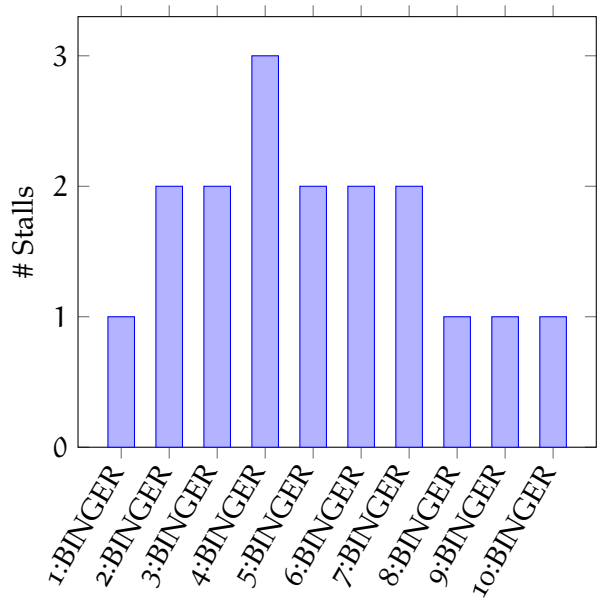


Figure 35: Total number of stall in the Exp.ID B10-p.
Half of the users experienced stalls other than the initial stall.

In terms of bandwidth there were slight improvements compared to the baseline experiment (Exp.ID B10) in which 3 users had around 300 MB or more received data. This overhead has now evened out between the clients, since the worst case of received data is around 250 MB (see Figure 37 compared to Figure 11).

The mean of the accumulated data received did however not improve and was around the same for the experiments compared to the baseline (see Figure 38), but the standard deviation decreased as the overhead was more evenly distributed among the clients. These results were very similar to that of the increased latency experiment as expected. Just like that experiment users get slightly more stalls, and the mean received data increased and the standard deviation decreased.

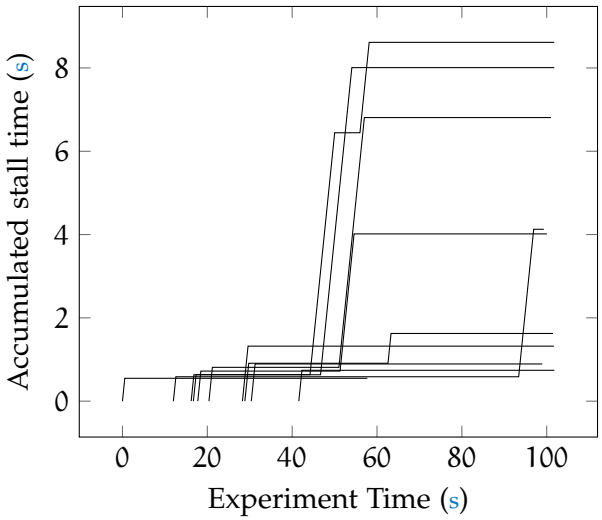


Figure 36: Stalls over time from Exp.ID B10-p.
Although there were not many stalls, the few stalls lasted several seconds during the playback.

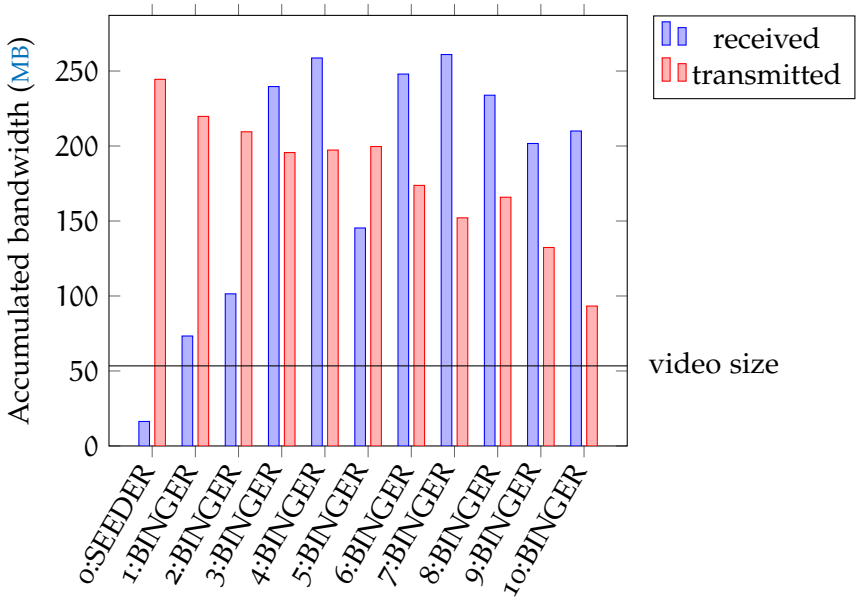


Figure 37: Total bandwidth for each user from Exp.ID B10-p.
Users do not tend to download more than around 250 MB unlike the baseline where some users could end up downloading much more.

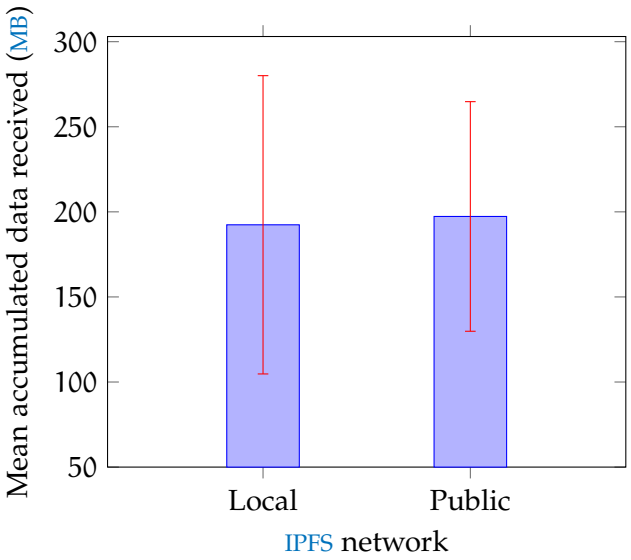


Figure 38: Mean of received data per client with std. dev. from Exp.ID B10 compared to Exp.ID B10-p.
The mean amount of received data is increased slightly, while the standard deviation lowers.

6.10 SUMMARY OF EVALUATION

IPFS has several shortcomings when it comes to performance, as discovered in this evaluation.

The first of which is its large CPU usage, which significantly limits how many peers can be simulated, and thereby the scalability of the experiments. This issue is further worsened by the high CPU usage of the DASH video player, which potentially could be lowered using another browser or other OSs as Linux does not have hardware accelerated video decoding available for Chrome[5]. Since the current setup utilizes Docker for emulation, a change in OS would be cumbersome as Docker only emulates Linux. However a change in browser to Firefox[6] would be very doable as Splinter also has a driver for this browser.

Despite the small network setup of maximum 12 IPFS peers (bootstrap, Seeder and up to 10 clients), another scaling issue of IPFS was apparent. As the number of peers seeding a file grew, the amount of duplicate data did as well. This transfer overhead meant that lowering the bandwidth to 10 Mbps resulted in very poor QOE, despite it theoretically being enough. Combine this with the fact that adding latency or using the public IPFS network, further increases the amount of duplicate data, and it is clear that IPFS is ill-suited for mobile devices as they are subject to these worsened network conditions, and often on a limited data plan making data overhead even more costly.

There are however ways to lessen the amount of duplicate data a user receives. The first is to have users only share their file segments while they are still streaming the video. with this strategy less peers seed the file segments at any given time, diminishing the connections serving the duplicated data. This strategy is however dangerous as the segments corresponding to the later parts of the video, are not stored for very long on a peer before it leaves the network, which means these segments could potentially disappear from the network if there were no one left to seed them.

Another way to reduce the data duplication was to give the users a larger time span between when they started the video. The cause of this is unknown, but a possible explanation could be that the BitSwap of IPFS, the protocol used for block exchange, puts more effort in distributing files that a peer recently received. This would be due to the less time there is between the introduced clients, the more wanted the same segments will be as the playback is contiguous. This phenomenon makes IPFS fairly similar to a centralized system as it does not handle sudden large influx in users, similar to a Distributed Denial-of-Service (DDoS) attack.

5 https://bugs.chromium.org/p/chromium/issues/detail?id=137247
6 https://www.mozilla.org/firefox/

The final way to reduce duplication of data was to increase the segment duration of the dashed video files. This comes at the cost of increasing the initial wait time before the video can start playing, due to the segment size being larger than what was necessary to begin playback. This behaviour indicates that IPFS is more suited for retrieving few large files rather than many small ones, which is further backed up by the fact that increasing bitrate also decreased the ratio of duplicated data compared to required data.

In terms of user behaviours the negative experiences caused by these were mostly isolated within the UX of the client causing these. Leeching resulted in more frequent stalling of the Leechers playback, while the other users could continue normally. This was until the population of Leechers was so large that the Seeder received too many requests to accommodate all. In terms of viewing patterns, skipping in the video resulted in stalls during the skip, which was expected behaviour as playback would require segments outside the buffer, but this had no negative impact on UX of the other clients in the network.

7 CONCLUSION

By utilizing containerization of users accessing a video sharing website from a Chrome browser and running an IPFS daemon as a backend, it was made possible to examine a UX under various circumstances, and determine what influence other users and network conditions have on DASH streaming fetching its video content from IPFS addresses.

The experiments where performed by simulating a steady state running network, with services hosting the video resource over IPFS and various logging services for collecting metrics of the experiment. Using the Chaos Engineering Principles, the steady network would then be introduced to different conditions, such as certain user behaviours in different populations as well as network conditions.

The population could easily be scaled after need, but due to a high CPU usage of both the IPFS daemon and the web player, a maximum of 10 clients were introduced to the steady network. Should the number of clients be increased, Docker starts to throttle the containers, resulting in artificial stalls for the video playback. This prevented a large scale evaluation of the project despite having a powerful host machine for the Docker environment.

Through evaluation of the system a major bug in IPFS was discovered, which resulted in a transmission of duplicate data that increased with the number of peers seeding a resource. The overhead of duplicated data was several times the size of the original video file, though the scale of the seeding peers were relatively small due to the CPU limitation. As a result of this, it could be confirmed that IPFS was unfit for the use of lowering bandwidth and would scale badly, therefore potentially flooding a network if high demand of the same resource arose.

The UX was not affected by the bug, so the experimentation of various behaviours and conditions were still conducted, to see if this would have an impact on the bug, for better or worse.

It was found that increasing the file size of any shared resource, the duplication would decrease. This is possibly due to IPFS being more suited for sharing larger files as each file then consist of more blocks (of fixed size), giving IPFS a larger influence in the coordination of downloads. Additionally, distributing the demand over a larger period of time also yielded a reduction in duplicated data, in contrary to the hypothesis stating that IPFS should be able to handle a surge in traffic and mitigate the burden.

As the duplication of data correlated with the number of Seeders, changing the ratio of the Seeders and the rest of the population seemed ineffectual and was therefore omitted. As a result of this and the limited scale of the network, churn rate was also omitted since having a single seed and removing it, only would result in a complete halt of the video playback.

In terms of user behaviours, the negative impact of these were largely isolated in the performing user's UX. The only exception to this being a large amount of leeching users that took the resource without giving back to the network. When there were few of the Leechers, only these received very poor UX, while the other well behaving users still had a passable viewing experience.

In conclusion, the use case of streaming video on demand from an IPFS hosted resource is ill-suited at its current state.

7.1 FUTURE WORK

To decrease the high CPU usage, it should be considered to change the browser, operating system (to gain hardware accelerated decoding) or the DASH player implementation.

To avoid duplication of data, using another DASH profile could be considered. This would allow the streams to be served as a single file, giving IPFS more control over the data flow, but presumably result in a much higher initial stall time.

Part II

APPENDIX

"Abandon all hope, ye who enter here."

— Dante Alighieri

A

MEDIA PRESENTATION DESCRIPTION

Listing 7: A full MPD example generated by `MP4Box`. The file have been slightly modified for readability by adding newlines.

```
<?xml version="1.0" ?>
<!-- MPD file Generated with GPAC version 0.7.2-->
<MPD
  maxSegmentDuration="PT0H0M3.008S"
  mediaPresentationDuration="PT0H9M56.480S"
  minBufferTime="PT1.500S"
  profiles="urn:mpeg:dash:profile:isoff-live:2011,
            http://dashif.org/guidelines/dash264"
  type="static"
  xmlns="urn:mpeg:dash:schema:mpd:2011">
 <ProgramInformation
  moreInformationURL="http://gpac.io">
  <Title>Big_Buck_Bunny</Title>
 </ProgramInformation>

 <Period
  duration="PT0H9M56.480S">
  <AdaptationSet
    lang="und"
    maxFrameRate="24"
    maxHeight="1080"
    maxWidth="1920"
    par="16:9"
    segmentAlignment="true">
   <SegmentTemplate
    duration="36864"
    initialization="Big_Buck_Bunny_video/Segment_0.mp4"
    media="Big_Buck_Bunny_video/Segment_$Number$.m4s"
    startNumber="1"
    timescale="12288"/>
   <Representation
    bandwidth="3915893"
    codecs="avc1.640028"
    frameRate="24"
    height="1080"
    id="video"
    mimeType="video/mp4"
    sar="1:1"
    startWithSAP="1"
    width="1920">
   </Representation>
  </AdaptationSet>
```

```
43    <AdaptationSet
44      lang="und"
45      segmentAlignment="true">
46     <SegmentTemplate
47      duration="144000"
48      initialization="Big_Buck_Bunny_audio/Segment_0.mp4"
49      media="Big_Buck_Bunny_audio/Segment_$Number$.m4s"
50      startNumber="1"
51      timescale="48000"/>
52     <Representation
53      audioSamplingRate="48000"
54      bandwidth="130055"
55      codecs="mp4a.40.2"
56      id="audio"
57      mimeType="audio/mp4"
58      startWithSAP="1">
59      <AudioChannelConfiguration
60        schemeIdUri="urn:mpeg:dash:23003:3:
              audio_channel_configuration:2011"
61        value="6"/>
62     </Representation>
63    </AdaptationSet>
64   </Period>
65  </MPD>
```

B

DOCKER COMPOSE FILES

Listing 8: A full Docker compose file used for running the stable network and introducing experimental users.

```yaml
version: '3.5'

# basic user setup, used for extending users easily
# source: https://medium.com/@kinghuang/a1e4105d70bd
x-user: &user-template
  image: andreasmalling/ft_user
  build: ./user
  volumes:
    - ./video_dashed/:/usr/src/app/video_dashed/
  depends_on:
    - bootstrap
    - metric
    - network
    - host

services:
  # EXP NETWORK:
  # user for debugging, with sound and display pass-through
  user_debug:
    <<: *user-template
    privileged: true                    # for sound
    environment:
      - DISPLAY=${DISPLAY:- :0}         # for display
    volumes:
      - /tmp/.X11-unix:/tmp/.X11-unix   # for display
      - /dev/snd:/dev/snd               # for sound
    command: ${USER_DEBUG:- -r 0 --head BINGE}

  # user adding content from /videos_dashed to IPFS network
  user_seed:
    <<: *user-template
    command: ${USER_SEED:- -r 0 SEEDER}

  # 3 users for persona client setup (expand as seen fit)
  user_1:
    <<: *user-template
    command: ${USER_1:- IDLE}

  user_2:
    <<: *user-template
    command: ${USER_2:- IDLE}

```

```
43    user_3:
44      <<: *user-template
45      command: ${USER_3:- IDLE}
46
47    # MAIN NETWORK:
48    # node used for bootstraping local IPFS network
49    bootstrap:
50      image: andreasmalling/ft_bootstrap
51
52    # python server hosting the website for the web player
53    host:
54      image: andreasmalling/ft_host
55      build: ./website
56      volumes:
57        - ./website:/usr/src/app/
58
59    # client that stores user reported DASH metrics in a mongoDB
60    metric:
61      image: andreasmalling/ft_metric
62      build: ./metric_server
63      restart: always     # due to failure if mongo boot is slow
64      depends_on:
65          - mongo
66
67    # collects network data from user containers
68    network:
69      image: andreasmalling/ft_network
70      build: ./network_logger
71      volumes:
72        - /var/run/docker.sock:/var/run/docker.sock
73      depends_on:
74          - mongo
75
76    # mongoDB microservice
77    mongo:
78      image: mongo:3.6.3
79      volumes:
80        - ./data/db:/data/db
81        - ./data/dump:/data/dump
82      command: --smallfiles --noprealloc
```

Listing 9: A full Docker compose file used for running the plot service, to evaluate to data collected in an experiment.

```
version: '3.5'
services:
  # plot service for evaluation of experiments
  plot:
    image: andreasmalling/ft_plot
    build: ./plot
    volumes:
      - ./data/plot/:/usr/src/app/output/
    depends_on:
      - mongo

  # copy of mongoDB microservice from experiment compose file
  mongo:
    image: mongo:3.6.3
    volumes:
      - ./data/db:/data/db
      - ./data/dump:/data/dump
    command: --smallfiles --noprealloc
```

C

EXPERIMENTAL SETUP OVERVIEWS

Collection of all named experimental setups and their Expiriment IDs, presented in table overviews from Chapter 6.

Table 12: Experimental Setup of Baseline (repeated from page 38)

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B1	1 Binger is introduced to the default stable network. Network bandwidth is at 20 MBps.
B5	5 Bingers are introduced to the default stable network. Network bandwidth is at 20 MBps.
B10	10 Bingers are introduced to the default stable network. Network bandwidth is at 20 MBps.

Table 13: Experimental Setup of Leecher (repeated from page 42)

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
L1B9	1 Leecher and 9 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.
L5B5	5 Leechers and 5 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.
L10	10 Leechers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.

Table 14: Experimental Setup of Skipper (repeated from page 47)

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
S5B5	5 Skippers and 5 Bingers are introduced to the default stable network over 60 s. Skipper watch time is 3 s and skip time 10 s. Network bandwidth is at 20 MBps.
S5B5-c	5 Skippers and 5 Bingers are introduced to the default stable network over 60 s. Skipper watch time is 1 s and skip time 25 s. Network bandwidth is at 20 MBps.
S10	10 Skippers are introduced to the default stable network over 60 s. Skipper watch time is 3 s and skip time 10 s. Network bandwidth is at 20 MBps.

Table 15: Experimental Setup of Incognito (repeated from page 51)

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
I5B5	5 Incognito users and 5 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.
I10	10 Incognito users are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps.

Table 16: Experimental Setup of Mobile (repeated from page 54)

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-m1	10 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 10 MBps.
B10-m2	10 Bingers are introduced to the default stable network over 60 s. Network bandwidth is at 20 MBps with an added latency of 100±10 ms.

Table 17: Experimental Setup of Leaver (repeated from page 58)

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-l	10 Bingers are introduced to a stable default network over 60 s. Each client will leave the network upon finishing the video. Network bandwidth is at 20 MBps.

Table 18: Experimental Setup of Socialness (repeated from page 59)

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-s1	10 Bingers are introduced to a stable default network over 10 s. Network bandwidth is at 20 MBps.
B10-s2	10 Bingers are introduced to a stable default network over 180 s. Network bandwidth is at 20 MBps.

Table 19: Experimental Setup of Segment size (repeated from page 61)

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-v9	10 Bingers are introduced to the default stable network. The desired video is divided into segments of approximately 9000 ms duration. Network bandwidth is at 20 MBps.
B10-c18	10 Bingers are introduced to the default stable network. The desired video has a approximately doubled bitrate of 10.4 MBps. Network bandwidth is also doubled at 40 MBps.

Table 20: Experimental Setup of Public Network (repeated from page 64)

EXP. ID	EXPERIMENTAL SETUP OF NETWORK
B10-p	10 Bingers having public IPFS access are introduced to a stable *near* default network over 60 s. The stable network differs by the 1 Seeder having public IPFS access and no bootstrap being present. Network bandwidth is at 20 MBps.

D

ADDITIONAL EXPERIMENTAL PLOTS

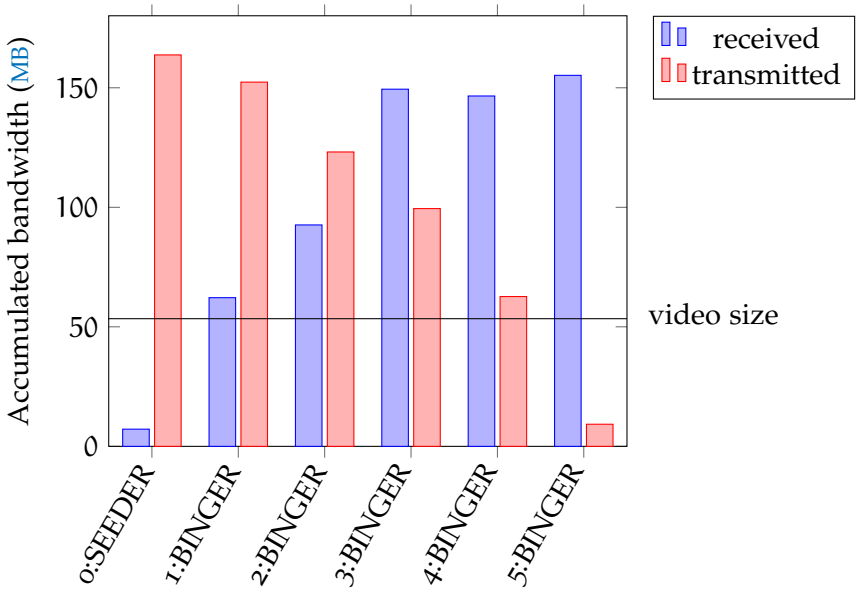


Figure 39: Received and transmitted bytes per client, from the baseline experiment (Exp.ID B5).
Download overhead is as large as factor 2.5 as exemplified by *5:BINGER* receiving 163 MB while asking for a 56 MB video.

E

SOURCE CODE

E.1 CODE REPOSITORY

https://github.com/andreasmalling/flixtube

E.2 THESIS REPOSITORY

https://github.com/andreasmalling/Master-Thesis

E.3 DOCKER IMAGES

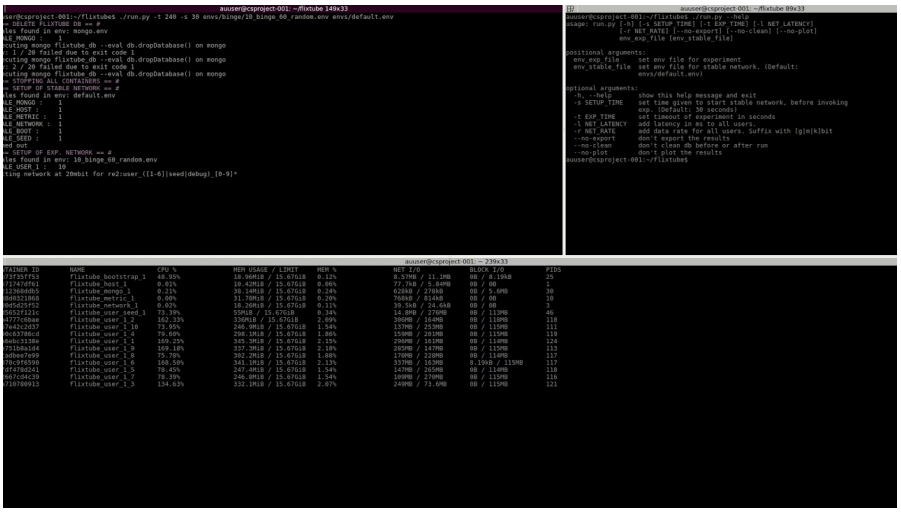
Container	URL
client	https://hub.docker.com/r/andreasmalling/ft_user
bootstrap	https://hub.docker.com/r/andreasmalling/ft_bootstrap
metric	https://hub.docker.com/r/andreasmalling/ft_metric
host	https://hub.docker.com/r/andreasmalling/ft_host
network	https://hub.docker.com/r/andreasmalling/ft_network
plot	https://hub.docker.com/r/andreasmalling/ft_plot

E.4 GETTING STARTED

1. Install Docker, Docker Compose, Pumba and Python 3.5
2. Download and unzip the project from the code repository (See Section E.1
3. Navigate into root folder of project
4. Run `./run.py envs/exp.env` for a simple experiment or check out `./run.py --help` for more options

E.5 VIDEO DEMONSTRATION

Video of demo run with a default stable network and an experimental introduction of 10 Bingers over 60 seconds. This experiment corresponds to Exp.ID B10 and shows the experiment run in its entirety.



Video can be found at https://youtu.be/_88_Kb7C6FY

BIBLIOGRAPHY

[1] ISO 23009-1:2014(E). *Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*. Standard. International Organization for Standardization, May 2014.

[2] A Aloman, AI Ispas, P Ciotirnae, Ramon Sanchez-Iborra, and Maria-Dolores Cano. "Performance evaluation of video streaming using MPEG DASH, RTSP, and RTMP in mobile networks." In: *IFIP Wireless and Mobile Networking Conference (WMNC), 2015 8th*. IEEE. 2015, pp. 144–151.

[3] Ingmar Baumgart and Sebastian Mies. "S/Kademlia: A practicable approach towards secure key-based routing." In: *Parallel and Distributed Systems, 2007 International Conference on*. IEEE. 2007, pp. 1–8.

[4] Juan Benet. "IPFS-content addressed, versioned, P2P file system." In: *arXiv preprint arXiv:1407.3561* (2014).

[5] Tom Broxton, Yannet Interian, Jon Vaver, and Mirjam Wattenhofer. "Catching a viral video." In: *Journal of Intelligent Information Systems* 40.2 (2013), pp. 241–259.

[6] Michael J. Freedman, Eric Freudenthal, and David Mazières. "Democratizing Content Publication with Coral." In: *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation*. NSDI'04. San Francisco, California: USENIX Association, 2004, pp. 2–15. URL: http://dl.acm.org/citation.cfm?id=1251175.1251193.

[7] Achraf Gazdar and Lamia Alkwai. "Toward a full peer to peer MPEG-DASH compliant streaming system." In: *Multimedia Tools and Applications* (2017), pp. 1–21.

[8] *Guidelines for Implementation: DASH-AVC/264 Interoperability Points*. Version 2.0. http://dashif.org/wp-content/uploads/2015/04/DASH-AVC-264-v2.00-hd-mca.pdf. DASH Industry Forum. Aug. 2013.

[9] Dirk Merkel. "Docker: Lightweight Linux Containers for Consistent Development and Deployment." In: *Linux J.* 2014.239 (Mar. 2014). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=2600239.2600241.

[10] Kien Nguyen, Thinh Nguyen, and Yevgeniy Kovchegov. "A P2P Video Delivery Network (P2P-VDN)." In: *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*. IEEE. 2009, pp. 1–7.

[11] Dongyu Qiu and Rayadurgam Srikant. "Modeling and performance analysis of BitTorrent-like peer-to-peer networks." In: *ACM SIGCOMM computer communication review*. Vol. 34. 4. ACM. 2004, pp. 367–378.

[12] Iraj Sodagar. "The mpeg-dash standard for multimedia streaming over the internet." In: *IEEE MultiMedia* 18.4 (2011), pp. 62–67.

DECLARATION

I, Andreas Østergaard Nielsen, hereby confirm that this thesis was made in collaboration between Andreas Malling Østergaard and myself. The workload of the written report and all of the associated code was evenly distributed among the two of us, and we have both contributed to all parts of the project.

Aarhus, June 2018

Andreas Østergaard Nielsen

I, Andreas Malling Østergaard, hereby confirm that this thesis was made in collaboration between Andreas Østergaard Nielsen and myself. The workload of the written report and all of the associated code was evenly distributed among the two of us, and we have both contributed to all parts of the project.

Aarhus, June 2018

Andreas Malling Østergaard

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of June 14, 2018 (`classicthesis` massive edition).