INVESTINGATING THE PROBLEMS AND SOLUTIONS OF PEER-TO-PEER BASED ARCHIVAL OF THE WEB

USING A REPUTATION SYSTEM TO GUARD ARCHIVE META-INFORMATION

LUCAS VICTOR PHILIPSTATT WINTHER, 201905215

BACHELOR REPORT (15 ECTS) IN COMPUTER SCIENCE

December 2022

Advisor: Niels Olof Bouvin

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

INVESTINGATING THE PROBLEMS AND SOLUTIONS OF PEER-TO-PEER BASED ARCHIVAL OF THE WEB

Using A Reputation System to guard archive meta-information

LUCAS W



Bachelor report (15 ECTS) in Computer Science

Department of Computer Science
Faculty of Natural Sciences
Aarhus University

December 2022

Lucas W: *Investingating the problems and solutions of peer-to-peer based archival of the web,* Bachelor report (15 ECTS) in Computer Science © December 2022

ABSTRACT

This report looked into the promise of a fully distributed peer-to-peer web archive. Five key aspects of a distributed archive were identified, being latency, its users and purpose, validation and trust of information in the archive, generation of archive pages in terms of both crawling and packaging information, and lastly, file storage, redundancy and deduplication of the archive. These were used in the analysis to create the frame in which the archive network was to function. In the frame, key missing necessary pieces for a completed system were identified, with the most important missing piece being the ability to determine which meta-information and thereby also information is truthful without having any pre-staked trusted parties. In the design, the report goes on to create a candidate solution for a reputation network based upon a peer-to-peer append-only log. It is further looked into how this system could help alleviate other problems described in the analysis, such as how higher file redundancy can be attributed to resources which embody more reputation. A proof-of-concept is constructed implemented with javascript and made in the node.js framework. The peer-to-peer network and websites are based on a simulation, where the reputation system uses the data from the simulation to give an estimation of website timelines based on the confidence of a resource. This confidence is calculated based on peers' reputations that they gain by interacting with each other, scaled by different factors. This prototype is then evaluated, and it is determined that the prototype can sort out a lot of incorrect and divergent information, though is unable to fully identify all of it. This is determined to be caused by both the design of the system, but also a key issue with the peer-to-peer nature of the network would appear in. It is lastly discussed how the purpose of a fully distributed peer-to-peer network needs to be reconsidered, and two example considerations were to accept the risk and let the network exist as a lesser archive with potentially malicious information or have the archivists be able to stake strategically chosen parties themselves.

CONTENTS

1 INTRODUCTION 1
- 1.1 Goal of Report 2
- 1.2 Overview 2

2 RELATED WORK 5
- 2.1 P2P Web Archival Problems 5
  - 2.1.1 Latency 5
  - 2.1.2 Users and Purpose 6
  - 2.1.3 Validation and Trust 8
  - 2.1.4 Generation of archive pages 9
  - 2.1.5 Resource storage, redundancy and duplication 10
- 2.2 Summary 12

3 ANALYSIS 13
- 3.1 Problems 13
- 3.2 Solutions 14

4 DESIGN OF ARCHITECTURE AND EVALUATION 17
- 4.1 What is reputation 17
- 4.2 The Append-Only Log Structure and Assumptions 17
- 4.3 The information on the AOL 18
- 4.4 System Behavior 19
  - 4.4.1 Crawling Peers 19
  - 4.4.2 Validator Peers 19
- 4.5 Using the Reputation System 20
- 4.6 Redundency 21
- 4.7 Bandwidth Savings and Space Considerations 21
- 4.8 Timeouts and Timestamps 21
- 4.9 Architecture Design Summary 22
- 4.10 Evaluation Design 22

5 IMPLEMENTATION 25
- 5.1 Limitations of the prototype 25
- 5.2 Language and Framework 25
- 5.3 Overall structure 25
- 5.4 Simulation Layer 26
  - 5.4.1 Time 26
  - 5.4.2 Peer Behaviors 28
  - 5.4.3 Websites and the AOL 29
- 5.5 TrustManager 30
  - 5.5.1 Reputation of User 33
  - 5.5.2 Temporal Scaling 34
  - 5.5.3 Trust of version at time 35

6 EVALUATION 37
- 6.1 Methodology 37
- 6.2 Total amount of data 39

6.3 Test Results 40
6.3.1 Parameters 40
6.3.2 Behaviors 42
6.4 Summary and Discussion 44
7 CONCLUSION 49

I APPENDIX 51

BIBLIOGRAPHY 53

LIST OF FIGURES

Figure 1 An example of the curve of a logistic function 35
Figure 2 Graph showing test results for how information density influences performance. 40
Figure 3 Graph showing test results for how churn rate influences performance. 41
Figure 4 Graph showing test results for how Minimum Confidence influences performance. 42
Figure 5 Graph showing test results for how different values for the temporal scaling function influences performance. 43
Figure 6 Graph showing how the score lowers with the number of peers with incorrect behaviour. 44
Figure 7 Graph showing average scores across behaviours that work alone 45
Figure 8 Graph showing how the grouped versions of either consistently malicious, site-specific malicious, or sometimes malicious affect prototype performance. 46
Figure 9 Graph showing average scores across all incorrect behaviours at a low population of incorrect actors 48

LIST OF TABLES

Table 1 The number of data points generated for each of the different simulation scenarios. 39
Table 2 Summary of Average Scores across different incorrect behaviours 47

LISTINGS

Listing 1 The control flow of a simulated peer 29
Listing 2 Estimation of the timeline of a given URL 32

Listing 3 Calculation of the trust one archivist peer has in another archivist peer 33
Listing 4 Temporal Scaling Function 34
Listing 5 Calculating the confidence level of a given version of a URL at a given timeslot 35

1 INTRODUCTION

Archival is an important aspect of conserving human history. Many countries have dedicated archives that specialize in preserving information about that country's past. When it comes to information in the modern age, the internet is the go-to place for both information storage and retrieval. The internet serves as a platform where many people can share domain-specific information, be it cooking recipes, academic papers or videos of cute cats. The internet is however a very transient place, as servers go down, website authors change content, and domains change owners. This means information can quickly disappear. As such, people wanting to access previously available information for a copious amount of reasons, be it historic academic research, to hold something accountable for something they published, or simply nostalgia will be unable to satisfy their needs.

Organizations and government subsidiaries exist that try to preserve these websites, such as *archive.org* and the Danish *netarkivet*. While having these archives are great, they do have their weaknesses. Issues like copyright and funding mean that these archives have to make concessions sometimes to things like restricting access to the archive and doing more selective and less frequent website crawling.

One might ask if voluntary enthusiasts could create some sort of global distributed peer-to-peer archive. Having a distributed base means that the archive could become more resistant to threats like slapp-suits[1], have greater reach and definition because of the increased processing and bandwidth power, and the archive itself could be used without relying on a single entity to keep up server costs. Examples of shutting down websites and forums containing specialized knowledge of niche topics aren't new, so being able to browse these sites into the future has value here too. If the archive is good enough, it might even be able to form a base from which professional archives could absorb information if it proves trustworthy enough.

However, upholding strict standards both for quality and authenticity in the archive proves to be a challenge when everyone can join. This report asks if it is possible to introduce some sort of mechanism or system, such that amateur volunteer archivists can create a great big distributed and trustworthy archive of the internet.

Multiple academic research papers have previously introduced their own candidate solutions for the non-distributed nature of modern-day internet archives. However while most of these papers are closer

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

to being fully distributed in nature such as transferring files in a distributed way, approaches still generally rely on staking trust in predetermined partners, be it government organizations such as libraries and universities or manually inputted peers like friends and family, and generally lack a fully-fledged image on how a fully peer-to-peer archive would work.

1.1 GOAL OF REPORT

This report investigates the potential of a distributed peer-to-peer archive. The report intends to describe what pieces of the puzzle are already there, what pieces are still missing and how these pieces fit together. The report intends to try and create some prototypes for missing pieces, primarily investigating and creating a proof of concept solution to the problem of resource validity and the hardships of creating a distributed network that has strong resilience to bad actors in a fully decentralized and distributed peer-to-peer archive where theoretically everyone could join. This report intends to create a reputation system for resources in the archive, that does not rely on predetermined staked trust, but rather on the work and consensus of other archivist peers in the network.

With the reputation system as its base, this report also intends to describe potential solutions to how a distributed network should allocate its resources in terms of bandwidth and storage. The goal of this report is then to make sure that even with a high concentration of malicious archivist peers, truthful, useful and precise information can still be retained.

The focus of this report will be on implementing a proof-of-concept to confirm or rule out the hypotheses:

1. It is possible to use a reputation system in a distributed archive to ensure that non-malicious peers of a distributed archive will be able to detect manipulated or divergent content submitted to the archive.

1.2 OVERVIEW

This section introduced the purpose of this report. Chapter 2 looks into related work and identifies 5 key aspects for how distributed archival and caching systems differ, and describes for each aspect what previous work different papers have researched. In Chapter 3, the current shortcomings standing in the way of a fully distributed archival system are identified and prioritized into what are essential missing components, and opportunities for improvement. In Chapter 4 solutions and ideas are designed to try and remedy the most important problems found in the Analysis. The implemented proof-

of-concept of the design is then described in Chapter 5 whereafter the evaluation in Chapter 6 shines a light on how the system compares to the hypothesis statement. In Chapter 7, the report is concluded with lessons learned and ideas for future work.

As an aside, do note that due to the limits in time and size of the bachelor project, this report has focused on the reputation system when implementing and evaluating the proof-of-concept and hypothesis. While developing a fully-fledged system would be desirable, the central issue of knowing which resources are trustworthy is a requirement for the archive to even make sense using, and is thus the focus of this report.

2 RELATED WORK

This report intends to find a way to increase the lifetime and record history of websites, through the creation of a distributed peer-to-peer archival network. This report is not the first that tests how web3 can be used together with newer website technologies. Thusly, this section intends to look at papers which share some of the common goals and/or aspects with Hypercache.

2.1 P2P WEB ARCHIVAL PROBLEMS

When creating a distributed archive, there are some different subjects to consider and plan accordingly to. From the other papers in the field, five key areas have been identified, and those are

- **Latency**: How quickly can a user find a resource in the system
- **Users and Purpose**: Who is participating in the system, and for who does the system exist
- **Validation and Trust**: How does a user validate that a resource from the system is correct
- **Generation of archive pages**: Which webpages are crawled from the archive, and what exactly defines the website
- **Resource storage, redundancy and duplication**: How is the archive stored and transmitted on a technical level

Some papers do, of course, try to research multiple of these subjects at once. This section presents the current status on these different fronts.

2.1.1 *Latency*

One of the key problems with peer-to-peer systems is the increased latency when it comes to fetching resources. In the current Web 2.0 architecture, the process of finding a website is relatively straightforward. You send a DNS request for an URL and then visit the IP address that the DNS request pointed to. However, in a decentralized system, it is usually not immediately clear who has a file or if the file/webpage even exists. Finding a file amongst many peers therefore usually has a cost in bandwidth and time before you get your resource.

One way popular newer peer-to-peer (P2P) networks like the BitTorrent network solve this issue in a distributed nature is by using a distributed hash table, such as Kademlia [8] or Chord. By organizing peers via routing/search tables, latency is greatly reduced. This comes at a cost of maintaining said distributed hash table, which is felt by the network as a whole. Even after building a DHT, the latency is still greater than the client-server architecture in almost all cases. For normal web browsing, this issue could be problematic [9], but for archivation purposes, it might be a worthwhile tradeoff to get the benefits of resilience that peer-to-peer provides. [1].

2.1.2 *Users and Purpose*

The archivists that are maintaining and using an archive, and the purpose of said archive has a large influence on its structure and workings. Are the peers trusted by each other? Who is doing the archiving and browsing, and to whose benefit? And how do copyright laws fit into the archive? These questions are discussed in this subsection.

From the perspective of the user, it is important to respect their privacy and choice in the peer-to-peer network. Here, it can be easy to overlook ways in which information about a user can be leaked. PeerCache [3] is a paper that intended to create a P2P cache of websites using web extensions to create caches of websites a user visits. In PeerCache, websites are theoretically 'cleaned' of private information by performing what is called private set intersection (PSI) with another peer on the network. On websites, PSI works by hashing every HTML element, and by comparing hash trees with another peer, you can figure out what elements you share based on matching hashes. A problem here is that there are no guarantees or knowledge about the other peers' relation with the original websites, which means you could theoretically also do PSI with someone who shares some private information with you on their webpage. In that case, you could do PSI with even more peers, but at that point, it becomes very hard to know when you've done enough PSIs.

Something that PeerCache can optionally do, and the paper by Røyeng [9] has as its main purpose, is that users have to manually select what caches to share with the peer-to-peer network. This, while giving full privacy control also means that many websites which could be clean and made available to the network aren't. Respecting peer privacy in the peer-to-peer network while still covering a large amount of website and historic data is, therefore, an important consideration in a full peer-to-peer web archive.

One thing to note is that especially Røyeng's project [9] is a personal-archive first, sharing second archive. This also has an impact on another problem in this subject: user recruitment. For a peer-to-peer network to be strong, it naturally needs a lot of peers, which need to be

'recruited' into the network somehow. In the case of Røyeng's project, the user installs the software of the project for their own purpose, and the archive is primarily used by the user themselves, not other peers. This means the user has an incentive of their own to install and join the network.

If the purpose of the archive is instead to act as a distributed large set of websites accessible for everyone, the circumstances change a lot. Some papers assume different things about how a network like this would be built. One approach would be to have the network primarily composed of bandwidth and compute sponsored by universities and public archives like is done in Herodotus [4], and to some degree in WARchain [7]. In these cases, it can be assumed malicious actors are close to non-existent. Others assume the peer-to-peer network to be fully compromised only by a flat hierarchy of users such as PeerCache [3]. In this case, bad actors should be considered to a larger degree.

There's also the consent of the webserver/webauthors to consider. With copyright laws, this can be a difficult topic to navigate. Website authors are generally people who will not want their content copied and redistributed without their consent [9]. A solution to this is to either delegate the responsibility to the users to not infringe copyright violations or to deliberately not share the archive with the public, which is the method Nettarkivet employs to tackle copyright [9].

If we instead see the user of the system as the website author, both parties might be able to gain benefit. In the case of Maygh [12], the purpose is to lessen the load on a webserver by having peers visiting the website distribute larger resources to other people visiting the website. While Maygh itself is not strictly web archivation, this proposes the idea of the server being the one instigating the peer-to-peer transfer, and users visiting the website being recruited into the network simply by visiting it.

How the users get interested in the network is one thing, but another is how the user gets to actively start participating. Ideally, this should be as easy as possible to have the best user experience. In the previously mentioned Maygh, users start participating as soon as the website is loaded, requiring them only to visit an URL in the webpage. Other alternatives require users to install browser extensions to participate, while others require the installation of software directly to the computer. To have the biggest network users should be easy to recruit. There are technical limitations of the different solutions which can require effort from the user to participate, so in the end, the network must make certain tradeoffs depending on the goal of the archive.

The topic of users and purpose is thus a question which initially will shape the archive and the required technology behind it. For personal and private archives, one might need to consider methods of recruiting peers to the network and copyright issues less. For public

peer-to-peer archives, there are more things to consider. These include copyright laws and how to recruit peers into the system. Also, one in this case has to consider bad actors. As a last point, webservers themselves could also look to draw benefit as a user of a peer-to-peer archive of their website.

2.1.3 *Validation and Trust*

This subsection tries to question how a user can be sure that the website they are viewing is unaltered and consist of what the real website consisted of at the time of archival.

When browsing through an archive, it is important that users viewing a website know that it is an authentic unaltered version. Without the guarantee that an archive's content is legitimate, the archive itself could be considered of little use. Even for one of the biggest online archive sites `arhive.org`, there have been reports of tampering with archived material[1]. Thusly, there needs to be a way to make sure what is being seen is non-disputably what was there at the time of archival.

How much thought and importance this has been given in peer-to-peer archive systems differ a lot. As discussed in Section 2.1.2 based on who is hosting the archive, there are fewer people you need to trust. The key idea here is that as long as you don't know what a specific website looks/looked like yourself, you will have to put/stake your trust in someone that delivers the information to you.

Some papers trust that all archivation peers act in good faith, such as Herodotus [4] and PeerCache [3]. However as described above, this naive approach can lead to bad actors changing the information on the web page. Another idea is to look at archives on a smaller scale. In Røyeng [9], you archive pages yourself, but don't share pages and can't browse other peers' archived pages by default. You can generate a link to share over peer-to-peer networking, thus staking trust in that those in the community sharing the link act in good faith.

However, for a completely public archive, in which bad actors exist, a more robust solution is needed. WARchain [7] is an approach where the idea is to put the trust in public universities and archives, which then agree via blockchain consensus which versions of an archived website are the correct ones. This is a noble idea, and staking trust in multiple trusted public sources could very well be safer than trusting just anybody. But in doing this, there is also a loss of truly distributed web archivation.

Alternatively one might opt to use reputation systems to gauge the trustworthiness and validity of information. Even in 2003, research was done to classify and use an opt-in reputation system on peers in distributed peer-to-peer networks [6]. Other work has also looked

1 `https://blog.archive.org/2018/04/24/addressing-recent-claims-of-manipulated-blog-posts-in-the-wayback-machine/`, last accessed 8. Dec 2022

into using reputation systems in the field of archival and protection of the validity of documents, such as how it is done by Vigil Et. Al [10, 11]. These do show good results and are a useful metric to gauge the trustworthiness of different documents such as electronic health records [11]. Something important to note is that these papers usually tend to rely on trusted third parties in the form of Certificate Authorities (CA's) or Timestamp Authorities (TSA). The issue is thus as above, that there is a loss of true decentralization.

This section looked at a solution to validate pages served from an archive. The problem presented is that a bad actor in an archive might send false information when a user tries to visit a webpage. It is difficult to find a solution to this problem in a peer-to-peer nature. For personal archives, you can trust yourself. For public archives, WARchain proposes to stake the trust in public institutions like state archives and universities, with a consensus blockchain underneath, while other papers suggest using reputation to ensure the validity of documents.

2.1.4 *Generation of archive pages*

For there to be an archive of content, someone needs to crawl the web and generate the archive files that are to be viewed. So the questions are: who crawls what websites, and what content from the website do we store?

In Herodotus [4], the idea is to crawl and archive the entire web daily, by using the Chord distributed hash table to determine which node has the responsibility for which website. From here, HTML and images are fetched, which resulted in an estimated 56TB for the first total web crawl, and 17.5TB of new data each month [Ibid]. Now the internet has arguably changed since 2002, and archiving every single thing on the internet seems unreasonable, especially with the advent of user-generated content on Web 2.0, such as YouTube and Twitter.

A lot of papers rely on individual choice of which websites to archive, meaning it is up to the single peer to choose what they want to archive and when. This is what this report will call passive archivation, meaning the archivation system isn't actively crawling and generating web archives. Peercache's [3] main idea is generating caches of websites when you visit them in your browser, where the website is then captured as the website has finished generating. This has the added benefit of being able to cache pages where it is required that the user is logged in. In Røyeng's paper [9], you have to submit URLs directly to an installed application. Another idea is to archive items based on the behaviour of you and the peers around you, meaning you could archive and cache websites in the background, that you or close peers will likely visit in the future (like a content recommendation system) [5].

Figuring out what content to archive from a webpage is another difficult decision. For instance, the WARC[2] format, which is used by the crawler that institutions like the Royal Danish Library uses[3], captures not only the HTML payload but also the response headers, which can be useful. Things like javascript files would also be important to archive, however, do note that archiving javascript files could easily create inconsistencies in successive replays [2].

To summarize, when it comes to archives, there are multiple options as to who is doing the generation and what is stored. There is passive archivation where users determine by manual submission or behaviour which websites they archive and there is active archivation where the network coordinates which websites each peer crawls in the background. Further, when archiving sites, you can choose how many of the website files and request information you want to store, optionally skipping things like images and javascript files to standardize behaviour and save space in the network at the cost of completeness.

2.1.5 *Resource storage, redundancy and duplication*

This section concerns itself with the more technical aspect of peer-to-peer communication. Here the main questions to ask are what network infrastructure to use and how files themselves should be stored on the network. These can impact the overall network performance and system requirements for peers.

One of the large technological hurdles in a peer-to-peer system is figuring out how the archived websites and also archive meta-information should be stored. There are a lot of different already made projects to consider in these cases, such as Gnutella, IPFS, PAST, BitTorrent, and more [9].

There are benefits to building on already popular and tested technological software stacks for both the archive itself and meta-information about the archive [7]. Therefore it is important to consider whether to build something new entirely [4, 12], use components of already established networks [3, 9], or simply just integrate into an already popular network [1, 7]. The benefits of using tried and tested networking technology are manyfold. There is the idea that there already exists some number of peers out there that can help with peer discovery. Initial issues and problems with the networking component can also have been fixed by early adopters and papers concerning themselves with these systems. On the other hand, control is given up, which can cause problems and unnecessary performance issues later. The same pros and cons also apply to using a ready-made implementation of a protocol versus coding your own from scratch [9].

2 https://en.wikipedia.org/wiki/Web_ARChive, last accessed 03. Dec 2022
3 https://sbforge.org/display/NAS, last accessed 03. Dec 2022

When storing the files on the network, you want to have high redundancy and low duplication. Think of duplication as the same file existing on the network with slightly different names. This could increase the total required amount of storage without having any benefit. Redundancy, on the other hand, means the same file can disappear from multiple peers without being unavailable on the network. While increasing storage requirements, this also increases the availability of files on the archive.

The simple way is just to store each website captured at a timestamp as its own folder in the network. This might increase duplication in the network, as even if the website has not changed, the network will be required to host a whole copy of the website again. In the InterPlanetary Wayback project [1], the content is instead stored with what is called content-based addressing. In this case, the website in its entirety is hashed and then put into the IPFS network. This hash is stored as meta information outside the IPFS network. When you then crawl a website, you can hash it and store it in the network. If the website is unchanged, the hashes would be the same and thus instead of duplication, you just add redundancy to the individual archived version of the website. Having a hash of the website stored as meta-information also increases verifiability as discussed in Section 2.1.3, as you can hash the website yourself and see if the transmitted information is correct.

There's also the question of active replication or passive replication of files in the network. In networks like the BitTorrent network, files are not actively shared before another user requests them [9]. Other networks employ what this report will call active file redundancy which tries to have at least a certain amount of copies of a file available between peers in the network. This comes with the added cost of keeping track of how many of these copies exist and requires extra space on the system's collective storage space. Balancing the number of copies each peer has with the desired amount of resilience to peers joining and leaving the network is an art in itself and requires careful consideration.

In this section, an overview was gained of the technological distribution of files in a peer-to-peer setting. Choosing the network stack is important, and choosing a ready-made solution can cut down on bugs and interact with peers already recruited in the network. However, control is lost which may lead to unwanted side effects such as performance concerns. The peer-to-peer filesystem should look to have high redundancy, meaning files will persist as peers join and leave the network, while having low duplication, meaning space wasted containing the same information.

2.2 SUMMARY

At the start of the section, five key subjects were mentioned when it comes to peer-to-peer web archives. Throughout this related work section, different methods, ideas and takes on these subjects were presented to the user.

First, regarding latency, the problem of how quickly users can see an archived page in the peer-to-peer network is raised. A proposed solution is most often to structure peers in some way using a DHT, however, there will still in most cases be more latency than traditional client-server structures.

Then, the report discussed the users and the purpose of a peer-to-peer archival network. Some distributed archives may be based on the users first, and only have sharing be optional. In these cases, it is important to consider the users' privacy. In archivation cases for the general good, it is important to consider how users will be recruited and potential issues with the copyright of the individual websites. It is also noted how in the situation of cooperation with the webserver, a peer-to-peer network might benefit the website owner.

Validation and trust was the next subject. Making sure the versions of websites in the archive are trustworthy and valid is important. In the past, this usually comes from trusting the peers in the network, but a newer solution to this is to have publicly staked institutions participate in a consensus-based blockchain, or focus on creating reputation-based metrics to gauge trustworthiness.

It was then discussed how the pages in the archive were to be crawled and generated. Some papers have users archive only the pages they visit, while in other papers peers strategically try to crawl as much of the web as possible. From here, deciding what parts of the website to archive is another question. Some archive only HTML, some HTML and images, and some also archive meta-information such as response headers using the WARC format. Furthermore, the question of how and if to archive javascript files is raised.

Lastly, the more technical aspect of a peer-to-peer archive is considered. Integrating into an already established network can have benefits, at the cost of control. Content-based addressing can be used to lower duplication in the network, and active distribution of files can help increase file redundancy.

3 ANALYSIS

This section identifies some key areas of peer-to-peer archival where problems or opportunities for improvement can be found, and argues which issues are most beneficial to solve.

As described in the problem statement, this report looks to employ peer-to-peer benefits to improve the availability of websites. With this perspective, the subjects from Chapter 2 are analyzed.

3.1 PROBLEMS

The elements that dictate the importance of the current issues are the users and the purpose of the system, as this will often dictate which tradeoffs can be done on the other key areas of the envisioned ideal peer-to-peer archival system. The purpose of the archive system described by this report is peer-to-peer archival with a focus on high availability and historical data. The users of this system are imagined to be interested activists. This means everyone should be able to browse the archive, and everyone should be able to participate in the archival process.

Therefore there are no staked parties to rely on in this system. This poses a challenge for the aspect of validation and trust, as you have to figure out who is telling the truth. For instance, resources might be archived by users located in different countries, resulting in resources with different languages and thus different hashes. The websites might be crawled at the same time too, meaning none of the peers are acting in bad faith even though they have different websites. For the system this report wishes to create, this is a problem that needs to be solved.

How resources are stored and shared in this network is also very dependent on which resources we trust are correct. This goes largely unexplored as most of the papers discussed in Section 2.1.5 don't discuss replication with bad actors present in the system. An attack vector for this kind of network could be to flood the network with 'garbage' files, which the network will replicate because of its goal of file redundancy.

Another aspect here is also how to efficiently transfer and store the resources of the network. In general, file storage is cheap while bandwidth is expensive. Papers previously suggested using differentials between different versions of resources, but none have ever actually implemented this. Furthermore, the bandwidth cost of setting up con-

nections and figuring out which differential parts a peer is missing might cost more than just sending the full new version to them.

The problem of crawling also still stands, with only selective crawling being the employed solution. For a full website archival, one would have to coordinate in a fully distributed way which peer is responsible for which part of the network. Something like this was done using a Chord ring in [4], however, the method incurred a very large cost when peers leave and join the network. This might thusly not be ideal in a public network context. Also, generating snapshots of pages can be difficult with the advent of javascript-based websites that are based on frameworks like react, vue and angular, with examples being things like single-page applications.

Latency is of course also a problem. While not as big of a problem for archival purposes, there could be a missed opportunity to help the servers host the resources that are being requested. In cases such as when a website is slashdotted or when bandwidth outside a LAN is pricey, it might be opportune to instead fetch these files from local peers in the network. Having a low latency allows the network to provide even more value and a great user experience, and the incentive could have web servers themselves joining the network in some form.

To summarize, the central problems needed to be solved for an ideal peer-to-peer archival system are as follows:

- A way to establish trust and validation without any parties being pre-staked
- A way to determine which files to replicate in the network
- Higher efficiency when storing and transmitting files
- Systematic crawling of the entire internet.
- Better way of snapshotting javascript-dependent websites
- Cooperation with web servers leading to improvements for both the network and website host

3.2 SOLUTIONS

This section discusses some proposed solutions to the problems described above. This section also intends to limit and choose which part of the problems to focus on, given the scope of this report.

This report argues that three points from the list are the most important problems to solve before an ideal peer-to-peer archival system is created. These are the trust and validation problem, the replication problem and the snapshotting problem.

Snapshotting websites with differentiating behaviour based on user interaction and procedural programming languages like javascript is

a difficult problem to solve. As more and more websites become frontends for applications that are running on server hardware, and as javascript is utilized to a higher degree, it gets harder for an archivist user to create a fulfilling archived version of a website since the current state of the website does not represent the website as a whole. As the difficulty of solving this problem is deemed too great for this report, this report will not focus on this topic.

The argument for trust, validation and replication is that without trust and replication of files in the network, the collective resource store is doomed to never grow, and therefore not be very useful. An archive is only useful if it contains resources worth archiving. Figuring out this problem will lay the groundwork for a completed ideal peer-to-peer archival system. These problems are what this report will focus on designing, implementing and evaluating a concept for while giving loose ideas for how to solve the rest.

The idea this report wants to propose is a reputation system. The main part of this system is, that if a peer can verify that someone did something good, the peer should be more trusting of their work. By having these messages cemented in a log-like structure with non-repudiation, peers would have to prove themselves before the resources they are storing are spread and trusted in the peer network. When the trust problem is solved, the replication problem can be somewhat solved by systematically having a higher amount of redundancy to trusted files. By doing this, the two other main problems of a wholly peer-to-peer archival system can be somewhat remedied.

When the system is running, the other issues can be solved over time and built upon the core foundation. Cooperation with web servers could for instance have a file at www.xyz.com/hypercache.conf which when an origin web server is under load asks new peers (which only prefetch the .conf) to grab a new version from the peer-to-peer network instead, thus alleviating traffic and improving both the historic and current availability of the website.

The idea of having higher efficiency will first of all likely change with time. As technology progresses, storage and bandwidth get cheaper while at the same time compression algorithms get better. However, there might also be smarter ways to save on bandwidth in the network. By knowing which files a peer already has in the network, bandwidth can be saved by only sending the new pieces of information the peer requires. This report will discuss this more in Section 4.7.

As a last discussion point, for crawling the entire web systematically. this might be of somewhat lesser importance. Since the number of peers are limited, it might be of greater importance to the network to crawl some websites deemed more important from a societal view daily, than it would be to crawl other websites of lesser importance. With the growth of a peer-to-peer network however, it could be that the amount of increased bandwidth in the network would allow a

considerably large amount of the internet to be archived. This report will not directly touch this problem but will orchestrate some form of timely crawling in Section 4.8.

This section discussed some identified problems in the ideal peer-to-peer archival system. The major problems included those of ensuring resource validity, trust, replication and usefulness. Because of the difficulty, this paper does not attempt to improve the usefulness of resources but will focus on implementing a trust/reputation system to solve the issue of the other three issues.

4 DESIGN OF ARCHITECTURE AND EVALUATION

This chapter presents the designed reputation system, and the more technical details and inner workings of how reputation is gathered and used in the evaluation of a resource's legitimacy. This chapter will also use the designed reputation system to design solutions to some of the minor problems discussed in Chapter 3.

4.1 WHAT IS REPUTATION

Before designing technical aspects of the system, it can be beneficial to think about what interactions would lead to a positive reputation. Some peer A should only think better of peer B if peer B has done an action which peer A can undeniably trust is correct. These two reputation-gaining actions are

1. If peer A approves a hash of a website peer B posted on the AOL
2. If peer B approves a hash of a website peer A posted on the AOL

The reputation system builds upon these two aspects to establish the reputation of archivist peers in the system. The trustworthiness of information in the network is then based upon the reputation each user has, which in turn is based upon the trust each archivists peer puts in other archivist peers. All messages should be public such that it is possible to calculate the reputation that other archivist peers have in another archivist peer, which can also be used to calculate a reputation score on a piece of information. With this in mind, it is time to design the technical aspects of the reputation system.

4.2 THE APPEND-ONLY LOG STRUCTURE AND ASSUMPTIONS

For the system this report envisions, the low-level communication makes use of an append-only log (AOL), such as the one Hypercore[1] provides. It is also assumed that the append-only log has non-repudiation, meaning once a peer appends something to the log, it cannot be taken back. For the sake of this report, it is also assumed that Sybil attacks[2] aren't possible on the AOL. It is assumed that all

1 https://hypercore-protocol.org/
2 Attacks where a bad actor has multiple identities, see more at https://en.wikipedia.org/wiki/Sybil_attack

messages get delivered to all parties with a reasonable upper bound in time. It is also assumed that the network consists of a majority of non-malicious peers. Lastly, it is assumed that the websites crawled by peers contain no information that could harm the individual user's privacy and that the peers appending websites to the AOL are okay with sharing that they have visited those websites.

When peers want to complete an action, they append a new message to the AOL. Using this kind of message-board approach, every interaction a peer will make will be public and locked in the AOL. Furthermore, to ensure the validity of the post, each user will sign their message. In the log, it is also assumed that when a user appends a message, a timestamp is added to the message.

Having gone over the technical communication layer of the system, the next section discusses what needs to be communicated in the log.

4.3 THE INFORMATION ON THE AOL

When an archivist peer connects to the network, they shall first grab a full copy of the AOL. On the AOL, there exist three kinds of messages, each corresponding to a peer doing a different action. They correspond to either a submission or validation of either a new website or a new private tree intersection.

The first message type is when a peer wants to add their view of a website to the AOL. This message contains a special kind of hash tree of the website. The top-level document hash is stored in full. The document hash can be used to verify website integrity when a peer in the network tries to browse the archive and is used for content-based addressing. The hash of all child nodes is also stored in the tree, but only the first few bits of the hash. These will be used as a way to have an idea about which content changes between versions of a resource in the tree while saving bandwidth. The URL of the website is also added. This first message type is thus a declaration/record of a website in the AOL.

The second type of message is also posted when a peer adds their view of a website. As above, the peer has visited a website and hasn't found an identical hash previously in the AOL. If they are not the first to submit a version of the specific website, they find the most similar hash tree in the AOL and from that create a hash tree with only the elements the peer has in common by means such as private tree intersection. The hash of child nodes in the tree are shortened.

The third type of message is a validation message. In this case, a peer has visited a website that is in a message type 1 or 2. This type 3 message contains a reference to the id of the message you want to approve. It also contains a special kind of signature added to the message too. This will be a signature consisting of 1. the top-level hash, 2. the full hash of the children of the top-level hash (which

isn't publicly available on the AOL) and 3. the verification key of the signee/peer who is doing the message.

With the message types described, it is time to explain why the messages are what they are, and how the control flow of the system works.

4.4 SYSTEM BEHAVIOR

4.4.1 *Crawling Peers*

When a peer crawls a website and wants to put information into the network they will:

- Calculate the hash tree of the visited website.
- Check if top-level resource hash exists on the AOL

  If hash is not on AOL

  - Create signature by hash(children, URL + my id).
  - Append hash tree with all nodes but root nodes shortened

  If hash is on AOL

  - Sign the website using the children of the root node the website URL and your id, saying you see this too.
  - Append to the AOL

4.4.2 *Validator Peers*

When a message type 1 or 2 is added to the AOL, peers in the AOL should act to verify the version, by visiting the corresponding website. This means if a bad actor adds a malicious hash tree to the AOL, it should quickly be fixed by a swarm of pure peers. However, care has to be taken to not slashdot a website. If all peers were to visit the website at the same time, the network could be maliciously used as a DDoS attack. A randomized approach is taken here, where depending on how many other peers a peer is connected to, there is a certain chance that this peer will visit the website in question and try to verify it. Until there is a certain amount of verifications, the chance that this peer will visit the website will increase more and more. Written in pseudo-code, it has the following structure:

- A message of type 1 is appended in the AOL for a website this peer isn't on timeout from, or this peer has scheduled its verification of a website.
- This peer generates a random value

- If the value is bigger than some threshold, the user visits the site else the user waits
- If some fixed amount of time has passed, and the resource isn't trusted yet, the threshold for the random value increases a fixed amount until this peer has either added their verification or made a new message type 1 on the network.

The peers should also regularly schedule redoing this behaviour for websites following the behaviour described in Section 4.8.

4.5 USING THE REPUTATION SYSTEM

Now, this is the most important part of the AOL. With all the information and behaviour described in the previous sections, it is now time to apply this to figure out which resources that most likely are trustworthy, and which resources aren't.

There are two cases for resources shared in the reputation system. In the first case, the resource in the AOL has no competitor within the timeframe of verification messages on the website. In this case, we just need to know if the resource is either valid or invalid. Here we use the approach, that as long as at least one pure peer has validated the website, it must be true. Conversely, this can also be thought of along the lines of as long as there are more verification messages than there are malicious peers in the system, the resource must be correct. However, a problem here is that the network size might change as peers churn. This report's solution to this is to look at the total amount of trust the system has at the point in time of a verification message. If this amount of trust is over a set threshold, the resource is set as trusted, as pure peers should have the majority of trust in the network.

When multiple versions of a resource exist within the same timeframe, this becomes a more difficult matter. In the case both an incorrect site and a correct site are submitted closely together in time, one of the versions would get a majority of the trust in the system per the logic that archivist peers will go to the URL to validate the website they see. This results in a singular most trusted version of an URL. If multiple correct versions exist for large enough groups of peers in the system, they would also be trusted as long as they can clear the required threshold.

However, for some websites where many specific groups of people get a personalized website, such as IP checker websites like `https://www.whatismyip.com/`, it may be the case that no two websites posted are alike, which is quite unsolvable for this reputation approach. This is where the private tree intersection comes into play. Anything different between the websites of the different peers will be removed, and from that comes a common base that the network will be able to agree is trustworthy.

4.6 REDUNDANCY

When it comes to ensuring somewhat redundancy of resources in the peer-to-peer archive, resource trustworthiness can also in this case be used to prioritize which resources should be kept on a peer. The peer should, first of all, keep all resources it has visited. These are the resources it has an infinite amount of trust in, as the peer observed the resource themselves as truthful.

From here different approaches can be taken to storing the files. One way is to use a tiered redundancy system. By using the reputation system to calculate the system trustworthiness of resources the archivist peer has not observed themselves, more redundancy could be given to more trustworthy files. A simple algorithm could combine the trust of the file, the number of potential peers who should already have said file and randomness to decide which files peer x should fetch and store the resource in the peer-to-peer filesystem without communicating with other peers.

4.7 BANDWIDTH SAVINGS AND SPACE CONSIDERATIONS

Using the meta tree also provides some opportunity to do smarter transfers between peers. To understand this take the example of two versions of a news article, with a typo fixed in the newer version. If a peer on the older version wants the new version, the newer peer can send only what's changed. They know which files the requester has, as it must be the set of websites they have posted, verified or requested. Therefore, sending only new parts of the tree could be beneficial. Furthermore, when storing the files on peers' hard drives, diffs could be used instead of storing different versions in all their glory if the similarity is big enough, granting even more storage and bandwidth capabilities for the network.

Another consideration is, that even when only storing the meta-information and not the actual resources, it can still give some insight. If the hash tree only changes at a certain node each day, the AOL can be used to show this to a user, such that they know what elements on a resource change. While being able to see the changes too would be preferable, it is better than nothing.

4.8 TIMEOUTS AND TIMESTAMPS

When it comes to the reputation system described in this report, there is an additional benefit when it comes to placing the resources in time. Most online archives only store the date that a resource was archived. By having an upper bound on the time a message takes to make it on the AOL, this reputation system also allows you to have an idea of the range of time a resource was a specific version.

In Section 4.4.2 it was hinted that peers could query websites regularly to keep the archive up to date. By checking websites daily, for instance, the resulting messages generated on the AOL allow the network to be much more precise in determining a version of a resource's timespan. Having different peers check different websites regularly also allows the network to potentially query and keep up-to-date with a much larger part of the internet.

4.9 ARCHITECTURE DESIGN SUMMARY

These sections described a design for a reputation system based on an append-only log (AOL). The design works by having peers append the version of a resource they see to the AOL, and have other peers in the network validate or update the log with their own version of a resource. Peers in the network regularly visit websites in the log to append new versions. Based on validation messages and relations between peers in the AOL, a reputation metric for peers that can be used to calculate resource trustworthiness in the network is generated. This metric is then used to solve issues of file replication and validity. Replication is based on the principle that more trustworthy files should be replicated to a higher amount. Validity is based upon the amount of reputation the archivist peers of the system as a whole placed into a file relative to the size of the system.

4.10 EVALUATION DESIGN

This evaluation of the reputation system happens via simulating the behaviour of the system. In a simulation, different scenarios can be tested without having to implement parts of the system that this report isn't focused on. The system needs to work under different conditions, and seeing and evaluating its behaviour can provide interesting metrics as to how much if at all the system contributes. To get generalized results, data can be generated at different simulated parameters such as:

1. Number of pure peers in the system
2. Number of malicious peers in the system
3. Types of malicious peers in the system.
4. Churn rate of peers in the system
5. Running time of the system
6. Number of resources to crawl in the system
7. Rate at which resources should change

8. Parameters for mathematical models used to calculate reputation.

After running different instances to make sure the results are somewhat generalized, the report will extract performance metrics from results and evaluate at which points the system shines and at which points it does not, with a focus on the original hypothesis made in Chapter 1. The performance metrics used to evaluate the system are split in two. The network needs to be evaluated based on both how well it separates malicious information from non-malicious information, and how well it places correct information on an estimated timeline of a given web resource. Further description will be described in Chapter 6.

5 IMPLEMENTATION

This section describes the prototype that was built following the design described in Chapter 4.

5.1 LIMITATIONS OF THE PROTOTYPE

Given the time constraints of the bachelor project, and to make sure this report is focused and concise, some choices have been made to limit the scope of the prototype.

This prototype has a high focus on the reputation system itself. It focuses on how well the system can date websites, and how it holds up under the attack of malicious peers. This prototype does not attempt to implement an underlying peer-to-peer network, nor the sharing of files between peers or supporting private tree intersection. The prototype aims to implement two systems:

- A P2P simulation system that can simulate the AOL and peer-to-peer system, including how different peers could behave in the system, but done locally.
- A reputation system which uses data from the simulated scenarios to calculate which files are trusted and not, and at which time a version of a website is presumed to have been live.

The implementation of these is described in further detail in the below sections.

5.2 LANGUAGE AND FRAMEWORK

The prototype is built using javascript and the nodeJS framework and can be found on `https://github.com/StereotypicalCat/HyperCache`. The choice of language and framework was made to support a possible inclusion of a real peer-to-peer backend in the form of Hypercore[1], which has a javascript implementation. The author also has previous experience in javascript, which lessens the burden of having to learn a new language.

5.3 OVERALL STRUCTURE

The overall structure of the prototype is split into two major subsystems, as described in Section 5.1. The AOL, simulated peers and

1 `https://hypercore-protocol.org/`

website data are simulated in one part of the codebase, whereas the reputation system and data collection are separated into another part of the system. A diagram has been included on page 27 which can give a graphic overview of the files and roles of the system in further detail.

A benefit of this, which is also an important point in the design of Hypercache as a whole, is that this sharp separation of the reputation system and the simulation system could allow further updates and refinements to the trust system in the future without having to throw away the underlying information in the AOL and start from scratch.

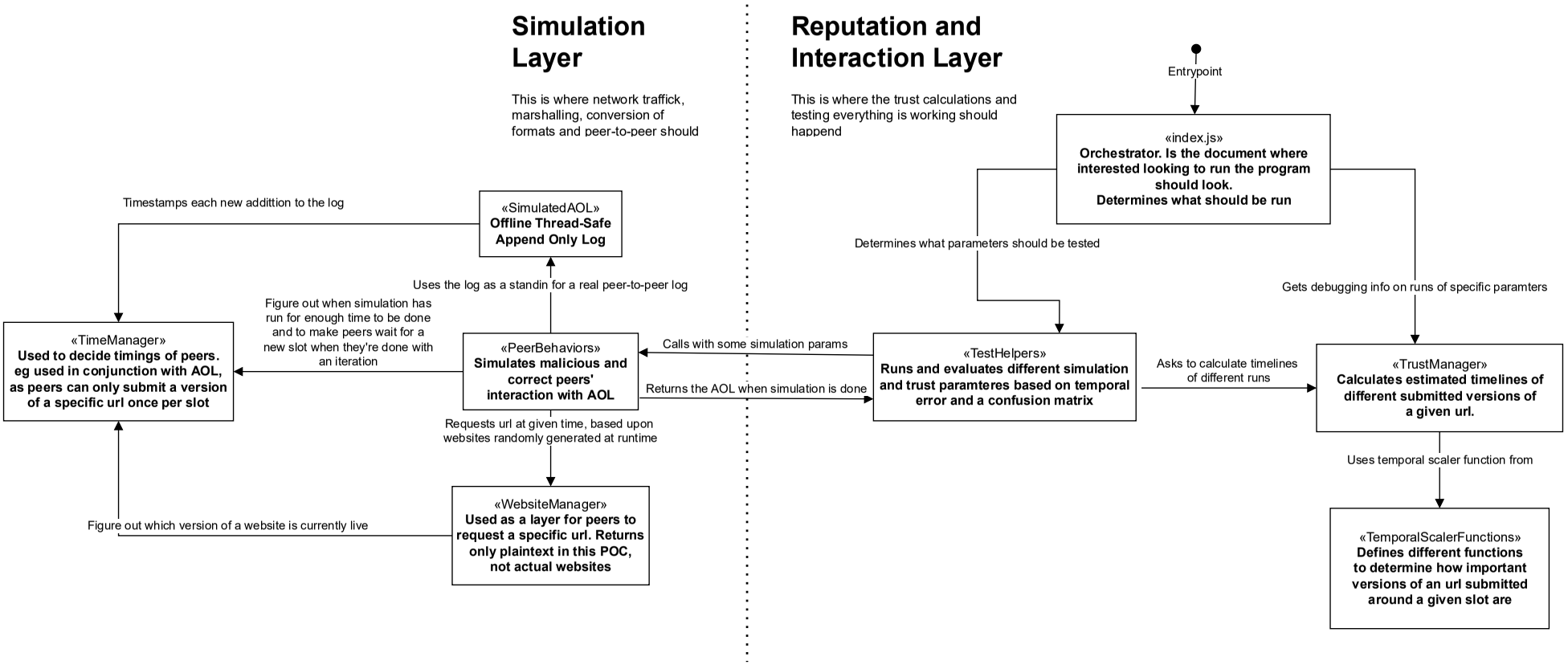
5.4 SIMULATION LAYER

This section describes the main parts and choices made when the mock for the network layer was created.

5.4.1 *Time*

HyperCache uses the idea of time that works in the same sense that slots in some popular peer-to-peer blockchain networks like the Bitcoin and Ethereum networks do. Time is divided into slots of a certain size. Behaviour is then restricted by these slots, such as how a peer may submit a version of a website only once a slot. A low slot length would create a more detailed timetable of different URLs, but too low and the network overhead might mean that the network grows uncertain on which websites are live in a specific slot because of lack of validations, and therefore malicious actors could become a problem. A shorter slot length could also require more bandwidth, given that peers can submit versions more often. Choosing a slot length is a tradeoff of archive precision and definition versus bandwidth.

One could in this case imagine that a slot length of 24 hours, or maybe even 7 days could provide sufficient historic information for most websites.


Simulation Layer
This is where network traffick, marshalling, conversion of formats and peer-to-peer should
Reputation and Interaction Layer
This is where the trust calculations and testing everything is working should happend
Entrypoint
«index.js»
Orchestrator. Is the document where interested looking to run the program should look.
Determines what should be run
Determines what parameters should be tested
Gets debugging info on runs of specific paramters
Timestamps each new addition to the log
«SimulatedAOL»
Offline Thread-Safe Append Only Log
Uses the log as a standin for a real peer-to-peer log
Figure out when simulation has run for enough time to be done and to make peers wait for a new slot when they're done with an iteration
«TimeManager»
Used to decide timings of peers. eg used in conjunction with AOL, as peers can only submit a version of a specific url once per slot
«PeerBehaviors»
Simulates malicious and correct peers' interaction with AOL
Calls with some simulation params
Returns the AOL when simulation is done
«TestHelpers»
Runs and evaluates different simulation and trust paramteres based on temporal error and a confusion matrix
Asks to calculate timelines of different runs
«TrustManager»
Calculates estimated timelines of different submitted versions of a given url.
Uses temporal scaler function from
Requests url at given time, based upon websites randomly generated at runtime
Figure out which version of a website is currently live
«WebsiteManager»
Used as a layer for peers to request a specific url. Returns only plaintext in this POC, not actual websites
«TemporalScalerFunctions»
Defines different functions to determine how important versions of an url submitted around a given slot are

5.4.2 *Peer Behaviors*

As one of the propositions of the report is to test the system's resistance to malicious peers, the system naturally needs to simulate peers doing different behaviour. The project models this with simulations having a selection of different peer behaviours to choose from, which are then instantiated given the appropriate simulation parameters. For each URL in the simulation to be crawled, a peer will request the URL, do something specific to it per its behaviour, and then perform the correct AOL behaviour to add their document to the chain. Which document they add is then the central piece that differs between them. In the project, the following behaviours exist:

- `The pure peer`: A peer which only submits correct information to the chain. They request the website, and when they get the response, simply just add it to the chain.
- `The solo consistently malicious peer`: A peer which consistently sends a malicious version of a website as the version they see on the chain. Works solo, and thusly the version they consistently submit should be unique.
- `The grouped consistently malicious peer`: Same as above, but instead of each malicious peer submitting a unique version, they all consistently submitted the same wrong version.
- `The solo sometimes malicious peer`: A peer which only sometimes submits a malicious version of a given website, based on simulation parameters. The malicious website they submit is not a specific site. Submits a unique malicious version.
- `The grouped sometimes malicious peer`: Same as above, but all peers submit the same wrong version of a random site.
- `The solo sometimes malicious specific site peer`: a peer which chooses a specific subset of websites and only posts wrong versions of them, and correct versions of every other site. Works solo, so the malicious version they submit is unique.
- `The grouped sometimes malicious specific site peer`: Same as above, but all peers target the same subset of websites
- `The past-focused one version peer`: Will consistently send in a correct version of a website, but chooses a version from a website's past and consistently submits it.
- `The past-focused last version peer`: Will consistently send in a correct version of a website, but from the version that just last appeared.

- `The new version spammer`: Will send in a random new unique version of each URL each iteration.

The configuration of these peers, such as how many peers there are in a system and which behaviour each peer employ are controlled by an instance of the simulation settings object. Other than the behaviour, there is some simulated delay between each peer's action with the chain, and the time it takes to request a website. The below pseudo-code illustrates how the peers function

```
startPeer(peerNum, urls, documentChangeStrategy) => {
    const delayToWait = ...//random delay calculated from simulation
        params that determine when a peer joins the network. Between 0
        and max_simulation_time

    // Shuffle the order a peer crawls the web in
    const thisPeersUrls = [...urls]
    this.shuffle(thisPeersUrls)

    let mainLoop() => {

        // Stops when the simulation is over.
        if (this.shouldStopPeersCheck()){
            return;
        }

        // Runs the peer behaviour included with the arguments this is
            called on.
        documentChangeStrategy(currentPeerNum, thisPeersUrls)

        // Can randomly churn after a peer has crawled all their websites
            .
        if (random() < this.simulation_parameters.chance_a_peer_churns){
            this.shuffle(thisPeersUrls)
            currentPeerNum = await this.getNewPeerNumber();
            //console.log("Peer " + peerNum + " has churned to " +
                currentPeerNum);
        }

        // Waits a bit before starting another iteration, based on
            simulation settings.
        setTimeout(mainLoop, this.simulation_parameters.peer_timeout *
            1000);
    }
    setTimeout(mainLoop, delayToWait)
}
```

Listing 1: The control flow of a simulated peer

5.4.3 *Websites and the AOL*

To simulate the network traffic between a peer and the network, both in terms of the websites they crawl and interacting with the AOL, the SimulatedAOL and WebsiteManager were made. They both use fake

local data structures to implement simulated behaviour, and function on the bases of plaintext messages in non-HTML structure, without hashing. This choice is made on the assumption, that if the trust system works with plaintext messages, it should work with any other kind of messages too.

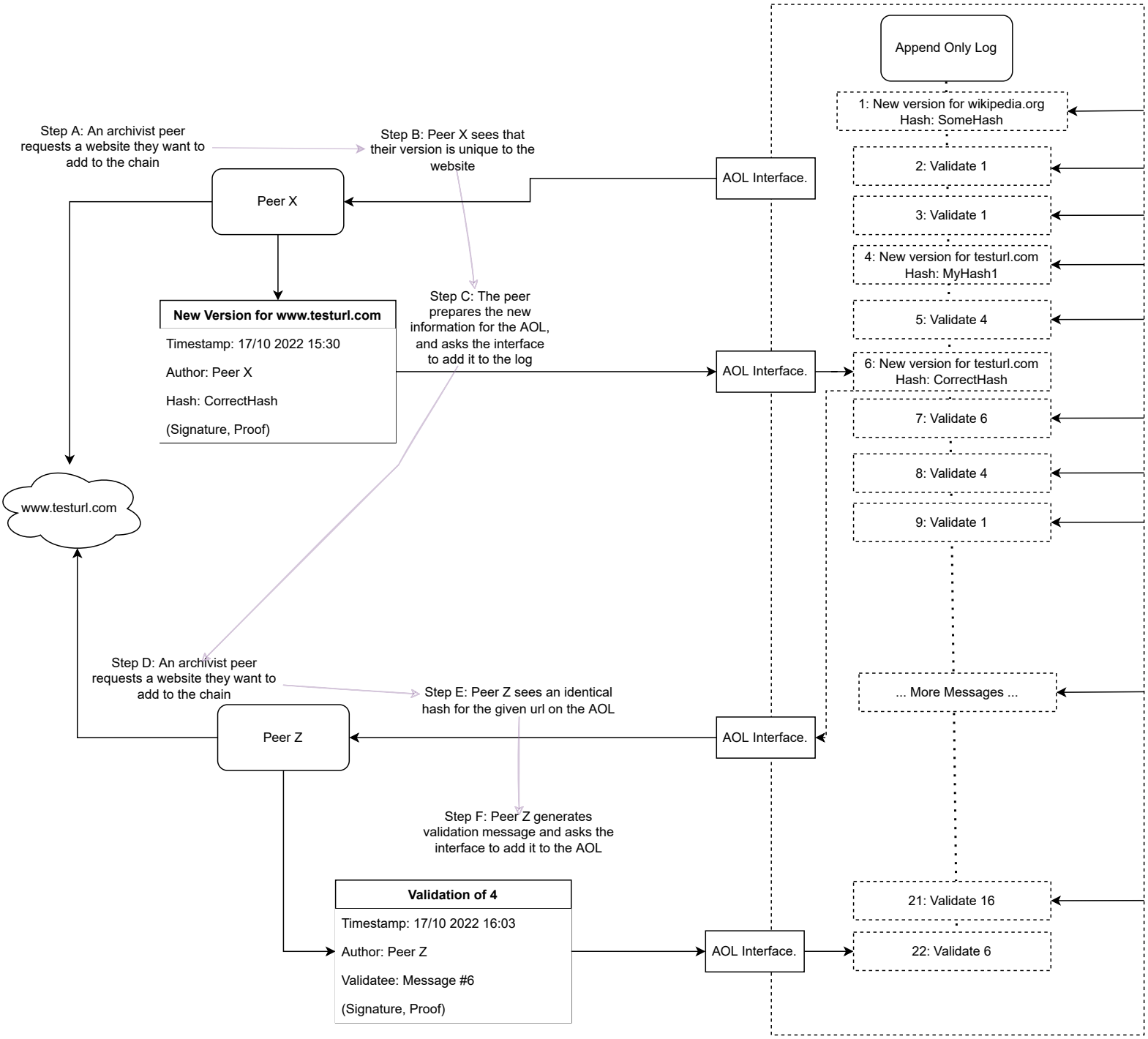
Concerning the websites, two things are done. Firstly, a random number of URLs and a random number of versions of those URLs are generated, per minimum and maximum values described in an instance of simulation settings. Next, the requester function is described, which finds the correct version of an URL in the current slot to serve a user.

The AOL is simply a data structure to which when one submits a new piece of information on an URL either adds it as a new version, or a validation on the version. This data structure could also be generated from timestamped messages in a log, which the data structure could also print the appropriate example of.

In terms of the AOL, in its simulated nature, it is most likely a lot more effective than a real peer-to-peer AOL chain would be. While one could work with delays here too, the author deems this delay a bit more complex to model and has thusly avoided doing it for the fear of creating something that would show an incorrect image. One could in this case argue, that the time it takes to orchestrate the network can be made insignificant if slot length increases. To see an example of how AOL works, see page 31.

5.5 TRUSTMANAGER

The other part of the prototype is the trust system itself. With the behaviour that creates the data described, this section is all about how the trust gets calculated. When a potential user wants to browse the archive, they will input an URL and get what the network is the proposed correct timeline. However given that the network cannot for sure know that no malicious versions have been submitted, the users need to choose some parameters based on how much they're willing to risk that a specific slot can have a wrong-correct version (correct version of the website but one that isn't live in this slot but is in the future/past) or a downright malicious one. How different values affect the correctness of the network will be evaluated in Chapter 6.


Append Only Log
1: New version for wikipedia.org
Hash: SomeHash
2: Validate 1
3: Validate 1
4: New version for testurl.com
Hash: MyHash1
5: Validate 4
6: New version for testurl.com
Hash: CorrectHash
7: Validate 6
8: Validate 4
9: Validate 1
... More Messages ...
21: Validate 16
22: Validate 6
Step A: An archivist peer requests a website they want to add to the chain
Step B: Peer X sees that their version is unique to the website
Peer X
AOL Interface.
New Version for www.testurl.com
Timestamp: 17/10 2022 15:30
Author: Peer X
Hash: CorrectHash
(Signature, Proof)
Step C: The peer prepares the new information for the AOL, and asks the interface to add it to the log
AOL Interface.
www.testurl.com
Step D: An archivist peer requests a website they want to add to the chain
Step E: Peer Z sees an identical hash for the given url on the AOL
Peer Z
AOL Interface.
Step F: Peer Z generates validation message and asks the interface to add it to the AOL
Validation of 4
Timestamp: 17/10 2022 16:03
Author: Peer Z
Validatee: Message #6
(Signature, Proof)
AOL Interface.

The way the system generates a timeline of an URL is described by the following pseudocode:

```
calculate_approximate_timeline_of_url(url) {

    // An array to store the timeline in, with each element representing a slot
    let calculatedTimeline = []

    // Loop over all available slots, from when the network started to now.
    for (let time = 0; time < endOfTimelineSlot; time++) {

        // Get the different unique versions submitted for an url.
        let hashes = AOL.get(url)

        // Calculates the total amount of trust in a certain slot, to be used in calculating confidence later.
        let totalTrustOfAllVersions = 0;
        let totalVersions = 0;

        for (const [hash] of hashes) {
            let trust = calculate_trust_of_version_at_time(url, hash, time)
            totalTrustOfAllVersions += trust;
            totalVersions += 1;
            }

        // Now that we know the total trust, we can calculate the relative amount of trust placed in each version.
        let trustedVersions = []
        for (const [hash] of hashes) {
            // Do note if minimum_confidence > 0.5, only 1 version of a website can ever exist :).
            if (trust / totalTrustOfAllVersions > minimum_confidence) {
                trustedVersions.push({hash: hash, confidence: trust / totalTrustOfAllVersions})
            }
        }
    }

    calculatedTimeline.push({timeStart: time, versions: trustedVersions, totalVersions: totalVersions})
    return calculatedTimeline;
}
```

Listing 2: Estimation of the timeline of a given URL

Specifically important here is the `calculcate_trust_of_version_at_time` function. To explain how this function works, two other functions need to be explained first. Namely, how a user accumulates trust (Section 5.5.1) and how a slot a validation is submitted in affects the trust it adds (Section 5.5.2).

5.5.1 *Reputation of User*

Calculating the trust of the user is done by following the two trustful actions described in Section 4.1. When evaluating the network without taking a specific peer's view, there's no bonus to peers you have interacted with. This approach was taken for the prototype for simplicities sake.

To calculate the full non-subjective Reputation of a user, the system loops over all peers in the system and calculates the trust other peers have in a source peer. The amount of reputation each trustful action carries is parameterized. The trust a source peer has in a target peer can be described by the following pseudo-code:

```
calculate_user_trust_of_user(aol, peerIdSource, peerIdTarget) {

    // We trust the work we did ourselves.
    if (peerIdSource === peerIdTarget) {
        return Infinity;
    }

    let trust = 0;

    // loop through all websites in the AOL
    for (let [versions] of websites) {
        for (let [versionInfo] of versions) {

            // if the source peer submitted the version
            if (versionInfo.peerId === peerIdSource) {

                // if the target peer validated the version
                for (let i = 0; i < versionInfo.validations.length; i++)
                    {
                    if (versionInfo.validations[i].peerId ===
                        peerIdTarget) {
                        // Add
                        trust += trust_for_validating_resource;
                    }
                }
            }

            // if the source peer validated the version
            if (versionInfo.peerId === peerIdTarget) {

                // if the target peer submitted the version.
                for (let i = 0; i < versionInfo.validations.length; i++)
                    {
                    if (versionInfo.validations[i].peerId ===
                        peerIdSource) {
                        trust += trust_for_submitting_new_version;
                    }
                }

            }

        }
```

```
    }

    return trust;
}
```

Listing 3: Calculation of the trust one archivist peer has in another archivist peer

By calling this method with source peer $\neq$ target peer, the reputation a peer has accumulated in the system can thusly be calculated. Do note this prototype expects peers to get a number from 0,1,2,... when they enter the network.

5.5.2 *Temporal Scaling*

When calculating an estimated timeline of a website, one wants to ideally use as much information as possible. An important aspect of calculating the trust of a version at a given slot is determining the importance of the difference in slot time on other versions and validations.

Given the difference in time for the current slot and the time a version is submitted/validated, how much influence should that submission get? This prototype allows any function that given input x returns a value between 0 and 1 to be used. Important here is to note that the function choice is very important for the network. When considering which function to use, one could ask questions such as: are validations/versions from a slot in the future more impactful than those of the past? How many slots back and forward should be considered in the trust calculation? This prototype uses a simple logistic function (see Figure 1), but testing different functions could be interesting in the future. The pseudo-code of the function is as follows

```
const inverseLogsticFunction = (x, trust_settings) => {

    // Curves maximum value
    const L = trust_settings.logistic_L;

    // The logistic growth rate of the curve
    const k = trust_settings.logistic_k;

    // The x value of the sigmoid's midpoint on the x-axis
    // Basically controls the bias of future versions vs previous
        versions
    let x0 = trust_settings.logistic_x0;

    let logisticPart = (L / (1 + Math.exp((-k) * (x - x0))));

    return 1 - logisticPart;
}
```

Listing 4: Temporal Scaling Function

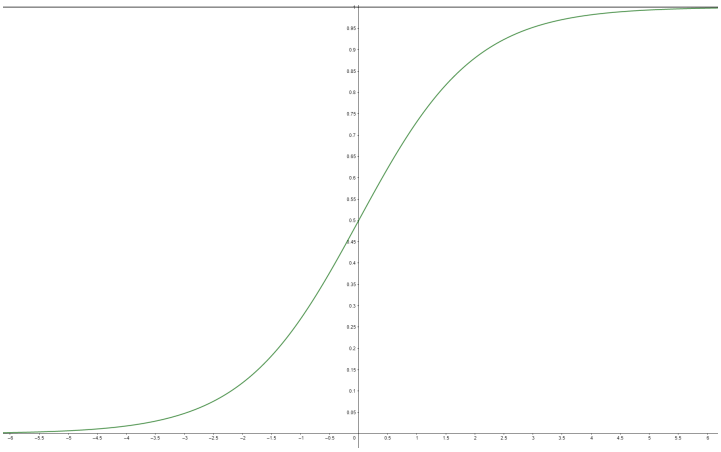


Figure 1: Example of a logistic function plot ($\frac{L}{1+e^{(-k)\cdot(x-x_0)}}$)

5.5.3 *Trust of version at time*

Finally, observe how the trust of a specific version at a specific slot can be calculated. Taking a look at this function in pseudocode, it looks like this:

```
calculate_trust_of_version_at_time(url, hash, slot) {

    // The initial trust of a version is determined
    let trust = calculate_full_trust_of_user(websites.get(url).get(hash).
        peerId);
    let timeSinceInitialSubmission = websites.get(url).get(hash).time -
        slot;

    // Scaled by the logistic function in this case
    let initialUserScaleValue = getTemporalScaleFactor(
        timeSinceInitialSubmission, this.TrustSettings)
    trust *= initialUserScaleValue;

    // Bonus works to incorporate some sense of majority vote, so one
        person with a lot of trust doesn't completely decide the correct
         version.
    let bonus = 1;

    // the trust of the version is the sum of the trust of the peers that
         validated it plus the peer who placed it there
    for (let i = 0; i < websites.get(url).get(hash).validations.length; i
        ++) {
        let user_trust = calculate_full_trust_of_user(websites.get(url).
            get(hash).validations[i].peerId);

        // Scale the trust accumulated with the time since the validation
        let timeSinceValidation = websites.get(url).get(hash).validations
            [i].time - slot
        let scaleValue = getTemporalScaleFactor(timeSinceValidation, this
            .TrustSettings)

        trust += user_trust * scaleValue;
```

```
23          bonus += this.TrustSettings.popolous_multiplier * scaleValue;
24      }
25
26      trust *= bonus;
27      return trust;
28
29  }
```

Listing 5: Calculating the confidence level of a given version of a URL at a given timeslot

With this, the most important parts of the implemented prototype are described. In the next section, the robustness of the system is evaluated in accordance with Section 4.10.

6 EVALUATION

This chapter evaluates the prototype built in Chapter 5. The hypothesis is that given no prior staked parties, a network of independent peers can agree, with high certainty, upon which version of a resource is the correct resource at a given time.

6.1 METHODOLOGY

To ensure meaningful tests that give an accurate representation of the performance, choices have to be made concerning simulation realism and which values to test. For this reason, randomness in the system is kept at a minimum, even if real-life behaviour might differ. Unless stated otherwise, this affects the following simulation parameters:

- In each simulation run, every website will have a constant number of versions per simulation.
- The time before a peer starts acting in the network is between 0 and 0.3 seconds, to make sure no peer gets a starting advantage. After a maximum of 0.3 seconds after the last peer is initiated, all peers are connected and participating in the network.

Some of the elements of the design will also be tested as constant values. The reasons are as follows

- `Different ratio of reputation for Validating vs Adding a new resource`: This is more of a behavioural change in the system, and when all peers are designed to crawl the entire URL space of the simulation, this parameter does not make sense to change. Therefore the amount of reputation gained for validating and adding is 1 point.
- `Popolous Multiplier`: The idea from this was to have an opportunity for the masses to outvote the few, as it is assumed the network consists of a majority of pure peers. However the current implementation in the prototype does not reflect the ideal behaviour, and the population bonus is thus set to 0.
- There will be a set amount of URLs that are crawlable in the system. This is set to 6, to ensure that the limitation of the simulation will not influence the results, as every peer should be somewhat able to crawl all sites in a given slot length.

- The number of slots the simulation runs should ideally run as long as possible. To represent a system and create some data, each simulation will run for 40 seconds, which each slot having a length of 1 second, meaning 40 total slots.

Having gone over the limitations of testing the prototype, the parameters that will be tested and is interesting to test will be described.

- Information Density: The total number of versions each URL will go through in the duration of the simulation. A density of 100% means there exists 40 versions / 40 slots, a density of 25% means 10 versions, etc. Values tested are $10\%, 35\%, 60\%$.
- Churn rate: The chance a peer will churn and come back as a new peer. A peer will complete a loop through the simulations URL space, and then randomly based on churn rate decide if they churn. Values tested are $10\%, 25\%, 40\%$
- Minimum Confidence: The confidence the network has to have that a given version exists in a timeslot to be valid. This will be tested at 30%, 45%, $60\% and 75\%$ to allow the timeline to potentially have 3, 2 and 1 trusted versions respectively.
- Logistic x_0: The higher x_0, the more history, both past and future, will be considered. Values tested are 0, 3.5, 7 and 10.5
- Logistic k: The slope of the curve. Lower values go towards constant importance of each document, higher is more like 0-1 classification. Values tested are 0.5, 1.5 and 2.5.
- Amount of Malicious Actors: The number of malicious peers divided by the number of pure peers. Values tested are $0, 3, 6, 9, 12$

When scoring these runs, 2 "measured" and 2 derived metrics are calculated.

- Confusion-Score: based on a confusion matrix. Given all time slots, counts how many trusted correct versions, non-trusted correct versions, trusted incorrect versions and non-trusted incorrect versions exist in the network. From these, it calculates the confusion score as $\frac{prototype_correct_guesses}{theoretical_best_score}$
- Temporal Score: Looks through each slot, and if the correct version of a website is one of the trusted versions from the reputation-systems generated timeline, it adds a point. Temporal score is then calculated as $\frac{TotalPoints}{TotalSlots}$
- Total score: $\frac{temporal_score + confusion_score}{2}$.

- Total score strict: if a trust simulation trusts even one wrong website, 0, else it is equal to the Total Score calculated above.

Having defined the parameters, it is time to define the scenarios. It is important to evaluate how the system runs both in realistic scenarios and how each erroneous/malicious behaviour affects the system. For more realistic scenarios, two configurations are tested. The pure simulation is a best-case scenario where there are no malicious peers. The High Chaos simulation then represents the opposite, having 10 pure peers and 3 of each of the different incorrect behaviours. For the individual actors, tests are run per the previously stated methodology.

6.2 TOTAL AMOUNT OF DATA

To give an idea of the number of data points for the next couple of sections, please refer to Table 1.

	Sim Params	Reputation Params	Total Combinations
Pure	27		1296
High Chaos	9	48	432
Sometimes Malicious Site-Specific	144	18	2592
Consistently Malicious New Version Spammers Past One Version Past Last version	36	48	1728

Table 1: The total amount of unique data points for testing and evaluating the behaviour of the system. To give an example of what this represents, if information density is tested at 3 different values and churn rate is tested at 5 different values, a total of 15 unique simulations (sim params) will be run, and each of these will generate a timeline with each unique set of reputations parameters, which results in a total number of data points as seen to the left. Do note this naturally includes both grouped and solo versions of each actor. So for instance, consistently malicious solo gets 1728 data points generated for itself.

Compromises of course have to be made when generating this data. In general, running the simulations is a cheap endeavour, but generating the timelines is expensive. The data is generated in a multithreaded manner. To make sure the simulation runs independently of high CPU usage, all simulations are run in parallel, and only when all simulations are finished do the threads start running the reputation calculations.

For sometimes malicious and site-specific malicious, more simulation parameters needed to be tested, which lead to testing 1 less of each of the reputation parameters.

6.3 TEST RESULTS

This section tries to give the reader a representation of the data gathered in this project. These graphs were generated in Microsoft Excel using the `.csv` files handed in with this report. In Section 6.3.1, generalized results are used to determine the impact that simulation and reputation parameters have on the performance of the network. The graphs shown are based on the results of all run test case scenarios. Next, in Section 6.3.2, the report dives into the influence of erroneous actors, showing both where the prototype succeeds as well as where it fails.

6.3.1 *Parameters*

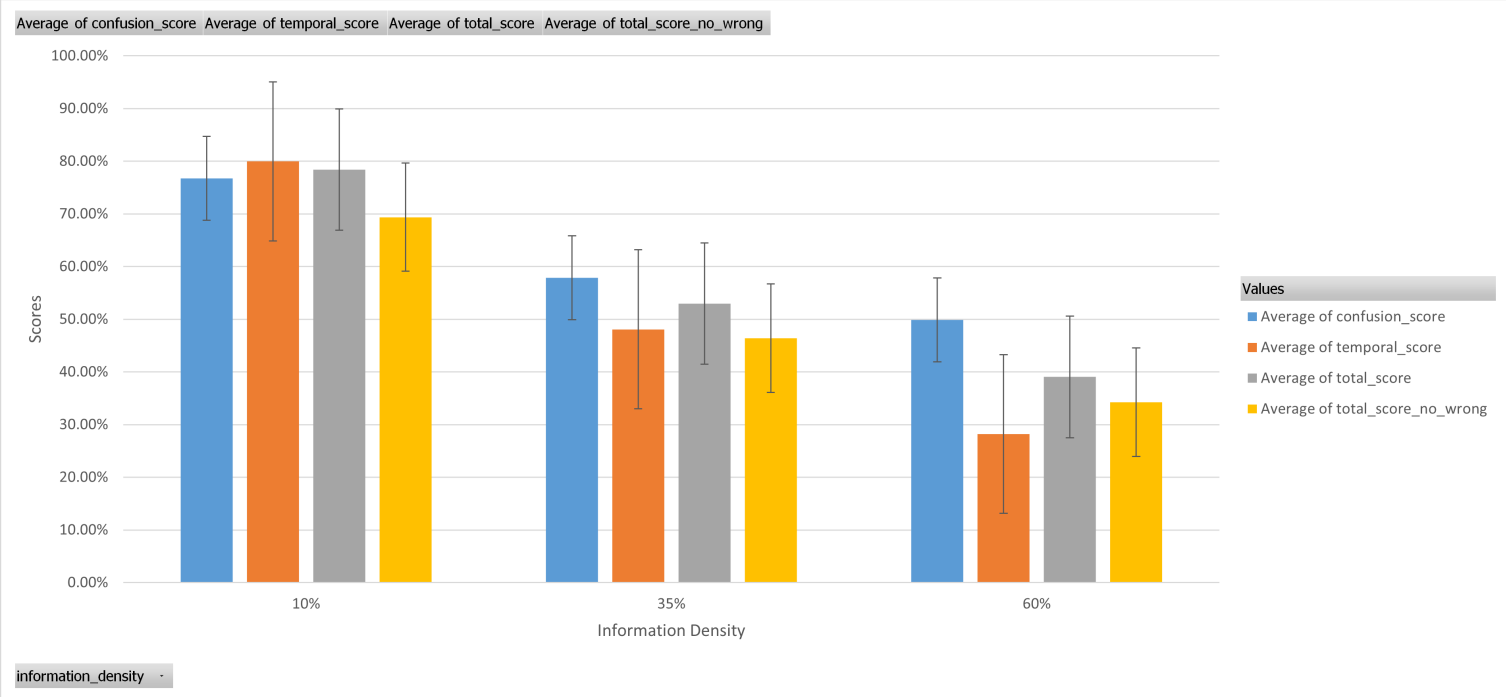


Figure 2: Average scores across all test case scenarios for how information density affects the performance of the prototype. Tested values 10%, 35% and 60% correspond to 4 versions, 14 versions and 24 versions spread out on the 40 timeslots the scenarios run.

First is the problem of information density (Figure 2). As seen on the graph, the information density is preferable as low as possible, meaning that choosing a short slot length is very important. This will come at a cost of network bandwidth and needs optimization in the

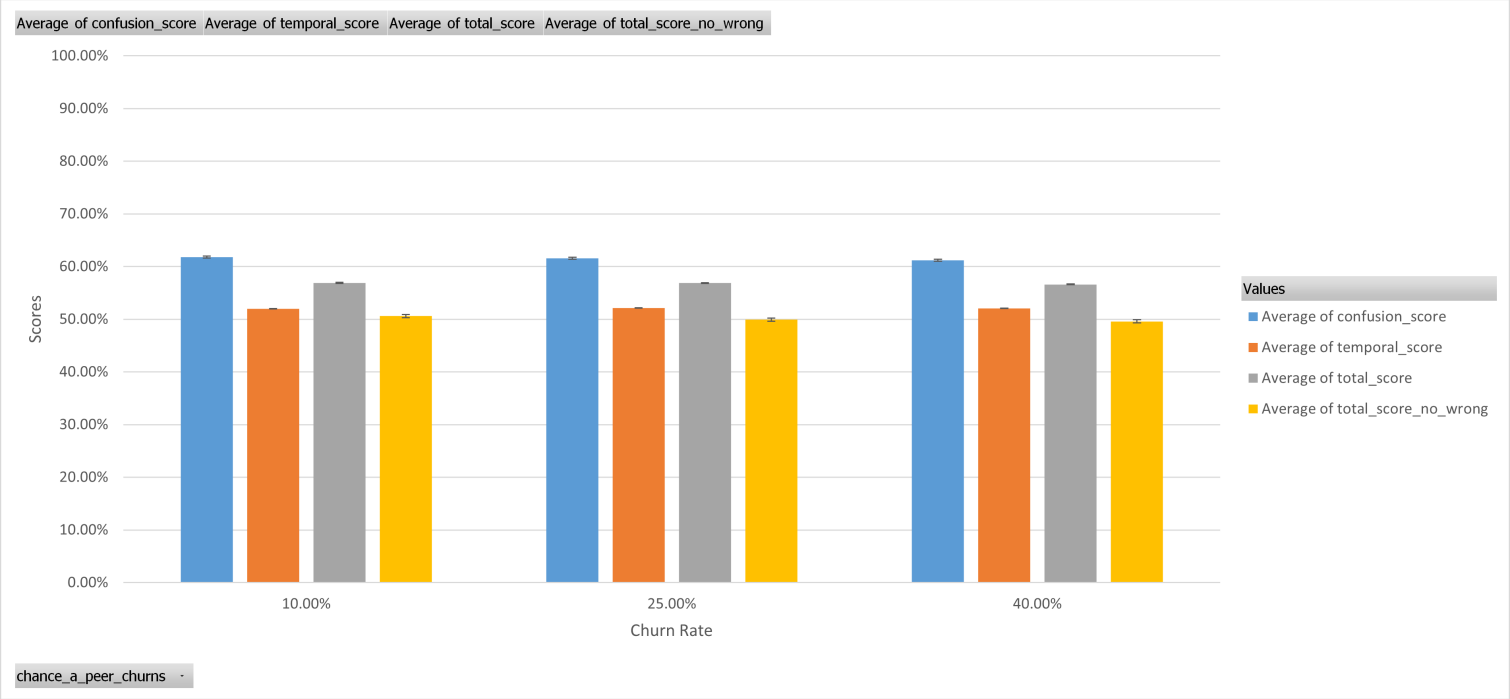


Figure 3: Average scores across all test case scenarios for how churn rate affects the performance of the prototype. Do note that peers only check if they should churn once they have crawled the entire URL space in their current iteration. See Section 5.4.2 for more information.

future as even if a website rarely changes, the slot length still needs to be great enough to support the fastest updating website, and if good results are wanted, at 10% information density.

The churn rate (Figure 3) seems to have no major effect on the system, however, in specific cases, a churn rate of 25% is beneficial for the system with erroneous behaviours that target a single version over the whole simulation timespan instead of multiple ones (such as consistently malicious, specific site, etc). This is likely because the median of the three tested churn rates strikes a good balance between having a high chance that the initial submitter of an incorrect website churns, which makes the malicious peers lose a lot of their total reputation, while still allowing pure peers to exists in the network long enough to generate meaningful information.

In regards to minimum confidence (Figure 4) the results are a bit more difficult to decipher. The important thing to note in the graph here is how total score and total score no wrong converges as minimum confidence increases, meaning less and less incorrect data enters the system overall, though at a significant cost to the total score. Balancing this parameter can be hard, as in the pure simulation, the lower the minimum confidence the better, but at higher concentrations of erroneous peers, higher minimum confidence is required to

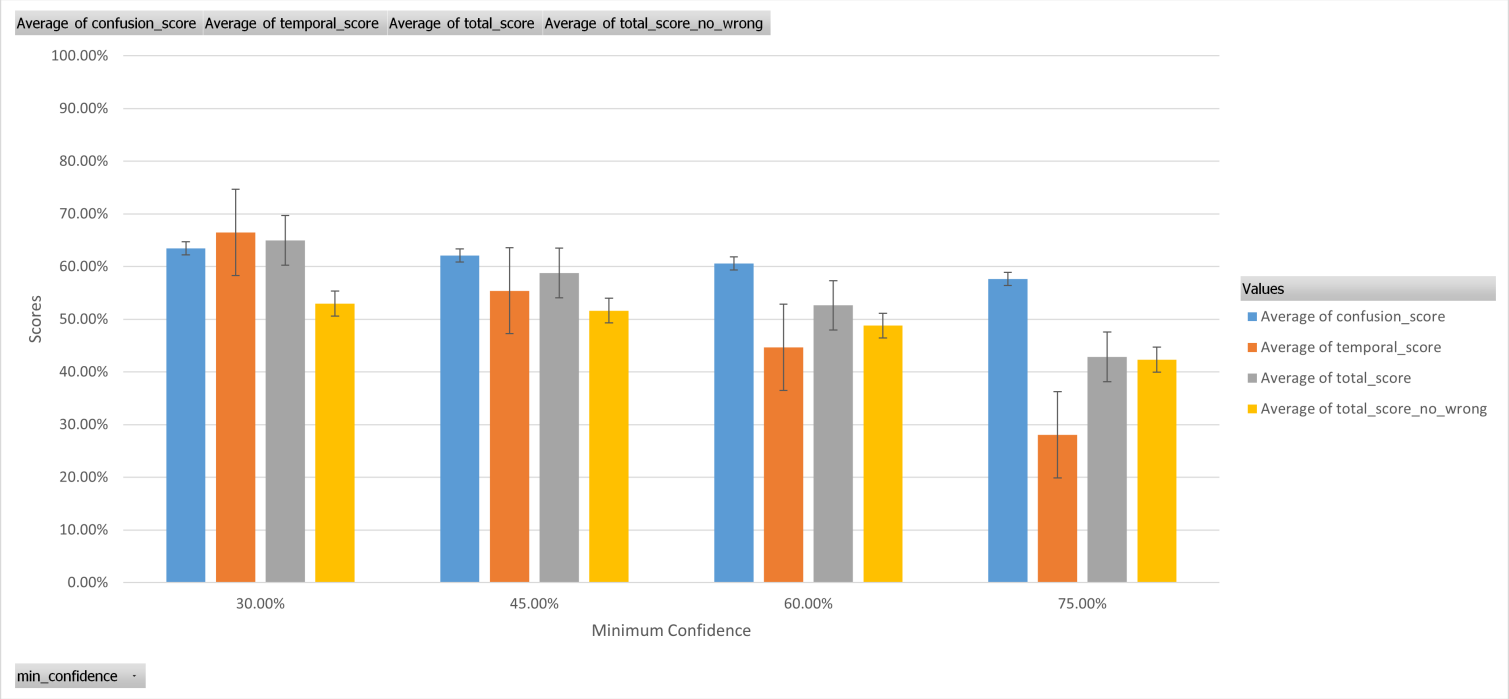


Figure 4: Average scores across all test case scenarios for how minimum confidence affects the performance of the prototype.

get good results. One needs to decide on a tolerance for erroneous behaviour in the network.

For the logistic function (Figure 5), it is generally a good idea to have a high k and low x_0. This means a curve that has a sharp fall-off and considers only documents a few timeslots away. Again it is important to note the difference between total score and total score no wrong in this graph. Total score no wrong peaks at 1.5, meaning there is some merit to looking back more than a single timeslot, however, if too many are considered the network has a hard time deciding on a specific version, meaning the temporal score is hurt a lot.

6.3.2 *Behaviors*

When it comes to the different types of erroneous behaviours, there's a clear difference in effectiveness between them. For instance, most maliciously behaved peers working alone (Figure 7) can have little effect on the system, even when they outnumber the total amount of peers. Working together however can have a drastic effect (Figure 8), and hit the system very hard. In general, the more percentage of the network is working together, the more effective just plain consistently posting erroneous information. At lower levels of maliciousness in the network, like at 3 peers, site-specific peers posting a malicious version of just 50% of the URL space proves a better approach.

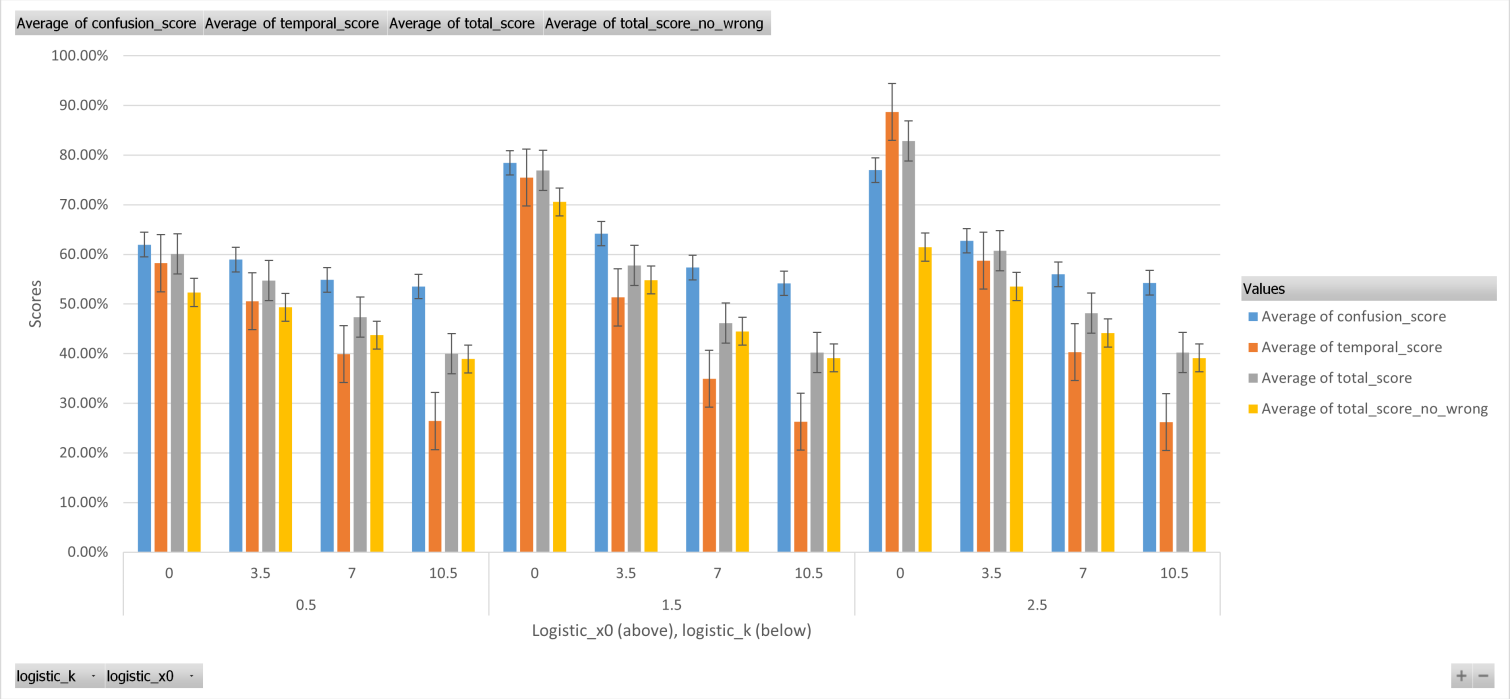


Figure 5: Average scores across all test case scenarios for how the temporal parameters logistic_k and logistic_x_0 affect the performance of the prototype. logistic_k determines the slope of the curve, and logistic_x_0 determines the number of surrounding timeslots to consider (do note this is not a one-to-one relation, $x_0 = 4$ does not mean 4 surrounding timeslots are looked at.)

Some simulations end up with better total score no wrongs because the malicious strategy ends up doing more correct work than incorrect (Figure 9). However, this is generally only true with solo peers, such as the solo sometimes malicious peer. As for the past focused versions, they successfully target the temporal score as can be seen in the graph, reducing it by upwards of 10%. They also indirectly affect the confusion score as some passing versions of URLs that arent live for very long can fail to get enough support in the timespace they are live.

One thing to consider with these results is what a realistic amount of malicious peers is. The tests show very low scores on grouped peers, especially at 12 malicious peers in the system, however, at this point the malicious peers outnumber the pure peers. It would be more strange not to trust the malicious actors when they consist of the majority of the network, as that is what the design of the peer-to-peer reputation network is based on. Furthermore, it is hard to distinguish between erroneous behaviour that the peer is not intentionally doing, and malicious behaviour a peer is intentionally doing to sabotage the network. Many of these behaviours could have plausible causes. Grouped site-specific peers could potentially be hitting a regional variant of a certain percentage of the site their visiting. Past-

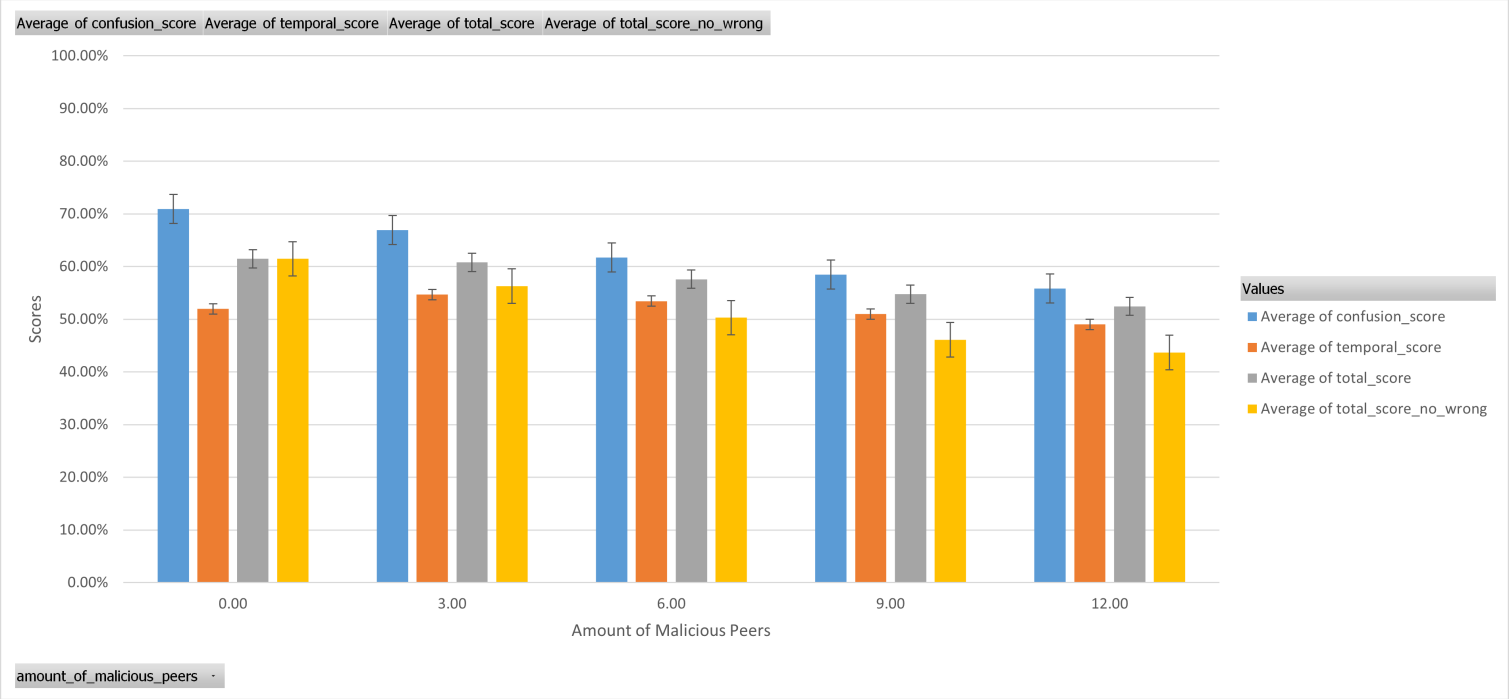


Figure 6: Graph showing how the number of malicious peers in general affects the scores. Useful when comparing with Figure 8

focused peers could be hitting non-updated cached versions from content delivery networks or simply be experiencing slow internet traffic. Grouped sometimes malicious could potentially have transmission issues that lead to incorrect data some of the time. Because of this, it becomes a cat-and-mouse game of deciding via parameters how much erroneous behaviour a peer is allowed to do before the work they submit carries a too great risk of being incorrect, no matter the reason.

6.4 SUMMARY AND DISCUSSION

Having tested the network in different simulations, it is now time to zoom out and look at the whole picture. The prototype had the intent of having high retention of information with a population of malicious actors in a peer-to-peer network that is initially stake-less. So does the prototype accomplish this to a satisfactory degree?

The conclusion is that it somewhat achieves the goals. The key problem of identifying malicious behaviour is that sometimes pure peers might not act intentionally malicious, but instead just erroneously because of other circumstances. Grouped specific site malicious peers might in this instance be deemed malicious, but it could be explained by a server serving different websites to different regions, and thusly no one is malicious per se. On the other hand, the line has to be drawn somewhere in the network, as the risk of accepting malicious

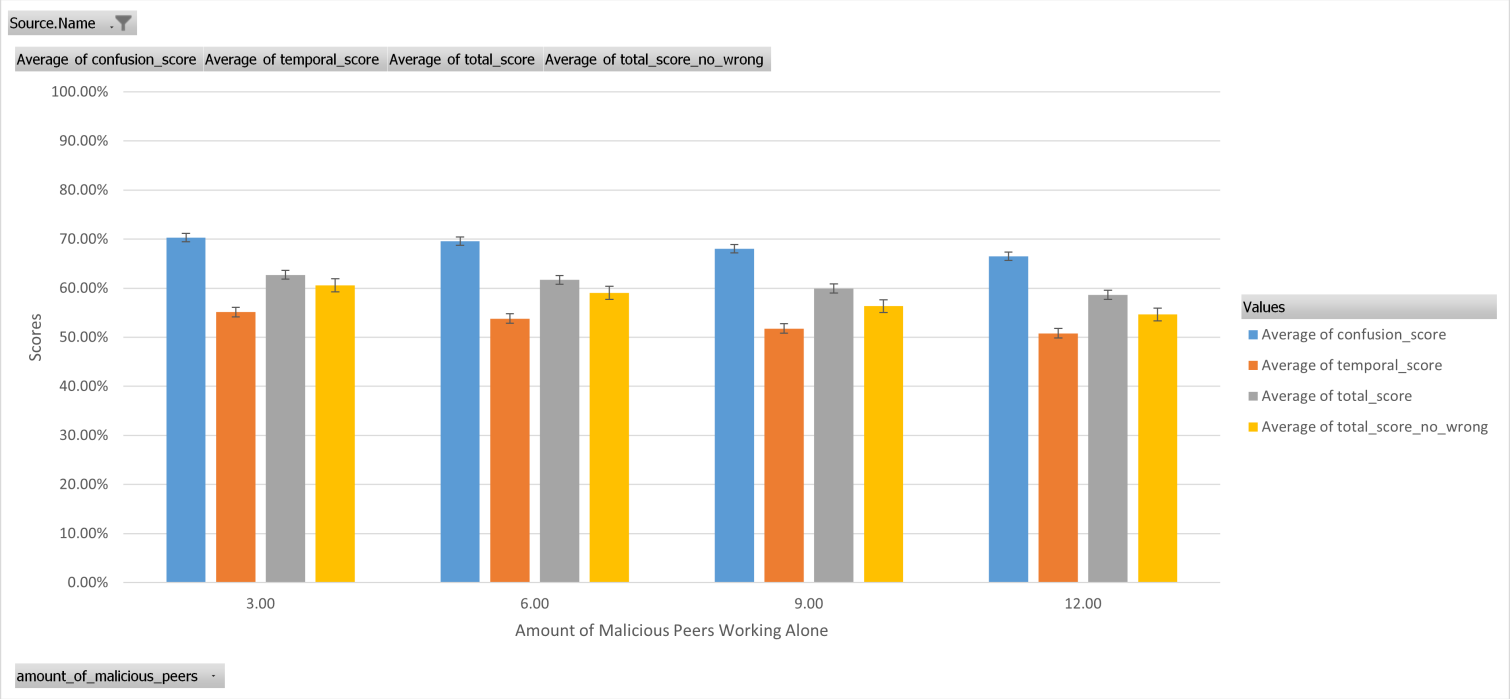


Figure 7: Graph showing the average scores across archivist peer behaviours which work alone. Since reputation is built upon the interaction with others, many incorrect behaviours get "filtered out" so the prototype does well in these situations.

information is a greater threat to a good archive than missing some of the total information.

When looking at average results such as those compiled in Table 2, it looks positive in many cases. The total no wrong score is the most important metric in this table. For even the worst case, this is still 48.4%, meaning a considerable amount of websites are retained, and these are placed in the correct respective timeslots. This is a clear win. However, this win is lessened by the fact that for most malicious actor groups, there is usually a noticeable difference between the total score and the total score no wrong. This means that given that the prototype does not know the correct results beforehand, it does not succeed in many cases to filter out every single malicious site.

This is a problem both in the calculations of the network, but as previously described this is also a central problem in a peer-to-peer archive. Some of these behaviours could come from pure peers and there is no easy way to distinguish between archivists who are being intentionally malicious and archivists who are mistakenly erroneous. In the end, it comes down to the choice of minimum confidence. Having minimum confidence of 100% would for instance require mostly the entirety of the network to agree on a version, but this also means 1 maliciously acting peer, or information density higher than the slot length, would make the network practically useless.

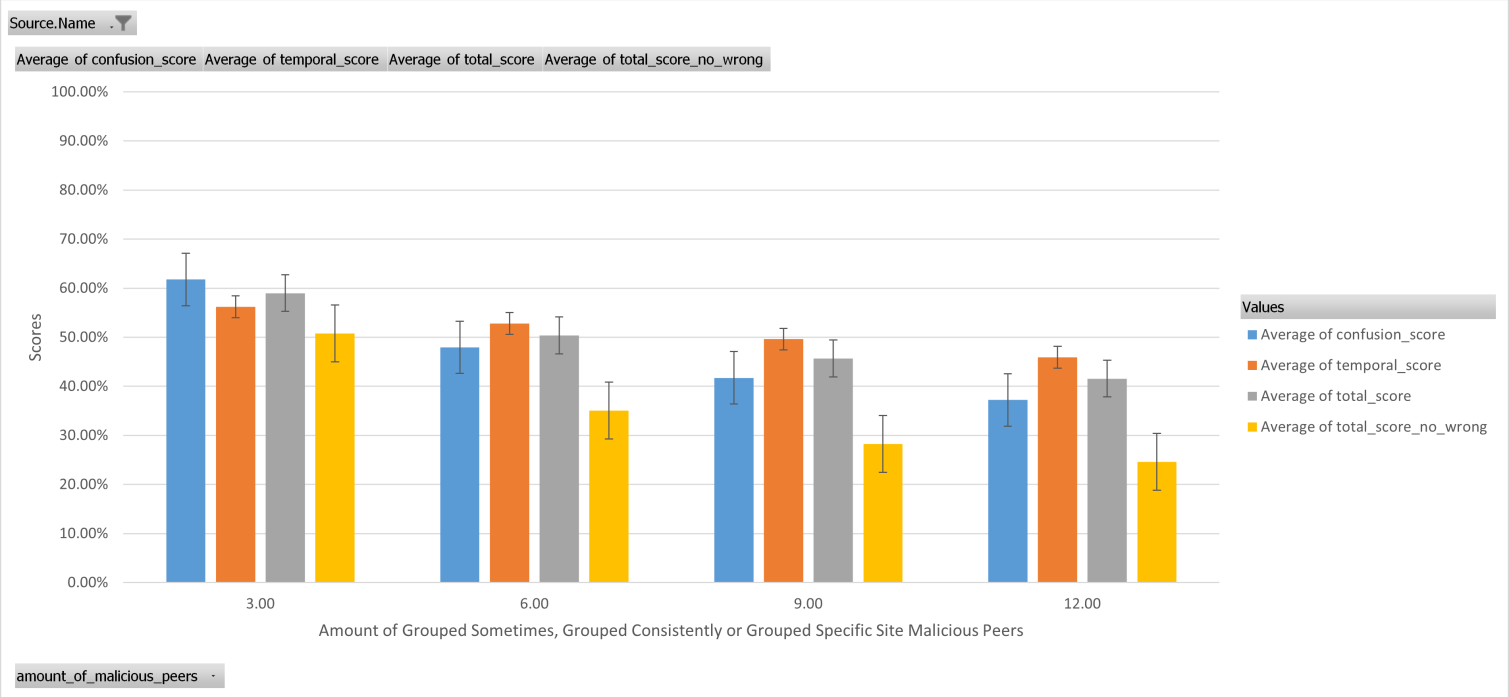


Figure 8: Graph showing average scores when only considering grouped peers that use either the consistently malicious, site-specific malicious or sometimes malicious strategy. Notice the much more severe cutoff than in Figure 6 and Figure 7.

Therefore, what this network has succeeded in doing is not creating a perfectly flawless archive, but a peer-to-peer network where archivists can retain a lot of probably correct meta-information. As this is not ideal for the goal stated in both Chapter 1 and Chapter 3, the purpose of a distributed peer-to-peer archive created upon this design as a foundation needs to make choices to remedy the inherent faults of this network. Examples could be those such as possibly having users themself decide who to trust by who they know outside the network in a kind of way like PGP-key trust work, creating a very shallow network with a very high minimum confidence, or using the network with the knowledge there exists potentially malicious information in the archive.

[%]	Confusion	Temporal	Total	Total no wrong
Pure	73.9	64.0	68.9	68.9
High Chaos	71.8	59.1	65.4	65.4
Consistently Mal. Solo	74.3	63.9	69.1	69.1
Consistently Mal. Group	56.0	60.7	58.4	50.6
Sometimes Mal. Solo	74.8	65.6	69.7	69.7
Sometimes Mal. Group	72.7	63.8	68.3	66.5
Specific Site Mal. Solo	65.6	64.6	65.1	57.5
Specific Site Mal. Group	58.2	60.9	59.5	48.4
Past Focused One. Group	73.8	61.8	67.8	67.8
Past Focused Last. Group	73.3	61.9	67.6	67.6
New Version Spammer	74.7	63.9	69.3	69.3

Table 2: Average scores across different simulations. For all the simulations 45% minimum confidence was used and $x_0 = 3.5$. For the individual tests, a total of 3 malicious peers meaning about 23% of the total peers should be malicious was used. For sometimes malicious and site-specific malicious, $k = 2.5$ and for everything else $k = 1.5$. In the chaos simulation, for site-specific peers, they post bad information on 50% of websites they visit, and the chance of a malicious peer acting maliciously is 5%, whereas these vary in the individual peer tests.

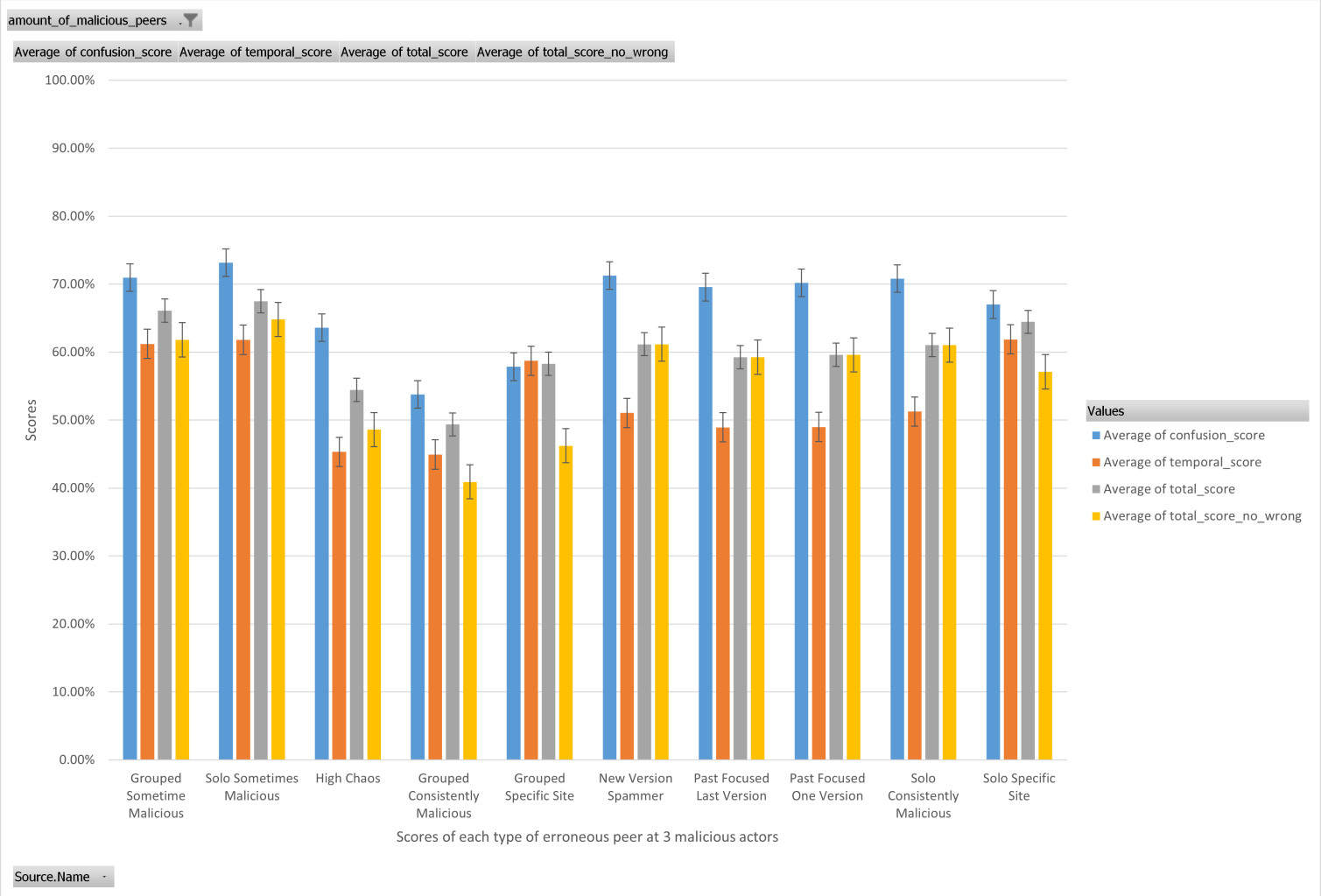


Figure 9: Graph showing the average scores across all incorrect behaviours at a population of 3 incorrect actors out of 13 total archivist peers. The performance of each individual actor is averaged over their specific parameters, which means some parameters may make actors perform better than shown, such as site-specific malicious actors.

7

CONCLUSION

From the evaluation, it is clear that stake-less distributed peer-to-peer networks do not exist yet not because no one has had the idea, but because of key problems in distinguishing bad archivist peers from good archivist peers from the network. Even though this is a problem, it is possible to sort through a lot of bad information and retain a lot of information that might be infected with small amounts of malicious meta information.

The problem is induced mainly because of the issue of choosing multiple parameters such as the resolution of the archive timeline (slot length) and the minimum confidence interval. Depending on the behaviour of different websites and archivist peers, it is by definition very difficult to decide if a peer is being malicious on purpose. These parameters then become a question of how much leeway we give peers before their reputation suffers. It is hard to make any theoretical guarantees, and the choice ends up being between an archive you trust a lot with a very little amount of information, or an archive you cannot fully trust but with a potentially very high resolution of its timeline.

The design and analysis chapters thusly need to be rethought with this in mind which leads to the question of what role a distributed peer-to-peer archival has in today's world. Having dismissed the hypothesis that malicious information can be completely sorted out, this report suggests two main ways to use the insights gathered from this conclusion to the experiment.

The first is to accept the limitations that the archive cannot be fully trusted. Instead of having the archive used for potential life-or-death situations or very important research, the archive is used for more recreational purposes, and only when the network has insanely high confidence in a site should it be considered to be valid. This could lead to the network being used to corroborate information for 'real' archivist institutions, while still allowing people with different tolerance for erroneous information to browse potentially lost websites.

Alternatively, archivist peers could choose parties to stake themselves, focusing on creating a larger difference in trust between those you know are good, and those you have no predisposed information about. One could for instance choose to stake universities and archivist institutions that participate in the network. If website authors decide to participate themselves to get high-definition archives of their websites, they could be staked high when it comes to their own site. One could also focus on communities like in royeng's paper

[9], trusting friends and family and possibly even having the like of PGP-signing conventions.

All in all, while this report has not succeeded in creating a mechanism that replaces the high standards of professional archivists, it has opened up to the idea that peer-to-peer systems could benefit and help create a richer archived version of the web for everyone to enjoy.

Part I

APPENDIX

BIBLIOGRAPHY

[1] Sawood Alam, Mat Kelly, and Michael L Nelson. "Interplanetary Wayback: The permanent web archive." In: *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries*. 2016, pp. 273–274.

[2] Sawood Alam, Michele C Weigle, Michael L Nelson, Martin Klein, and Herbert Van de Sompel. "Supporting Web Archiving via Web Packaging." In: *arXiv preprint arXiv:1906.07104* (2019).

[3] Kristian Borup Antonsen. "Peer Cache: Secure peer-to-peer web caching using private set intersection." MA thesis. Aarhus University, 2016.

[4] Timo Burkard. "Herodotus: A peer-to-peer web archival system." PhD thesis. Massachusetts Institute of Technology, 2002.

[5] Davide Frey, Mathieu Goessens, and Anne-Marie Kermarrec. "Behave: Behavioral cache for web content." In: *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer. 2014, pp. 89–103.

[6] Minaxi Gupta, Paul Judge, and Mostafa Ammar. "A reputation system for peer-to-peer networks." In: *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video - NOSSDAV '03*. ACM Press, 2003. DOI: 10.1145/776322.776346. URL: https://doi.org/10.1145/776322.776346.

[7] Imre Lendák, Balázs Indig, and Gábor Palkó. "WARChain: blockchain-based validation of web archives." In: *International Workshop on Socio-Technical Aspects in Security and Trust*. Springer. 2020, pp. 121–134.

[8] Petar Maymounkov and David Mazieres. "Kademlia: A peer-to-peer information system based on the xor metric." In: *International Workshop on Peer-to-Peer Systems*. Springer. 2002, pp. 53–65.

[9] Tonje Røyeng. "A collaborative internet archive for personal and social use." MA thesis. 2020.

[10] Martín Vigil, Daniel Cabarcas, Johannes Buchmann, and Jingwei Huang. "Assessing trust in the long-term protection of documents." In: July 2013, pp. 000185–000191. ISBN: 978-1-4799-3755-4. DOI: 10.1109/ISCC.2013.6754943.

[11] Martín Vigil, Denise Demirel, Sheikh Habib, Sascha Hauke, Johannes Buchmann, and Max Mühlhäuser. "LoT: a Reputation-based Trust System for Long-term Archiving." In: July 2016.

[12] Liang Zhang, Fangfei Zhou, Alan Mislove, and Ravi Sundaram. "Maygh: Building a CDN from client web browsers." In: *Proceedings of the 8th ACM European Conference on Computer Systems*. 2013, pp. 281–294.

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 December 8, 2022 (`classicthesis`).