Decentralized Internet Archive using the Cothority framework

Nicolas Bernard Lucien Plancherel
nicolas.plancherel@epfl.ch

School of Computer and Communication Sciences
Decentralized and Distributed Systems lab

Master Thesis

January 2018

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Eleftherios Kokoris
Kogias
EPFL / DEDIS

Supervisor
Kirill Nikitin
EPFL / DEDIS

Abstract

Since the 1990s, Internet takes more and more importance in the everyday life. People use it to store any data from holidays pictures to government's communications. They also use it as a reliable source of informations and knowledge. Nevertheless, we observe a centralization of the providers of services and a fragility of the long-term sustainability of the data. Because of that, the latter are very vulnerable to censorship either by deletion, modification, or access denial.

In this report, we present the definition and the implementation of a consensus-based system to create a decentralized internet archive. The particularity of this protocol is that the servers involved in it will first agree on the *biggest common subset* of the website using a tree comparison based approach. Then they will only store that common subset on a blockchain. The implementation is done inside the Cothority[1] framework.

Once a website is archived, our solution makes it impossible for a single entity to remove or modify its content. The consensus algorithm is considered efficient since it grows polynomially with the size of the website and linearly with the number of machine involved in the consensus. Furthermore, our simulations shows that it can save a news web page in under 40 seconds. Finally, from a security point of view, if the leader of the consensus protocol and the majority of the machines involved in the archiving protocol are honest, the latter will store a honest, immutable snapshot of the web page that can later be securely retrieved by any user of the system.

Keywords

web, hash, decentralized, distributed, consensus, tree

Contents

1 Introduction

By studying today's internet architecture and usage, we can see a heavy centralization of data and source of services. That effect can be highlighted by observing the lack of diversity of tools used to access the internet and its content[1]. Those tools are good when you see them as services that are free of charge and that allow every user to have access to a high-quality and reliable infrastructure. Nevertheless, despite having some positive points, this state of affairs have multiple drawbacks. Centralized systems are very vulnerable to censorship that could come from public or private entity. They also often have an identifiable single point of failure. And finally it poses some privacy concerns mainly coming from the fact that the user does not has any power when it comes to privacy except by avoiding using some services that could be hard or impossible to realize in today's world. Before exposing briefly the solution developed during this work, we will first expose a bit more in details the three problems identified: the censorship problem, the single point of failure problem and the privacy one.

The first and most concerning drawback is the censorship vulnerability of centralized systems. The latter are under one legal jurisdiction and owned by a single entity. Furthermore, the content of a website is owned by a single legal entity. All these factors allow governments to easily put pressure on people in order to remove parts or the whole content those people provides. The fact that a limited number of person can decide of what is available or not on the Internet is a threat to the users and goes against one of the fundamental principle of it which is facilitating the exchange of ideas whatever they are. Furthermore, the manipulation of information coming from governments or private entities are not future possible scenarios but happens nowadays. For example, the Chinese government censors some informations available on the internet and modifies Google Search result in order to make them fit its standard. We can also cite the *fake news* problem that became a critical subject during the US presidential election of November 2016. Social network like Facebook had a relatively great influence about the opinion of Americans about the candidates thanks to their ability to highlight some advantageous or disadvantageous information about the latter. Another example, more harmless, happened in Switzerland were Ignazio Cassis, a member of the executive power, deleted tweets he made before his election. Those can still be viewable thanks to external entities that regularly make backups of some Twitter accounts. That last example, even if it is of smaller importance that the other two, shows explicitly the importance of a distributed archiving system. A system that does not depends from one single entity especially when it comes to addition, modification or deletion

[1] We can see this lack of diversity through the analysis of the following different market shares. The market share of Google's Chrome from December 2016 to November 2017 is 58%[2] and the market share of Google's search engine is 74%[3].

of informations archived on it. Finally we can say that the censorship point of view, the two other vulnerabilities of centralized system, which are the single point of failure and the privacy concerns, can be seen as tools in order to obtain the desired result. But some particularities of both are worth being exposed.

The second drawback is that if you rely on a single entity, there is a single point of failure in both the technical and legal system making it vulnerable to technical failure and attack but also to legal action against it. One quite recent and famous example of technical attack is the mirai bot ddos attack[4] against Dyn that took down Twitter, Reddit, The Guardian and multiple other websites. The legal actions can lead to the de-indexing of a website from the Google Search Engine result and/or the deletion of that page from its cache. Those actions can come from judges applying, for example, the *right to be forgotten*[5] which can be considered a good trade-off between freedom of information and protection of individuals but they can also come from governments or technically powerful organizations in order to censor certain ideas or people. Centralization makes that task relatively easy. For example, an organization shared their traffic data when they removed themselves from the Google index[6]. In short, about 60% of their traffic they thought were direct traffic (meaning not obviously related to search engine) was in fact dependent of Google index. One can easily infers the impact of a removal from Google Index on the visibility of a website, even if the host server is still serving incoming request without problem. The last aspect of the *single point of failure* drawback is the fragility of the availability of the data on the long-term. Indeed, one of the principle of the initial architecture of internet is that a single machine holds an information and makes it accessible to any machine connected to the network. This was an obvious choice at the time mainly because the number of computer and their power were extremely low compared to today's machines. But this architecture makes the deletion or tampering of data easy since you only have to take control of a single computer. The analogy in the legal world is straightforward; you only have to make pressure on a single entity. For example, some entity could put pressure on Google and thus could "take control" of any of its services. Nevertheless, the limit of this analogy is that not only it is very difficult for any entity to lobby Google in a global way. But for governments on its own territory and own juridiction it is easy to do that. The drawback of centralization we exposed here cannot be imputed to the actors of the internet (Google being heavily cited as it is one of the, if not *the*, biggest provider(s) of internet services nowadays). Those actors play at worse a passive role at best an indirectly active role against that drawback.

The third drawback of the centralization of internet mainly originates directly from the providers of services. It is the privacy concerns, the centralization, analysis, storage and use of data of internet users. With the de-

mocratization of internet, users observes the appearances of advertisements that were first generic but soon becames content-based, location-based, and finally now, thoses advertisements are targeted at a single individual user. For some, this is as a good thing or at least a neutral one: it allows people to have access to good services for free and advertisements become more relevant for the user. The data can also be collected and used by the governments, where the main argument to justify this is to increase the security of the country. However, the collection of data by private or public entities, especially the personally identifiable ones, should be avoided if you have the interests of the user in mind. Firstly, even if we assume that the entity collecting the data is honest, it is impossible for it to guarantee that their data will never be stolen by malicious third party that can then use the data for blackmailing, stealing banking information, impersonate or defame the users. Secondly, if we assume that the entity acts as a rational agent maximizing its utility function without any ethics considerations, the function being the profit for private actors and control over information for public actors, then some more threats appear for the users. For example, the data brokers sell personal informations such as medical records to other private third parties[7]. In a worst case scenario, we could imagine these data being used to compute your insurance bill or to discard you from certain work position. In the case of a public entity, the collection of data often goes together with mass surveillance which leads to a tighter control over what citizens say and legal action against those that do not respect the laws. Even if it can be seen by a few as an effective security measure, it leads to the arrest of hundreds of political opponents in some countries.It is also a threat to freedom of speech and to democracy since one of the fundamental democratic principle should be the guarantee that a debate of ideas is always possible. Finally, keeping any assumption on the entities, we observe that with today technology, which is capable of identifying us as a unique "token", it is possible for these actors to filter what we see as individual. They already do it with targeted ads, adapting the website to the user-agent of the browser or simply with accreditation systems. The most well know example of this filtering is Google filtering out explicit content by default. But between protecting children from inappropriate content to prevent adults to inform themselves to defend an ideology or a system, there is one step that some governments already did.

We can see that in the two drawbacks, a potential user must trust the central entity. And from a user point of view, the trust is the key point for choosing centralized or decentralized system and which one to choose. He must trust the central entity to have robust and reliable architecture and to not tamper data when they send it to him. Furthermore, he must trust the actor not to collect data implicitly and to handle data with respect of his privacy. In most distributed systems on the other hand, the user is

not required to trust any entities participating but only that a significant number of them do not collude where *significant number* depends on the system. Until today, it was difficult for distributed system to store verifiable data without using a central system which stop them from being a real alternative to centralization. But with the development of blockchains, such systems become a viable alternative. The Cothority framework developed at DEDIS is an effort to create a common base for these systems.

Our contribution to the Cothority framework is the development and the implementation of a decentralized internet archive. The aims of this work, in addition to preventing the user from having to trust a single entity, are (a) to be able to store website where individual based-content are removed, which should result in a gain of space, (b) to create a censorship resistant archive by requesting that the content of the website is verified by all the machine participating to the protocol called *conodes*[8] and (c) to give a user a way of requesting a website without revealing informations to the host of the website. Some tools already exist and handle one or two of these aims like ZeroNet[9] that proposed a decentralized web, *archive.org* providing a centralized web archive structure or all the proxy systems that allows to "hide" a machine behind another. Theses already-existing solutions are presented and compared to the solution we provided in section 2.

2 Background

In this section, we realize a quick overview of the Cothority framework and define some vocabulary related to it. We also present different solutions currently used on the internet to either archive webpages or create a distributed web and briefly compare them with our solution.

2.1 Cothority

The *Cothority Framework*[8] regroup a list of different services. The inital goal of the project was to provide a distributed, decentralized certificate authority[1]. To do so, you need an efficient way to communicate between distant machine, a efficient signature scheme and a way to store and to access the signature in a decentralized way. Those missions were respectively fullfilled by Onet[10], the CoSi service[11] and the skipchain service[12]. In this section we will first define some vocabularies that will be used throughout all this document especially in Section 3.

2.1.1 Vocabulary Definition

We define the following concepts:

- A conode is a machine that takes part in a distributed protocol using the onet library communication protocols.

- A Roster is multiple conodes participating in the same protocol or service.
- A collective signature is a cryptographic signature that is both created and validated by a Roster. The collective signature is generated using the CoSi protocols.
- A skipchain is a blockchain that is run by a Cothority skipchain Roster that allows us to interact with it through the Skipchain API.

2.1.2 Onet, CoSi and Skipchains

The Onet library and service [10] is a network library that offers a tree-based communication protocol and facilitate the creation and simulation of distributed protocol.

The skipchain API provides an implementation of a blockchain designed in the Chainiac paper[13].

The CoSi API is an implementation of the CoSi signature scheme allowing independent servers to create an aggregated Schnorr signature. The protocol is described in the paper *Keeping Authorities Honest or Bust with Decentralized Witness Cosigning*[1]

2.2 ZeroNet

The ZeroNet[9] is a software that create a decentralized censorship-resistant network by using the BitCoin cryptography and the bit torrent technology. It handles static and dynamic websites that user can consult and serve in a peer-to-peer manner. It provides very similar functionality and guarantees that the project developed here. Indeed, it allows to access to website in a distributed decentralized way, and allows to create and maintain any kind of websites. It resolves almost all the problems that we highlight in the beginning of our work. Nevertheless, since it is not an archive, it does not handle external website. The content it contains is and must be done explicitly for the ZeroNet network. Furthermore, the content of a ZeroNet website can be changed unilaterally by the owner of the associated private key. This makes the conservation of the content over a large period difficult since it can disappear by the will of a single person. Nevertheless, this is an extremely interesting project that is worth time time it takes to understand it. Our goal is also different. ZeroNet wants to create an internet that is parallel to the one we know today. Our goal is to create an archive of today's internet to ensure the access to the website content despite the centralized design we described in the introduction.

2.3 Archive.org - Waybackmachine

The wayback machine accessible on archive.org[14] is a centralized system that allows anyone to make a request to archive any website available on the internet. It creates snapshot of different websites and for some websites that were regularly saved, you can follow the evolution of their design and content over months or even years. It is a very good initiative that allows to increase the longevity of the contents. This solution provides all the functionality one could expect from an archive. The only drawback of this project is its centralized design that makes it vulnerable to the same threat as any other website on the internet. Our solution is try to create an archive as close as this one in terms of functionality but with the advantages of being distributed and decentralized.

3 Decentralized Internet Archive

This section describes the final system that has been implemented during this project. The presentation of the old system's implementation is available in the Appendix of this document.

The Decentralized Internet Archive system's flow can be describe as follow: A user provides a url to a group of independent parties. The latter request the web page pointed by the url and reach a consensus on a unified view. Once a consensus of the view is reached, they store it on an immutable ledger. Anyone can then securely retrieve the view from the ledger.

3.1 Definitions

In this section we recall some definitions of the ONet and Cothority framework. It not meant to be read sequentially but more as a quick helper for this section.

- A conode : it is a computer part of an ONet network. It runs at least one service or one protocol and most often allows an interaction with those through a command-line API.
- A Roster : it is a set of conodes that are fully connected to each other through the network.
- A protocol : it only has a state that lasts the time of one execution. It allows conodes to communicate with each other. It has an entry and an exit point and a well-defined number of steps.
- A service[8]: it is run by nodes. It can keep a persistent state and be restarted if the node crashes. The service can initiate or join protocols.
  - Can start protocols

- Communicate with other service-instances
- Communicate with the clients through an API
- Have state that is kept over reboot of the cothority binary

3.2 General Network Architecture

By general network architecture, we mean the high-level representation of the machines involved in the web archiving as well as how they communicate with each other. There are four main entities that need a small description:

- the conodes executing the save and retrieve protocol. (the blue square of Figure 1)[2]
- the conodes executing the creation, storing and retrieval of the block in the skipchain. (the green square of Figure 1)[3]
- the user initiating a save or a retrieve protocol.
- the web server on which we request the website we want to archive.

In the Figure 1, we can see the different entities. A bold coloured arrow between two entities means that they communicate as entity. For example, the user make a save request to a roster of conodes. Even if it can happen that it communicates effectively with only one machine, it is only made as *a machine being part of the entity* so the chosen machine could have been replaced by any other of the same entity. The black thin arrows represent a communication between two machines ignoring if the machine is part of a distributed system or not. We can see them as regular network communication. The light gray arrows are identical to the black arrows in terms of what they represents; the different color being here for readability reason. It is important to notice that the roster responsible for the skipchain (in orange) and the roster that communicate with the user (in blue) may be made of the same physical machine but they are two different entities in term of functionality.

It is important to note that the conodes executing the save and retrieve protocol (the blue square of Figure 1) and the conodes executing the creation, storing and retrieval of the block in the skipchain (the green square of Figure 1) can be the same machines. One Roster can also be a subset of another. In our Simulations in Section 4, for example, the two entities are composed of the same machine.

[2]That entity can be made of the same conodes as the skipchain handling entity.
[3]That entity can be made of the same conodes as the saving/retrieving entity.

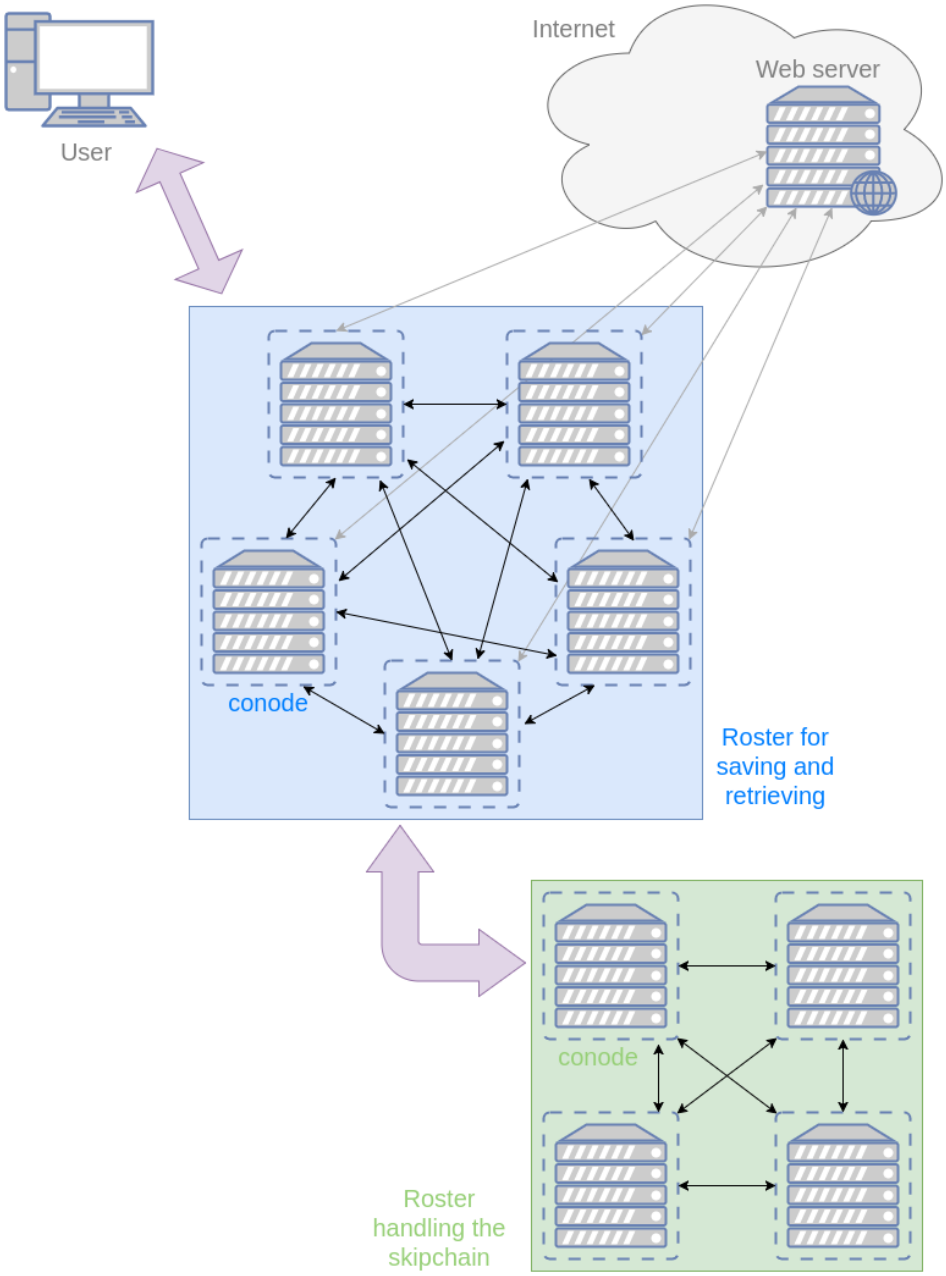

Figure 1: General Network Structure by entity

3.3 Theoretical Assumptions and Guarantees

This section lists the assumptions we had to make for the different protocols and the guarantee provided by it, in other words, its scope. Since those considerations are tightly bound to the protocols themselves, it may be

necessary to first read the sections 3.4, 3.5 and 3.6.

3.3.1 On the Consensus and Saving Protocol

The saving protocol runs on a tree of height equals to 1. So there are two roles for the conodes : leader or child respectively root of the tree and leaves of the tree. The leader only communicate directly with the user and is responsible to aggregate the answer of the other conodes. The children only have access to its own answer and send it (or not) to the leader. In the lists below, we will use *conode* when an assumption or guarantee holds for both leader and children. Otherwise will we use respectively *leader* or *children*.

The assumptions are the following:

1. The leader is considered honest when creating the reference tree.
2. At least one conode in the Roster fulfill the assumption 1.

Under the assumptions, the protocol guarantees the following:

- A conode cannot forge the signature of another conode.
- A conode cannot learn the plain text data seen by another node except if it has the plain text data itself.
- A consensus is either the best that is reachable or a subset of it. It never contains data that would not have been in the consensus reached by only fully-honest, functioning conodes.
- Once a consensus is reached and signed, it is not possible for any conodes to tamper the data if there is at least one honest conode.

3.3.2 On the Retrieval Protocol

The retrieval protocol run on a single conode of a roster and the own user machine. The user and the user machine are seen as a single entity in all the section 3.

The assumptions are the following:

- The consensus and saving protocol produced a valid and honest archive.
- The url and time stamp associated with the signed data is correct.
- The user has enough space on his file system.

The guarantees are the following:

- The data send are always accompany by a verifiable collective signature (Cothority CoSi tool).
- A valid signature means an interpretable data that is either an html document, an image or a css document.

3.3.3 On the Skipchain Handling Protocols

The skipchain handling protocols act as a proxy between the Cothority Skipchain service[12] and the Decenarch Services. All the assumptions and guarantees of the Cothority Skipchain service are implicitly taken as granted as soon as the request from the Decenarch Skipchain service is send and do not hold anymore as soon as a response is received by the Decenarch Skipchain service.

The assumptions are the following:

- At least one conode of the Roster will serve the user request.
- The serving conode is honest, it follows the protocol.
- The user knows the exact url of the page he requests. Aliases are not used.

The guarantees are the following:

- The user receives the data that are the closest to what he asks for.
- If the user receives no data, then it implies that either the page was not saved or there is no data old enough.

3.4 Consensus and Saving Protocol

Here, we describe the service and protocol executed by the conodes when the user requests the saving of a website. We will consider that the roster is composed by three nodes: one leader and two children. Furthermore, we will assume that the user has an SSH access to the leader and can execute API request directly on the conode. In practice, any reasonable[4] number of conodes is possible. Furthermore, it would be possible for a user to contact a machine without the help of an SSH machine. That last assumption was made in order to focus on the protocol itself.

3.4.1 High-Level Presentation

The consensus and saving protocol has 5 phases:

1. The Consensus Phase where the conodes retrieve the website and reach a consensus. It happens on the protocol level.
2. The Request Missing Data Phase where the leader asks the other nodes for plain text data that are in the consensus but that it does not have. It happens on the protocol level.

[4] See Section 4 for a more precise definition of reasonable in this context.

3. The Cosigning Phase where the conodes collectively sign the plain text data they agreed on. It happens in the service level only.

4. The Additional Data Phase where the conodes where the conodes redo the phases below for each associated resource of the web page[5]. It is launched automatically in service level.

5. The Skipchain Saving Phase where the leader sends the data that should be archived to the conodes responsible for the skipchain.

3.4.2 Protocol Details

The consensus and saving protocol has the following steps (see Figure 2, 3 and 4 for a graphical representation of it):

1. Through the API, the user launches a save request giving the url he wants to be archived.

2. The leader of the protocol receives the request and starts the service related to saving.

3. The leader locally starts the save protocol by (The Start and Consensus Phase are represented in Figure 2):

   (a) Initiating the Consensus Phase

   (b) Requesting the website to the web server.

   (c) Creating a MasterTree or a MasterHash that will be used as a reference for the other nodes. A MasterTree the tree that can be inferred from the html code but every data in every node is hashed. A MasterHash is a hash associated to an array of signature.

   (d) The leader creates a signature associated with the MasterTree as shown on Figure 6. The seen array is not send since it is implicitly an array whose length is the number of nodes in the MasterTree and whose value is always 1. In general, in the seen array, if $seen[i] \geq 1$ it means that the i^{th} node discovered using Breath-First-Search on the master tree was seen by the conode. If $seen[i] = 0$ it means the the conode did not see that anonymized html node in its own locally computed tree.

4. The leader send the url, the MasterTree and the MasterHash to all the conodes including himself.

[5]Those resources are images and css files.

5. Each conode, upon receiving the url, MasterTree and MasterHash on the Consensus Phase will prepare itself to answer to its parent (see next point), and if it is the leader will pass the message down to its children. This is the top-down part of the Consensus phase.

6. The bottom-up part of the Consensus Phases begin when the message reaches the leaves of the conode tree. Every conodes execute the same steps:

   - The conode requests the website from the internet web server and compute either the anonymized, hashed html tree or the hash of the unstructured data and it sign the output.
   - The conodes stores the plain text data associated with its hash locally.
   - The conode use the local anonimized result it computed previously and create a signature associated to the MasterTree or directly sign the MasterHash according to what it has. The signature scheme used for the hash is shown on Figure 7. The signature scheme used for the tree is shown on Figure 6.
   - The last step depends of its role:
     - If it is a child, it sends the MasterTree and MasterHash and its Seen Array and Signature associated with the MasterTreeto the leader.
     - If it is the leader, it either extracts the nodes signed by a sufficiently high number of conodes from the MasterTree or the hash signed by most conodes from the MasterHash. It could happen that no nodes and/or no hash are selected. The extraction of the most signed hash is done simply by verifying that all the signatures are valid and then by choosing the hash with the higher number of associated valid signature. The selection of the consensus tree is a bit more complex. First, it verifies that all the couple (seen array, signature) are valid. If some are not valid, it simply discards them. Then using only the valid seen array, it identifies the nodes that was seen by a sufficiently high number of conodes. FInally, it removes all the nodes that are not in its *sufficiently seen* list.

7. The leader initiates the Request Missing Data phase graphically represented in Figure 3. From the Consensus Phase, it has the consensus MasterTree or the consensus MasterHash, meaning that it has the hashed data that represent the biggest common subset of the resources they must agree on. It then does the following:

   (a) It checks if it does have all the plain text data related to the html nodes or to the hash chosen.

(b) It creates a list of missing data (that can be empty)

(c) It sends that list to its children.

8. Upon receiving the request, the children check if they have the plain text data. If they have it, they send it to the leader. Else they send an empty list.

9. The leader now have all the plain text data. It constructs the plain text html tree from the anonimized one or get the plain text data from the hash of the unstructured data.

10. The leader ends the protocol and send the plain text data to sign to its service level.

11. On the service level, the leader uses the CoSi API to make the conodes collectively sign the plain text data.

12. On the service level, the leader checks if any additional resources are needed. If yes, it redoes the current protocol up to here for each of the required additional data. Else, it ends the CoSiging Phase.

13. On the service level, the leader turns the data into a format that is usable by the conodes responsible to maintain the skipchain and send the data to them.

14. The leader ends the Saving Service.

In the Figures 2, 3 and 4, L0 is the leader conode and C1,C2 are the children conodes.

In the Figures 5 and 6 and the Figure 7 we detail the process done locally by each conode to transform respectively the plain text html code to an obfuscated tree and to transform the not hashed image or css file into a hash.

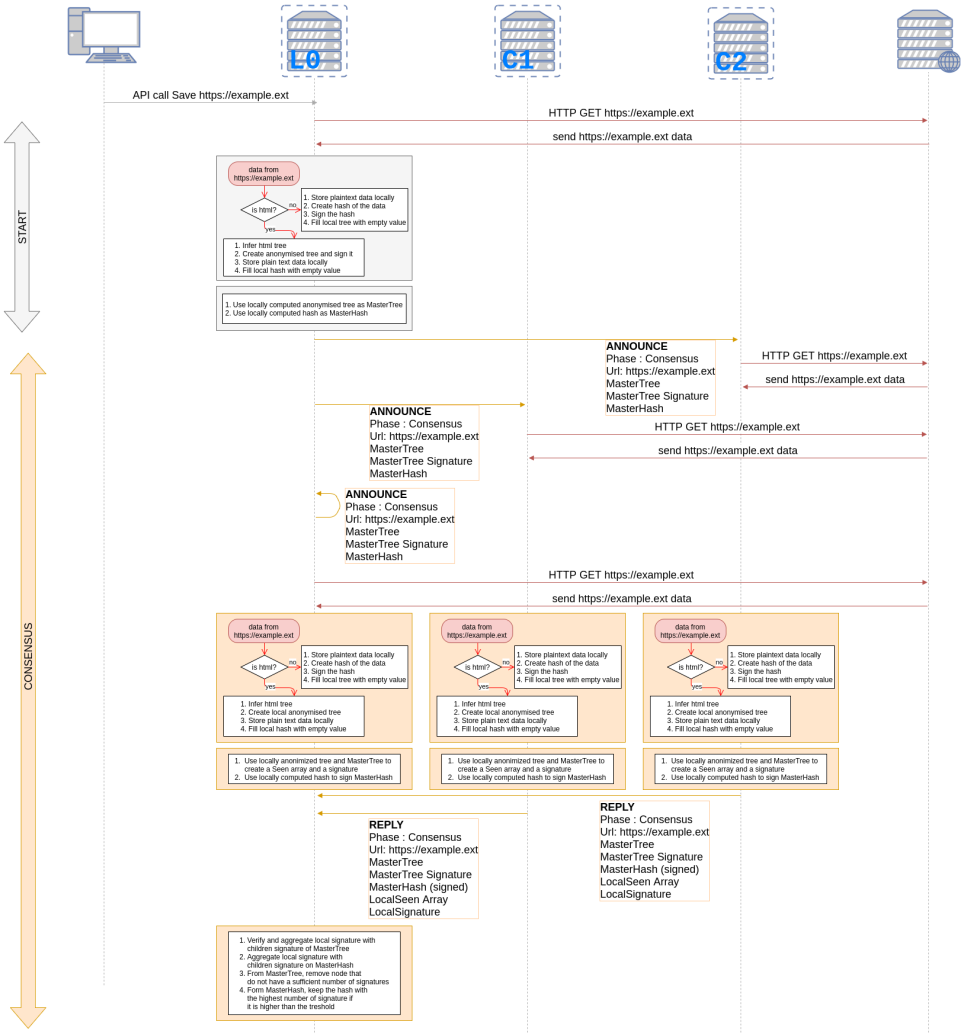

Figure 2: Start and Conensus Phase

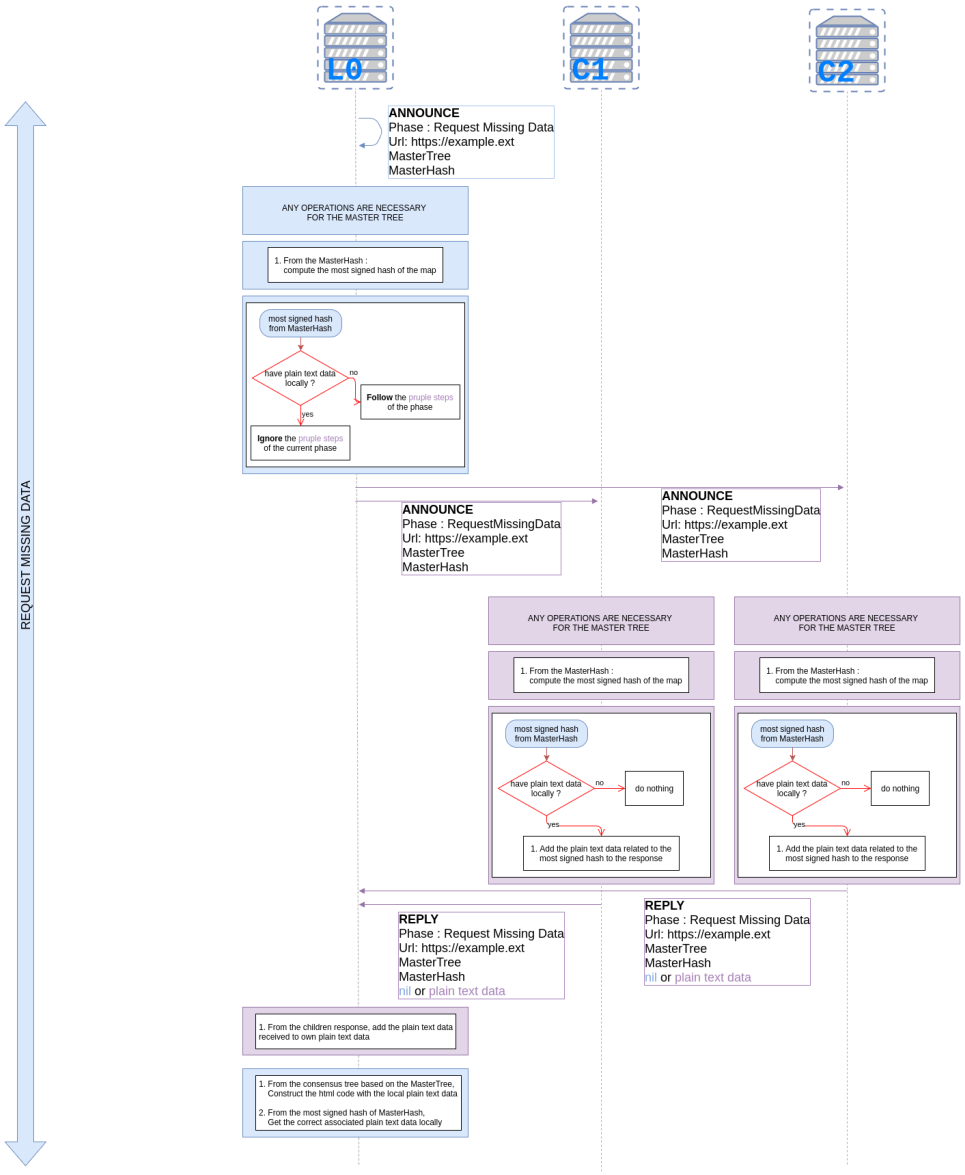

Figure 3: Request Missing Data Phase

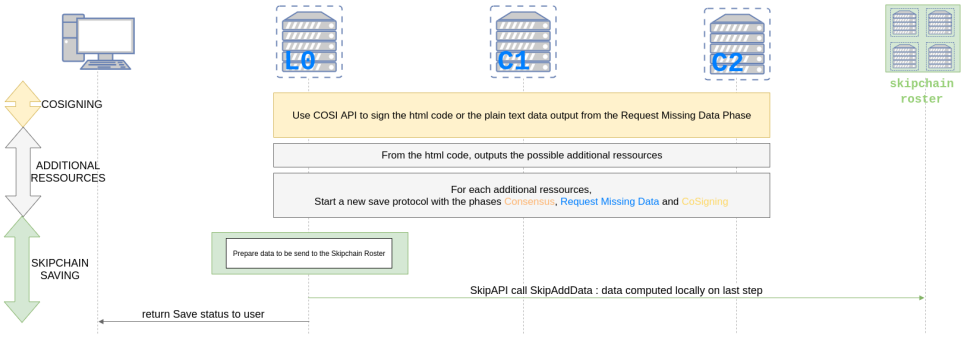

Figure 4: CoSigning and Skipchain Saving Phase

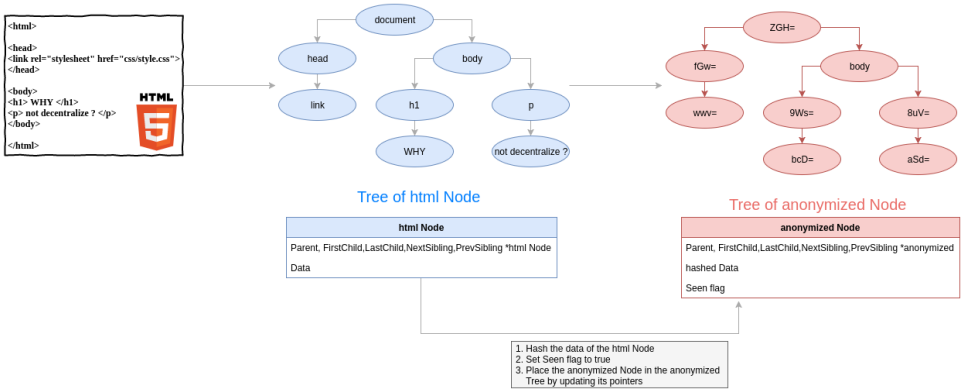

Figure 5: Detail of turning an html document to an anonymized tree

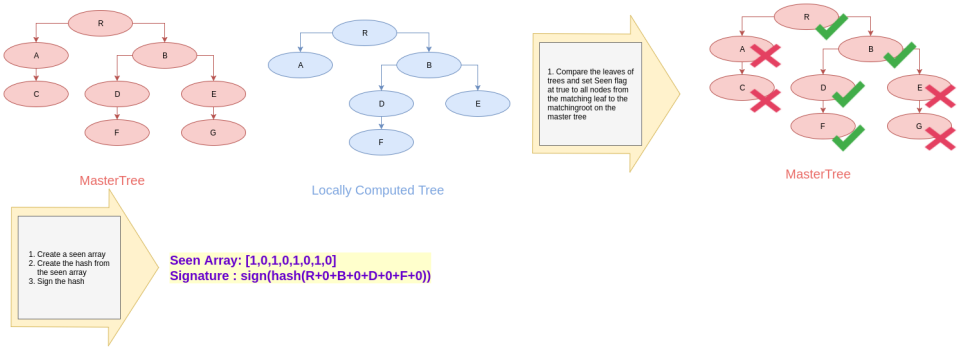

Figure 6: Detail of creating the signature and the seen array from the local and master trees

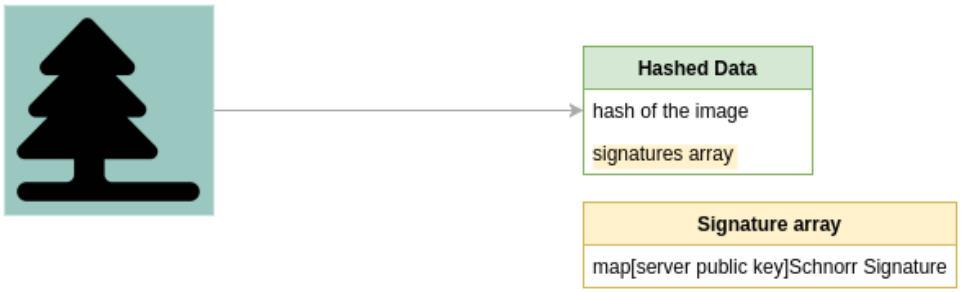

Figure 7: Detail of turning an image or any unstructured data into a signed hash

3.5 Retrieval Protocol

The retrieval protocol is the simplest from this project point of view since it heavily relies on the Cothority CoSi and Skipchain API. Its aim is to provide a tool for a user to retrieve a webpage stored in the skipchain at a given time. The details of the interaction to with the skipchain are developed in section 3.6. In this section, we focus on the relations between a user and the Decenarch Retrieving service. We only show the API requests and responses with the Decenarch Skipchain Service that is represented as a green square called skipchain Roster in all the figures that will be used in this section.

3.5.1 High-Level Presentation

The main steps of the protocol are the following:

1. The user provides an url and a time stamp to the Decenarch Retrieve Service via the Decenarch client API.

2. The Decenarch Retrieve Service forward the information to the Decenarch Skipchain Service

3. The Decenarch Skipchain Service retrieves the correct data (see Section 3.6.2 for details) and send the data back to the Decenarch Retrieve Service

4. The Decenarch Retrieve Service verifies the signature of the data and send it to the user

5. The Decenarch client on the user machine saves the data on the user file system and output a path to the requested file.

3.5.2 Protocol Details

On a more detailed level, we apply the following protocol, illustrated in Figure 8[6]:

1. The user makes a request to the Decenarch Retrieve service through the API. The request contains an url and optionally a time stamp (that will be the current time if left blank).

2. The service receives the url and the time stamp and send it to the Decenarch Skipchain service applying the *get data* operation[7].

3. The Decenarch Skipchain service send the data requested with their associated signatures.

[6]The details about the skipchain part is describe in section 3.6.2.
[7]See section 3.6.2 for details.

4. The Decenarch Retrieve service checks the signatures and if they are valid, send the data to the client in an encoded format that contains the signatures.

5. The user Decenarch client receives those data and register them in the user file system. The file path is constructed as following:

   (a) For the url : https://subdomain.domain.ext/folder/file.html

   (b) $\$PREFIX/ext/domain/subdomain/folder/file.html$

   (c) with $\$PREFIX$ being a constant of the Decenarch client.

6. The Decenarch client outputs the local file path to the requested page to the user.

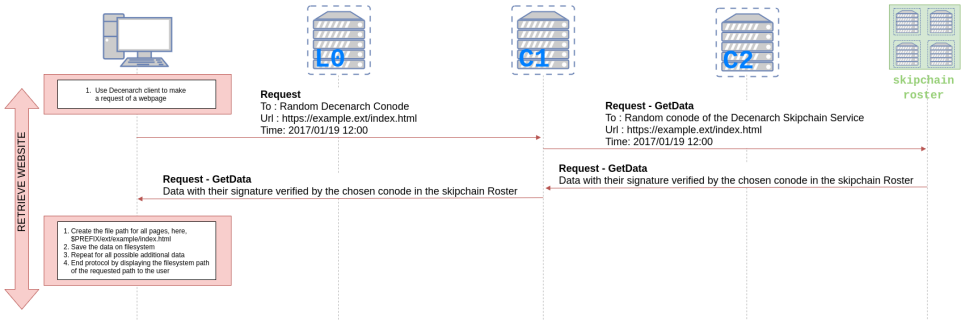

Figure 8: The Decenarch Retrieval Protocol

3.6 Skipchain Storing, Maintaining and Retrieval Protocol

In the decentralized archive we developed, we created two services, one service to save a website by reaching a consensus as well as to retrieve the saved website and one service to handle all the operations involving the skipchains. In this section, we detail the latter. Its role is to serve as a proxy between the skipchain and the user that want to use it. It provides a limited number of operations ensuring that the request to the skipchains are valid and that the data added to the skipchain were correctly signed.

An important remark before reading this section is to understand what are the key entities involved in the processes. There are (a) the Decenarch Saving/Retrieving service defined on section 3.4 and section 3.5, (b) the Decenarch Skipchain Service (also called SkipService in this section) that are the proxy between the skipchain and the Decenarch Saving/Retrieving service and finally (c) the Cothority Skipchain API[12] that is an implementation of the skipchain[13] developed before this current project and used as a black-box tool.

3.6.1 High-Level Presentation

The Decenarch Skipchain Service is responsible to start,stop and maintain the cothority skipchain[13]. It uses the Cothority v1 skipchain API[12] in order to create and retrieve skipblock and to verify the validity of blocks.
In the context of the decentralized internet archive architecture, the Decenarch Skipchain Service is used as an interface between the Decenarch Consensus Service and the user. Mainly, the user only makes call to the Decenarch Saving and Retrieval services and they are responsible for making the call to the Decenarch Skipchain Service. The only exception to this rule is the start and the stop operation of the Decenarch Skipchain Service which are directly made by the responsible of the Skipchain Roster using directly the Decenarch Skipchain Service.
To resume, we have the following entities:

- The Cothority Skipchain API which was developed before this project and whose role is to handle all low-level operation on the skipchain.
- The Decenarch Skipchain Service, called SkipService in the Figures 9 and 10 which is an interface between the low-level Cothority Skipchain API and the Decenarch Saving/Retrieving .
- The user of the Decenarch Service that will never call directly the SkipService and is represented as the computer with a screen in the Figure 10.
- The responsible of the Roster handling the skipchain that is represented as the computer with a screen in Figure 9

- The Roster responsible of saving and retrieving the website called *conodes roster* in both Figure 9 and Figure 10.

In terms of entity relations, we have the start and stop operation that are made directly by a user which is assumed to be the responsible for the skipchain and the add and get data operation that are done through the Decenarch conodes Roster that act as a proxy.

3.6.2 Protocol Details

The Skipchain Service has 4 operations: *start*, *stop*, *add data* and *get data*. The Figure 9 represent the start operation. The Figure 10 represents the add and get data operations. The stop operation does not have a special figure since its steps are quite easy to understand with text only.

The start operation , shown in Figure 9, has the following steps:

1. The responsible for the SkipService Roster use the API and make a SkipStart call. On Figure 9 it is represented by the computer with a screen on the most left part of the image.

2. The SkipStart function first handle the creation of a skipchain and thus of a genesis block:

   (a) The Start Root, where the user sends a SkipStartRoot request to a conode chosen, on Figure 9 it is L0.

   (b) L0 then create a genesis block with no data and use the Cothority Skipchain API to dispatch the information to the other member of the Roster. **it is here that a verification method for the blocks are defined. We used the standard one[12]**.

3. The SkipStart function then make a SkipStart request to each member of the Roster, even the root of the skipchain. Its aim is to create a go-routine on each conode responsible of creating a block regularly[8]. A go-routine is mainly a new thread on the server, so it is important that a **channel** is created in parallel in order to be able to communicate with the thread. The channel is read by the go-routine responsible of creating the block and is written on by any user of the API that want to add data to a block. It can correctly handle the format of website produced by the Decenarch Saving protocol.

[8] It was coded as 10 minutes and on the Figure 9 it is represented as X minutes. This is a parameter of the skipchain

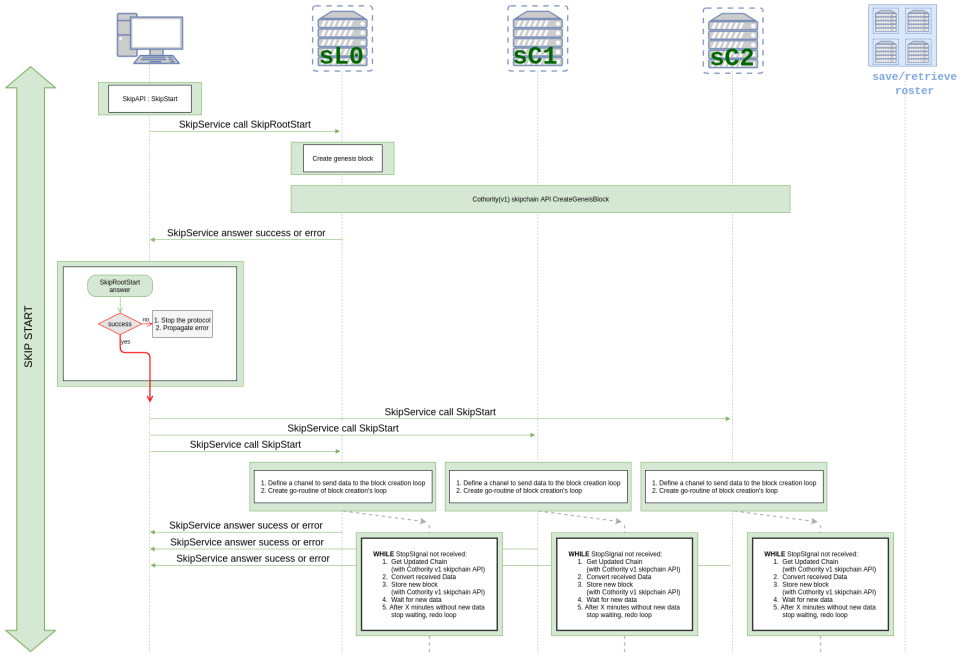

Figure 9: SkipService - Starting the service

The add data operation , shown in Figure 10, has the following steps:

1. Before the actual operation, a user of the decentralized service make a saving call on the Decenarch Service. It is the latter that will actually perform the call to the Skipchain Service since the format used to store website on the skipchain are tightly bound to the consensus and saving protocol. This step is represented on Figure 10 by the blue arrows. The user and the Decenarch Service being respectively the computer with the screen on the most left and the blue square with servers in it on the most right of the figure.

2. The actual protocol begins when the Decenarch Service responsible for saving the website, called *conodes roster* in Figure 10 makes a SkipAddData request to a random conode of the Skip Service Roster. The request contains all the data needed to recreate the website **as well as their respective Cothority CoSi signatures**.

3. The conode that receives the request, called sC1 on Figure 10 verifies all signatures and sends the correctly signed data to the go-routine responsible of creating the block via the channel created during the start operation.

4. The adding operation returns an error state to the Decenarch Service that propagates it to the user. It is important to note that the data will

be effectively added to the skipchain only when a new block is created so they can be a significant delay between the add data operation request and the possibility to get the added data.

The get data operation , shown in Figure 10, has the following steps:

1. The user makes a request to the conodes Roster in which it indicates an url and a time stamp indicating which website. The snapshot retrieved will be the one a user could have seen if it had consulted the website at that time.
2. The Decenarch Roster will then send the same data to the SkipService by making a SkipGetData request.
3. The SkipService will explore the blocks from the latest to the genesis block and for each website store in the block will check that the url of the web page save is equal to the requested and if the time stamp provided by the requester is after the time stamp saved. Once both match, it outputs the website and look **in the same block** for all the possible additional resources associated with the page requested.
4. The data are then sent to the conode roster that launch the retrieve protocol detailed in Section 3.5.

The stop operation has the following steps:

1. The user makes a SkipStop request to each nodes of the SkipService Roster.
2. The conodes will put their stop signal to true and thus stop the go-routine responsible for creating the blocks.
3. On a side node, the creation loop can also be stopped by killing the process but there will be no way to restart the skipchain. The creation of a new chain will be necessary.

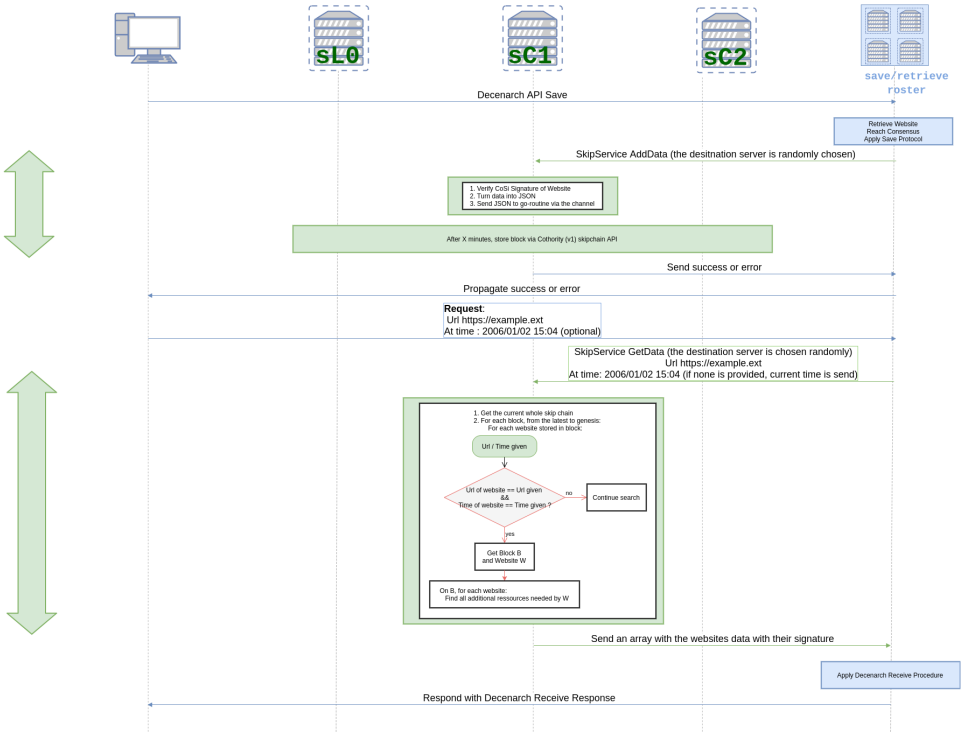

Figure 10: SkipService - Adding and Getting data from a block

4 Evaluation

4.1 Theoretical Analysis

In this section, we try to obtain order of complexity for different part of the Decentralized archive services. In the end of this section, we propose general order of magnitude for the consensus/saving protocol, the skipchain handling and the retrieval protocol.

We define the following variables:

- N the number of conodes in the Decenarch Save and Retrieve Roster.
- M the number of conodes in the Decenarch Skipchain Handling Roster.
- K the number of nodes in an html-tree generated from an html page.
- W the size in bits of the html page.

We first try to find some relevant bounds in term of space.

The bandwidth use depends obviously on the size of the web page one wants to retrieve or save. Furthermore, it also depends on the number of the conodes involved in the protocol. We can identify the following proportions:

- The number of request done by the Roster to the website in the saving protocol are $N+1$ since the leader request the website to create the reference and then every conode in the Roster (including the leader) requests the web page. So we have that the bandwidth use for the main web page is of order $O(N * W)$.
- On the number of messages passing between one conode and another, we have:
  - The leader passes its reference to all its children so it is in $O(N * W)$.
  - The children pass their signature and the reference back to the leader so it is in $N * O(W) = O(N * W)$.
- Finally, if the page has i external resources of size W_i we can add $2 * O(N * W_i)$ messages.

So we have $2 * O(N * W) + \Sigma_i 2 * O(N * W_i) = O(N * W) + \Sigma_i O(N * W_i)$.

Now that we have studied the cost in term of space, let's look on the different time bound that are relevant.

The cost of comparing and signing an html tree is one of the most impacting part of the consensus protocol. It must compare the locally computed html tree and the reference tree. In the intermediate results on the list below, we put that $K = max$(number of nodes in html reference tree, number of nodes in local html tree) :

1. First the local node must assure that the reference is correctly tagged, its complexity is $O(K)$.

2. Then, the local conode must compare both of its tree with the reference. To do so, the worst case is comparing each of its node with the nodes of the reference tree, so we have a complexity of $O(K^2)$.

3. Finally, the local conode must create the signature, going through the reference tree again with a complexity of $O(K)$.

So, for the comparison of a tree we have a complexity of $O(K^2 + 2 * K)$ which is $O(K^2)$. Since each N conode will do this operation, the overall complexity of the signature creation is $O(K^2 * N)$

The cost of verifying the consensus signature is also a factor that vary, depending on the number of conode N. It does also depend on the number of nodes in the html tree K since the verification of a signature requires to create a hash that can be infer from the reference tree. The leader must control each of them. So we have a cost of $O(N * K * cost_{verification})$ with $cost_{verification}$ begin the cost to verify a signature. We consider it constant, so we have a final order of $O(N * K)$.

The cost of the handling additional resource step can highly vary since it depends on the number of additional data and their type. But if we assume that the cost of handling all the data on a single conode is A, we will have a complexity of $O(K * N * A)$. we have the K, which is the number of nodes in the html tree of the main web page requested, on the formula because the leader conode must extract the links that could point to additional resources from the main page. To do so it must parse the page and the time taken to parse a page is strongly correlated to the number of nodes in its tree since a high number of nodes means either a high complexity in the structure or a high number of lines and all two leads to an increase of the parsing time.

The cost of saving the web pages to the skipchain from the consensus protocol point of view is quite straightforward since the operation is only made on the leader conode. The data are simply send to the Decenarch skipchain Roster so the complexity is $O(W + \Sigma_i W_i)$ with W_i the size of the i^{th} additional resource.

The cost of the collective signature request is quite difficult to evaluate since we use the Cothority CoSi API as a black box. But we can assume that it is $O(N)$ since all the conodes must participate at least once in constant time in the collective signature.

The overall complexity of the saving protocol is the following:$O(K^2 * N)+O(N * K)+O(K * N * A)+O(N)$ which can be reduced as $O(K^2 * N + K * (1 + A)N + N$.

4.2 Simulations

This section evaluates the Decentralized Webarchive. The simulations were all realized locally so the possible influence of the network connection between the conodes of a Roster were not considered. Furthermore, the Roster responsible for saving and retrieving the web page and the Roster responsible for handling the skipchains are composed of exactly the same machine. All simulations were all made on a Lenovo T470s with a CPU Intel Core i7-7600U 2.80GHz, with 20GB of RAM.

The variables defined in the beginning of Section 4.1 are kept in this section. Furthermore, we make the following keywords association (used in the Figures of this Section) :

- consensus : the start consensus phase. More precisely from the user request to the creation of a plain text html code issued from the consensus Phase in Section 3.
- cosi : the collective signature. More precisely from the reception of the plain text html code to the reception of the collective signature by the service. It corresponds to the CoSi phase in Section 3.
- additional : It corresponds to the moment where the signed html code is parsed to find additional links to the moment where all the additional resources are retrieve and signed. It corresponds to the Additional Resource Phase of Section 3.
- skipchain add : Correspond to the Skipchain saving Phase of Section 3 and takes place from the moment the data are being sent to the Skipchain Handling Roster to the moment the confirmation response is received.

All the Figures of this section have linear scaled axis except axis where the text *log scale* is present. These axes have a logarithmic scale.

The first simulation was made on standardized website created for the purpose of these simulations. Their source codes are always of the form:

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>\o/ nibelung.ch \o/</title>
<meta name="viewport" content="width=device-width,initial-scale=1">
<link rel="stylesheet" href="css/style.css">
</head>
<body>
<h1> DecenArch - dummy site</h1>
<p> me voila ! </p>
</body>
</html>
```

With an increasing number of `<p> me voila ! </p>` lines. The result of the simulation can be seen on Figure 11 and Figure 12 and the plain text data used in the Appendix. Since the host is the same for every of those simulations, it allows to reduce the noise that could come from a significant bandwidth difference between two distant hosts. For example, when someone requests a website hosted in Switzerland and a website hosted in United States of America. Another possible bias introduced real life website is a significant difference in the size W of the website that would not impact the number of nodes K of the generated html-tree, for example comparing a website that contains multiple images and a website that contains multiple paragraph of texts.

Let's study the result of this simulation. The first thing we can say is that the main time component here is the consensus phase. Nevertheless, this result is not surprising. As we have seen in Section 4.1, the CoSi signature scheme used depends on the number of conodes in the Roster, since this number is constant, the CoSi time can be considered as constant. Furthermore, since the web pages have no additional data, the time taken to treat this part is also negligible but increases slowly because this step involves the parsing of the web pages to look for external links meaning the longer the page, the longer the parsing. Finally, the step of sending the data to the skipchain is also negligible here, growing slowly, similarly to the part involving retrieving additional data. The validity of this last assumption may not hold on real life structure since the service will possibly need to send data of size $O(W)$ to another machine. On this simulation, the Roster responsible to reach a consensus on the website and to store it in a skipchain were not only the same but also simulated on the same machine. Nevertheless, here we have that the significant parameter is the consensus phase which is not surprising as it is but its growth worth the study.

As we have seen on Section 4.1, the complexity of the consensus phase

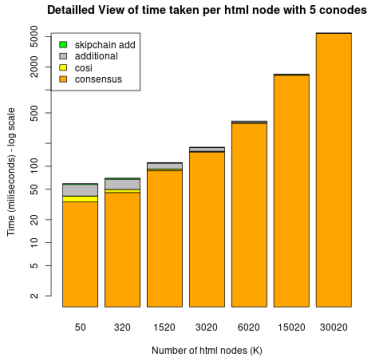

Figure 11: Step-separated simulation on 5 conodes on different standardized website

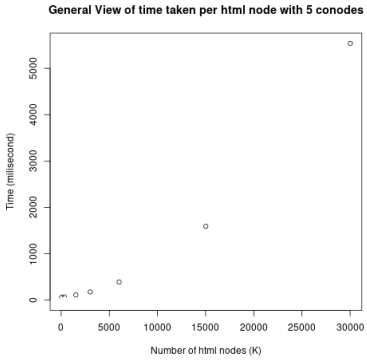

Figure 12: Step-aggregated simulation on 5 conodes on different standardized website

in terms of html nodes is $O(K^2 * N + K * (1 + A)N + N$. On this simulation, especially on Figure 12, we can see that the growth is polynomial in terms of number of nodes in the html tree. The most probable explanation can be found in the considered most expensive part of the consensus protocol: the comparison between the reference tree and the locally retrieved tree in order to create a signature. In the simulation, both html tree were exactly identical so their number of nodes are equal and is K. Furthermore, the signing node has to compare all its nodes with all the node of the reference tree so we reach a worst case complexity of $O(K^2)$ and thus found an explanation for the distribution seen on the simulation.

The second simulation was made on real life website but otherwise keeping the same setup as in the first simulation. The result can be seen on Figure 13. The websites[9] are in order from left to right on the figures. The numbers below correspond to the Web Id:

1. `http://decenarch.nibelung.ch`
2. `http://google.ch`
3. `http://www.cbc.ca/radio/quirks/how-an-[...]-1.4470200`
4. `http://blog.golang.org`
5. `http://www.bbc.com/travel/story/20180102-[...]language`
6. `http://www.20min.ch/ro/news/monde/story/Un-avion-[...]-16596126`

[9]Some of the websites url were shortened using [...] to fit in a line. The whole url can be found in the Appendix

7. `https://www.lenouvelliste.ch/articles/lifestyle/sortir/melanie[...]728845`
8. `http://tass.com/science/984384`
9. `http://tass.com/politics/984968`
10. `http://www.bbc.com/news/world-us-canada-42680070`
11. `http://20min.ch/ro`

On the real-life simulation, the results are quite different from the first one. We can see that the main factor of the time taken to finish the saving protocol is clearly the additional data part. Furthermore, its growth seems arbitrary. Nevertheless, none of those facts contradicts what we already know. Since a website can have a lot of additional data as it seems to be the case for the golang.org blog and the 20min.ch pages, the protocol is restarted for each of additional data. Thus, the impact of an additional data can be big even if it is only an image that does not require a complex signature scheme. This leads us to the explanation of the seemingly arbitrary growth of that factor: the time it requires does not depends on K, the number of html nodes of the main page requested but on the number of additional resources a web page needs.

To conclude the second simulation, we can say that the importance of a good managing of the additional resources is critical on the reduction of time it takes to reach a consensus and save a website. Since it makes the time to reach a consensus on the main page almost insignificant. On a more positive note, we can say that our protocol takes less than *40 seconds* (the red line of Figure 13) to reach a consensus on the websites. Obviously, this does not take into account the fact that blocks are added to the Skipchain at regular interval of times. Assuming we make a saving request just after the creation of a block, we would have to wait until a new block is stored on the skipchain, 10 minutes in the current implementation, before being able to access the website using a retrieval request.

The third simulation measures the the impact of the number of conodes in the Saving Roster on the time it takes to reach a consensus. The simulation was realised on the same website `http://nibelung.ch/decenarch/5000p.html` but changing the number of conodes in the roster. The results can be seen on Figure 14 and Figure 15.

Before beginning with the analysis of the results of the simulations, it can be relevant to highlight that a simulation with 75 conodes runs in parallel on the computer used for the simulation failed. So we are unable to tell how really the protocol would behave with a hundred or more conode. Nevertheless, we will assume that it follows the same growth that we see on the current simulations.

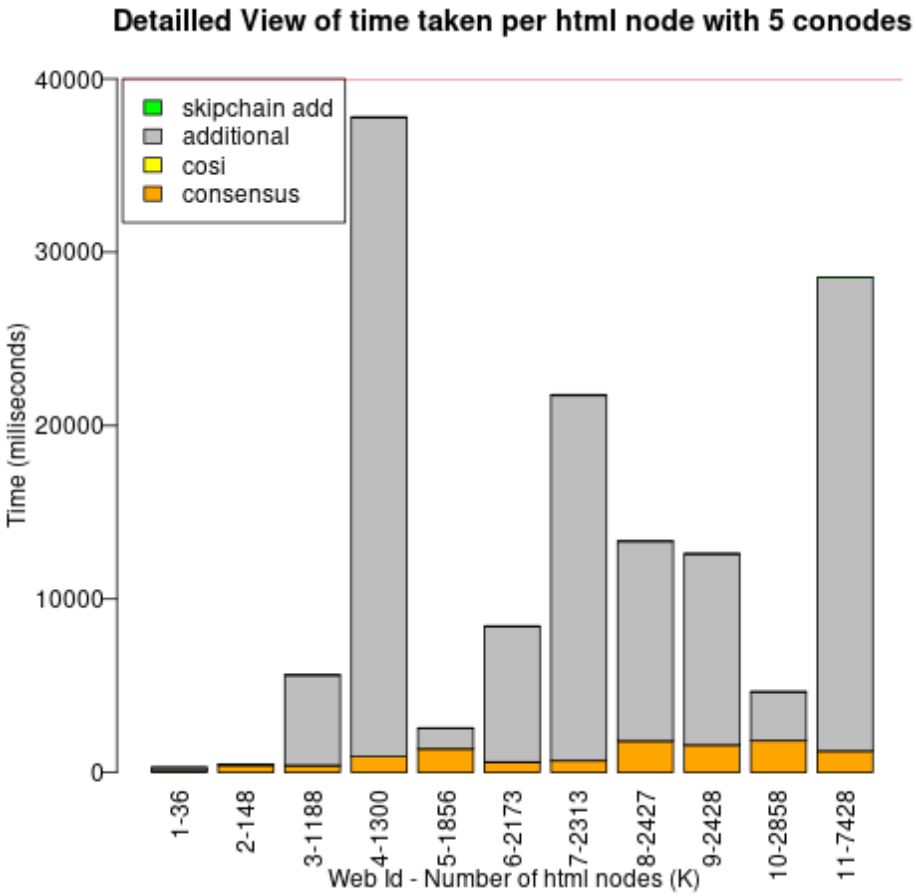

Figure 13: Step-separated simulation on 5 conodes on different real-life website

As for the first simulation, we see that the main time factor of the protocol is the consensus part. But on the contrary of the first two simulations, all steps take more and more time except the skip operations. The explanation for the growth of the consensus phase has been explained in both the Section 4.1 and the first simulation, but the difference here is that we change N and not K. So we expect a linear growth since the overall complexity of the consensus step is $O(K^2 * N)$ and indeed, on Figure 15 we can see that it is indeed very close to a linear function. The explanations for the behaviour of the cosignature and adding resource steps are also explainable with the theoretical result. For the cosignature, the more conodes are involved, the more signature will be needed. Even if we consider that signing has a constant time complexity, the cosignature will be at least in $O(N)$ since each conode must participate. For the adding ressources step, the explanation

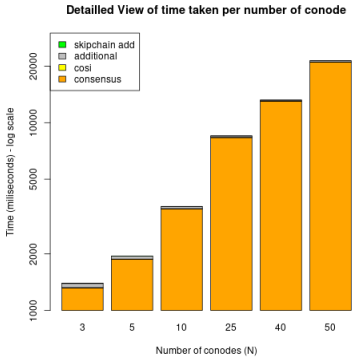

Figure 14: Step-separated simulation on the same website on different number of conodes

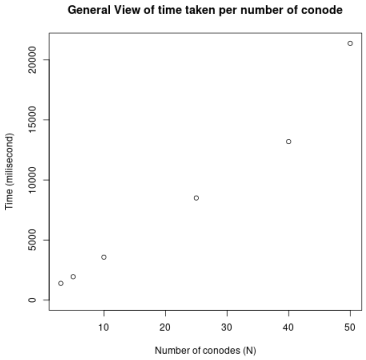

Figure 15: Step-aggregated simulation on the same website on different number of conodes

is very similar. Finally the relative stability of the step responsible to send data to the skipchain is explained by the fact that it is realized by the leader only whatever the number of conodes involved in the process so N has no influence on it.

5 Discussion and Possible Improvements

As we have seen in Section 3, the protocols implemented in this project have a lot of constrains like the trust one must put in the leader. Furthermore, the choice of a tree to handle html data as well as the way to sign it and reach a consensus on it seems to be arbitrary. In this section, we explain our choices and quickly summary the iterative process that make us reach the current implementation. We finish each explanation by a proposition of improvement or some clues about a possible direction that could lead to it. Finally, we talk about global improvement of the concept of the distributed internet archive without focusing especially on the current implementation of it.

The consensus structure for html documents was an early choice in the project. We implemented a tree-based approach, mainly opposed as a byte-per-byte one. The goal of the consensus is to be able to keep only parts of the html document that are common to the majority of the conodes in a Roster. The first but naive way to do so is to hash the document and drop the whole pages if the hashes do not match. But in that case, we have no granularity in our choice and two documents that are identical but one word, for example if a *Hello Alice* becomes *Hello Bob*, then we would not keep any part of it. To introduce granularity to this first draft, we can simply cut the document per-byte and compare them. Even if this can handle the problem of very similar but different documents, another problem happens: the html page is a structured document and removing a byte of it randomly can damage the file and make it unreadable. Even if one could test that the document stay valid for each removed byte is possible, the cost of doing it is exponential since we have to test all the possibilities for all the possible group of bytes. From these considerations, we assumed that a byte-per-byte approach was not the clever one and that the solution proposed must handle both the granularity of the document but insure that the structure of it is kept thanks to a way of comparing that only allows the removal of unnecessary part of the document, making it a valid document at any point of the comparison. The tree-based approach fits all the criteria. We first turn the html document into a tree and allows the removal of the leaves of the tree only. Indeed, in a tree representing an html document, the content is only at the leaves level, all the other levels being related to the structure of the document only. Furthermore, a decent granularity can be reached since most website separate their content in different structural entities.
Nevertheless, the current solution is not free from criticism. The first one is on the fine grain granularity. Indeed, when the hashes of two different leaves are not equal, we consider them unequal despite the fact that they can vary by only one word. And even if the problem is less critical than in the first naive approach, it remains. One improvement that could be made

is that instead of considering the leaf that contains text as a single entity it could be broken in half or quarter and compared with a byte-per-byte approach. The structure of the document will remain valid and so will the document itself but the drawback is that there is a risk of outputting text that has no meaning. Another solution to avoid that is to have a semantic approach instead of a syntactic one. For instance, for images, we can imagine a pixel-by-pixel comparison that would have the advantage of detecting some watermarks. The drawback of those improvements are a rise of the time cost of the consensus phase and the need to inform the conode about the type of data they have to handle.
The second criticism is that the link between the collective signature of the consensus plaintext output and the tree used to realize this consensus is impossible to do only with the collective signature and the plaintext. This creates a security weakness since the leader of the protocol could tamper the plaintext without the other conode knowing. A solution to this problem is currently implemented: the creation of another proof which is mainly the tree used for the consensus and the associated resources that allows to build the plaintext that should have been outputted. Nevertheless, this proof is not actually used and can be seen as a proof used in an audit. An obvious improvement would be to integrate this poof in the collective signature phase of the plaintext to avoid the need of keeping it.
This last criticism leads us to the strongest assumption of the saving protocol: the need of having a trusted leader.

The trusted leader assumption , more precisely the trust we have to put in the leader of the saving protocol to create reference and consensus trees honestly is the strongest constrain on the protocol. Thus, we will develop a small explanation of the reason of this assumption. Our starting point will be the fact that the consensus will be made on a tree. So ideally we want to be able to compare two trees and output the union of those tree with a verification scheme that allows us to say that some nodes were seen twice and some only once. Then doing a similar operation for all the conodes to participate and outputs the tree containing only the tree-nodes seen by the majority of the conodes.
For the creation of this protocol, we put ourselves under the following constrains: a limited bandwidth use, a time-complexity that scale to real world websites to create the tree and finally, the display of the fewer information about what a conode has seen.
The first solution that we implemented, detailed in the Appendix of this document, is a path-based approach where every conode creates a hash of each path in their tree and signs it. If two hashes collide, then we considered the two path identical thus the path was signed by the two conodes. This solution was bandwidth efficient since it reduced the number of information

to sign to the number of leaves. Furthermore, it was impossible for a node that does not have the plain text path to guess what the other nodes had seen. Nevertheless, this solution had two drawbacks. The first one is that the number of signatures to verify is in $O(number_{conodes} * number_{leaves}$ which does not fit real-life webpage that typically have $3000 - 5000$ leaves. The second drawback was fatal to the solution. The path was not a sufficient information to identifies the leaves deterministically. For example, the two path $R{-}A{-}B$,$R{-}A{-}B$, given from root to leaves with $R, A, B \in htmlNode$, even when we assume that the root R are common, can represent the tree $R^{-A^{-}B}_{-A-B}$ or the tree $R - A^{-B}_{-B}$. This case being very frequent on webpage, we abandon this path-based approach for a tree-based approach. Considering that displaying the structure of the tree, without revealing the tags and most importantly the content of the leaves were a reasonable loss of privacy.
In the tree-based approach, the obvious solution is to apply exactly the initial aim of the consensus protocol: creating the union of two trees and when the union between all the trees has been made, outputting only the nodes that are seen by a majority. This solution is not viable as it is since the complexity of the union of two tree is in NP[15] mainly it reduces to the max-clique problem. Furthermore, the signature would have to be made node-by-node since the tree could be modified at every union operation thus the complexity of verification of the signature is not decreased. For these reasons, we decided to introduce a reference tree that is compared to the locally computed tree by the conodes. Nevertheless, it imposes us to have an honestly made reference thus the need of having a trusted leader. This solution, presented in Section 3, allows us to have a complexity of order $O(number^2_{treenodes}$ which in polynomial.
Even if the solution can be considered good, the assumption of a trusted leader weakens the censorship resistant properties of the protocol since a dishonest leader could censor any part of the website without being detected. A great improvement would be to be able to add nodes to the tree thus preventing a leader to hide information and allowing the other conodes of the Roster to detect a malicious leader. One obvious solution to do so is to have a multiple round protocol where each conode takes the leadership once but such a solution would probably not scale with a large number of conodes. Another solution that could be worth exploring is making the protocol a two-round protocol where in the first round, the conodes collectively create and sign the reference tree and in the second round the protocol implemented here is used.
But the creation of the collective tree also relies on a efficient signature and verification scheme that was a problem in the first path-based solution. Thanks to the *trusted leader* assumption, we were able to create a more efficient signature scheme.

The consensus-tree signature scheme is also a key component of the consensus protocol. As we have seen, the initial path-based solution would not allow us to have an efficient signature scheme, but the trusted leader assumption have make it possible for us to have a relatively efficient signature scheme with a complexity in $O(number_{conodes})$. The implementation described in Section 3 can be summarized by saying that the conodes generate a hash from the nodes that were both in the reference tree and their own tree. This hash depends entirely of the reference tree and the knowledge of which nodes were taken. The signature is then a list of node seen and a signed hash with the conode private key. The number of signature is then always equal to the number of conodes whatever the initial tree thus the time-complexity of the verification is $O(number_{conode})$.
This complexity is difficult to improve since we can assume that $number_{conodes} << number_{htmlnode}$ if we use the protocol on an average sized web page. Furthermore, since all the conodes must participate at least once in the protocol thus producing a signature it seems difficult to have a better complexity. The only way to improve such a complexity would be to have an aggregated signature or a set of aggregated signature of constant size that would provide the number of time a certain node was seen and the insurance that all participants were allowed to vote and voted exactly once. Unfortunately, we do not find such a signature scheme.
To conclude on the consensus-tree protocol, we can say that the censorship resistance and the privacy properties of the whole decentralized archive also rely heavily on the consensus part of the protocol. Furthermore, both the theoretical analysis and the simulations shows that an efficient protocol is essential for the scalability of the project. Nevertheless, the simulations realized on real website shows that the handling of additional data has a great impact on the scalability as well.

The handling of additional data is another important component of the decentralized archive. Even if, from the technical point of view, the consensus protocol applied to those data are exactly the same as the one applied on the main page, from the scalability, performance and content-preserving points of view, this question becomes critical. Indeed, for the scalability and performance points of view, the simulations of Section 4 showed that the additional data handling is the part of the protocol that impacts the time it takes to save a website the most. Furthermore, since those data includes images, the weight of the data that will be stored for a given web page greatly depends on the data associated with it, assuming that the web page saved is an html text file. From a content-preserving point of view, this question is also of great importance since we can imagine cases where the text talk about an image or use the images as a support for an explanation. In this case, the image becomes critical for preserving the interest of the

web page.
In our current implementation, is considered as additional resources only the images and the css files. We considered to be a good trade-off between simplicity of implementation, content-preserving archive and storage efficiency. Nevertheless, one could argue that a video can be as relevant as an image or that a css file is useless in a lot of cases. Once could also take the counterpart of our choices, focusing on storage and performance saying that images could simply be dismissed and the choice of having associated image is left to the user willing to save the web page. Finally, the last main criticism we can do is not about what to save with the page but how it is done. Indeed, in the current implementation of the protocol, if a link in the saved html web pages is given in absolute path ($http://...$), then the image considered as additional data and saved too but since there is no edition of the code, the user willing to open the page in his browser will request the distant image and not the one saved.
On the three criticisms exposed here, the latter is certainly the first to consider and to resolve to improve the protocol. For the moment we did not edit the source code in order to be able to verify the associated signatures of it easily ,but we can imagine a bijective deterministic way of editing the html source code in order to avoid external internet request but being able to reconstruct the original source code. For example, by editing the link only when a user retrieves the web page from the decentralized archive turning absolute path to an internet resource to an absolute path on the local user file system. The first and second criticisms are more related to what we consider relevant to preserve the meaning of the web page. We consider that giving the user of the archive the responsibility to choose which image is important should be absolutely avoided since the process will not be deterministic from one user to another and that we would give the user the power to censor certain part of the web page he saves. Nevertheless, a harsh but deterministic selection of the additional data is another axis one could take to improve the protocol. We could imagine a natural language processing tool that could give us clue about the importance of an image. Some tools maybe already exist. We did not do any research on the subject since we considered it was out of the scope of this initial protocol. This could save not only times to resolve the saving protocol but also allows us to be more space efficient.

The storage efficiency was not considered a central and critical problem of this project. Nevertheless, it can be useful to expose the current situation and to suggest a few improvements that could be done to it. In the current implementation of the project, the plain text data and their associated collectively signed signature are all stored without any transformation on a single skipchain. This solution is the simplest to implement keeping in mind

that we have to keep the data stored in a decentralized way to be able to claim any censorship resistant related properties. From this solution, multiple possibilities of improvement *without loss of decentralization* exist. The most obvious improvement that can be made is storing all the data but the proofs on a peer-to-peer system for example. Indeed, the skipchain storing provide a security layer that is critical to have for the signature making them useless if stored on a peer-to-peer system. But the data could be stored on a system that does not have that kind of layer without drawback since the signatures are made to control the validity and the integrity of them. The only criterion to have in mind is that it is necessary that the system chosen to store the data is distributed or at least cannot be made inaccessible by shutting down a single machine or a single entity to keep the benefits of the distributed system.
An improvement that can be made in a second time is to use multiple skipchain to store the signatures, allowing to have multiple Roster that runs in parallel distributing the work load if we imagine a heavy use of the system.The main task to do in order to apply that kind of idea is the implementation of a communication protocol between the different Roster and an efficient way to avoid having unsolicited copies of the same signatures stored on different skipchain.
In parallel to the improvement of the signature storage, we could also improve the storage of the data itself by adapting solutions that were developed for backups. For example, we can imagine to have an incremental backup style storage where we save the whole website the first time a save request is made but then only patches to apply to obtain the new version of the page.
In conclusion, we see that the current implementation leaves plenty of room for storage improvement. Again, it was not a primary concern of the project even if an archive that should be *real-world resistant* should definitely focus on those problems. The latter are not the only one that are of great importance. The security issues are fundamental in the decentralized internet archive project. All the signatures scheme involved here are mainly a consequence of those possible issues.

The passage from the anonymized consensus tree to the plain text html code collectively signed is the first weak point of the protocol from a security point of view. Indeed, the signature scheme used in the consensus and the whole consensus protocol's design does not allow for a user to control that the plain text html code collectively signed is indeed the actual result of the consensus protocol. To partially reinforce this flaw, we created a webproof that is mainly the reference anonymized tree and the signatures sent by the conodes to the leader during the consensus protocol. That webproof allows someone to control that the html code collectively signed is indeed the result of the consensus protocol on the reference tree

given. We recall that we assume that the leader is honest when it creates the reference tree. Nevertheless, this webproof is used nowhere and not stored. It can be seen more as an audit tool or as a proof that can be given on demand. An improvement to the protocol would be to integrate this proof or to develop a tool that makes it possible to verify the link between the anonymized reference tree and the html code. This point is indeed critical for the security guarantee of the protocol as well as for the censorship resistance of the whole. But other criterions, not directly related to the implementation should also be considered.

The conodes requirements to guarantee a censorship resistant archive are also a critical security issue that are not explicitly given by the design of the decentralized web archive. It is necessary to have a sufficient number of conodes. Indeed, if you run the protocol on only one conode, the result is no different than a centralized system and even adds useless complexity. Furthermore, running the protocol on small Roster of 3-5 conodes can still not be considered decentralized enough since a collusion between such a small number of participant is quite easy to make. The minimal number of conodes required to avoid collusion is difficult to estimate but there are some criterions that we could put in place to reduce the risk of collusion significantly. Firstly, it is important that the conodes are geographically distributed, on particular, the jurisdiction under they are should be different. This avoids not only that a government impose a collusion to all the conodes. by blocking certain contents for all its country or by imposing a government monitoring system on all the computer for example but also a possible collusion between malicious actors since it is more difficult to stay aware of all the possible nodes available to a user at a given time. Another requirement that could limit the risk of collusion is to impose that the conodes are run by different kind of entities. For example, universities from the public sector, industry from the private sector and individuals. Since those actors often have different interest in mind for running conodes, it avoids the risk of intentional or even unintentional collusion by serving their own interest. Finally, it is important to require that the conodes run an up-to-date version of the decentralized archive since no software is born perfect and exploits are always found after a given time. This avoids collusion that could be made not by the actors running the conodes but by a hacker corrupting the majority of the conodes in a Roster giving it the opportunity to remove the advantages of decentralization. In conclusion, the conodes must be geographically distributed, operated by diverse actors and regularly updated. Improvements in this domains are probably the least obvious to make. One would be to impose some of the requirement directly inside the protocols for example by refusing to run the protocol on a too small Roster or insuring that the conodes are from different countries. Another interesting research

would be to try to measure how easy it is for malicious user to create a collusion sufficient to compromise the protocol, try to identify the critical variables and impose strict policies that would be justified by experiments and results and not only rule of thumbs.

To conclude this discussion, we will propose a few scenarios and very briefly expose what guarantees we have.

1. Let's assume the protocol was correctly processed.
    - The website is archived on a skipchain. A modification or deletion is impossible without every conodes of the skipchain being malicious since the plain text data is collectively signed and the skipchain stay accessible as long as one conode serves it.
2. Let's assume that the leader created a majority of conodes were honest.
    - The case is identical to the scenario 1.
3. Let's assume the leader produce a correct tree and plain text code but the majority of conodes voted maliciously.
    - The website image contains only valid elements but can omit some of them. The worst case is when the web page is totally empty. The storage is still done with the same guarantee as in the scenario 1 and 2 but the archive of the web page is useless.
4. The conodes are honest but the web page itself was maliciously modified before the archiving.
    - Then we fall back to the scenario 3. Older version of the website, if it was regularly archived could help to still have a useful snapshot.

6 Conclusion

In this report, we have presented and evaluated a proposition of a decentralized internet web archive. We have seen that the actual internet is heavily and more and more centralized. That makes it censorship vulnerable and forcing the user to have a significant level of trust toward very few imposed entities. We have proposed a decentralized protocol that allows the user to request the saving of a web page whose content will be the result of a consensus between multiple machine. The decentralized archive uses the collective signature (CoSi) protocol and the skipchain structure provided by the Cothority framework. Through our simulation and theoretical analysis, we have seen that the protocol seems to scale for large website since its complexity is of order $O(number_{conodes} * number^2_{htmlNode})$ making it a polynomial time algorithm from the web page's size point of view and a linear one from the number of conodes point of view. Nevertheless, there is plenty of room for improvement either by relaxing some constrains or optimizing the storage management or the choice of the additional data that we need from the web page. Finally, we also saw the importance of the localization of the machine that runs the protocol as well as the need to be connected through the standard internet exposing them to the same threats as any other machines. Anyway, this decentralized archive is a small stone on the edifice of a more censorship resistant and distributed internet that becomes more and more accessible, usable and useful to all thanks to the rise of blockchains technology and the good will of computer scientists.

7 Appendix

7.1 Decentralized webarchive - Path-based consensus

The protocol presented in this section is the first tentative to reach a consensus on an html file represented as a tree. It is an adaptation of the old documentation written for the github repository[16]. Please take note that this protocol does not work and appears in this report for the sake of exhaustivity, Its assumptions are the following:

- a conode can only vote or not vote.
- a conode do not discard the vote of another server.

It also has the following guarantee:

- a conode cannot forge the signature of another node.
- a conode cannot forge the signature for a path it does not have.
- a conode learns nothing on the data another node has seen except if it posses those data itself.

It does not work because one can not reconstruct a tree in a deterministic way from list of the paths. Even using an order in the listing. For example, for the paths received in order $[R-A-B, R-A-B]$ with $R, A, B \in$ hashed data of html tree node, the path are given from the most left to the most right, from the root to the leaves. We have two possible tree[10]:

- $R \to A$, $A \to B_1$ and $A \to B_2$
- $R \to A_1$, $R \to A_2$, $A_1 \to B_1$ and $A_2 \to B_2$

7.1.1 General Network Architecture

In the Figure 16, we can see that there is two conodes network. One is used to get the webpage, reach a consensus on it and create the data for a block in the blockchain as well as requesting block from the chain. The other must create a block in the skipchain regularly and handle the data received by the blue conodes to create a block.

7.1.2 Protocols Descriptions

Saving a webpage When a user request to save a webpage, the conodes will go through different **phases** in order to serve it. THere will always be a leader of the protocol. **The leader is assumed to be honest**. The phases are the following:

[10] We consider that an html tree have exactly one root so we can remove certains possible trees.

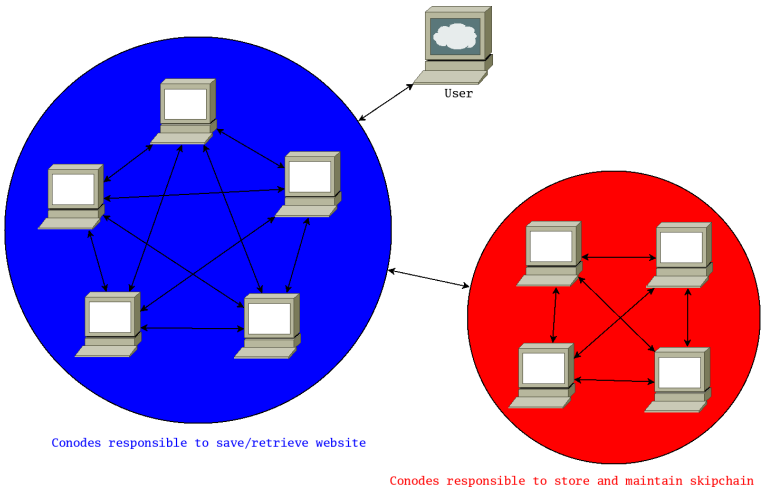

Figure 16: General Network Architecture

- **Consenus** : The conodes will all download the webpage locally. Then they will exchange messages until they find which parts of the website are found in a number of conodes sufficiently high which is 80% of the conodes of the Roster.
- **Request Missing Data** : Once a consensus is found. The leader will request for the plaintext data of the part of the website that it may not have but that are present in a sufficiently high number of conodes.
- **Cosigning** : Once a common website is created, the leader ask the conodes to sign the data using the CoSi client provided by the dedis/cothority framework.
- **Additional ressources** : The leader send the signed data to the conodes in order to reach a consensus on the additional data requested by the webpage. By additional data we mean images and css files.
- **Skipchain saving** : The leader emits a request to store the signed website in the skipchains to the conodes ersponsible of the chain.

In all the figures of this subsection, we assume that there are five conodes responsible for the consenus. Further more, we assume that for communication purposes, they organize themselves as the tree represented in FIgure 17.

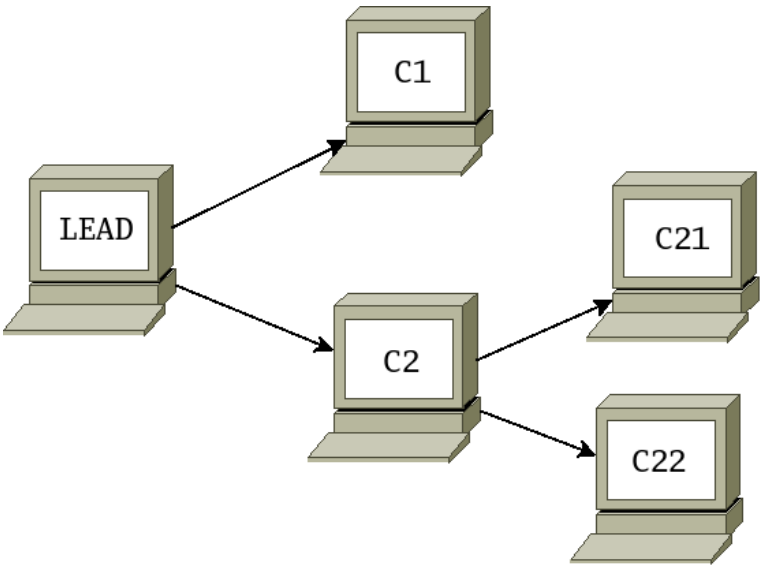

Figure 17: Consensus Tree

Consensus	
Step	Explanation
1	The user send a request to the leader with the url to save. The leader then simply pass the information down to its children and the operation is repeated until we reaches the leaves of the conode tree. The Figure 18 explains this step.
2	The conodes indicates how many time it sees the differents parts of the webpage, aggreggate it with the result of the children and send it to the parents. The leader will finally have a vision where for each parts of the webpage is associated the number of conode that have it too. The consensus protocol can handle two types of data: the structured one which are html and css document and the unstructured one which are images. For each type of data, you have two step, the conode that individually treat the data and outputs its answer and the aggregation of that answer with the ones the conode received form its children. The steps below detail those behaviours. The Figure 19 explains this step.
2.1	**structured data** 1. The conode take the structured document. It turns the document into a logical tree. Then for each path of the tree, it creates a hash and signs it. The hash of a path is done by hashing the leaf, then concatenate that hash with the plaintext data of the parent and hash that new string and repeating the process until the root is reached. 2. From its children, the node receives a map with the keys being the hashes of the paths of the tree infered from the structured document and the values a list of signature of the associated hash. For each of its own hashes, the nodes checks if it is equal to an exsiting hash in the map given by its chlidren. If yes, it adds its signature to the associated list. Else, it adds an entry to the map with its hash. The updated map will be send to parent. The Figure 20 explains this step.

2.2	**unstructured data** 1. The conode take the data and turn it into an array of bytes. Then it hashs it and sign that hash. It outputs the couple $(h(data), sig(h(data))$. 2. From its children, the node receive a map with the keys being the hashes of the data and the value a list of signature of the hash by the conodes. If there is a match, the conode add its signature to the list of signature associated with $h(data)$. Else, it adds an entry to the map. That updated map will be send to parent. The Figure 21 explains this step.
End of Phase	It is important to be aware that the nodes send the map to the parents after **verifying that *every* signature of the map is valid**. If a signature is invalid, it is removed from the list and not send to the parent. When the leader is reached in the conode tree and that the leader has done the operations described in this section, it passes to the next phase: Request Missing Data.

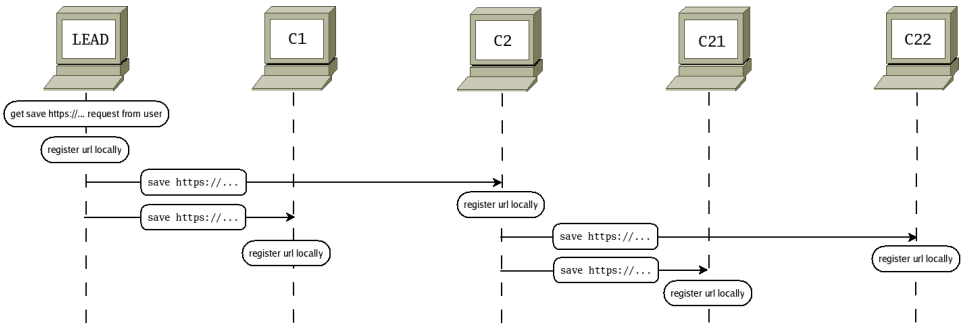

Figure 18: Consensus Step 1

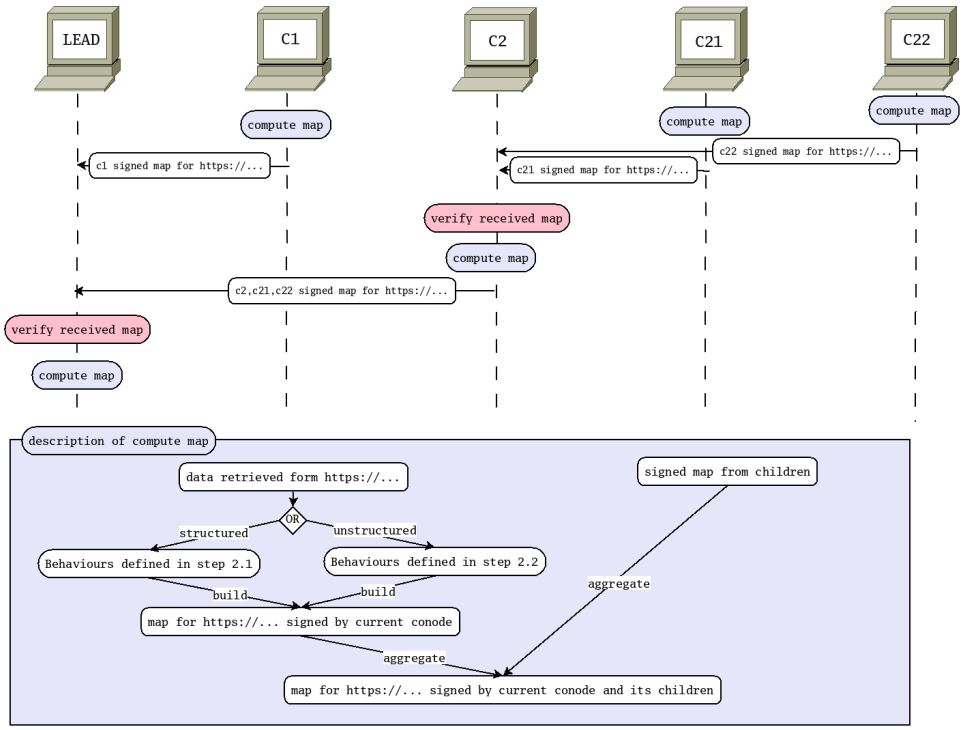

Figure 19: Consensus Step 2

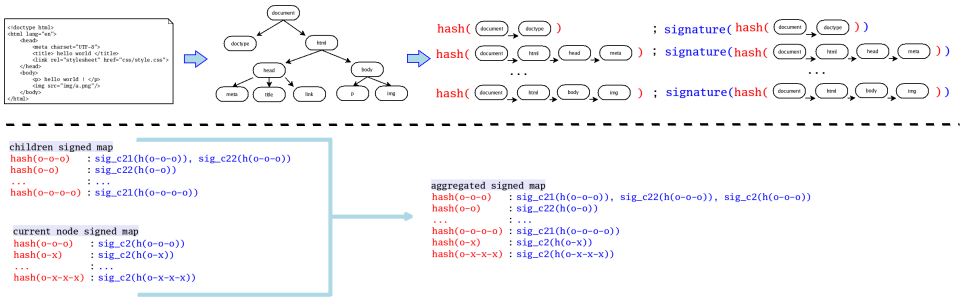

Figure 20: Consensus Step 2.1

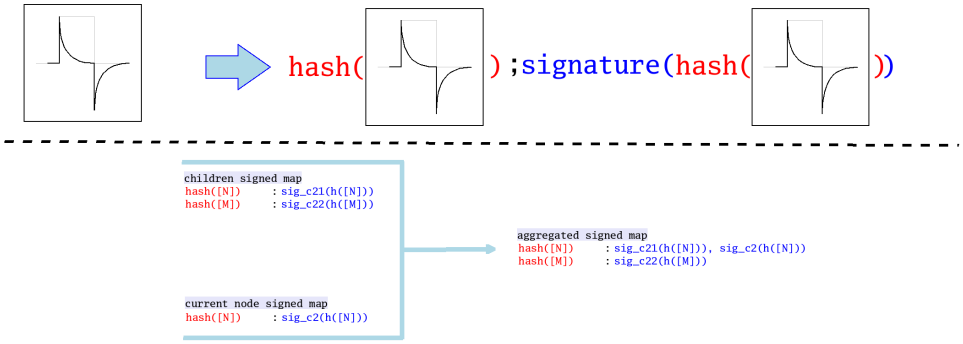

Figure 21: Consensus Step 2.2

Request Missing Data	
Steps	Explanation
Preliminaries	At this phase, the leader has a map where the keys are the hashes of all the data that has been seen by a sufficient number of conodes and the values are the list of the signatures of these hashes. For structured data, the hashes are the hash of the paths of a consensus tree. For unstructured data, the hashes are the hash of an images and only one of them should be valid. From now, will will refer to it as the *consensus map*. For structured data, the Request Missing Data phase is necessary in case the leader does not have a certain path but a sufficient number of nodes have it. The leader will request the plaintext data for those paths. If the leader does not require any paths form its children, this phase is skipped. For unstructured data, the Request Missing Data phase is necessary if the unstructured data chosen is not the one that the leader has. The phase is skipped if the leader has the correct corresponding unstructured data or if no hash has a sufficiently high number of signature. In the latter case, the unstructured data is dropped.

For Structured Data	1. The leader send all the entry of the consensus map for witch it has not the plaintext path already. 2. Upon receiving the part of the consensus map, a receiving conode check each entry signatures and then choose between two options after checking that the path was indeed signed the correct number of time: (a) The node has the plaintext information for the path. Then it removes it from the missing paths list and store the plaintext associated with the hash of the path. (b) b. The node does not have the plaintext information. It keeps the entry in the missing paths. 3. The conode send the missing paths left to its children. 4. The process continue until the leaves are reached. 5. The leaves send to its parents the plaintext paths it has. 6. The parents aggregate the paintext paths of their children with theirs after having checked that the plaintext path correspond to the hash. 7. The leader now has all the paths in plaintext. It verifies that the plaintext paths correspond to the hash of the missing paths. 8. The leader then construct the consensus tree from the plaintext paths. From that tree, it builds the original structured page (for example html code). The Figure 22 explains those steps.

For Unstructured Data	• The leader check if it has the plaintext data of the hash that were signed by a sufficiently high number of conodes. If it has it, the phase stop. Else, it send the hash and the signatures to its children. • The receiving conode check the signatures. If all are valid and the number is sufficiently large, it checks if it has the plaintext data. If it has it, it sends the plaintext data to its parent. Else, it sends it to its children. • The protcol continue until the leaves are reached and all nodes reply to its parent. • Finally, the leader check that the plaintext data correspond to the hash.
End of phase	At the end of this phase, the leader has the plaintext data seen by a sufficiently large number of conodes. So it ends the phase and go into the Cosigning phase.

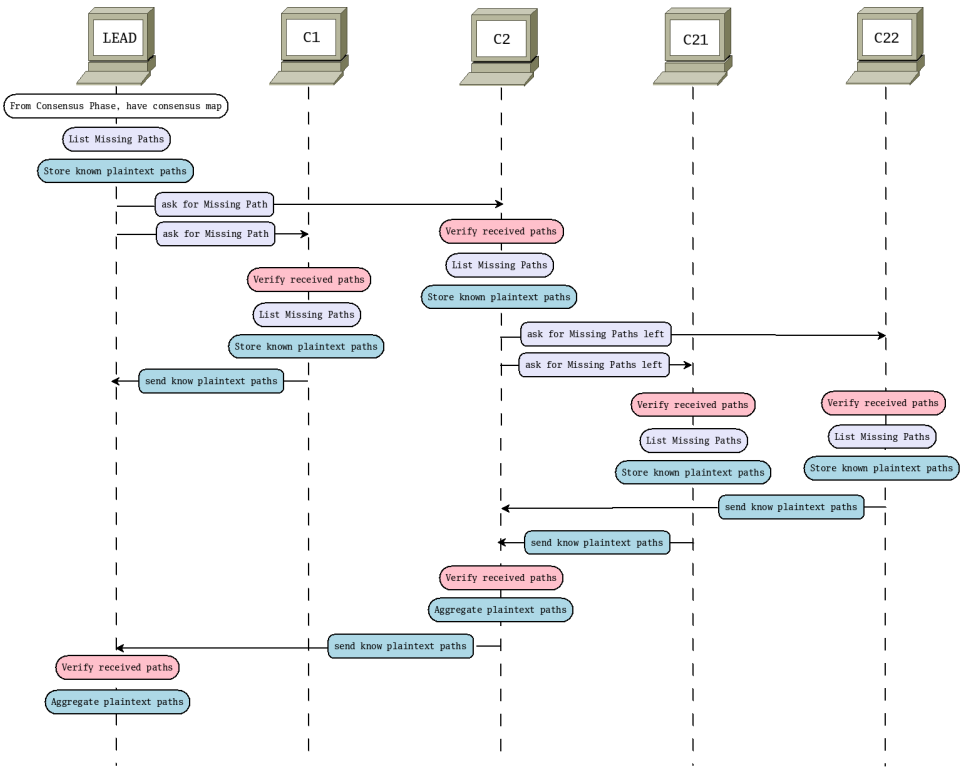

Figure 22: Request Missing Data Steps for structured data

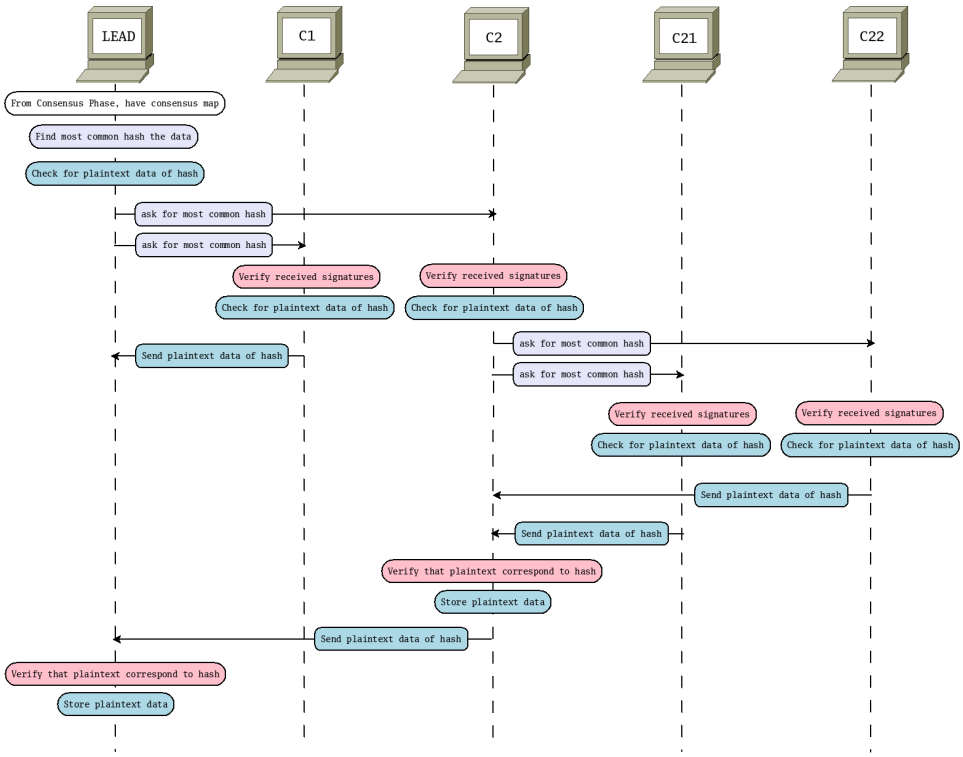

Figure 23: Request Missing Data Steps for unstructured data

Cosinging	
Steps Cosinging	Explanation The leader request a cosignature from all the servers on the html,css page or the byte of the image. This phase use the CoSi API provided by the Cothority framework**??**.

Additional Ressources	
Steps Additional Ressources	Explanation In case of an html page, it is possible that the page require additional ressources like a css file or an image in order to be correctly displayed. We then launch the consensus phase for all the possible additional paths. Mainly we do: 1. The leader parse the signed html page and output the list of additional ressources. 2. The leader restart the whole saving a webpage protocol 3. The leader store the informations retrieved in a table. An entry is of the form: (data, co-signature on the data).

Skipchain saving	
Steps	Explanation
Skipchain saving	The leader now has all the pages and the signatures required to save the webpage on the skipchain. So it sends a request to the skipchain conodes in order for them to create a block with the new data and signatures. To do that, it uses the skipchain API of the cothority framework.

Retrieving a webpage is identical to the one presented in Section 3.

7.2 Plain text Data Used In The Simulations

For the Figure 11 and Figure 12 , we use the following plaintext data (the time being given in miliseconds):

```
numHtmlNodes   reqS cosi adds skip
50   34.1  6.6 17.0  1.6
320   45.3  4.4 18.2  2.0
1520   87.8  4.0 18.1  1.8
3020  153.8  3.5 17.5  3.0
```

```
6020  366.7  3.7 17.5  2.1
15020 1558.6  5.1 22.7  5.1
30020 5497.8  6.5 26.0  7.1
```

On the following website (in order):

1. `http://nibelung.ch/decenarch/10p.html`
2. `http://nibelung.ch/decenarch/100p.html`
3. `http://nibelung.ch/decenarch/500p.html`
4. `http://nibelung.ch/decenarch/1000p.html`
5. `http://nibelung.ch/decenarch/2000p.html`
6. `http://nibelung.ch/decenarch/5000p.html`
7. `http://nibelung.ch/decenarch/10000p.html`

For the Figure 13 , we use the following plaintext data (the time being given in nanoseconds):

```
web,numConodes,numHtmlNodes,start,reqS,cosi,adds,skip

http://www.bbc.com/travel/story/20180102-how-sausage-flavours-the
-german-language
,5,1856,0,1350700000,11900000,1193500000,11700000

http://decenarch.nibelung.ch,5,36,0,88800000,9300000,192500000,3800000

http://20min.ch/ro,5,7428,0,1232100000,12100000,27306200000,6630000

http://google.ch,5,148,0,420900000,15600000,600000,4700000

http://blog.golang.org,5,1300,0,911400000,11200000,36820000000,64000000

http://www.bbc.com/news/world-us-canada-42680070,5,2858,0,1829500000,9100000,
2804900000,23700000

http://www.20min.ch/ro/news/monde/story/Un-avion-manque-de-peu-de-tomber
-dans-la-mer-16596126,5,2173,0,597500000,5300000,7808600000,34200000

http://www.cbc.ca/radio/quirks/how-an-oddly-large-beetle-penis-is-
inspiring-new-medical-technology-and-more-1.4470178/this-food-additive
-is-hard-to-avoid-and-could-make-hospital-superbugs-more
-deadly-1.4470200,5,1188,0,404500000,8300000,5190700000,26300000
```

https://www.lenouvelliste.ch/articles/lifestyle/sortir/melanie-pitteloud-amene-un-nouveau-regard-poetique-sur-le-rhone-728845,
5,2313,0,672300000,7000000,21062900000,40500000

http://tass.com/politics/984968,5,2428,0,1584300000,5100000,11002100000,33900000

http://tass.com/science/984384,5,2427,0,1808500000,4900000,11507300000,30700000

For the Figure 14 and Figure 15 , we use the following data (time in miliseconds):

numConodes	reqS	cosi	adds	skip
3	1317.1	5.7	69.4	4.8
5	1871.9	4.7	68.5	3.7
10	3475.5	7.8	89.0	4.6
25	8329.1	29.6	141.4	4.0
40	13003.5	44.9	143.5	3.4
50	20989.0	61.8	315.3	5.4

For the web page http://nibelung.ch/decenarch/5000p.html.

References

[1] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities" honest or bust" with decentralized witness cosigning," in *Security and Privacy (SP), 2016 IEEE Symposium on*, pp. 526–545, Ieee, 2016.

[2] "Chrome market share 2016-2017 statistics." `https://www.netmarketshare.com/browser-market-share.aspx`. Accessed: 05 December 2017.

[3] "Google market share 2016-2017 statistics." `https://www.netmarketshare.com/search-engine-market-share.aspx`. Accessed: 05 December 2017.

[4] "Mirai Attack Analysis." `https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet`. Accessed: 05 December 2017.

[5] "Right To Be Forgotten." `https://en.wikipedia.org/wiki/Right_to_be_forgotten`. Accessed: 05 December 2017.

[6] "Effect of de-indexing statistics." `https://searchengineland.com/60-direct-traffic-actually-seo-195415`. Accessed: 05 December 2017.

[7] "On the Data selling." `https://www.cbsnews.com/news/the-data-brokers-selling-your-personal-information/`. Accessed: 05 December 2017.

[8] D. l. EPFL, "Cothority." `https://github.com/dedis/cothority/wiki`.

[9] K. Tamas, "ZeroNet." `https://zeronet.io/`. Accessed: 05 December 2017.

[10] D. l. EPFL, "Cothority ONet." `https://github.com/dedis/cothority_template/wiki/ONet-principles`.

[11] D. l. EPFL, "Cothority CoSi." `https://github.com/dedis/cothority/wiki/CoSi`.

[12] D. l. EPFL, "Cothority Skipchain." `https://github.com/dedis/cothority/tree/master/skipchain`.

[13] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford, "CHAINIAC: Proactive software-update transparency via collectively signed skipchains and verified

builds," in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 1271–1287, USENIX Association, 2017.

[14] "Archive.org wayback machine." `https://archive.org/web/`. Accessed: 10 January 2017.

[15] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, pp. 85–103, Springer, 1972.

[16] "Decentralized Archive github devlopement repository." https://github.com/nblp/decenarch.