TÉCNICO LISBOA

DClaims: A Censorship-Resistant Web Annotations System

João Ricardo Marques dos Santos

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Nuno Miguel Carvalho Santos
Eng. David Miguel dos Santos Dias

May 2018

Acknowledgments

I am deeply grateful for the support my family has provided me, over all these years. This work is the culmination of all the effort they placed in my education. I would also like to leave a word of appreciation for the support provided by my advisors. Professor Nuno's diligence and clear-thinking were critical in organising the ideas that led to the work developed over this last year. David's contribution was equally important. His motivation and vision for this new decentralised world were vital in motivating me to pursue this theme and becoming part of a broader community.

Abstract

Unreliable and misleading information on the web can have a severe impact on democracies. Web annotations are a way to add information to web pages. They can be used to provide context, clarification or fact check any website, and, for that reason, can be a useful tool for improving the quality of websites, and reduce the spread of misinformation. Unfortunately, the current web annotations services have a centralised architecture, which makes them vulnerable to censorship attempts. A motivated censor that can exert control over an annotation service will be able to delete, tamper with, or prevent new content from being published. To solve the problem of censorship in web annotations services, we present DClaims, a decentralised web annotations system. DClaims' decentralised architecture relies on two main building blocks, the Inter-Planetary File System (IPFS) and the Ethereum blockchain, both of which offer desirable censorship resistant properties. We present a DClaims reference implementation in the form of a web annotations browser extension, called DClaims-News, which allows for the classification of news articles, on news websites. We test the system's performance and carry out a cost analysis. From this study, we conclude that a large scale implementation of the system is possible and that the cost of operation is close to 20% of the cost of operating a centralised platform, vulnerable to censorship.

Keywords

Web Annotations, Internet Censorship, Ethereum, IPFS, Decentralisation

Resumo

Informação pouco fiável e enganosa na Web pode ter um sério impacto nas democracias. Web Annotations (anotações na Web) são uma forma de acrescentar informação a páginas Web. Podem ser utilizadas para adicionar contexto e clarificar qualquer website, como tal, têm potencial para melhorar a qualidade de todos os websites e reduzir a disseminação de informações incorrectas. Infelizmente, os serviços existentes de anotações na Web têm uma arquitectura centralizada, o que os torna vulneráveis a tentativas de censura. Um censor motivado e que possa exercer controlo sobre um serviço de anotações, terá a possibilidade de apagar e adulterar anotações, ou impedir que novas anotações sejam publicadas. Com o objectivo de resolver o problema de censura em anotações na Web, desenvolvemos o DClaims, um sistema descentralizado de anotações Web. A arquitectura descentralizada do sistema são dadas pelas principais duas tecnologias na qual está assente, o Inter-Planetary File System (IPFS) e na blockchain Ethereum – dois sistemas que oferecem variadas propriedades de resistência a censura. Apresentamos uma implementação de referência para o sistema DClaims, sob a forma de uma extensão de navegador de Internet, chamada DClaims-News, que possibilita a classificação de artigos de notícias em páginas Web. Testamos o desempenho do sistema e fazemos uma análise de custos. Concluímos que uma implementação em larga escala do sistema é possível e que o custo de operação é cerca de 20% do custo de operar uma plataforma centralizada e vulnerável a censura.

Palavras Chave

Anotações Web, Censura na Internet, Ethereum, IPFS, Descentralização

Contents

List of Figures

List of Tables

1

Introduction

Contents

Over the last two decades, the web has become the primary source of information for humans living in developed countries. On the other hand, this shift has been accompanied by the dissemination of unreliable or even misleading information throughout the web. Over time, commentary sections on websites and social networks have become popular platforms for people to exert their freedom of speech, by correcting inaccurate information, providing additional context or engaging in discussions. However, existing web platforms tend to be controlled by single authorities which, if the shared information is considered to be inconvenient to their interests, can engage in censorship by denying others users access to such content. This work aims to provide a fully distributed a censorship-resistant platform which people can use to comment on information available on the web.

1.1 Motivation

The web plays a critical role in informing modern democracies. A study [2] by the Reuters Institute for the Study of Journalism at the University of Oxford, conducted in 2016, reveals that the Internet makes 53% to 84% of the primary sources of information used by UK citizens ages 18-44. Studies conducted in the United States reached similar conclusions. [3–5]. Unfortunately, we have become more aware of often lack of credibility, falsehood and incompleteness of that very same information – which can have serious consequences. Democracies are fed by public opinion which in turn is shaped by the information individuals receive. Campaigns of disinformation have the power to start wars [6], influence government elections [7], endanger human lives [8] and even jeopardise the future of the human species [9]. Therefore, it is essential to allow people to have access to reliable sources of information – to the point of the identification and classification of low quality information having become one of the most active areas of research [10–16].

There have been several attempts to improve the reliability of information on the Web. Fact-checking platforms and social networks are two. Fact-checking organisations [17], such as PolitiFact[1],WikiTribune[2] and FactCheck.Org[3] constantly analyse news articles and issue classifications about the truthfulness of statements. People have also used social-networks and commentary sections on news websites to correct inaccurate information, provide additional context or engage in discourse [18, 19]. The problem with these platforms is that they too can be censored. Media outlets control the commentary section on their websites, meaning that they can selectively remove comments, and governments can exert pressure over social network providers. There are multiple instances of state censorship of information. China is one of the high profile examples. It has legislation in place which financially incentivises organisations to self-censor [20]. This results in user-generated content being constantly deleted from online forums and

[1] http://www.politifact.com/
[2] https://www.wikitribune.com/
[3] http://factcheck.org/

social networks. There is evidence of efforts by governments of at least 13 countries (including China, Egypt, Turkey, United Kingdom, Russia and France) to either block access to these platforms or censor user-generated content [21, 22]. Google, Facebook and Twitter publish annual reports of the number of government requests of remove content [4] [5] [6].

Besides being vulnerable to censorship, the information created by these platforms is scattered across multiple profile pages and websites - for users to get the complete set of social-network publications about a news article on a news website, they need to search for it on all the profiles that publication has been shared on. An alternative way to achieve the goal of informing would be to display the information directly on the source webpage. Web annotations[7] is a new way of interacting with information on the web, which allows for users to make and share annotations of web pages, just like they would annotate a physical notepad. It enables end-users to highlight text, create sticky notes or comment specific parts of a web page. A typical usage scenario consists of a user visiting a news webpage article which showed portions of the text highlighted by her friends (or anyone chosen by her) and when she places her mouse over the highlighted portions she sees comments made by the users about the highlighted text. What web annotations allow is for the creation of a new layer of data, on top of the existing websites, without changing the original resources. This layer can be used to provide context, clarification, additional information about the resource a person is viewing.

Currently, there are several web annotation services available to the general public [23], but they have a centralised infrastructure. The most popular service is Genius Web Annotator[8] which allows for the annotation of any webpage and offers a complete feature set. However, Genius and other similar platforms are proprietary and closed, meaning that they are not compatible with each other. An annotation made by a user using service A cannot be used by service B. We call these services silos because the information they hold and services they offer are only useful and relevant to their ecosystem. A siloed architecture is characterised by low interoperability, low control of the data by the users. Other platforms such as Hypothes.is improve in the interoperability aspect, by using a standard type of web annotation data model, but still, have total control over the data and offer no guarantees of permanent storage. Consequently, these services are vulnerable to the same type of censorship as the one present in social networks.

[4] Google Transparency Report: https://transparencyreport.google.com/government-removals/overview
[5] Facebook Transparency Report: https://web.archive.org/web/20180501211320/https://transparency.facebook.com/government/
[6] Twitter Transparency Report: https://web.archive.org/web/20180501211317/https://transparency.twitter.com/en/gov-tos-reports.html
[7] https://web.archive.org/web/20180502111019/https://www.w3.org/annotation/
[8] https://genius.com/web-annotator

1.2 Goals And Requirements

The goal of our work is to build a system that allows for uncensored access to web annotations on the Internet by eliminating central points of control where a powerful actor can exert pressure to fabricate, modify, or suppress exchanged messages. Examples of such actors include governments or powerful media organisations.

In addition, our system must satisfy the following requirements:

- **Authenticity and Integrity Assurances:** Ensure that the end-users know who created the web annotations they are reading and that they have not been tampered with.
- **Data Permanence and Portability:** Links to the web annotations should not get broken if they change location. This means that the links should remain the same, independent of the web server where content is stored.
- **Financial Cost Efficiency:** The infrastructure cost of operating such a system will be supported by several voluntary institutions, and should not be higher on one of current platforms which provide similar services.
- **Scalability:** The system should handle workloads similar to the ones on Facebook's most active news pages
- **Compatibility with standards:** The data model used should be compatible with any current standards. This is important to ensure interoperability between applications.

1.3 Contributions

This thesis presents a novel system for the censorship-resistant exchange of web annotations over the Internet, called DClaims. DClaims has a decentralised architecture, which is based upon two building blocks: the Ethereum blockchain[9] and the Inter-Planetary File System(IPFS)[10] . To marry the feature set of web annotations, with the integrity and censorship resistance assurances of IPFS and the ordered registry and freshness of Ethereum, our system stores web annotations on IPFS and records the IPFS links of these files on Ethereum.

Here is an example of how the platform works: A user, Alice, visits the Fox News website and opens an article about global warming. She notices that the chart that plots the rise in temperature in the last 50 years contains inaccurate information. Then, she makes an annotation on the website, detailing the inaccuracies in the chart. That web annotation gets stored on IPFS, and a registry entry is added to the

[9] https://www.ethereum.org/
[10] https://ipfs.io/

Ethereum blockchain. Later that day, Bob visits the same news article and sees Alice's annotation. Bob is assured that the annotation was made by Alice and that it has not been tampered with. Bob is also assured that no annotation is being withheld from him. Now Bob has all the information he needs to form an opinion about the issue being discussed in the article.

We use the Ethereum blockchain to keep a permanent, canonical record of all the annotations made. This blockchain is entirely decentralised and cannot be controlled by any government authority or media outlet. The records that are assured to be fresh and ordered. These properties guarantee that every time a user queries the system, he is receiving the latest information, unfiltered and uncensored.

The data is stored on IPFS, a distributed file system where data is stored and scattered all across the web. IPFS has been used as a tool against state censorship by granting access to Wikipedia to Turkish citizens, following the government's blockage of the website [24]. Besides decentralisation, IPFS offers strong integrity assurances of the files it stores. That happens because the link to an IPFS file depends on the content of the file rather than the file's location. Typical HTTP links point to a server on the web that is hosting a given file. The hash of the file partially forms an IPFS link it is addressing. That means that if a file gets tampered with, the link changes. An IPFS link will always point to the same file.

Developing a platform with a blockchain based architecture comes with its own set of challenges. The Ethereum blockchain works as a giant computer, a single machine being run by a community of nodes scattered all across the globe. Every node runs all the operations and maintains a copy of the state of the chain. Usually, systems which use a blockchain are slow and expensive. Designing this system meant developing a scalable and cost-effective blockchain architecture.

In summary, the contributions of this thesis are:

- The development of the DClaims Protocol, which defines how web annotations are generated, transported and stored.
- Implementation of the DClaims platform, which implements the DClaims Protocol and can run on any modern web browser or web server.
- Implementation of a web annotations application, called DClaims-News, which allows for the classification of news articles, on news websites.
- An experimental evaluation of the DClaims and DClaims-News platforms, comparing its performance with other widely adopted platforms.

1.4 Document Roadmap

The rest of this document is structured as follows: Chapter 2 is the related work where we overview Web Annotations, Ethereum and IPFS. In Chapter 3, we present the architecture of the DClaims system.

Next, in Chapter 4, we offer our implementation, followed in Chapter 5, by the evaluation of the system. Finally in Chapter 6, we present the conclusion and discussion.

2

Related Work

Contents

In this chapter, we present the related work relevant in DClaims' scope. We start, in Section 2.1, by analysing the state of art of censorship-resistant systems to motivate the need for a new one. Then, in Section 2.2, we present the current systems for Web Annotations and explain how they can be a useful tool for fact-checking information on the web. In Section 2.4, we introduce a building block of our solution, Ethereum, followed by the second building block, IPFS in Section 2.5. Finally, in Section 2.6 we present a summary of this chapter.

2.1 State Of The Art On Censorship-Resistant Information Sharing

The desire of access to information on the internet has led to the appearance of several censorship resistance systems. Khattak et. al [25] characterise the censorship resistance panorama as an arms race. Censors block information, so new systems appear with capabilities to circumvent the current censorship techniques, which in turn leads censors to develop new censorship capabilities. They offer a classification of censorship systems, dividing them into two main categories, communication establishment and conversation. Each censorship resistance system focus on different properties, that is, it employes mechanisms to circumvent the type of censorship that is more relevant in the context they operate. We present an overview of the two categories to situate DClaims' focus.

The most common censorship circumvention techniques applied in the communication establishment phase are making many changes to the initial stage of the protocol (Flashproxy [26]), limit the number of credentials which provide access to the system (Defiance [27], Tor Bridges [28]), obfuscation of the communication(SilentKnock [29],ScrambleSuit [30]), and allowing access only to known peers (Proximax [31]). The communication establishment phase is not important in the context of DClaims. We assume that the users can establish communication with the servers of the network, with IPFS and with Ethereum.

In the case of circumventing censorship during the conversation phase, three main techniques are used. Content and communications flow obfuscation (by tunneling [32, 33] or covert channels [34]) and destination obfuscation (Tor [35]). These three techniques are used to provide end-users with access to other systems. Tunneling and covert channels consist of creating a secure and private communication channel with a service, which is not the problem DClaims is trying to solve.

There is a different class of protocols which have the goal of providing access to specific documents and publications. This is achieved by content replication and redundancy and distributed storage. The goal of providing access to specific online resources which are controlled by a central authority is precisely the scope of DClaims. We will now review some of these systems and discuss their shortcomings.

Freenet [36], Tangler [37], Publius [38], and Serjantov [39] are examples of systems designed to circumvent censorship of documents. They achieve this by replicating content across the network. In

Freenet, files are split into chunks and spread out through several servers on the network. The network is made of volunteered servers, each server offering some amount of storage space. Tangler operates in a similar way. The key difference between Tangler and Freenet is that Tangler makes it harder for servers to delete content deliberately. This is because each file that is added to the network contains blocks of other files already hosted. The result is that is a server deletes a certain file, he will be affecting many other files, making it easy for the network to detect the attack. Publius also spreads files through a network of servers, but since its main goal is anonymity, there is no way to verify the authenticity of the files.

The three methods described in the previous section – for resisting censorship in the conversation phase – focus on providing end-users with access to a specific service or website, for instance, Tor allows users to connect to websites that may be blocked in their geographical area. This other type of services has the goal of providing access to specific documents and publications. This is achieved by content replication, redundancy and distributed storage. Freenet [36], Tangler [37], Publius [38], and Serjantov [39], are examples of such systems. Tangler and Freenet have a special focus on making the content really hard to delete from the network, while Publius and Serjantov focus on privacy.

These systems could be used to store and share web annotations, but they rely on a small number of server, which can be compromised. Since each system supports the storage and sharing of files, those files could be web annotations. Adapters could be built to interface the system with current web annotations services. However, a significant amount of trust is placed on a small number of servers. All these systems depend on the existence of a network of trusted servers to work, all the content is stored on those servers, and they are the ones who serve the content to end users. Would that small network of servers to be compromised (by internal or external attacks), all the integrity and freshness assurances of the system would seize to exist. These systems need a way to overcome the trust dependency on a small number of servers, and, as we explain in Section 2.4, blockchain technology is one of the ways to attain such a trustless architecture.

The DClaims architecture places the trust on a blockchain, which offers much stronger guarantees. DClaims also makes use of a small network of servers, but these exist only to increase the scalability and availability of the system.

2.2 An Overview of Web Annotation Services

Unlike the case with social networks and fact-checking websites, with web annotations the additional layer of information that is created (comments, classification of statements) is made available directly on the original site, so a user is automatically exposed to all the available information about the information content she is consuming. A user can subscribe to specific individuals or organisations; this prevents the

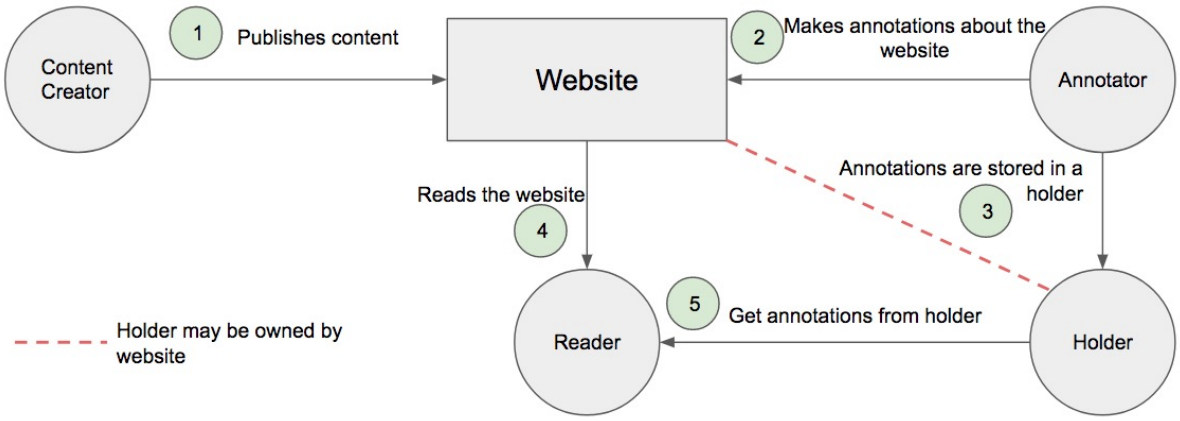


Figure 2.1: Basic Model Of A Web Annotation Service.

problem of commentary saturation that happens in social networks. Just as if Fact Checking websites, annotations platforms are independent of the site of the original content.

In this section, we present the current status of web annotations systems and platforms. We start by offering the general model through which annotations are made. Next, in Section 2.2.2 we present the most common model for web annotations services architecture, which is based on information silos, and discuss the inherent problems with this design. Finally, in Section 2.3 we present some relevant alternatives to the siloed architecture that have been proposed to overcome this limitation.

2.2.1 The General Model of Web Annotations Services

Figure 2.1 illustrates the workflow of a web annotation service in its simplest form. Essentially, this diagram represents the workflow through which an *Annotator* created an annotation about a *webpage* published by a *content creator*. When a *reader*, which is just a normal internet user, visits the website, the web annotations are displayed to him in the browser's UI. The web annotations are stored by a *holder* which may or may not be controlled by the website.

To better clarify this model and understand the role of each participant, consider a simple example. Alice is a blogger who just published an article on Medium (a blog hosting website). Bob visited the website and found a paragraph of the blog post particularly interesting, he highlighted that portion of the text and used the website's highlight feature. Later that day, Charlie, visits Medium and reads Alice's blog post. The paragraph that Bob highlighted is displayed highlighted to Charlie. In this scenario Alice is the *content creator*, Bob is the *annotator*, Charlie is the *reader*. Medium is the *website* and the *holder* (since the information about the highlights is kept under their control), and the highlight is the *annotation*.

A – Ideal Properties of Web Annotation Services Ideally, web annotations should have a set of desired properties, which are enforced differently depending on the type of annotation service implementation. The main of such properties are the following:

- **Revocability:** Revoking an annotation consists of invalidating a previously published annotation. The revocation mechanism lets other users in the network know that that annotation should no longer be considered valid.
- **Verifiability:** There should be a well-defined mechanism for annotations to be validated. That is, for any entity to be able to check the authenticity, integrity and correctness (non-revoked) of any given annotation.
- **Self-sufficiency:** Ideally, none of the steps of annotation (creation, verification, storage) should be entirely dependent on a third party.
- **Compatibility:** Annotations should obey a certain standardised schema, to be able to link to each other. Even those created by different entities.
- **Portability:** Entities should be able to exchange annotations with each other and change the location where they are stored.
- **Permanence:** Once created, an annotation should not disappear (without the consent of its creator) and should be retrievable at all times.
- **Censorship Resistance:** No one party should be able to restrict access to annotations or modify existing annotations.

2.2.2 Siloed Web Annotations Services

The web annotation service model presented above can be instantiated in different architectures. Now we present the architecture most commonly adopted in real-world systems: the *centralised siloed architecture*. To do that, we analyse some of the most successful web annotation platforms, Medium and Genius. Both these services allow the annotation of web pages, and, in both cases, they control the data. In relation to the annotation model presented in Figure 2.1, the *content creator* is a user, who creates a new webpage (or makes a blog post). The place where the creator puts the content is the *website*. The *holder* is Medium or Genius since they store the data. Regular website users, are the *annotators* and *readers*. We list their positive features and then enumerate the desirable properties that are lacking. This latter point will serve as motivation for establishing the need for moving towards a decentralised non-siloed architecture.

A – Medium Launched in 2012, Medium[1] is an online platform for publishing blog posts. Anyone can create blog posts and share them with the community. The website has many features similar to social networks like Facebook, such as the possibility to comment or clap (like) on the articles but what sets it apart is the highlighting, tagging and note-taking features. As described in Alice and Bob's example, a user can highlight portions of the blog post which are then visible to other users that *follow* the user who highlighted the content. The highlights are also shown to the original creator of the article, providing a fine-grained level of feedback. The tagging feature is another shift from the old blogging platforms such as Blogger[2] as it allows users to tag posts by theme and content rather than by author. Finally, the note-taking feature allows users to associate written comments to their highlights. These comments can remain private or be shared with the entire community. In summary, Medium provides three forms of web annotations, highlights, tagging and note taking (or comments), in their platform.

B – Genius Web Annotator Genius started out as a digital media company with a focus on music. It allowed for users to interact directly with artists through annotation and commenting on musical lyrics. The platform has evolved a lot since its genesis. Now it enables users to generate annotations about any website, which can happen in two ways. The first is when the owner of the site wants to embed genius web annotations on the website and to do that he merely needs to add a javascript line to the website's code or install a Wordpress[3] plugin. The second way allows for users to use Genius web annotator even in sites which do not have the plugin embedded. To that end, a user only needs to add the prefix `genius.it/` do the website's URL, which will reload the site on Genius' platform and allow the user to annotate and view annotations made on that website. Genius' web annotator is feature complete, allowing for annotations via highlights, comments and images. The service also offers a reputation counter, called *IQ*, which results from voting on a user's annotations.

C – Main Weaknesses of These Platforms So far in this section, we have presented a few popular siloed web annotations systems, namely Medium and Genius Web Annotator. However, despite their benefits, these platforms still fall short of attaining all the desirable web annotations properties, listed in Section 2.2.1, more precisely:

- **Lack of interoperability:** Annotations done on one platform should work on another platform. In Medium's case, this is not an issue since their annotations are only relevant inside the publications hosted by them. However, in the case of Genius and other similar services that allow for the annotation of any website, one would hope that the annotations made in one platform could be moved to another, which is not the case. This is one of the leading problems with siloed architecture

[1] https://medium.com/
[2] https://www.blogger.com
[3] https://wordpress.com

– which is that data generated inside these services are only valuable *inside* that ecosystem, a mechanism purposely designed to make it harder for users to leave the service.

- **No data permanence assurances:** The annotations produced by the users are stored, managed and maintained by the service providers. These offer no guarantees that the data will be preserved. They own it; they control it, they can delete it.
- **Vulnerable to censorship:** This is closely related to the previous point. Since its owners of the service who control the way data is stored and shared, they can choose to dismiss annotations that can be damaging to them. Genius', for instance, has special users with the roles of curators and moderators who can delete other users' annotations.

Systems such as Medium or Genius offer very little control to their users. It is true that the user has some control over the visibility of his annotations, but the platform has the last say in the sense that the platform operators can delete a user's entire history. Moreover, all the information contained inside these silos has no meaning in any platform other than the one it was created at, so a user has no choice to transfer the data from one platform to another due to the lack of compatibility between these platforms. Siloed systems are characterised by having total control over the information they hold and not making that information compatible with any other platform, nor offering guarantees of permanence and future compatibility. Next, we are going to introduce relevant platforms that aim at overcoming the limitations of siloed architectures.

2.3 Towards Breaking Down Siloed Web Annotations Platforms

In this section we introduce a new standard for Web Annotation, proposed by the W3C Web Annotation Working Group[4] and one service, Hypothes.is[5] which implements it. This standard and service aimed at taking the best features of the systems described in Section 2.2.2, while overcoming some of their shortcomings. We will start by introducing the Web Annotation standard, followed by an overview of the Hypothes.is service. We conclude that while Hypothes.is is an improvement over siloed systems in the sense that the web annotations it uses are standardised and can interact with others, it still has the problem of offering no assurances of data permanence and is vulnerable to censorship.

2.3.1 W3C's Web Annotation Recommendation

The World Wide Web Consortium (W3C) is an international community which focuses on developing Web standards. In 2014, the W3C Web Annotation Working Group was created with the goal of produc-

[4] https://www.w3.org/annotation/
[5] https://web.hypothes.is/

ing a specifications recommendation (The intent of this recommendation is for it to be adopted by the community, thus becoming a standard) for a distributed, interoperable and shareable Web Annotation architecture. The group finished its job in 2017, having produced the Web Annotation Data Model and Protocol. The data model describes the annotation data model abstraction, and the protocol defines the transport mechanisms for the creation and management of the annotations based on REST best practices.

A – Data Model A Web Annotation is a JSON-LD[6] file, which allows for files to reference other files and in its most basic model is comprised of a `body` and a `target`. When Bob makes a comment (an annotation) about Alice's blog post, the comment is the target, and the blog post is the body. The standardisation of the data model is an essential step to allow for the annotations to be compatible across platforms. The shortcoming of this data model is that it does not deal with authentication and integrity. All the verification checks are to be carried out by the protocol, between the users and the servers, never dealing with the objects themselves, which means that the data model is not self-verifiable.

B – Transport Protocol The Web Annotations protocol defines the transport mechanisms for annotations that follow the Web Annotation data model. The protocol is based on HTTP and REST. It defines that Web Annotations are to be stored in `containers` which expose a standard API to be used by services. Relating to Figure 2.1, the container is the *holder*. These containers are essentially servers managed by some entity. There can be many containers, managed by different entities. Revisiting Alice and Bob's example, when Bob is making an annotation on Alice's Medium article, the annotation could be stored in a container which is owned by an entity completely independent from Medium. This is a step in the right direction towards decentralisation. For instance, it allows for people to organise in communities, each running their containers to keep annotations that are relevant to them. However, as we will see later, this architecture does not provide strong enough permanence, integrity or authenticity assurances, as there is still a need to trust in the party who maintains the container.

2.3.1.A Hypothes.is

Motivated by usefulness and positive impact that web annotations can have on the world, Hypothese.is is a non-profit organisation building an open source platform which implements W3C's Web Annotation recommendation. As illustrated in Figure 2.2, Hypothes.is inherits the properties of the standard's data model and transport protocol. The features offered by their platform are comparable to the ones of Genius' (users can annotate any webpage and share it with anyone) with the advantage of interoperability, meaning that the annotations produced by Hypothese.is' platform will be compatible with

[6] JSON for Linked Data

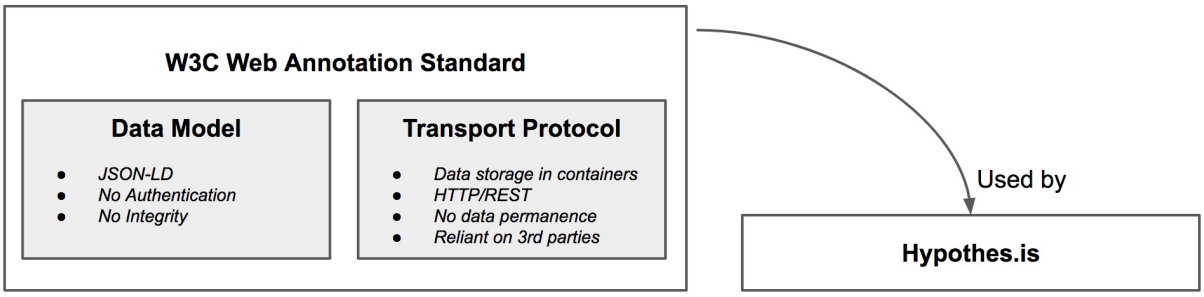


Figure 2.2: The Relation Between Hypothes.is And W3c's Web Annotation Standard.

the ones produced by any other platform that implements the same standard. Furthermore, they have developed specific software for use cases in education, journalism, publishing and research. However, the annotations are being stored by Hypothes.is containers (servers run and controlled by Hypothes.is) and users still need to trust the organisation to keep their best interests in mind.

2.3.1.B Discussion

Web Annotations are a handy tool for the web. They allow for the creation of a layer on top of the existing websites, enriching their content and improving the way information about them is shared, all of this without the need to change the original resources. Table 2.1 is a comparison between the previously presented services. Platforms such as Genius are feature rich, offering great functionality for their users, but a cost of no interoperability. All the value generated by the millions of Genius users is locked inside that silo. Open source platforms such as Hypothes.is offer similar services with the bonus of using a standard data model for their annotations, which allows for users to move their annotations between different services that use the same standards. The standard data model for Web Annotation is the one produced by W3C which has interoperability and decentralisation in mind. Unfortunately, as of yet, there is no real fully decentralised implementation of web annotations, as all the existent services use their services to store the data. Furthermore, users have to trust the service to not withhold data from them or tamper with the data.

In the context of this work, censorship should be defined as denying access to information stored on web annotations services by deletion or content withholding; Tampering with information stored on web annotations services by supplying modified versions of the data (created by an original user) to the end users; Preventing new information from being stored on web annotations services without informing the user trying to add the new content. All this, with or without the cooperation of the content provider and without the cooperation of the end user and the applications running on his machine.

We argue that to make web annotations resilient, two objectives need to be achieved. First, the annotations need to be provided with integrity assurances and, second, the whole architecture needs to

Table 2.1: Comparison Of All The Presented Web Annotation Services As Well As Dclaims

	Revocability	Multiple Website Support	Cross-Platform Compatible	Self Verifiability	Permanence	Censorship Resistant
Medium	**Yes**	No	No	No	No	No
Genius	**Yes**	**Yes**	No	No	No	No
Hypothes.is	**Yes**	**Yes**	**Yes**	No	No	No
DClaims	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

be decentralised, to not have centralised points where censorship can occur.

Blockchain and IPFS are decentralised technologies that can be leveraged to attain the goals described above, and for that reason, the next two Sections are dedicated to an overview of it.

2.4 Primer on Ethereum

In this section, we present Ethereum [40, 41], which is one of the building blocks of DClaims. Ethereum is a blockchain, which is a distributed application that runs on a peer-to-peer network with the goal of maintaining a state. The technological novelty behind blockchains is in the way the nodes on the state of the network in a trustless manner, that is, not trusting in any node to act correctly. Agreement on the state is achieved through a consensus protocol. There are several consensus protocols, the most used to date is the proof-of-work protocol [42], which consists of having nodes solving a complex mathematical problem. The node that solves the problem dictates the next state of the application. There are several blockchains working today [43–49], and while Bitcoin [1] is the most famous, Ethereum offers a more significant capability. Bitcoin is used primarily as a currency, that is a financial instrument used to buy and sell products[7]. The state that the Bitcoin network maintains is the balance of each user's account, which enables the network to validate all the transactions. On the other hand, Ethereum serves a much broader purpose, which is to run a global virtual machine, on the blockchain, that anyone can use by paying a small fee. The state that the Ethereum blockchain maintains is the Ethereum Virtual Machine's state.

In Section 2.4.1 we explain the consensus protocol of the network. Next, in Section 2.4.2 we explain how the Ethereum virtual machine works, then, in Section 2.4.3 we present Smart-Contract, which are arbitrary programs that run on top of the Ethereum virtual machine. Finally, in Section 2.4.4 we discuss the challenges in implementing a large scale system using Ethereum.

[7] Bitcoin can also be seen as a store of value, but that topic is beyond the scope of this project

2.4.1 Consensus Protocol

One of the key technological innovations behind blockchains is the way the network reaches consensus, meaning the way that all the nodes agree on the transactions that have taken place in the past, and which ones are to be included in the future. In a typical centralised architecture, there is an entity responsible for making such decisions, but in a trustless configuration, such as one of blockchains, it is essential to devise a way that prevents bad actors from damaging the network. The consensus protocol used by Ethereum is called proof-of-work and was first implemented by Bitcoin, which is why it is commonly referred to as Nakamoto consensus [1].

2.4.1.A An Overview of Bitcoin's Nakamoto Consensus

A Bitcoin transaction consists of nodes exchanging unspent coins and is identified by a chain of digital signatures. To complete a transaction the sending node digitally a hash that identifies the past transaction and the public key of the next owner. This ensures that each transaction is linked to the ones preceding it, creating a chain. The next key challenge is getting all the nodes to agree on the set of past transactions, to prevent double-spending. To that end, transactions are packed into blocks and added to an append-only data structure, the blockchain. Figure 2.3 depicts the format of Bitcoins transactions. Each transaction is linked to the one before it and is signed by the owner of the UTXOs used as input in that transaction.

To add a block to the blockchain, nodes have to mine the block, and the way to do that is to solve a mathematical challenge with a pre-defined difficulty level, this is called a proof-of-work [42]. The first node to solve that challenge mines the block, permanently adding it to the Blockchain. Each time a node mines a block, a new Bitcoin (coin) is created and given to that node. This constitutes an incentive for nodes to mine. The algorithm is designed to automatically adjust the challenger's difficulty in regulating the currency issuing rate. All the mined blocks are made of transactions that are linked to each other, which means that if a dishonest node was to change a past transaction that change would be eventually be detected by an honest node.

Sometimes a node will mine a block that has already been mined by another node, but due to network latency, the first is unaware of this. In this situation, a fork occurs. Both of these blocks are propagated through the network, and eventually, a new block will be mined and appended to one of those forks. The network will choose the most extended fork as the standing one, and the blocks on the branch that is not accepted are discarded. Those blocks are called stale blocks. The probability of the occurrence of a fork of depth n is $O(2^{-n})$ which gives miners a high degree of confidence that the transactions they process will eventually be included in the ledger.

Bonneau et al. [45] provide an overview of Bitcoin's consensus protocol stability features. It ensures consensus, eventually all correct nodes will agree upon the set of blocks that exist on the blockchain, it

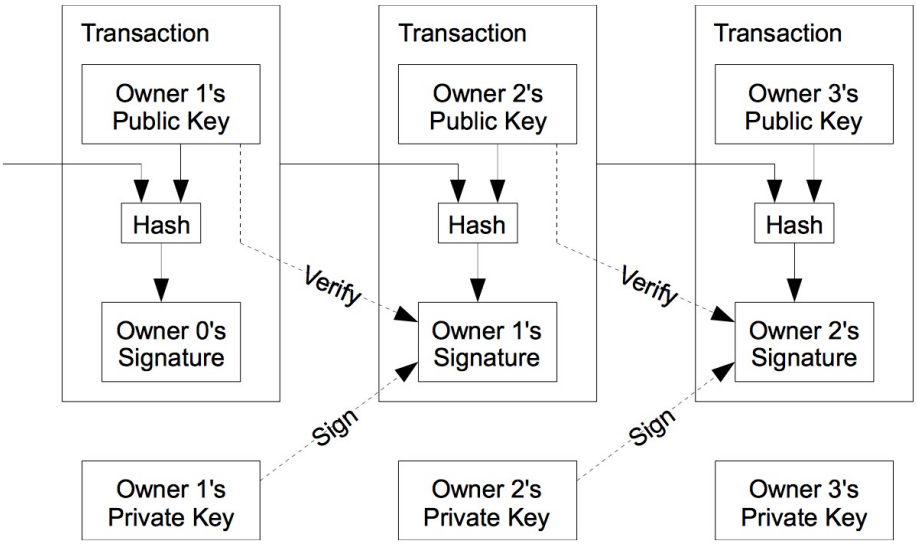


Figure 2.3: Bitcoin's Transactions [1].

has exponential convergence, giving confidence to miners that the blocks they mine will be eventually be added to the blockchain, it has liveness as new blocks will continue to be added, motivating miners by enabling them to collect fees, it has correctness since the most extended chain (the active one) will not include stale blocks, and it has fairness in the sense that the probability of a miner mine the next block is directly proportional to the amount of computing power that miner has.

2.4.1.B Ethereum's Genesis and Consensus Protocol

Ethereum was created as an alternative to all the highly specialised forks of Bitcoin. Following Bitcoin's success, a large and active community has emerged and worked on modifications and extensions to the original protocol to improve some of Bitcoin's shortcomings like performance, introduce new consensus protocols, add new features and even new banking systems [50]. These modifications and extensions, which are usually forks of Bitcoin or abstractions that sit on top of existing blockchains, are called "alt-coins" [45].

One of the criticisms of Bitcoin's proof-of-work mechanism is that it motivates an uneven distribution of mining power. This happens because the time it takes to solve the computational challenge can be decreased by using more CPU cores concurrently. To address this problem, Litecoin[8] introduces a proof-of-work mechanism that is not solved faster by adding CPU cores. Other systems propose completely different consensus mechanisms such as proof-of-space [51] where nodes are rewarded by storing data, rather than performing computations; this is the case with Storj [52] and FileCoin[9] crypto-currency

[8] https://litecoin.org
[9] https://filecoin.io

whose goal is to leverage the blockchain to store data. Blockstack [53] uses the Bitcoin blockchain to provide a global naming service (like DNS). Peercoin [43] proposes a Proof-of-Stake mechanism where nodes' stake in the network is calculated based on the age of the coins they own and is based on the premise that a node will act in its own best interest, so if it has a high stake in the network he will be well behaved and thus contribute to a good operation of the network. DDOS-Coin [44] proposes a malicious consensus mechanism, proof-of-denial of service, where nodes are rewarded by proving they participated in a DDOS attack. Another consensus mechanism include proof-of-bandwidth [54], proof-of-luck [55]. Some crypto-currencies proposals have focused on improving Bitcoin's transaction rates and scalability, which is the case of Bitcoin-NG [56,57], and others, such as ZCash [46] and HAWK [58], have focused on transactional privacy [59].

While Bitcoin's consensus protocol challenge relies on CPU power, Ethereum's lies on memory requirement. The goal of this different implementation is to make the network more democratic. One of the consequences of Bitcoin's consensus protocol was the creation of mining farms, i.e., infrastructures provided with Bitcoin mining specialised hardware, which resulted in an uneven distribution of the mining power among active nodes. Memory is hugely optimised in modern electronic devices, more than CPU power, so this measure empowers lighter miners to the detriment of more powerful ones.

As is the case with Bitcoin, the whole ecosystem of Ethereum is built around making the system heavily decentralised. This is achieved through architectural decentralisation – the number and distribution of machines that make the network – and political decentralisation – the machines are owned by a large number of different entities [60].

It is this level of decentralisation that makes the network fault tolerant, and resistant to attacks, and collusion resistant. Heavily decentralised systems rely on a lot of separate components to work, it is unlikely that a lot of these components will fail at the same time. Thus it is unlikely that the network will fail. Attacking a blockchain is expensive because there are no sensitive points of failure for an attack to be successful, many nodes need to be compromised, which removes the financial incentive to attack. Collusion is also discouraged by the way the protocol is implemented. Using a combination of game theory principles and cryptography – called *cryptoeconomics* – the protocol incentivises nodes to be well behaved.

In summary, the features of Ethereum that make it a good design choice for a censorship-resistant system are the following:

- **Decentralisation:** The political and architectural decentralisation of Ethereum makes it resistant to attacks and collusion;
- **Trustless:** No trust needs to be placed in any one entity;
- **Event ordering:** The state machine which runs on top of the blockchain constitutes a source of

ground truth for the ordering of events. This means that users who query Ethereum for the state of a smart-contract are assured to be viewing the latest valid state, and it is tough for any entity to change the state illegally;

2.4.2 Ethereum Virtual Machine

The Ethereum virtual machine is a computer which runs on top of the Ethereum virtual machine and was created to solve some of Bitcoin's shortcomings. One of the shortcomings of Bitcoin was the lack of contract customisation, as while it did allow for parties to create some rules regarding a given transaction (such as requiring a third party to approve the transaction), doing so required learning a not very intuitive scripting language and the functionality was insufficient. This lack of flexibility led to the creation of several Bitcoin forks, each one designed to meet a particular application need. These Bitcoin forks are called 'alt-coins' [45]. The problem with all these forks is that they are extremely specific, and end up suffering from the same lack of flexibility as Bitcoin. Ethereum's goal is to be a blockchain on top of which all future protocols can be built on. Its flexibility compensates for the lack of optimisation for specific applications.

The key for the flexibility of Ethereum is the Ethereum virtual machine, a 256-bit computer, which runs on top of all Ethereum nodes. This virtual machine runs programs written in Solidity and compiled into EVM Bytecode. All the nodes in the Ethereum network run the same operations. Since the virtual machine is deterministic (for a given input, the output is always the same), all nodes will reach the same state, which results in the network achieving consensus.

2.4.3 Smart-Contracts

Smart-contracts are pieces of arbitrary code which run on top of the Ethereum Virtual Machine. They can be written in various programming languages, the most used being Solidity. Smart-contracts are Turing-complete[10] which provides enormous flexibility and allows for the creation of arbitrary programs.

From a programming standpoint, one can look at smart-contracts as objects in the Ethereum universe; these objects can interact with each other to fulfil any business rules required by an application.

A smart-contract has to define an upper limit of computing power it will use. This limit of computing power is known as Gas. The amount of Gas a program has is what defines how long Ethereum nodes can run it. Gas costs money and miners are rewarded with that Gas. This condition is critical to stop programs that halted and malicious programs designed to enter an infinite loop. If an application runs and terminates successfully and does not spend all the Gas, the remainder is returned to the smart-contract and can be used in future executions. On the other hand, if a smart-contract runs out of Gas

[10] A case can be that smart-contracts lack Turing-completeness for their lack of support for infinite loops.

before ending its execution, all the changes made by the smart-contract are rolled back (to prevent inconsistent states), but the Gas is not returned to the smart-contract, it is given to the miners.

Smart-contracts are designed to be financial contracts. They implement some business logic that can be automatically enforced and which can move value between accounts. However, the flexibility of the smart-contract's programming language, called *solidity*, allows for the creation of a registry, meaning a data structure that can be used to time stamp particular events. These events are ordered in a canonical order – all the nodes in the network view the data structure in the same state – and can be used to keep track of arbitrary occurrences. For example, Alice can create a smart-contract that points to the PGP key she is currently using for digital signing documents. Alice is the only one with write permissions on the variable to change the key, and her Ethereum wallet signs the transactions she sends to the smart contract (to update the PGP key, whenever she decides to rotate it). Alice then broadcasts the contract's address across her network (she can share it on social networks for instance). This way, Alice's friends now know where to get the most recent key Alice is using whenever they need to verify a document signed by Alice.

2.4.4 Challenges in Developing Large Scale Systems Using Ethereum

Smart-contract's provide enormous flexibility, and the way they are executed in the Ethereum virtual machine ensures a correct operation at all times, but there are scalability and cost limitations that need to be overcome.

A – Cost Limitations: Paying transaction fees for deploying and interacting with smart-contracts is an essential part of how Ethereum works. The transaction fees are given to the miners, which ensure the correct operation of the network, consequently, if there were no transaction fees, there would be no motivation for miners to participate in the protocol. Adding to transaction fees, there is also a cost associated with storage. Since all nodes are replicas of the blockchain, each piece of data that is added to a smart-contract is replicated across all Ethereum nodes. To discourage an over usage of storage in smart-contracts, the fees associated with it are high. As an example, the storage cost per GB has been estimated to around USD 76000[11] .

B – Scalability Limitations: Just as a regular server has performance limitations, so does Ethereum. While the Ethereum blockchain is public and anyone with an Ethereum wallet with funds is allowed to send transactions (send Ether to other account or interact with a smart-contract), there is a limit to the

[11] Based on an estimate from StackExchange. The value in USD varies according to the value of Ethereum. https://web.archive.org/web/20180505133812/https://ethereum.stackexchange.com/questions/872/what-is-the-cost-to-store-1kb-10kb-100kb-worth-of-data-into-the-ethereum-block

number of transactions that Ethereum can handle per second. The current limit of transactions per second is around 20, which is high when compared to Bitcoin's 4, but much lower than traditional payments network such as Paypal, which supports around 193, and Visa, with 1667. A system which needs to support a high rate of state changes and is based on Ethereum needs to find a way to address the scalability bottleneck of the network. It is also important to notice that transaction rate limit means that all the entities making transactions on Ethereum can, collectively, make 20 transactions per second at most. If there are 1000 entities making transactions, each can only make 0.02 transactions per second[12].

In summary, developing DClaims with Ethereum as one of the building blocks carries two constraints:

- Keep the data stored on the blockchain to an absolute minimum;
- Minimise the number of transactions.

2.5 Primer on the Inter-Planetary File System (IPFS)

As highlighted in Section 2.4.4 Storing data on a blockchain is expensive and IPFS a good solution to that problem. IPFS [61] is a peer-to-peer distributed file system. The motivations behind its creation are a better use of bandwidth, the ability to permanently addressing content and moving the web towards a distributed architecture. As the web grows, the amount of bandwidth required to make it also work increases and the HTTP file transfer model that is widely used today does not optimise bandwidth usage. If 50 people in a classroom all download the same file from Dropbox at the same time, 50 independent requests are made to the backbone of the Internet. By addressing that same file with IPFS, nodes can download the file from other nodes who have it.

Figure 2.4 provides an overview of how IPFS, with an example. Suppose, that Alice wants to send a picture of her cat to Bob using IPFS. Assuming both Alice and Bob have IPFS installed, Alice starts by adding the `cat.jpg` to her local IPFS repository (*steps 1 and 2*), using the IPFS protocol. IPFS then provides Alice with the link to the file she just added (*step 3*), the link is formed by the hash of the file. Alice then sends Bob the link to the file (*step 4*). Using the IPFS protocol, Bob requests the file(*step 5*). Bob's IPFS node searches for the file in IPFS' Merkle DAG (*step 6*) – a data structure distributed across the IPFS network which holds a list of all the files available and routes to those files – then IPFS' networking library (`libp2p`[13]) handles the transport from Alice's to Bob's node (*step 7*). The `cat.jpg` is now stored on Bob's IPFS repo, and Bob can view it (*steps 8 and 9*).

[12] https://web.archive.org/web/20180505160148/https://altcointoday.com/bitcoin-ethereum-vs-visa-paypal-transactions-per-se
[13] libp2p: https://web.archive.org/web/20180505173610/https://libp2p.io/

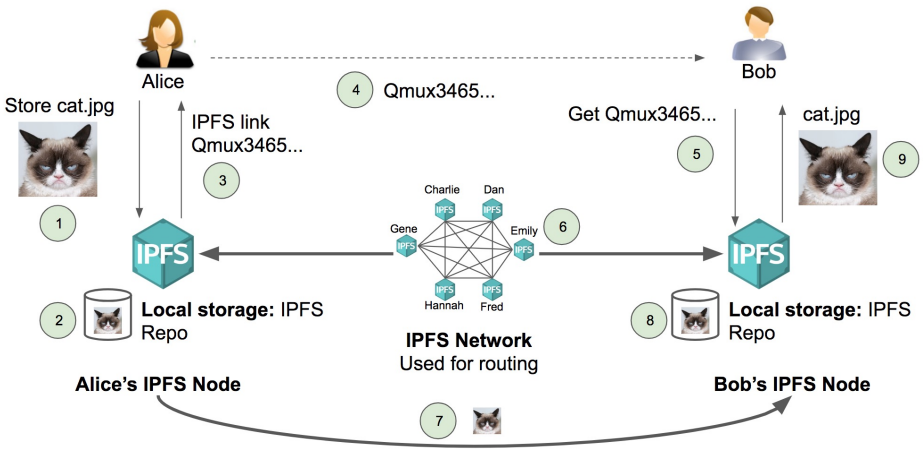


Figure 2.4: IPFS Overview. Transferring A Cat's Photograph.

2.5.1 Addressing Files By Content

Rather than being addressed by their location (IP address based link), IPFS files are addressed by their content in the form of a self-describing cryptographic hash (a multihash[14]) that is immutable and permanent. Content addressing offers two main properties:

- An IPFS link serves as both version control and integrity check on the file it addresses. The link is a hash of the file, so if the file changes, the link changes as well. Consequently, when a user retrieves a file from IPFS, she is sure that the file has not been tampered with.

- Files can be moved, and the link is permanent. A study by Harvard Law [62] found that 49% of the HTTP links cited in the United States Supreme files are unresolvable. This happens because HTTP links address files by their location, that is, the web server where they are stored. If the file is moved to another server, an HTTP link can easily stop working. That is not a problem with IPFS. Whether a file is stored, or not, in its creator's node is of no importance to the rest of the network. As long as at least one node has the file, the link will work, and all other nodes can access the file.

2.5.2 Using IPFS With DClaims

IPFS can be used to work with any application. The main application has two main implementations, in Golang and JavaScript, and there are API client libraries in over 14 programming languages and, in January 2018, Mozilla Firefox started supporting the protocol handler[15].

[14] https://github.com/multiformats/multihash
[15] https://web.archive.org/web/20180506003706/https://blog.mozilla.org/addons/2018/01/26/extensions-firefox-59/

One of the challenges in using IPFS for storage purposes in our system is that it does not offer replication by default. This means that when a user adds a file to IPFS, the file only exists in that users' IPFS node. Returning to the example of Figure 2.4, until step 5, the `cat.jpg` file only exists in Alice's IPFS repo. The consequence of this is that for Bob to retrieve the image, Alice needs to have her IPFS node running (assuming she is the only user who has the image). In the context of DClaims, we need to assure that everyone has access to the web annotations, regardless of the creators have their IPFS nodes online. For that reason, a replication mechanism needs to be built.

2.5.3 Censorship Resistance

Besides being a good solution for storing content anchored on a blockchain, IPFS offers a set of desirable features when it comes to decentralisation and censorship resistance, namely:

- **Logically decentralised:** An IPFS node does not need a connection to the backbone of the Internet to work. All the nodes are the same partitioned network can transfer files between them.
- **Does not need DNS:** DNS poisoning can be employed to prevent machines from translating a name to an IP address. This does not affect IPFS.
- **Does not rely on Certificate Authorities:** This means that Certificate Authorities that have been compromised will not affect the system.
- **Only one node needs to have the content:** If at least one node on the IPFS network as a file, all other nodes will be able to access it. Moreover, since the link to the file does not depend on the location of the file, the file can be moved around nodes without any disruption of access.
- **All files are cryptographically verified:** The IPFS application automatically verifies the integrity of the data. Since a hash of the file forms the address, it is easy to verify that it has not been tampered with. Consequently, even is a file changes location and the node serving the file is not its original creator, the receiver is assured to be receiving an exact copy of the original.

2.6 Summary

We began this chapter by presenting the state of the art of censorship-resistant systems. We looked at systems such as Tangler and Freenet whose goal was to provide access to files and documents, similar to the goal of DClaims. However, these systems do not offer a trustless architecture because they depend on a small number of servers controlled by arbitrary parties to assure critical operations, such as data integrity freshness.

Next, we motivated the use of web annotations as a tool that can be part of the process of fact-checking the information available on the web and showcased several web annotations services. Some of these annotations services came in the form of siloed services, characterised by low interoperability. We then presented an improvement on web annotations, which introduced interoperability. This improvement came in the form of a W3C Web Annotation standard and is implemented by a service called Hypothes.is. Then, we explained that despite the improvements introduced by the standard and Hypothes.is, there were still fundamental problems in the underlying architecture of these services, namely, lack of data permanence, integrity and prone to censorship.

At that point, we introduced Ethereum, which has a strongly decentralised architecture, a desirable feature to overcome some of the limitations of current web annotations services. Ethereum is a blockchain on top of which runs a virtual machine with the ability to run smart-contracts, scripts written in a programing language similar to javascript which can be used to write traditional programs, such as keeping track of events.

Finally, we presented IPFS, a distributed file system which offers desirable properties to circumvent content withholding while providing strong guarantees of data integrity.

In the next Chapter, we present DClaims, a system for Web Annotations built on top of Ethereum and IPFS which is resistant to censorship and scalable.

3

Architecture

Contents

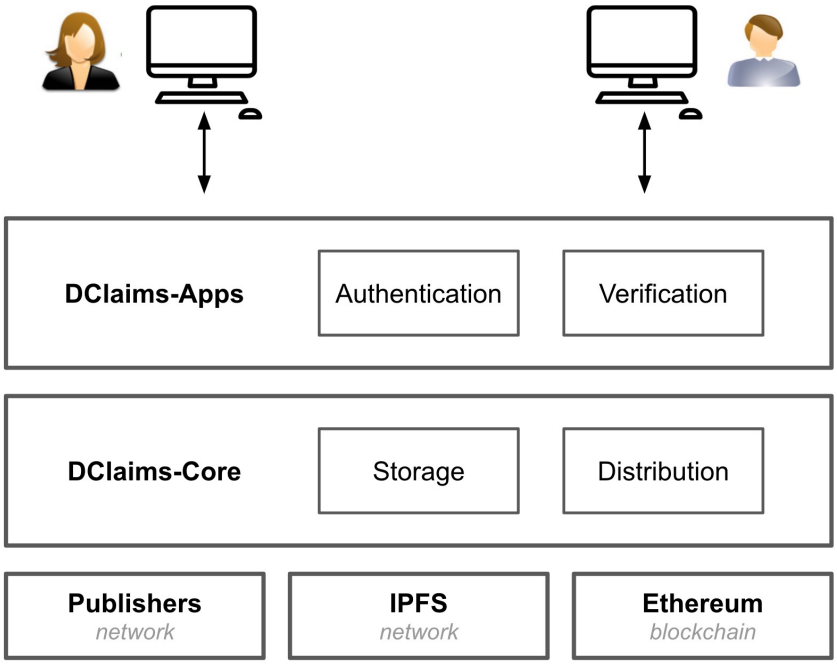


Figure 3.1: DClaims' Architecture

In this chapter, we present the DClaims system. We start by over-viewing the DClaims-Core layer, in Section 3.2. In Section 3.3 we analyse DClaims-Apps layer. Finally, in Section 3.4 we summarise the chapter, providing further insights into the system.

3.1 Overview

DClaims allows for the creation, storage and sharing of web annotations in a decentralised and censorship-resistant way. End-users create web annotations on a web browser. After creation, users send the annotations to an end-point, which belongs to a network of servers called Publishers, who are in charge of storing those dclaims on the IPFS network and registering them on the Ethereum blockchain. The novelty of the system is in the way web annotations are made into self-verifiable objects and in the way they are stored and distributed across the network.

As shown in Figure 3.1 The system's stack is divided into two primary layers, the core layer – called *DClaims-Core* – responsible for storing and distributing the annotations by DClaims' participants, and the application layer – called *DClaims-Apps* – responsible for generating new annotations, sending them to the publishers network, retrieving, filtering and verifying the integrity and authenticity of dclaims from DClaims-Core. This stack can be used by any web-annotations application that wishes to be resistant to censorship. We decided to divide the system into these two layers to provide application-level flexibility. The core layer is the same for any application. This means that all the web annotations are stored and

distributed the same way. On the other hand, the application layer allows applications to chose how they want to implement the verification procedures, what cryptography packages to users and how to identify users. A DClaims-App is an application which uses DClaims-Core for storage and distribution of web annotations and follows the DClaims protocol for integrity and authentication verifications.

DClaims' architecture was designed with the requirements discussed in Chapter 1. They include revocability, cross-platform support, self-verification, permanence, availability and censorship resistance. The attacker model we are considering is an entity capable of exerting the kind of censorship presented in Section 2.3.1.B, that is, a centralised entity which stores web annotations and, without the consent of end-users, can deny access, delete or tamper with the annotations it holds or prevent new annotations from being generated and stored by their service. We are not contemplating an attacker model which engages in censorship techniques on the transport level (such as IP blocking, packet dropping or content inspection).

3.2 DClaims-Core

This section describes the DClaims-Core architecture. Section 3.2.1 presents the dclaims data format. Next, in Section 3.2.2 we present the system's entities. In Section 3.2.3 we present the way data is stored on the platform. DClaims-Core information storage and propagation are achieved through a combination of IPFS and Ethereum. In Section 3.2.4, we discuss ways to implement the IPFS and Ethereum hybrid architecture. In Section 3.2.5, we settle on a specific architecture, using a network of special nodes called publishers, and then proceed to explain it.

3.2.1 Data Format

DClaims encapsulates web annotations inside a new data model, called *dclaim*. Two of the requirements of DClaims are that web annotations are self-verifiable and compatible with current standards. Unfortunately, the current web annotation standard data model does not provide any integrity and authenticity assurances. One way to solve this problem would be to change the web annotation data model to accommodate the necessary fields for self-verification – user identification and digital signature fields –, but that would result in the new data model not being compliant with the standard. For that reason, we encapsulate web annotations into a new data model, called a dclaim. A dclaim is a data structure, based on the Verifiable Claims Data Model[1] , which supports digital signatures and user identification. The difference between verifiable claims and dclaims is the context in which they are used. Verifiable claims are part of a standard, created for entities to make statements about themselves. DClaims can be used in other contexts, such as making statements about other entities.

[1] https://web.archive.org/web/20180507221511/https://www.w3.org/TR/verifiable-dclaims-data-model/

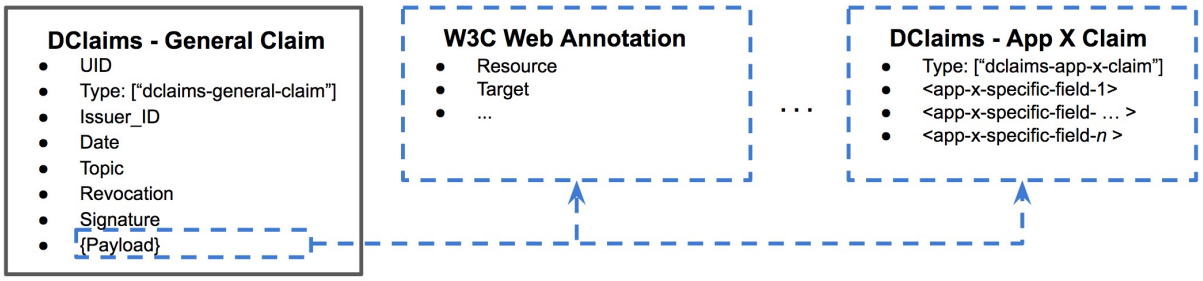


Figure 3.2: A Dclaim Object And Payload Field.

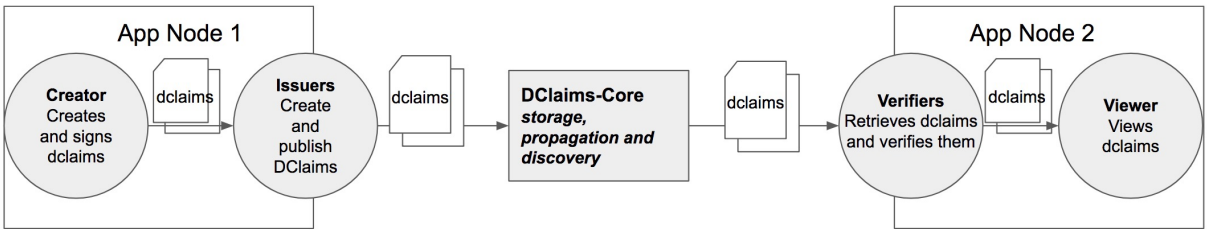


Figure 3.3: DClaims Entities Interactions.

Figure 3.2 illustrates the schema of a dclaim. It has a Unique Identifier, a type (which provides information about the application using this dclaim, such as the signature type, issuer id type and revocation information type), the id of the issuer (which depends on the type), the date of creation, the topic (which is used for dclaim discovery), information about revocation (a pointer to a resource which provides information as to how one should proceed to verify if this dclaim has been revoked), the digital signature (all the fields are signed) and finally the payload. The payload field is where web annotations are nested.

3.2.2 Entities

Figure 3.3 shows the interactions between the different entities. In this context, an application node is an end-user's terminal (browser, or smartphone) running DClaims. DClaims-Core has four basic entities, *creators*, *issuers*, *verifiers* and *viewers*. Using a DClaims application, a creator creates a web annotation, embeds it into a dclaim and signs the dclaim. An issuer stores and shares dclaims. A verifier retrieves dclaims and verifies (authenticity, integrity, validity) them. Finally, a viewer is a consumer of dclaims, he retrieves and uses the web annotation nested inside the dclaim. Figure 3.4 shows the lifecycle of a dclaim. It starts as an empty object, then it receives a web annotation as a payload, gets signed, and published. Finally, a user retrieves it, the verifier of its application node makes the necessary verifications to assure the correctness of the dclaim, and finally, presents it to the user.

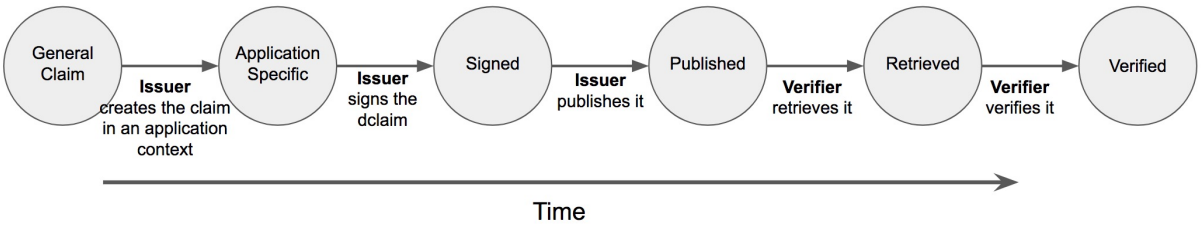


Figure 3.4: Lifecycle Of A Dclaim

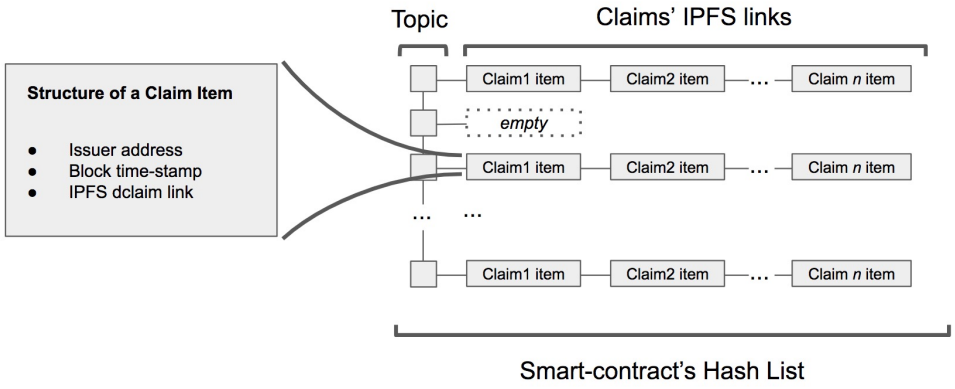


Figure 3.5: Smart-contract's Hash List That Keeps Track Of The IPFS Link Of The Issued Dclaims

3.2.3 Storage and Discovery: The IPFS and Ethereum Hybrid

In this section, we explain how IPFS and Ethereum are used to store dclaims. As presented in Sections 2.4 and 2.5, IPFS and Ethereum offer a set of desirable properties for censorship-resistant systems.

Dclaims are stored on IPFS and pointers to the data are put on an Ethereum smart-contract. For the DClaims application to retrieve the dclaims for a particular topic, it queries the smart-contract to get the IPFS links (pointers) and then fetches the files from IPFS. Dclaims are indexed by a topic. Claims with the same topic correspond to web annotations for the same resource. For example, web-annotations about the website `https://www.acme.com/index.html` are going to have the same topic.

The function of the smart-contract is to keep track of the dclaims issued. Figure 3.5 represents the smart-contract's data structure that maintains the dclaim's IPFS links. The smart-contract holds a hash list where the key is the dclaim topic, and the list contains the IPFS links, issuer addresses and time stamps of all the dclaims that exist about that topic. When a new dclaim is created, its IPFS link is added to the list, for that topic. When a dclaims-app requests the dclaims from a given topic, it provides the topic and the Ethereum client returns a list with the IPFS links to the dclaims (the issuer addresses and timestamps are also returned for the verification procedure). The user then proceeds to retrieve the dclaims via IPFS. Imagine a web annotations application where users make annotations about news

articles on websites. These annotations could be a classification of the article as being *true* or *false*. Web annotations made for the same article would have the same topic. This means that these annotations would be registered in the smart-contract under the same hashlist's key (which is the topic).

In the next two sections, we present the ways this hybrid storage mechanism can be implemented.

3.2.4 Possible Architectures For Storage and Discovery

In this section, we present two different approaches to issuing dclaims. The first approach discusses an implementation that only uses Ethereum smart-contracts, while the second, introduces IPFS for data storage, an approach in line with the one presented in Section 3.2.3. Neither of these approaches fulfils the requirements. However, it is valuable to discuss them as they are stepping stones for the final architecture solution.

At this point, it is useful to elaborate on some of the requirements posed by the constraints of using a blockchain based architecture:

- Low financial cost: As described in Section 2.4 there is a fee associated with each blockchain transaction. Moreover, in blockchain which offer Turing complete smart-contracts, such as Ethereum, there is a high price of storage. The two challenges are to minimise the number of transactions and the amount of data stored on the chain.
- Scalable: While traditional centralised architectures can support several thousand, or even millions, of operations per second, blockchain is much slower. Bitcoin supports about four transactions per second, while Ethereum supports close to 20. The challenge of scalability is to ensure that the transaction performance does not become a bottleneck in the system.
- User-friendly: Blockchain technology is still in the beginning and can be easily become overwhelming to a regular user. It is important to assure that the difficulties of dealing with the technology are dealt with by our architecture and not passed on to the final user.

Using only an Ethereum smart-contract to support the entire system offers the highest degree of decentralisation. Each issuer is responsible for issuing its dclaims directly on the Ethereum smart-contract, which deals with the dclaim storage, meaning that all the necessary information about the state of the system is kept on the smart-contract. The advantage of this approach is its simplicity, fully decentralised nature and high availability. On the other hand, it carries high operation costs. Storage on Ethereum smart-contracts is costly, and in this case, everything is stored on the smart-contract. This approach would also generate an enormous amount of transactions, as a transaction is required for issuing each new dclaim. In conclusion, this approach fulfils the web annotations improvement requirements but fails at being a useful blockchain architecture due to scalability and cost-prohibitive issues.

With operating cost reduction in mind, we developed another architecture model in which dclaims are stored by the issuer, instead of on the smart-contract. The issuer runs an IPFS node in which dclaims are stored and through which another DClaims user can access them. Only the IPFS links to the dclaims are stored in the smart-contract (as opposed to the first approach where the entire dclaim was stored on the chain). Storing the dclaims on IPFS offers the same integrity assurances as storing them on the smart-contract since the IPFS link of a file is the hash of that file, meaning that if the file changes the link changes as well. This approach improves cost efficiency but reduces availability. The user is required to run an IPFS node for the dclaim to be made available to the rest of the network. When nodes request dclaims from other nodes, they cache that dclaim (pinning), which increases its availability in the network. However, if an issuer issues a dclaim (adds the dclaim to IPFS and then puts its IPFS link in the smart-contract) and immediately proceeds to disconnect his computer from the network, no one else will be able to access the dclaim until the issuer reconnects to the network. In summary, this approach improves on the previous one by reducing the cost of issuing new dclaims (by moving the storage away from the smart-contract) but not without penalising availability. Moreover, just as the previous architecture it still requires one transaction per new dclaim issued, thus, having the same scalability issue.

In the next section, we present the final storage and discovery architecture which uses a network of specialised nodes, called publishers, to achieve all the desired requirements. In this section, we refer to DClaims application nodes to a client-side application running DClaims' software, which is executed in a terminal controlled by the end user, such as a web browser or a smartphone. We refer to publisher nodes as web servers running the DClaims publisher's software, which are part of the DClaims publisher network.

3.2.5 The Publishers Network

The problems pointed above (software requirements, availability and issuance costs) can be solved by having a network of dedicated nodes that run the required software to issue dclaims, replicate, or pin, dclaims issued by users and issue dclaims in batches. This approach has the potential to dramatically reduces the issuance cost. We call these special nodes, publishers. We start, in Section 3.2.5.A by explaining how the issuance of dclaims works with the Publisher network. In Section 3.2.5.B we highlight how Publisher issue dclaims as batches, dramatically reducing costs. Then, in Section 3.2.5.C we provide insight into how the system deals with misbehaved publishers. In Section 3.2.5.F and Section 3.2.5.E explains how the system protects against denial-of-service spam attacks. Finally, Section 3.2.5.D, overviews the economic model which supports publishers.

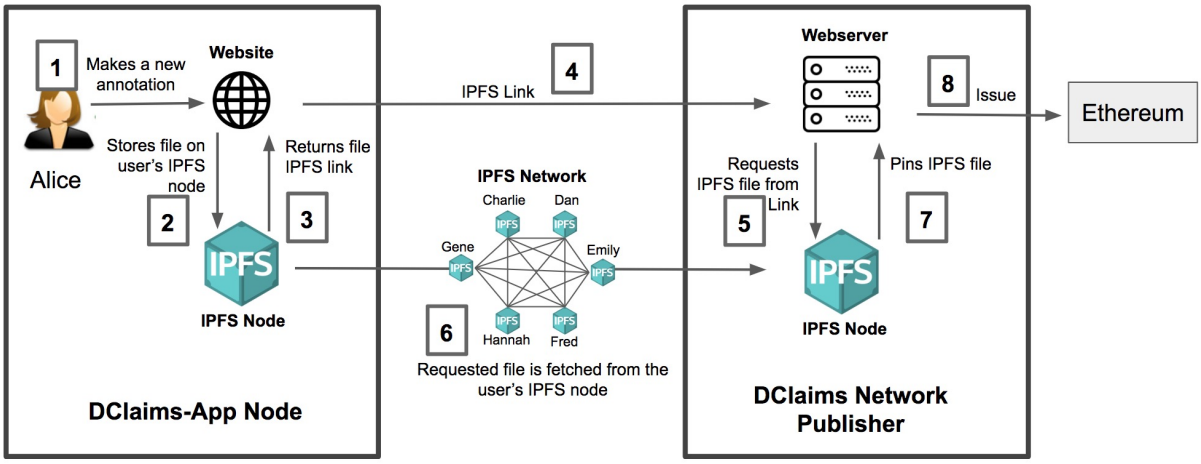


Figure 3.6: How A Publisher Makes A Copy Of A Dclaim

3.2.5.A Issuing Dclaims

Publishers act as a proxy between a dclaims-app who wants to issue a dclaim, and IPFS and Ethereum. To issue a dclaim the app's issuer uses DClaims to store the dclaim on its IPFS node and send the link to a publisher (the network can have more than one publisher). The publisher then proceeds to pin the dclaim in its IPFS node and adding the dclaim to the smart-contract (adding the IPFS link, just like in the data issuance with IPFS storage model).

Figure 3.6 shows how a dclaim is issued using a publisher. Alice, (creator) creates a new dclaim in a dclaims-application (*step 1*). The application, using dclaims-core, adds the dclaim to the user's local IPFS node (*step 2*) which returns the IPFS link (*step 3*). The application then sends the IPFS link to one of the publishers in the DClaims' network of publishers (*step 4*), who stores a copy of the dclaim in its IPFS node (*steps 5-7*). Finally, the dclaim is added to the Ethereum smart-contract (*step 8*).

A positive aspect of this approach is that the user who created the annotation keeps a copy of the file, so even is a publisher later deletes it, the object still exists on the network (it will, however, be unavailable until the user reconnects to the network).

3.2.5.B Batch Issuance

Upon receiving a dclaim, a publisher needs not to issue it on the smart-contract straight away since that would result in one new Ethereum transaction per dclaim. A more cost-efficient way of doing it is to batch dclaims (that have the same topic) together and issue them all in the same transaction. Figure 3.7 illustrates the batch issuance mechanism. Let us look at the dclaim with UID 123 – each dclaim has a unique identifier, a *uid*–, as an example. The dclaim is sent to the publisher, who places it in a buffer. After having received the dclaim, the publisher returns an issuance receipt (the receipt mechanism is

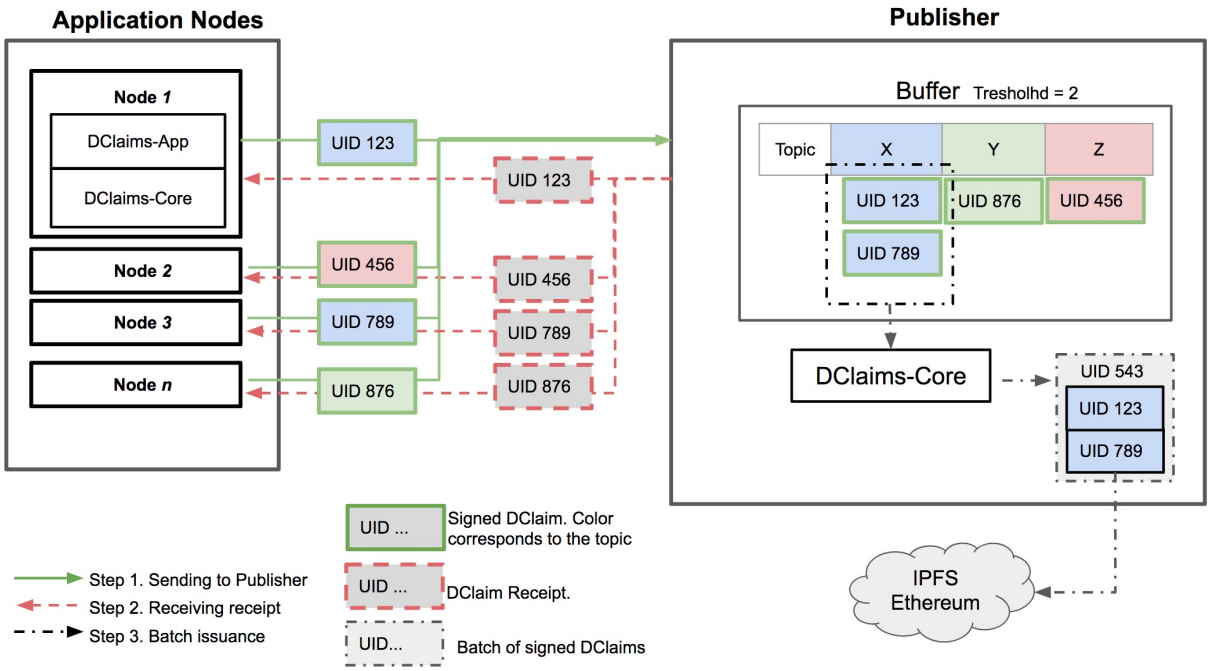


Figure 3.7: Publisher's Batch Issuance And Receipt Issuance Mechanism

explained in Section 3.2.5.C). Each buffer has a threshold, controlled by the publishers. Publishers can set a threshold, X, on the number of dclaims to include in a given batch. This threshold is defined based on the activity level of dclaims issuance for a specific topic. Topics with a higher level of engagement (for instance, a webpage receiving hundreds of web annotations per hour) can have larger batch sizes, as the batches will fill fast. Consequently, topics with a lower level of engagement should have smaller batch sizes, as the batch will take longer to fill. When a publisher has received X dclaims, a batch is formed, which is added to IPFS and issued on Ethereum as one transaction. In the case of the example, the threshold is two. The dclaim with UID 789 arrives after UID 123, and both have the same topic. At this point, the buffer is full, and the dclaims are issued as a batch. Publishers can then choose if they want to carry the cost of issuing to the issuers (100 dclaims from 100 different issuers, the Ethereum transaction price was 1USD so that each issuer would pay 0.01USD), or pay for the issuing themselves. This can be achieved by implementing other mechanisms to generate revenue (display ads, have the issuers mine some cryptocurrency for them, mass adoption like review websites). The way to interact with an Ethereum smart-contract is by sending transactions to it. Each contract has its address, just like a personal Ethereum account's address. The cost of transactions is determined by the amount of Gas they use (Gas is a measure that takes into account a given operation's computational cost and storage cost and is paid for in Ether).

3.2.5.C Receipts

A potential concern is that the publisher network can be seen as a point of centralisation in the system architecture. If publishers are blindly trusted, they can exert all kinds of malicious behaviours such as deletion based censorship. One can easily imagine a case where a publisher decides to delete, or not publish, dclaims that can cause it financial damage. It is important to mention that an attack by a publisher would only compromise the temporary availability of the system, causing no permanent damage. This is due to the permanent record of dclaims being stored in the Ethereum smart-contract. We present a mechanism to discourage publishers from misbehaving by making it easy for any user to spot a badly behaved publisher and to inform the rest of the network about the bad publisher. The mechanism is called *receipts* and is shown in Figure 3.7. When an application requests the issuance of a new dclaim to a publisher, the publisher returns a receipt acknowledging the request and promising to issue the dclaim in a given time frame. When that time frame ends, the user can query the smart-contract (either by running its ethereum node, or using another party as a proxy, such as MyEtherWallet, EthBase or BlockCypher) for the dclaim link, to verify that it was issued and request the dclaim from IPFS (again by running its node or using a public gateway), to verify that the publisher pinned the dclaim in its IPFS node.

After having verified that a publisher failed to publish a dclaim, a user can generate a new dclaim, called a complaint, which is issued in the smart-contract (issued just like any other dclaim, stored on IPFS first and then sent to the smart-contract). The dclaim is then added to a publisher-complaints-list which lives on the smart-contract. This list is meant to be consulted by other users when making a decision on which publisher to use. If a user sees that a particular publisher has many complaints, he may decide not to use that publisher. Complaints are verifiable and auditable. Each complaint contains the receipt that the publisher gave the user upon issuance request which allows any other user to audit the validity of the complaint. If a user analyses a complaint and verifies that the original dclaim (which is the subject of the complaint) was issued correctly, meaning that the dclaim link is listed on the smart-contract, was issued by the publisher and is available on the publisher's IPFS node, then the user can and should disregard that complaints. A possible spam attack would be a user issuing many complaints with false receipts (which would fail to be verified) to flood the network. To prevent this, complaint dclaims cannot be issued by publishers. They have to be issued directly by the end user, with him paying for the transaction. This provides a financial disincentive to spam the network.

It is possible to implement more sophisticated complaint and punishment mechanisms, such as requiring a proof of stake from the publishers to ensure their correct behaviour, or use a token curated registry of publishers. However, that will be left for future research. [63] proposes a *proof-of-censorship* mechanism, yet the type of data in which their work acts upon is different from ours.

Users can have different uses for publishers. In the case of issuing dclaims, some users may want to

use them only as Ethereum proxies, to defray the cost of issuance, while others may want to run their own Ethereum nodes and directly issue their dclaims on the smart-contract and only want to use publishers to replicate their dclaims on IPFS. In the case of issuing dclaims, users need just to run their IPFS nodes when they are using the system, which is an improvement to the alternative which would require them to permanently run their IPFS nodes or, in the very least, run them until some node replicated their dclaims. The different use cases are implemented by the publisher software and accessible via the publisher API.

3.2.5.D Paying For The Publisher Network

We consider two possible financial models to support the publisher network, donations and pay as you go. Each publisher is responsible for supporting its costs so that different publishers could have different financial models.

As we discuss in the evaluation of our system, in Section 5.2.2, the cost of running a full-scale deployment of DClaims would have a cost 5 to 6 times lower than the one of Wikipedia[2]. If the community found the system valuable, a donation based financial model could be a viable alternative.

An alternative to that model would be to have users pay for issuing their web annotations. We calculate that the cost of creating 1000 annotations is a little over USD 2. Compared to the cost of using current free web annotations platforms, USD 2 is significant. However, DClaims offers the value proposition of being censorship resistant, which in some cases may be well worth the money. Publishers could charge regular payments to users (PayPal, credit card), or they could employ more sophisticated methods, such as requiring users to mine some cryptocurrency on their website. Platforms such as Coinhive[3] offer a service where websites can run a Captcha[4] which is running a proof-of-work algorithm to mine Monero. Ultimately, publishers could also run ads on their websites.

3.2.5.E Protecting Against SPAM

Publishers dramatically reduce DClaims' cost of Ethereum transactions and, depending on the financing model they use, issuing dclaims may be free. For that reason, there needs to be a way to protect the system against attackers who want to spam the network by issuing large amounts of meaningless dclaims. We envision two independent mechanisms that can be put in place to mitigate this type of attack:

- **Demand a proof-of-work:** Publishers can request users to expend CPU power by performing some computationally intensive calculation. For individual users, issuing individual dclaims, the

[2] https://www.wikipedia.org
[3] https://web.archive.org/web/20180508132739/https://coinhive.com/
[4] Captcha: Completely Automated Public Turing test to tell Computers and Humans Apart

time spent performing this action would be minimal and cause little to no impact on their user experience. However, for malicious users engaging in much activity, the proof-of-work would become a strong disincentive to attack. Moreover, this proof-of-work could serve as a financing model for the publisher, as described in Section 3.2.5.D.

- **Authenticate and Identify users:** Publishers could choose to only issue dclaims for users who are authenticated in their platform. This way, if a user engaged in an attack, the publisher could remove his credential. The requisite for this mechanism to work is that it has to be expensive to create new accounts, to prevent a banned user from just creating new accounts. To this end, publishers could request users to anchor their accounts on social network platforms (by having the user posting on his social network accounts a string provided by the publisher, to prove control of the account), or provide some proof of identification (such as a photo of a government-issued ID). A valid concern is that this mechanism can be employed by publishers to censor individual users. If this happens, a blacklisted user has two options. The first, and easiest, is to use another publisher (we assume there are several). The second, which could occur if all publishers conspired against this user, would be to issue the dclaim directly in the smart-contract. The downside of the second alternative is that the user would have to pay the full Ethereum transaction price. In both cases, the dclaims issued by this user would be available to all other DClaims users.

3.2.5.F Replication of Data

DClaims creates one copy of each dclaim issued. As discussed in Section 3.2.4, IPFS does not automatically replicate data across the network, only when nodes request the content, are new copies generated. This leads to a bootstrapping problem, which we solve by having publishers copying each new dclaim that is issued, as described in Section 3.2.5. With this mechanism, we ensure that there is at least one node online which can serve the content. As more users request a dclaim, more copies start becoming available in the network and remain available while these users are online, unless the users choose not to share content with other nodes.

However, this mechanism assumes that publisher nodes are always online. To solve this, a replication procedure between publishers can be put in place. This way, even if a publisher fails, there are others who can serve the same content. Freenet and Tangler propose replication mechanisms which could be used in DClaims. A more robust replication mechanism could be the topic of future work.

It is important to emphasise, however, that the default DClaims replication mechanism, based on IPFS' replication of files across the network, already offers a high degree of replication. At any given point, an online node – a regular end-user using DClaims – can serve all the content that he has consumed. If Alice is checking the web annotations for the `https://www.acme.org/index.html` website, she can serve the annotations of that website to any other DClaims users who require them. Users

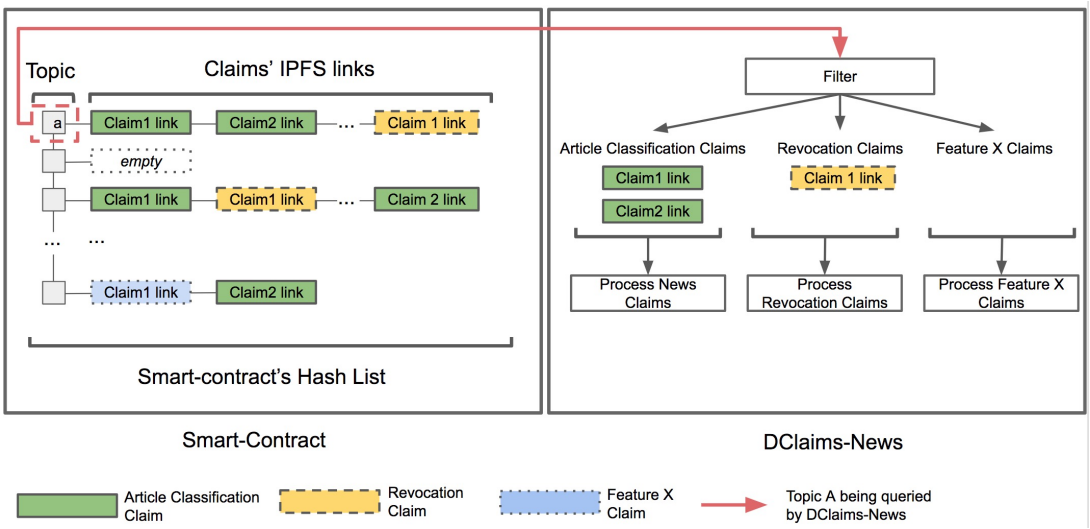


Figure 3.8: The Dclaims Aggregation Process In DClaims-News. In This Example The Application Is Querying Article A (Topic A In The Hash List)

can choose not to share the content they have on their IPFS node, however, given the purpose of this system, we believe nodes would be motivated to share the data. In the case of less popular dclaims, the replication factor would be lower, but the dclaim would still be made available by the publisher who issued it, and by the creator of the dclaim. In the next section, the DClaims-Apps layer will be introduced.

3.3 DClaims-Apps

The DClaims-Apps layer deals with the application level aspects of DClaims. It works on top of DClaims-Core and can be programmed to work with multiple web annotations applications. In this section, we analyse the critical components of DClaims-Apps. We start in Section 3.3.1 by clarifying how DClaims can be used to work with many different applications. Next, in Section 3.3.2, we explain the dclaim verification procedure. In Section 3.3.3 the user authentication procedure is revealed. Then, in Section 3.3.4, we highlight how dclaims can be revoked. In Section 3.3.5, the SPAM protection mechanism is explained. Finally, in Section 3.3.6 we describe a mechanism to reduce the amount of network data used by end-users.

3.3.1 How DClaims Supports Multiple Applications

DClaims offers support for multiple data formats, and web annotations are just one of them. Moreover, DClaims supports different data formats inside the same application. A dclaims based application is not restricted to use only one type of dclaim. Figure 3.8 illustrates the process through which dclaims are interpreted and segregated. All dclaims for a given application are kept track of in the smart-contract,

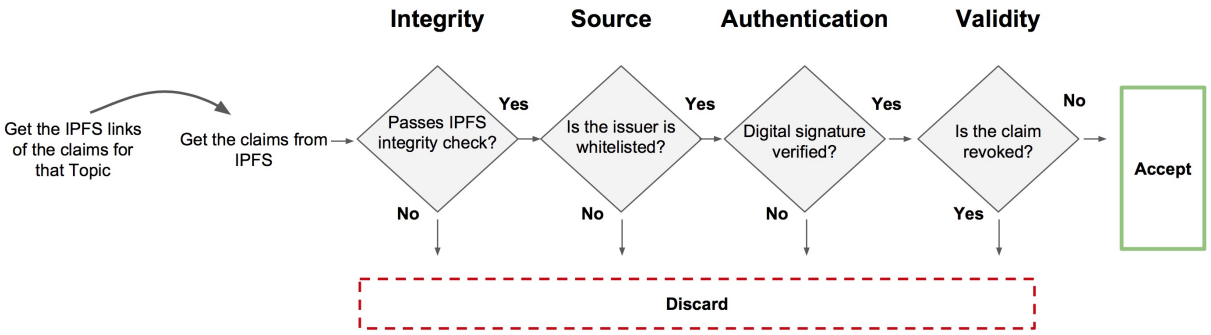


Figure 3.9: The Verification Process Of A Dclaim.

regardless of the type of dclaim. When the application queries the smart-contract for a given topic, all the dclaims are returned (the IPFS link is returned, and then the dclaims are retrieved from IPFS, this part is omitted in the figure). The application then proceeds to filter the dclaims. In the scenario represented in this figure, the dclaims news application (the proof-of-concept application we developed, detailed in Chapter 4) is querying the smart-contract for dclaims about the article with id A (so it retrieves the list for the topic A). When the application received the dclaims (after fetching them from IPFS), it filters and then processes them. In this case, there is a revocation dclaim for Claim 1, meaning that this one would be discarded in the verification process (described in Section 3.3.2).

An application that wants to use DClaims starts by customising DClaims-Apps to its needs. Using the Web Annotations use case as an example:

- **Define the data format:** An application defines the data format it wants to use. For a web annotations application, the data format can be the W3C Web Annotations standard.
- **Define the cryptography package:** The application chooses the cryptography package it wants to use and implements the digital signature and verification methods. As an alternative, the application can use DClaims-Apps reference implementation (described in Section 4).
- **DClaims-Core Setup:** The application uses DClaims-Core to deploy a smart-contract for the application. This handles storage and distribution of data. Different web annotations services would run different smart-contracts, same code but run in different addresses.
- **Publisher Network Setup:** The application sets up the publisher network by running the DClaims publisher software on many servers. Then, it makes the addresses of those servers known to its end users.
- **Use DClaims-Core:** The application then interacts with DClaims-Core to issue and retrieve dclaims. Once dclaims are retrieved from DClaims-Core and properly verified, the payload, web annotations, is extracted and the application can run normally.

3.3.2 Dclaims Verification

All dclaims are digitally signed by their creator. The digital signature is the main form of integrity and authenticity check, but there are more verification steps. Figure 3.9 illustrates the verification process. After acquiring the dclaims' IPFS links from the smart-contract, the first layer of verification is handled by IPFS which checks that the link of the dclaim matches its content. This is possible because the IPFS link of an object is its hash (the hash function is described in the link's prefix). The second step of verification is to filter the dclaims. As will be described in Section 3.3.5, users can choose whose dclaims they want to see, so all the dclaims from issuers who are not whitelisted by the user are discarded. The next step of the verification is to check the digital signature of the dclaim. The verifier checks that the public key used to sign the dclaim is the same as the one in the Issuer ID field. The last verification step consists of ensuring that the dclaim has not been revoked. If all these checkpoints are passed, the dclaim is considered valid verified.

3.3.3 User Authentication and Identification

Dclaims Issuers are authenticated by the digital signatures in the dclaims they issue. The reference implementation of a DClaims-App (DClaims-News, revealed in 4.5) uses an Ethereum library to sign dclaims using the Ethereum address. Other applications can choose to use other cryptography packages. There is also the scenario where a publisher requests an entity (issuer, verifier) to authenticate itself before him to be granted access to published dclaims or publish dclaims (this application relates to the financial model used by publishers, discussed in Section 3.2.5.E). This can be enforced via API authentication. In this situation, the authentication procedure is left in charge of the publisher, who can choose from a variety of different authentication protocols available. It is important to emphasise that this authentication does not replace the digital signature of dclaims by the entities' Ethereum wallets, it is merely a step that publishers can choose to require to allow entities to interact with them. Different publishers can have different authentication mechanisms. For instance, publisher A can request a combination of Google and Facebook authentication, publisher B can request a DID based authentication, using uPort or Sovrin and publisher C can choose not to request any authentication.

3.3.4 Revocation of Dclaims

The way a dclaim is revoked is by issuing a new dclaim, which states that a previous dclaim is now revoked. As detailed in Section 3.3.1, DClaims supports various types of application specific dclaims. Each different type of dclaim can be interpreted differently. We leverage this mechanism for revocation.

A – Revoking a Dclaim The process of revoking a dclaim consists on issuing a dclaim-news-revocation-dclaim. This dclaim is issued just like any other DClaims-News dclaim, it can be issued directly by a user, or by using the publisher network, it is stored on IPFS and kept track of on the Ethereum smart-contract. This dclaim has a different type (which states that it is a revocation dclaim) and payload, which includes the UID of the dclaim to be revoked.

B – Verifying the Revocation Status of a Dclaim When a user requests the dclaims from a given topic, all the existing dclaims on that topic, regardless of the dclaim type, will be returned. The dclaims interpreter (explained in Section 3.3.1) will separate the different types of dclaims, so at this point, the application has access to the web annotation dclaims and the revocation dclaims. Each of these dclaims has a UID, so to verify that a dclaim has not been revoked what is done is a search for each web annotation dclaim's UIDs in the revoked dclaims list. If we have a web annotation dclaim with UID 123 and there is no revocation dclaim for that UID, then dclaim 123 has not been revoked.

3.3.5 Protection Against SPAM

To ensure that end-users are not exposed to an excessive amount of dclaims, users are required to whitelist the issuers whose claims they can see. In Section 3.2.5.E we presented the way DClaims-Core protects against SPAM in the publishing phase, here we highlight SPAM protection for end-users. Both IPFS and Ethereum are permissionless, meaning that all users can see all the dclaims made about all the topics. This could mean that a user could end up being exposed to too many dclaims (especially if some users engaged in spam attacks by generating a lot of dclaims, similar to twitter bots) which is precisely one of the problems with Facebook comments and one we want to avoid. For this reason, we designed a mechanism where users whitelist the users whose dclaims they want to see. This way if a user goes rogue and starts posting spam, other users need only to remove him from their list.

3.3.6 Limiting Data Consumption

We envision that end users can run DClaims applications on their smartphone browsers, and for that reason, we designed a way to limit the data consumption of the application. The most obvious way for dclaims to be downloaded is to query the ethereum smart-contract for all the dclaims about a particular topic, and then download all those dclaims from IPFS. That would result in an enormous amount of network data consumption.

To prevent this from happening, when a DClaims application queries the smart-contract for the dclaims about a specific topic it first downloads only the first dclaim of every batch (the batching procedure is described in Section 3.2.5.B). This first dclaim contains metadata about the remaining of the

batch. From this, the application only downloads the dclaims from the issuers that the user subscribes too.

This mechanism can be taken one step further by only loading a small number of dclaims at a time (20, for instance) and requesting user input to download more.

3.4 Summary

In this chapter, we started by seeing the way DClaims is divided into two layers. The DClaims-Core layer handles data storage and discovery, while the DClaims-Apps layer handles data verification.

DClaims-Core uses a data format, called dclaims, which is self-verifiable and can encapsulate web annotations inside it. Dclaims are stored in the IPFS network, and a registry of those dclaims is maintained in an Ethereum smart-contract. Two elementary architectures for the usage of IPFS and Ethereum were presented, but both had problems of scalability. A third architecture, which used a network of servers to replicate data and process transactions as a batch, was chosen. The network of servers is called the publishers' network. We explained how the publisher network had the potential to dramatically lower the cost of Ethereum network transactions in the DClaims system and provided two viable economic models to support the network. We then discussed some mechanisms to counter SPAM attacks on the network and to report misbehaved publishers.

We also presented the DClaims-Apps layer. We started by reviewing the dclaims verification process, followed by the several components on which it depends, dclaims signatures, user authentication, revocation mechanism and the SPAM protection mechanisms. Next, we proposed a way to reduce the usage of network and, finally, explained how dclaims could be used by other applications, highlighting the steps necessary to convert an application into used DClaims' architecture.

Next, in Chapter 4 we present our reference implementation of DClaims, which consists of the implementation of DClaims-Core and an application, a DClaims-App, for web annotating news web pages.

4

Implementation

Contents

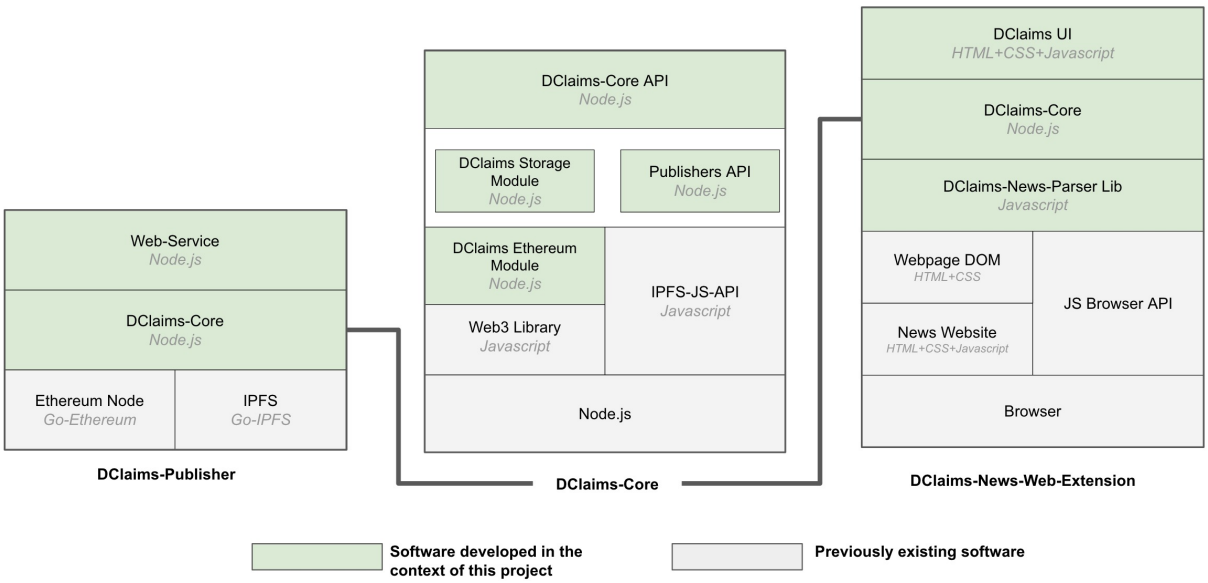


Figure 4.1: DClaims Core, Publisher And Web Extension Software Stacks.

In this chapter we present the developed software stack, as described in Figure 4.1. We start, in Section implementation:selected technologies, by overviewing the selected technologies. Section 4.2 focuses on the implementation of the DClaims-Core library while Section Section 4.4 focuses on the Ethereum smart-contract. The Publisher software is covered in Section 4.3. Finally, Section 4.5 covers the details of the DClaims-News application.

4.1 Selected Technologies

In this section, we explain the technologies that were used to implement the DClaims-Core and DClaims-News platforms, as well as the ones used for the system testing. All the libraries and most of software development platforms used are open-source. Furthermore, all the DClaims source code is available, open-source, on GitHub[1].

4.1.1 IPFS Tools

IPFS was used as the storage and transport platform due to the unique properties it offers as well as large community adoption. We utilised the following tools to work with IPFS:

- **Go-IPFS**[2]**:** We used the reference implementation of IPFS, written in Go-lang. This is the software that every DClaims node (application users and publishers) runs to connect to the IPFS network and run a local IPFS node.

[1] https://github.com/inesc-id/dclaims-pm
[2] https://web.archive.org/web/20180508232713/https://github.com/ipfs/go-ipfs

- **JS-IPFS-API**[3]**:** The JS-IPFS-API is a client library for IPFS, written in Javascript. We used it to have the browser extension communicating with the Go-IPFS node running on our machine.

4.1.2 Node.js and Javascript

Given the need to develop libraries that would be used both in a server environment as well as a browser environment, Node.js and Javascript were chosen as programming languages. Modern browsers such as Chrome and Firefox have native support for Javascript and Node.js is compiled to Javascript. The codebase can be shared. A Node.js module can easily be compiled to run on the browser (using a module bundler, such as Webpack[4]). Moreover, developing in Javascript benefits from NPM[5], an open-source registry with over 600.000 modules (which are essentially the building blocks of code).

4.1.3 Ethereum Tools

One of the reasons Ethereum was a good platform to use in the implementation of DClaims, was the great set of development tools it offers:

A – Writing the Smart-Contract: We wrote the smart-contract in Solidity, which is similar to Javascript in syntax, for it is the most widely adopted smart-contract language. There were alternatives such as Serpent, which is based in Python, but the community adoption is much lower. This was a deciding factor due to the novelty of programming smart-contracts. The development environment used was the Remix integrated development environment, which runs on the browser, for the rich feature set if offers. Remix makes possible not only writing the smart-contract but also deploying it and interacting with it, provided a Web3 client is running in the browser (more about the Web3 library later in the chapter).

B – Connecting to the Ethereum Network: To deploy smart-contracts on the Ethereum network, an Ethereum node is required. In the development environment, we used two Ethereum clients. In the Node.js environment (when Publishers are running DClaims-Core) we ran an Ethereum node using the Go-Ethereum (Geth) client. Geth starts by syncing with the selected Ethereum network by downloading the blocks of all past transactions (there are faster-syncing methods which only download the block headers). Geth then runs a remote procedure call, on a pre-selected port, to which applications can connect to.

[3] https://web.archive.org/web/20180508232808/https://github.com/ipfs/js-ipfs-api
[4] https://webpack.js.org/
[5] https://www.npmjs.com/

In the web browser environment (when developing and testing the browser extension) we used Metamask which is a browser extension that functions as an Ethereum node proxy. Contrary to Geth, Metamask is not an Ethereum node in that it does not process transactions. It is instead a gateway to Ethereum nodes being run by a third party (Infura[6], which runs public Ethereum nodes).

Node.js and Javascript applications can connect and interact with an Ethereum node via the Web3 library. In the Node.js environment, Web3 connects to Geth and in the browser, to Metamask.

4.2 DClaims-Core

The DClaims-Core module is responsible for handling the issuance and retrieval of claims and is implemented in Javascript. The module is composed of several sub-components. The Storage module handles the communication with IPFS, and the DClaims-Ethereum module handles the communication with the Ethereum Smart-Contract. There is also the Publisher API which is used for interactions between applications and the Publishers.

4.2.1 Application Program Interface

The API exposes the methods that applications built on top of DClaims-Core (DClaims-Apps) can use. The main exposed functions are:

- `issueClaim(claim)`: Stores the claim on IPFS and issues it on the Ethereum smart-contract. Returns the Ethereum transaction identification.
- `issueClaimWithPublisher(claim)`: Sends the claim to be issued by a Publisher. The way a publisher proceeds upon having received a claim is described in Section 4.3.
- `getClaimsForTopic(topic)`: Retrieved the list of claim links from the smart-contract and then fetches the claim from IPFS. Returns an array with the claim objects.

4.2.2 DClaims-Ethereum Module

The DClaims-Ethereum module has three main purposes. First, it establishes a connection to an Ethereum node (using the Web3 Library), second, it connects to a deployed smart-contract and finally, it makes calls and transactions to the smart-contract.

A – Establishing a Connection to an Ethereum Node: Using the Web3 library, the DClaims-Ethereum module receives the IP address and port of the running Ethereum RPC and connects to that node.

[6] https://blog.infura.io/

B – Connecting to a smart-contract: To connect to a deployed smart-contract two things are necessary. The first is the contract address, which is a regular Ethereum address (just like a personal account one). The second is the smart-contract's *Application Binary Interface*, an object that defines the interface of the smart-contract, that is, the names of the functions it exposes, as well as the data types that should be passed as arguments to interact with such functions.

C – Making Calls and Transactions to a Smart-Contract: There are two ways to interact with a smart-contract. The first is making a call to it, which essentially translates to a reading operation. A call can be made by anyone running an Ethereum node and is free, meaning it costs no Gas (Ether). The second way is through *Transactions*, which are just like *write* operations. They change the state of a smart-contract, for that reason, they cost gas. The DClaims-Ethereum module exposes functions that other modules can use to interact with the smart-contract, for call and transaction operations. Using the Application Binary Interface (ABI), the module ”translates” regular javascript functions to something that the Ethereum smart-contract can interpret.

4.2.3 Storage Module

The DClaims-Storage module is responsible for interacting with IPFS, through the IPFS-JS-API and with Ethereum, through the DClaims-Ethereum module. The storage module retrieves the claim IPFS links from the Ethereum smart-contract, using the DClaims-Ethereum module and then retrieves the claims from IPFS. This last task is particularly important in the case of batched claims since a batch claim only contains the IPFS links to single claims, not the claims themselves. For that reason, the first part of the process consists of taking the IPFS links inside a batched claim, then retrieve them from IPFS and then return them to the module that made the request.

The main exposed functions are:

- `addItem(dclaim)`: Issues a new claim by, first, storing it on IPFS and, second, making a transaction to the smart-contract.
- `getClaimsListFromIPFS(topic)`: Returns the claim files from IPFS
- `getClaimsCount(topic)`: Returns the number of claim links for a given topic. Note that this is not the same number as claim issued. Claims can be issued in batches, and for every batch there is only one entry in the smart-contract.
- `getFileFromIPFS(ipfsLink)`: Returns a claim from IPFS.
- `addFileToIPFS(file)`: Adds a claim to IPFS

4.3 Publisher Module

The DClaims-Publisher module is a web server which exposes an API. It is implemented in Node.js, using the Express library [7] and the DClaims-Core module. The API allows for applications to request publishers to issue claims. Publishers batch the requests and issue them periodically. This software module is responsible for exposing the publisher API, keep track of the issuance requests and issue claims in batches.

A – API: The Publisher API is the interface that applications can use to issue claims using publishers. It currently exposes only one method, `/issueclaim`. As a parameter, it receives the IPFS link of the claim to be issued. Note that the way application issue claims with publishers are by first storing the claims in their local IPFS nodes and then sending the claim links to the publisher.

B – Buffering and Issuing Claims: Publishers do not issue claims as soon as they receive them as they are meant to be issued in batches. To that end, they keep track of the claims in a hash-map, where the key is the topic of the claims. When the number of claims for a topic reaches a certain threshold, the Publisher creates a new batch claim. After having created the batch, it issues it in a single transaction, using `DClaims-Core.issueClaim()`.

4.4 Ethereum Smart-Contract

The Ethereum smart-contract maintains a record of all the claims issued using the dclaims platform. The smart-contract maintains the IPFS links for each topic in a *mapping*[8] data type, the variable is called `claimLinks` which is essentially a *key,value* store, where the *key* is the hashed claim topic and the value is a list of structure elements composed by the issuer's Ethereum wallet address and the claim's IPFS link. The wallet address is saved as an *address* data type (a 20 byte array, the same size of an Ethereum address) and the IPFS link as a *bytes[32]*, which corresponds to a *string*. The smart-contract exposes one transaction function, `issueClaim(topic,ipfsLink)` and two call functions, `getClaimListSize(topic)` and `getClaim(topic,index)`:

- `issueClaim(topic,ipfsLink)`: This transaction function adds a new entry on `claimLinks`. The issuer address can be attained by the `msg.sender` property, which can not be spoofed as it is automatically filled by Ethereum (based on the account that signed the transaction).
- `getClaimListSize(topic)`: This call function returns the size of `claimLinks` by accessing the `.length` property.

[7] https://www.npmjs.com/package/express
[8] http://solidity.readthedocs.io/en/develop/types.html?highlight=data%20type#mappings

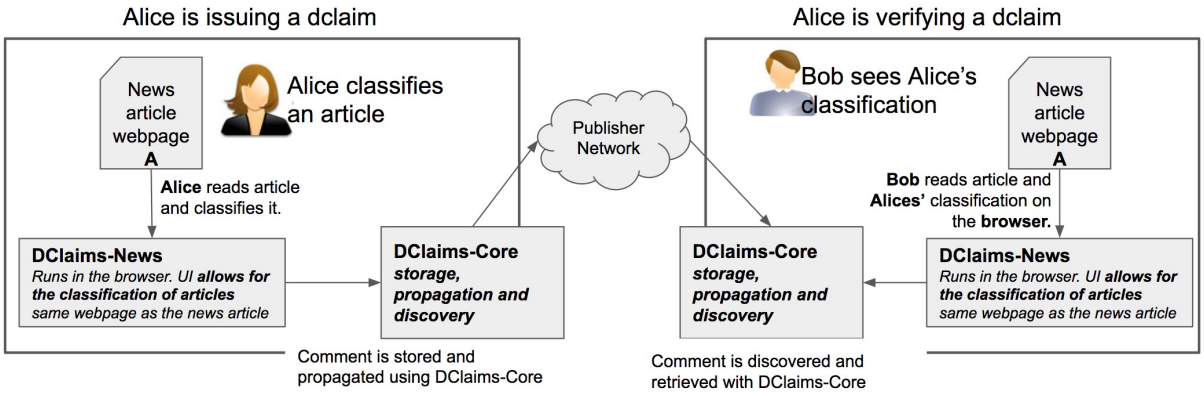


Figure 4.2: DClaims-News Basic Architecture.

- `getClaim(topic,index)` : This call returns an individual item from `claimLinks`. The item is accessed by `claimLinks[topic][index]`.

The `claimLinks` mapping maps hashed topics to a list and at the date of writing this document, Solidity does not allow for lists to be returned in functions. For that reason, the way to get all the elements from a Solidity list is to first query the smart-contract for the size of the list, to learn the number of elements, and then query each element individually.

4.5 DClaims-News

As a proof of concept of an application using the DClaims platform, we built a web annotation application for news websites to allow users to classify, and view classifications, on news articles. There was no significant interest in making the application feature rich since the many existent web annotation applications already cover that ground. The primary goal of developing the application was to provide a reference implementation to dclaims-apps verification, authentication and revocation mechanisms.

Figure 4.2 shows the basic operation of the platform. Alice visits a news website and reads a news article. Her browser has a DClaims-News browser extension, which allows Alice to classify on the news article she is reading, directly on the news article's website. DClaims-News then sends the classification to DClaims-Core, which handles storage, propagation of the classification. Later, Bob visits the same news article website. He also has the DClaims-News browser extension installed. Upon opening the article's website, the browser extension uses DClaims-Core to request the classification made about that news article. Alice's classification (which, let's assume, until that point had been the only person classifying) is then displayed to Bob on the news article's website. The classifications are recorded as a dclaims-news-claim data format, which can be seen as a form of web annotation. We decided to use our data format due to the complexity of the W3C's standards recommendation.

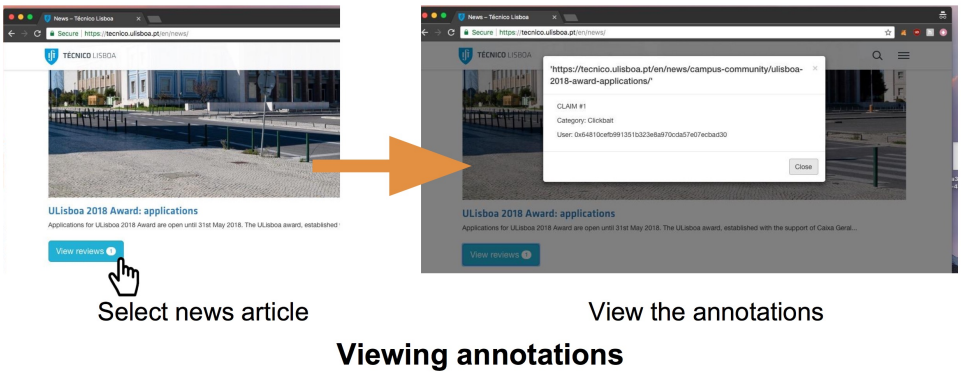


Figure 4.3: Using The DClaims-News Application To View Web Annotations.

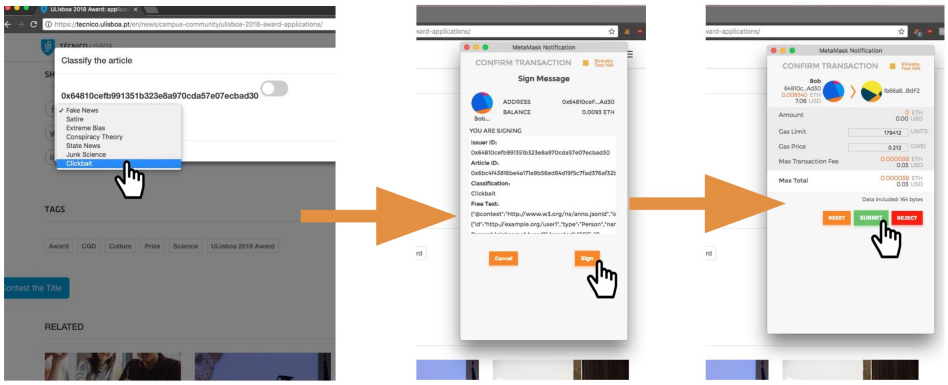
4.5.1 Interaction With News Websites

The Dclaims news platform is a browser extension that runs on top of most news websites and that allows users to classify and view comments about news articles. In this section, we are going to highlight the application's features and workflow.

The browser extension allows users to classify news articles and view classifications about news articles. News websites usually have two types of web pages. The first type is the homepage where a summary of some articles (usually the most relevant at any point in time) is presented. This webpage will be referred to as the webpage. The second type is web pages dedicated to news articles, meaning a webpage where an article is displayed in full, these will be referred to as article pages. Figures 4.3 and 4.4 illustrate how the application works.

A – Behaviour on the Homepage The features made available by the extension on the homepage are different from the ones on the article pages. On the webpage of a news website, there are many articles displayed, and DClaims-News provides two features. The first is to display a counter next to each article which shows the number of classifications made for that article. This is the sum of the number of classifications all users have made about that specific news articles. The second feature is to allow users to read the classifications made about that article. To do that a user clicks a button that reads "View reviews" and UI element pops up which displays the classifications created by users in the user's whitelist, more about this on Section 3.3.5. This process is illustrated by Figure 4.3.

B – Behaviour on Article Pages On an article page, DClaims-News allows the user to classify (issue a dclaim) on the article of the page. Commenting an article consists of classifying it on one out of seven categories, as shown in Figure 4.4. To do that, the user clicks the button which reads "classify this article" that appears on the bottom of the page. Next, the user is prompted to sign the dclaim, using his Ethereum Wallet (the signing process is handled by the Metamask web extension). Finally, if the user is



Choose the classification | Sign the dclaim | Confirm the transaction*

Creating annotations

*This last step only occurs if the user is not using the publishers network.

Figure 4.4: Using The DClaims-News Application To Create Web Annotations.

issuing her own dclaims (instead of using the publisher network), she is asked to confirm the Ethereum transaction. To issue the dclaims using the publisher network, the user would simply need to activate the toggle next to the Ethereum wallet address, in the ”Choose the classification” step.

4.5.2 DClaims-News Data Format

In Section 3.2.1 we presented the dclaims data model as the data format for the Dclaims platform. This data format supports application-specific claims, which are nested inside, such as web annotations. This ensures that each dclaim has the standard fields (UID, Signature, Revocation info, Date, Issuer, Topic) while allowing for extreme flexibility by enabling each application to extend the claim to its needs.

The payload used by DClaims-News is a dclaims-news-claim. It includes a version field, which keeps track of the version of the news article (a hash of the body of the news article) and a classification field, which keeps track of how the issuer evaluated the article. It is this the evaluation field that enables a user to classify an article as fake news, fake science, biased, among others. The system currently supports the seven different types of classifications shown in Figure 4.4 , which are based on the bs-detector[9] classification. We now offer an overview of how the data model is used:

A – Dclaim Fields

- **UID** The UID of a dclaims-news-claim contains if formed by the DClaims-News namespace, which is DClaims-News concatenated with a UUID (RFC4122 v4[10]).

[9] http://bsdetector.tech/
[10] https://tools.ietf.org/html/rfc4122

- **Type** Provides information about the type of signature and revocation that should be expected as well inform about the application specific fields. Points to an IPFS object that describes the description of the signature, verification and revocation mechanisms.

- **Issuer ID** The DClaims-News browser extension uses the Metamask web extension to connect to Ethereum and to sign claims. For that reason, we are using the Ethereum Wallet Address of the Issuer as the Issuer ID. In Section 3.2.5 it was stated that one of the reasons to use Publishers was not to require the users to run extra software, such as running an Ethereum node and also do not require the users to maintain an Ethereum Wallet with funds, so it may seem like a contradiction to be using the Ethereum Wallet address and an ID mechanism. Addressing the point of running extra software, Metamask is a lightweight client that works as an Ethereum endpoint. It does not require the user to run an Ethereum full node. Furthermore, the user can still choose to use the Publisher network as an Ethereum provider. Metamask is only absolutely required to sign the claims. This is an application level choice, a compromise for usability. A user can maintain an Ethereum account, using Metamask, not have any funds in it and still make full use of the system.

- **Topic** In DClaims-News, dclaims are indexed by the Article ID, which is explained in 4.5.3. The topic field has the SHA-3[11] hash of the news article's URL.

- **Revocation** As shown in Section 3.3.4 revocation in Dclaims news is done by issuing claims directly to the smart-contract, for that reason this field contains the address of the smart-contract of the Dclaims news application. This allows for future compatibility in the case that the application evolves to a state where the revocation claims are hosted on a separate smart-contract. Regardless of revocation claims being hosted in the same place of the news classification claims, the verifier always knows what smart-contracts to query.

B – DClaims-News Application Payload

- **Classification** This is the field users use to classify the news article. The classification currently supports seven categories: , Fake News, Satire, Extreme Bias, Conspiracy Theory, State News, Junk Science, Clickbait.

- **Article Version** The article version is used to match a claim to a specific version of the article. This prevents the author's of the article to completely change it after creation and change the context of the claims issued for that article. An example would be an article that denied the effects of global warming. This article would receive several claims classifying it as fake science. The author could then go back and change the article to say that actually, the effects of global warming are

[11] https://www.sha-3.com/

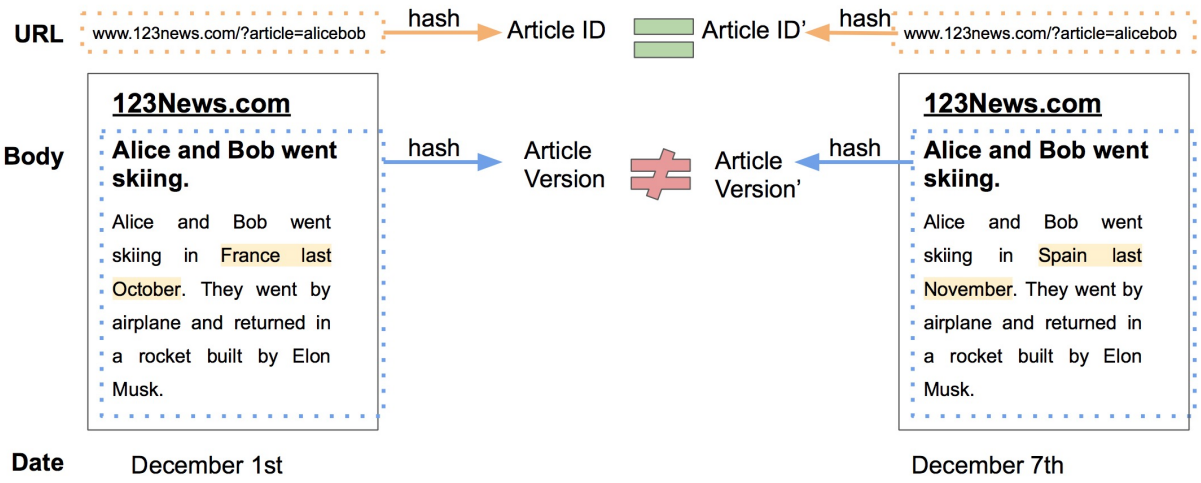


Figure 4.5: Generating An Article Id From An Article's Webpage

devastating. Without a mechanism to pin the version of the article to a claim, the previously issued claims could be interpreted as being denying climate science (since there would be classifications of fake science for an article that was now defending climate science). Section 4.5.3 expands on this topic.

4.5.3 Identifying News Articles

In DClaims, news articles are identified by their URL, as shown in Figure 4.5. Version control is done by hashing the body of the text. This means that if an article is changed (spelling error corrected, paragraphs eliminated, retractions, among others), its identifier will do the same, but the version will change. As described in Section B –, DClaims data structure, each claim has both the identifier of the news article it refers but also has the version of that article. This allows us to notify users that a claim they see about that article might have been created when the article's content was different, meaning that the context might have changed. Figure 4.5 shows how to different versions of the same news article (one published on December 1st and another on December 7th) yield the same article ID but different article versions.

4.5.4 DClaims-News Implementation

A – Visual Elements The visual elements module is responsible for a visual overlay that is placed on top of the news websites, which allows for users to interact with the application. To draw the visual elements (buttons to interact with the application) we injected several javascript files (including the Boot-

strap[12] and jQuery[13] libraries) via a Chrome browser extension[14] , that changed the HTML of the web pages. The Chrome extension was configured using the `manifest.json` file.

B – Using DClaims-Core The web application uses the DClaims-Core library to send and get data from Ethereum and IPFS. To connect to the Ethereum network we used the Metamask web extension, which exposes the Web3 library - this is the library DClaims-Core uses to connect to the Ethereum network. Claim issuance using Publishers is also mediated through DClaims-Core.

C – Claim Integrity and Authenticity Verification When issuing, even if using Publishers, the users must sign it using its Ethereum Wallet Address. To sign and verify signatures we used the Ethereum Signed Type Data method (which is made available in the browser environment by Metamask). When a user creates a claim (by classifying an article), a pop-up box from Metamask appears, showing the piece of data the user is about to sign.

4.6 Summary

In this Chapter, we started by presenting the technologies used to develop and implement the DClaims system. Next, we presented the developed software stack, starting with the DClaims-Core module, including its API, ethereum and storage sub-modules. After that, we presented the publisher module and the Ethereum smart-contract. Finally, we analysed the DClaims-News application, which allows for end-users to create and view web annotations on news websites, in a decentralised and censorship-resistant way. Next, in Chapter 5, we test the DClaims system, in order to evaluate its scalability, cost and performance.

[12] https://getbootstrap.com/docs/3.3/
[13] https://jquery.com/
[14] https://developer.chrome.com/extensions

5

Evaluation

Contents

In our evaluation, we wanted to gauge the extension to which the DClaims system serves as a viable alternative to current web commentary platforms, such as social networks. To make that assessment we analysed several parameters. First, in Section 5.1, we evaluate the performance of the DClaims-News browser extension, to assess if end user's web browsing experience is impacted by having her running DClaims. Next, in Section 5.2 we studied if the system could deal with the level of requests that a popular platform such as Facebook received, and calculate how much this system would cost to implement. The first evaluation reviews the end user's application, while the second, provides insight into the publisher network. The two tests were conducted using different techniques, so the methodology of each one is presented in each of the test's section.

5.1 Performance of the User Interface

For the performance evaluation of the user interface, we used two different testing scenarios. To test the web extension's performance, we injected Javascript to measure the loading time of the webpage. The measured values corresponded to the elapsed time between the `requestStart` and `loadEventEnd` events provided by the browser. For our tests, each webpage was loaded thirty times on each condition (with and without DClaims running). Our experiments were conducted on a computer equipped with 2.4GHz CPU and 8GB memory connected to 20Mb network. The web extension was connected to the IPFS network through a daemon running locally on the computer and connected to an Ethereum testnet via Metamask, a web extension that acts as an Ethereum proxy.

To test the performance of IPFS and Ethereum, we ran DClaims on sixty nodes on Amazon Web Services. Each node had 4 GB of memory and was running IPFS and Go-Ethereum (Geth) on Docker containers. The goal of this test was to simulate a regular usage by ordinary users. To do that we started by having each node randomly issuing five dclaims, each about a randomly selected article (from a pre-selected list of thirty, which corresponded to the articles on SkyNews' webpage on January 28th). To avoid the extra burden of configuring a different account on each node, the dclaims were issued sequentially so that all nodes could share the same Ethereum address.

After the dclaims having been issued, each node started fetching dclaims, randomly selecting an article (from the same list used for the issuance) and fetching all the dclaims for that article. Each node selected a new article every 10 seconds. We ran this experiment for twenty minutes, which resulted in each node querying 120 articles. The overlap (querying 120 articles from a list of 60) was intentional to increase the odds of every article being queried at least once.

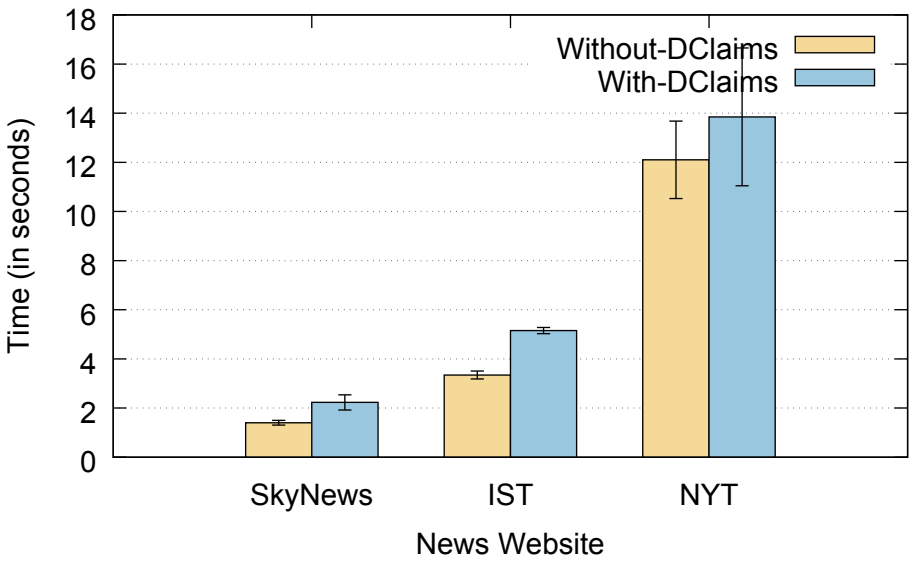


Figure 5.1: Loading Time Comparison Of Popular News Websites With And Without DClaims Running.

5.1.1 Performance of the Web Client

In this section, we report the performance evaluation results of our DClaim's web extension. To provide an idea of the impact of DClaims to end-users' experience, we adapted our web extension to support three websites: SkyNews, New York Times (NYT), and Instituto Superior Técnico (IST)—the news front page of our university's website.

Figure 5.1 shows the time that the three news websites take to load, with and without DClaims. The overhead DClaims introduces is expected since the web extension needs to connect to an IPFS node, to an Ethereum node and then, for each article, it needs to generate the news article ID (which is the SHA-3 hash of the referenced news article's URL).

To better understand the impact that the number of news articles had on DClaims' introduction overhead we conducted a benchmark test whose results are displayed in Figure 5.2. In this case, we used IST's news website as a basis and created ten different versions of the website, each one with ten more articles than the previous one. We also stripped the website of the Javascript code it had, to ensure that DClaims was the only source of performance overhead. This test showed us that the overhead introduced by DClaims increased linearly with the number of articles each page had. In Figure 5.1 Sky News' website should have taken longer to load than IST's, being that the first has more than double the number of news articles (19 from IST, 42 from Sky News) than the second, however, the opposite occurs. Even though there is an increase in the overhead that depends on the number of articles a website has, the determining factor in the website's loading time is the Javascript running on the website. This is especially noticeable in the NYT's website, where the standard deviation is exceptionally

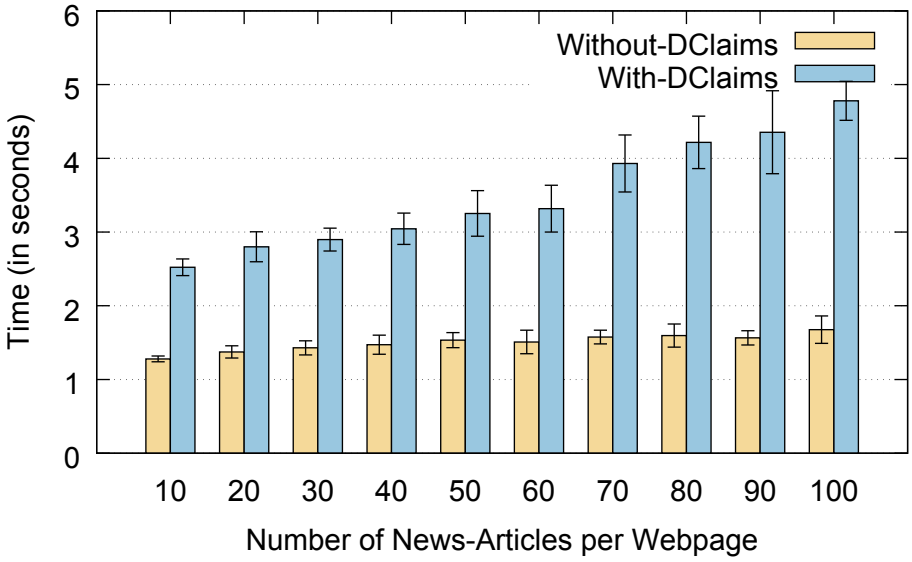


Figure 5.2: Time To Load A Webpage As Function Of The Number Of Articles.

high due to the substandard javascript code it has. Furthermore, the latency introduced by DClaims does not affect the user experience, as the original elements of the website (news titles, images, among others) appear just as fast as they did before, only the elements introduced by DClaims (view dclaims button, dclaims counter) take longer to appear. In conclusion, the performance overhead introduced by running DClaims-News is negligible in the sense that it does not impact the user experience. Users can use the news websites with DClaims-News running, with the same ease as they would without the web extension.

5.1.2 Performance of IPFS and Ethereum

IPFS and Ethereum are a crucial part of DClaims since dclaims are stored in a combination of the two. For that reason, we wanted to evaluate the performance their performance concerning the time it takes to retrieve dclaims. The backend is being evaluated in regards to the time it takes to retrieve claims. Evaluation of the time it takes to issue dclaims is not relevant as most of the time is spent on waiting for Ethereum's transaction confirmation and not DClaims' operations. In this experiment we had sixty nodes running DClaims, each node made five dclaims about random news articles (from a poll of twenty) and then fetched dclaims from random articles, fetching a new article every ten seconds, for 20 minutes. The process of retrieving dclaims from DClaims, for a given article, starts by retrieving the list of claim IPFS links (where retrieving each link corresponds to an Ethereum call) followed by retrieving each claim from IPFS. To evaluate the time dclaims take to be retrieved, we measured the time it takes, for a given article, to get a list of claim links from Ethereum as well as the time it takes to retrieve individual

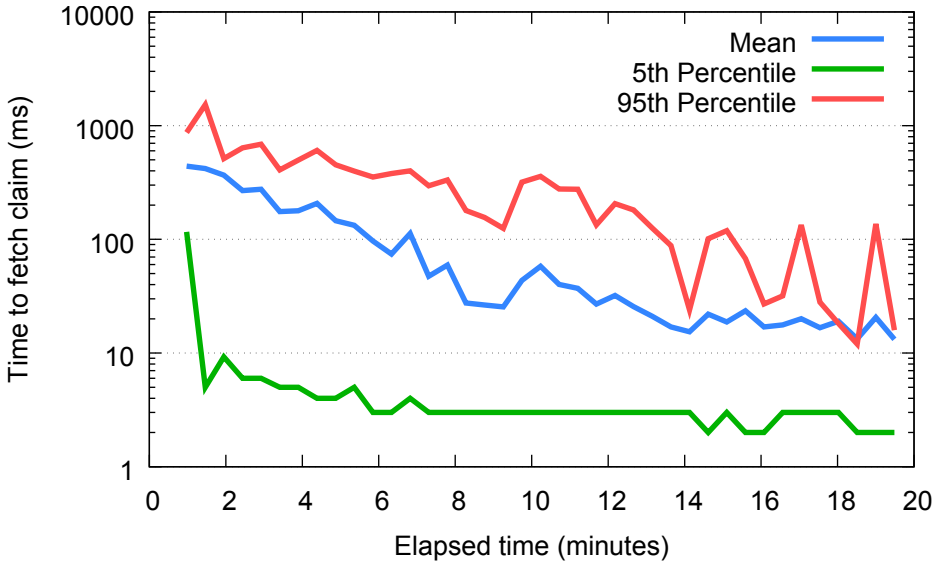


Figure 5.3: Time To Fetch Individual Claims From IPFS

dclaims from IPFS.

Figure 5.3 shows the time it takes to retrieve individual dclaims from IPFS. The 95th percentile is represented by the red line, the 5th percentile by the green line and the mean by the blue line. Fetching times under 10ms represent cases where the node had the claim stored locally (meaning that such claim had been created by this node or that it had previously fetched it from other nodes and kept a cached copy).

As we can see, at the beginning of the experiment (in the first minute) most of the dclaims are being fetched from other nodes, which is why there are no fetching times under 10ms (the small number of dclaims that were available locally were treated as outliers). As the experiment goes on, nodes start having local copies of the dclaims (everytime a new claim is fetched, a copy is maintained in cache), and fetching times start decreasing very quickly (notice that the Y axis has a logarithmic scale). The discrepancy between the 95th and 5th percentile is explained by the difference in time to fetch a claim that is stored locally from one that has to be retrieved from another node.

Around minute 14 the system stabilises. At this point, most nodes have local copies of all the dclaims, which is why the meantime to fetch is only slightly over 100ms. There are still some dclaims that have to be retrieved from other nodes, but with a much lower frequency than in the early stages.

The time span with the most interest to study is the one between minutes two and seven because it represents the period where most dclaims still have to be retrieved from other nodes, which best represents the system working in the real world. In a real-world deployment, new dclaims would continuously be being generated, and most nodes would not have local copies of those dclaims.

Figure 5.4 shows the time to retrieve the list of dclaims from Ethereum. Requesting the dclaims of

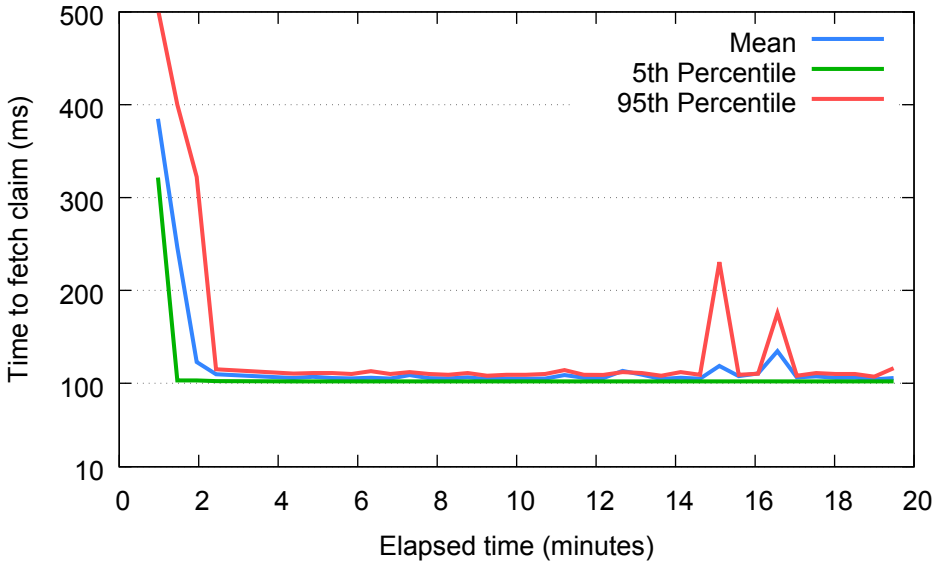


Figure 5.4: Time To Retrieve The List Of Claims From Ethereum.

a given article with *n* dclaims results in *n+1* requests to Ethereum, one requests to find the number of dclaims for that article, and then a request per claim to get the ID of the claim (this occurs because Solidity, the smart-contract programing language does not yet support returning lists. If it did support returning lists, the number of requests to Ethereum would be 1, instead of *n+1*). Looking at Figure 5.4 we can see that in the beginning (first minute) the values are remarkably higher than in the rest of the elapsed time. We attribute this to Geth (Ethereum daemon) establishing connections to other nodes in preparation for querying them for information. This occurs because we are using the light sync mode, meaning that Geth only gets the current state of the ethereum blockchain, rather than the full ledger, and for that reason needs to query other nodes to get information about previous blocks. We used the light sync mode due to time and storage constraints. Light sync mode takes about 30 minutes to sync while full sync mode takes more than 3 hours[1] . After having established the connections to other nodes, the requests to Ethereum take about 100ms to be completed. The outliers in minute 15 and 17 were due to some node's Geth losing connection to other nodes (this occurs because we are using a testnet with a small number of nodes that can quickly become unavailable. The same would not occur in the main net).

Figures 5.4 and 5.3 show the scalability of DClaims. Ethereum request times are constant while IPFS' decrease over time. This result is expected. Over time, more IPFS nodes have start having the files cached and do not need to request from other nodes. This also demonstrates that in a real-world scenario, if IPFS nodes were distributed all across the globe, the time to retrieve dclaims from the IPFS network would decrease because nodes closer to the user requesting the dclaims would have those

[1] The 2.5-hour difference on the 60 nodes amounted to 150 hours on AWS costs

Table 5.1: Activity Of News Pages On Facebook

Facebook Data	
Time duration analysed (hours)	24
Number of pages analysed	4
Average number of followers per page (Millions)	27
Total number of posts analysed	200
Total number of interactions	1214549
Average number of posts (per page, per day)	50
Average number of interactions (per post, per day)	6073
Average number of interactions (per page, per day)	303637
Average length of comment (characters)	148

dclaims cached, and could serve the files quicker than nodes further away. As for Ethereum, the time to get responses remains constant because the requests are local requests, each Ethereum node has a copy of the Ethereum blockchain.

5.2 Evaluating DClaims Costs

In this section, we determine the cost of a full-scale deployment of DClaims. We start, in Section 5.2.1, by estimating the level of activity our system would have to endure, using Facebook data as a proxy. Next, in Section 5.2.2, we calculate the costs based on the considered activity level. Finally, in Section 5.2.3, we provide an analysis of the cost of the system, offering an example as to how it compares to real-world systems in use today.

5.2.1 Analysis of News Pages on Facebook

We analysed Facebook data to learn the level of activity our system would have to support. In our system, users perform a task similar to the one of commenting on Facebook posts. At the same time, one of the primary uses for the platform will be for users to annotate news websites. For these reasons, we decided to use the rate of interactions (comments, likes, reactions) on four of the most active Facebook's News organisation pages as a proxy for the activity level that we might encounter in our system. That is, we analysed the rate of interaction that end-users have with these pages, and used those levels of activity to evaluate the cost and performance of our system.

A – Methodology: We used the Facebook Graph API[2] to analyse the posts of four of the largest Facebook news pages (CNN, Fox News, The New York Times and BBC News), which average 27 million users each, during 24 hours (a limitation of the API). For each Facebook post on these pages,

[2] https://developers.facebook.com/docs/graph-api/

Table 5.2: Cost And Performance Analysis

Storage	
Average size per web annotation (KB)	30
Data generated per day, per News Website (GB)	9,11
Annual cost of storage on AWS (USD)	**2203**
Computation	
Operations per second to support 300 active articles	25
Anual cost on AWS T2.XL Virtual Machines (USD)	**1880**
Ethereum Transactions	
Batch size	100
Publishing waiting time (min)	30
Number of transactions per day	3037
Annual transactions cost (USD)	**277069**

we obtained all the comments and a count of the number of likes and reactions[3]. From that dataset, we were able to calculate the values presented in Table 5.1.

B – Results: Observing Table 5.1, we can see that, on average, each news facebook page posts 50 news articles per day, and each post receives around 6073 interactions per day. This means that the activity level DClaims needs to support, per day, per news organisation, is 303637, which corresponds to the activity level per post, multiplied by the number of posts.

5.2.2 Estimating DClaims' Cost and Performance

We used the activity values presented in Table 5.1 to estimate, per News Website, the load our system would have to withstand. We assumed that the volume and rate of web annotations our system would receive, per news outlet, is that same that the news Facebook pages receive. We separated the load analysis into three components, storage, computing power and Ethereum transactions. Since the activity level depends on the number of news organisations the system is supporting, we made the calculations per news organisations. This means that the results, presented in Table 5.2, represent the cost of running DClaims, for one year, to handle the activity level generated by a large news organisation (such as CNN, Fox News, The New York Times or BBC news). Next, we analyse values from Table 5.2.

A – Storage: To calculate the necessary storage of our system, we started by measuring the storage cost of individual annotations. To do that we measured the average length of Facebook comments on those news pages, which is 148 characters. We inserted a 148 character string into a default web annotation (the schema used is the W3Cs standard), then placed the web annotation inside a dclaim, which was then digitally signed. We repeated this procedure for 10000 dclaims which were then added

[3]Facebook reactions are interactions similar to *likes*, which allow to express the following feelings: *love*, *haha*, *wow*, *sad* and *angry*.

to our test machine's IPFS repository. We calculated the increase in storage in the repository and divided by the 10000 dclaims, which resulted in the measurement that each claim occupies 30KB of storage. We took special care measuring the size of the repository as we were adding the dclaims to ensure that its size increased linearly, that is, that each claim continues to add 30KB to the repository and not more (or less) than that.

Now that we had the size of each claim, we proceeded to calculate the storage requirements for an entire year. Using the Average number of interactions (per page, per day), from 5.1 we were able to calculate the daily storage increase, per news outlet, which is 9.11GB, as shown in Table 5.2's *Data generated per day, per News Website (GB)* . From there we multiplied by 31, which gave us the monthly volume. We multiplied the monthly volume by Amazon Web Services' storage price[4] , which gave us the monthly price. We then calculated the price for one year of operations, taking into account that the price of each month's increase has to be added to the previous month's price. This resulted in the value of USD 2203 in storage costs. We consider the storage costs to be low. They represent less than 1% of the total cost of the system. If need be, this value could be optimised by using less expensive storage options, such as hard drive disks for storing older content.

B – Computation: To calculate the computation costs and performance, we started by calculating the number of requests our system would have to process, and then stress-tested a Publisher server running DClaims to evaluate how many servers would be necessary per news outlet. From there we could calculate the annual cost.

We started by benchmarking a Publisher server running DClaims to see how many issuance requests per second it would be able to handle. To do that we used the *Artillery.io*[5] test framework to simulate a high volume of requests. We make a total of 3000 requests over a period of 100 seconds, 30 requests per second, and evaluated that the server dropped 2% of the requests. We concluded that the server could withstand 25 requests per second.

We used Table 5.1's *Average number of interactions (per post, per day)* to estimate the number of requests per second we would receive per news article, to then see how many active articles one server would support. With each server supporting 1500 requests per minute (25 per second), and the interaction level per article being 4.2 per minute we calculated that each server could serve around 357 active articles simultaneously. This number is well above the *Average number of posts (per page, per day)* from Table 5.1, which means that one server can serve the number of requests expected per news website. The server used was Amazon Web Service's T2.2xlarge, which costs USD 1880 per year[6].

[4] EBS General Purpose SSD: https://web.archive.org/web/20180503104411/https://aws.amazon.com/ebs/pricing/
[5] https://artillery.io/
[6] https://web.archive.org/web/20180503113204/https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/

Table 5.3: Ethereum Price Calculations

Fixed Values	
Ethereum Fiat Value (USD)	631
Publisher's Batch Size	100
Transaction Gas Price (Gwei, nanoEther)	3
Results	
Batch filling time (min)	23
Transaction Confirmation Time (min)	5
Price per Transaction (USD)	0,25

Table 5.4: Ethereum Confirmation Time As A Function Of The Gas Price

Category	**Gas Price (Gwei)**	**Time (min)**
Safe-Flow	2	30
Standard	3	5
Fast	5	2

C – Cost of Ethereum Transactions: To calculate the costs on Ethereum transactions, three variables had to be fixed. More precisely, the value of Ether (the Ethereum blockchain's currency) in fiat currency, USD[7] , the size of the batch the Publishers would use and the Gas Price for the Ethereum transaction confirmation. The selected values and calculations are presented in Table 5.3.

The Gas Price of a transaction influences how long that transaction will take to be confirmed. The gas price of a transaction is directly proportional to the reward an Ethereum miner node receives for processing such transaction. So, miners are incentivised to process transactions with higher gas prices faster. The time a transaction takes to be confirmed on the blockchain as a function of the transaction's gas price, varies over time. When the network is operating at peak capacity (many transactions), gas prices rise, when it is at average or low capacity (lower number of transactions). To choose the gas price for our calculations, we used a popular website for Ethereum consumer metrics (EthGasStation[8]), the values are shown in Table 5.4. We chose the standard value, 3 Gwei[9] , for it is a good compromise between waiting time and price. 30 minutes was too long to wait, and 5 Gwei was too expensive.

The next step was to define the Publisher's batch size, that is, the number of dclaims Publishers will batch together and issue in the same Ethereum transaction. The size of the batch is a delicate balance between cost and time. If the batch size is too small, it will result in a high number of Ethereum transactions, which is expensive. On the other hand, if the batch size is too high, it will take too long to fill up, and end-users could end up waiting hours to see their dclaims Published. Looking at Table 5.1's *Average number of interactions (per post, per day)* we know that the number of interactions per article

[7] Value consulted on April 26th, 2018, from https://web.archive.org/web/20180426195440/https://coinmarketcap.com/currencies/ethereum/

[8] Value consulted on April 26th, 2018, from https://web.archive.org/web/20180426195130/https://www.ethgasstation.info/

[9] 1 Gwei = 1 nanoEther

Table 5.5: Final Analysis Of Costs Per News Outlet

Final Costs	
Storage	2203
Computation	1880
Ethereum	277069
Total cost for 1 year	281152
Useful Metrics	
Cost per 1000 Claims (USD)	2,54
Cost per User for 2,7M users (USD)	1,041

is 4.2 per minute, which means a batch of size 100 would take 23 minutes to fill and would reduce the transaction cost per claim by a factor of 100 (when compared to one transaction per claim). 23 minutes per batch with a 100 fold reduction in cost is a good compromise, so the selected batch size for this test was 100.

Now that we know the gas price (3 Gwei) and batch size (100), we can calculate the yearly cost in Ethereum transactions, which is of USD 277069 as presented in Table 5.2. It is important to note that this is the cost of the entire system, the cost per user is small. In Section 5.2.3 we provide an analysis of the cost.

5.2.3 Main Findings of the Cost Analysis:

By adding the costs of storage, computation and Ethereum, we arrive at the cost of the system for one year and news outlet. The results are summarised in Table 5.5.

Running the DClaims system for one big news outlet, such as CNN, Fox News, BBC News or The New York Times, would approximately cost USD 281152 per year. This value was calculated for only one of these large news outlets, so the value presented does not represent the real world cost. However, even if we assume that around the world there are 30 news outlets the size of the ones analysed, the costs are still at significantly smaller than the ones for real-world systems with donation based financial models, such as Wikipedia, as shown in Figure 5.5.

These values may seem high but should be put in perspective. The four Facebook news pages analysed have, on average, 27 million users. Even if we assume that DClaims only attracts 1% of those users the cost per user, per news outlet, would less than USD 1 per year. Alternatively, USD 30, assuming a network with 30 publishers. Moreover, this is maintaining the same number of interactions per post, meaning that a smaller number of people is dividing the same cost and level of activity. DClaims is a system built for users who need to circumvent censorship. Many people pay monthly fees for services such as VPNs (also a defence against censorship), ranging from USD 5 to 10 per month, which equals USD 60 to USD 120 per year. There is no way to calculate the number of people who use VPNs for censorship resistance since there are other reasons to use one, but it is clear that there is a market

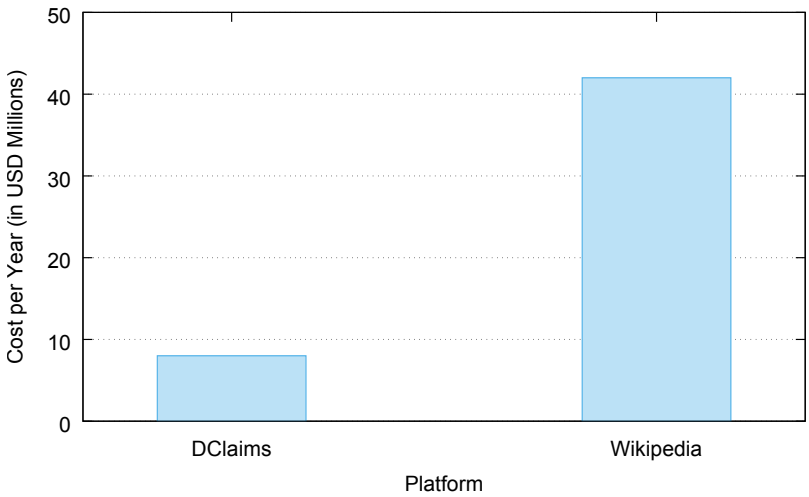


Figure 5.5: Cost Comparison Between A DClaims System For 30 News Outlets And Wikipedia

for kind of product. Therefore, if a donation based financial model such as Wikipedia's did not work, there is reason to believe a subscription-based service, would.

5.3 Summary

In this chapter, we provided an extensive evaluation of the DClaims system. We started by evaluating the performance of the DClaims-News web extension and scalability of the DClaims system. We concluded that the performance overhead introduced by the web extension does not affect the web browsing experience of the end-users when visiting news websites. We also concluded that IPFS' performance improves over time, as content gets more distributed through the network, while Ethereum's performance remains constant. Next, we analysed the costs of a full-scale deployment of DClaims, using the activity level of Facebook's news pages as a proxy for expected demand. We concluded that while the costs of a full-scale deployment of DClaims are significant, it is possible to finance a system like this based on a donation financial model. We also noted that a subscription-based model would be a viable option. Next, in Chapter 6 we present the conclusions of this work.

6

Conclusions

Contents

In this chapter, we start by offering, in Section 6.1 the main results of this thesis, followed by a discussion of the system's limitations and possible directions for future work, in Section 6.2

6.1 Summary

Over the last two decades, the Web has become the primary source of information for humans living in developed countries. On the other hand, this shift has been accompanied by the dissemination of unreliable or even misleading information throughout the Web. Over time, commentary sections on Web sites and social networks have become popular platforms for people to exert their freedom of speech, by correcting inaccurate information, providing additional context or engaging in discussions. However, existing Web platforms tend to be controlled by single authorities which, if the shared information is considered to be inconvenient to their interests, can engage in censorship by denying others users access to such content.

To solve this problem, we built DClaims, a decentralised web annotations platform which is resistant to censorship. DClaims stores data in a distributed network and keeps a registry of metadata on the Ethereum blockchain, which is a tamper-proof, permanent record of information. Blockchain technology has some limitations. It is expensive to use, and hard to scale. To address this issue, we created a novel blockchain-based architecture, which makes use of a small network of dedicated nodes who batch transactions together. This results in a decrease of blockchain transactions cost and increases the number of operations the system supports per unit of time. We built a reference implementation of the system on the form of a browser extension, which allows for the web annotation of news websites, allowing users to classify news articles, and view the classifications made by others.

Our evaluation of the system shows that it can support the same level of activity of Facebook's news organisations pages (such as CNN, Fox News and BBC News). The architecture is built in a way in which the bottlenecks of the Ethereum blockchain are not a limiting factor in the level of activity DClaims can support. We also show the system can scale to support higher levels of activity if necessary. The cost of running DClaims on a global scale, supporting around thirty news organisations the size of the ones analysed (CNN, Fox News, BBC News and The New York Times), is five to six times the cost of running a service such as Wikipedia. We use Wikipedia for comparison due to the economic model for supporting Wikipedia – donation based – is similar to the one we envision for DClaims. We also evaluated the user interface's performance, which showed that running DClaims in a browser for web annotating news websites has little impact on the user experience.

6.2 System Limitations and Future Work

We identify three main limitations on DClaims, data replication and overall cost. This section offers an overview of each, and possible solutions to be carried by future work.

The replication mechanism implemented by DClaims results in the system's availability being dependent on publisher servers being available. A way to counter this problem is to implement a more sophisticated data replication mechanism, where publishers replicate data from other publishers. That way, if a publisher becomes unavailable, others in the network can serve the same content. An alternative to this would be to move data storage away from the publisher network and use a service such as FileCoin [64] (once its deployed), which is a distributed storage network, built on top of IPFS, where users can choose the degree of replication of the files they want to be stored.

The overall cost of the system can also be considered a limitation. Currently, 90% of DClaims' cost is due to Ethereum transaction fees. The problem of high transaction fees on the Ethereum blockchain is an active area of research, and several solutions are being developed, which will significantly lower the cost of using the network. These solutions include moving the Ethereum consensus to proof-of-stake protocol [65], sharding the blockchain [66], to reduce the cost of storage, or to use side-chains and state channels, which are way of running computations off-chain, anchoring only the results on the main chain [67–69]. Once some of those solutions are deployed, the cost of DClaims' operation will lower dramatically.

Finally, it would also be interesting to investigate how DClaims could be used to support other kinds of services. The dclaim data format can encapsulate data formats other than web annotations. With some adjustments, the protocol could be used to support platforms such as Wikipedia, or Twitter, both of which could also benefit from censorship resistance.

Bibliography

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” *self-published paper*, pp. 1–9, Mar. 2009.

[2] S. Cushion, A. Kilby, R. Thomas, M. Morani, and R. Sambrook. (2016, May) Newspapers, Impartiality and Television News. [Online]. Available: https://web.archive.org/save/https://medium.com/oxford-university/where-do-people-get-their-news-8e850a0dea03

[3] (2006) Where Do We Get Our News? [Online]. Available: https://web.archive.org/save/https://https://web.archive.org/save/https://www.pbs.org/wgbh/pages/frontline/newswar/part3/stats.html

[4] (2012) How People Get Local News and Information in Different Communities. [Online]. Available: https://web.archive.org/save/http://www.pewinternet.org/2012/09/26/how-people-get-local-news-and-information-in-different-communities/

[5] (2014, Mar.) How Americans get their news - American Press Institute. [Online]. Available: https://web.archive.org/web/20180502011138/https://www.americanpressinstitute.org/publications/reports/survey-research/how-americans-get-news/

[6] (2007) Official’s Key Report On Iraq Is Faulted. [Online]. Available: https://web.archive.org/web/20171023180238/http://www.washingtonpost.com/wp-dyn/content/article/2007/02/08/AR2007020802387.html

[7] H. A. Gentzkow and Matthew, “Social Media and Fake News in the 2016 Election,” *Journal of Economic Perspectives*, vol. 31, pp. 211–236, Apr. 2017.

[8] (2018) Crisis Pregnancy Centers: Last Week Tonight with John Oliver (HBO). [Online]. Available: https://web.archive.org/web/20180502005942/https://www.youtube.com/watch?v=4NNpkv3Us1I

[9] (2018) The EPA’s Website After a Year of Climate Change Censorship. [Online]. Available: https://web.archive.org/web/20180319165205/http://time.com/5075265/epa-website-climate-change-censorship/

[10] C. Buntain and J. Golbeck, “Automatically Identifying Fake News in Popular Twitter Threads,” in *2017 IEEE International Conference on Smart Cloud (SmartCloud)*. IEEE, Oct. 2017, pp. 208–215.

[11] R. J. Sethi, “Spotting Fake News: A Social Argumentation Framework for Scrutinizing Alternative Facts,” in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, Jun. 2017, pp. 866–869.

[12] A. Colborne and M. Smit, “Identifying and Mitigating Risks to the Quality of Open Data in the Post-truth Era,” in *IEEE International Conference on Big Data*, Jan. 2018.

[13] M. R. Siddiki and et al., “CrowdsouRS: A Crowdsourced Reputation System for Identifying Deceptive Online Contents,” in *2017 27th International Conference of Computer and Information Technology ICCIT*, Dec. 2017, pp. 1–6.

[14] J. Kim, B. Tabibian, A. Oh, B. Schölkopf, and M. Gomez-Rodriguez, “Leveraging the Crowd to Detect and Reduce the Spread of Fake News and Misinformation,” in *the Eleventh ACM International Conference*. New York, New York, USA: ACM Press, 2018, pp. 324–332.

[15] L. Wu and H. Liu, “Tracing Fake-News Footprints,” in *the Eleventh ACM International Conference*. New York, New York, USA: ACM Press, 2018, pp. 637–645.

[16] R. J. Sethi, “Crowdsourcing the Verification of Fake News and Alternative Facts,” in *the 28th ACM Conference*. New York, New York, USA: ACM Press, 2017, pp. 315–316.

[17] (2016) Fact-Checking Organizations Around the Globe Embrace Code of Principles. [Online]. Available: https://web.archive.org/web/20180424193829/https://www.washingtonpost.com/news/fact-checker/wp/2016/09/15/fact-checking-organizations-around-the-globe-embrace-code-of-principles/?utm_term=.f2f14a4e4f2c

[18] J. Fábrega and J. Sajuria, “The formation of political discourse within online networks: the case of the occupy movement,” *International Journal of Organisational Design and Engineering*, vol. 3, no. 3/4, p. 210, 2014.

[19] V. Qazvinian and D. R. Radev, “A Computational Analysis of Collective Discourse,” *CoRR*, 2012.

[20] (2006, Jan.) Freedom House Condemns Latest Act of Media Repression in China. [Online]. Available: https://web.archive.org/save/https://freedomhouse.org/article/freedom-house-condemns-latest-act-media-repression-china?page=70&release=325

[21] Censorship of Facebook. [Online]. Available: https://web.archive.org/web/20180424201915/https://en.wikipedia.org/wiki/Censorship_of_Facebook

[22] Censorship of Twitter. [Online]. Available: https://web.archive.org/web/20180424201603/https://en.wikipedia.org/wiki/Censorship_of_Twitter

[23] Making web annotations persistent over time. [Online]. Available: https://web.archive.org/save/https://en.wikipedia.org/wiki/Web_annotation

[24] (2017) Turkey Can't Block This Copy of Wikipedia. [Online]. Available: https://web.archive.org/web/20180425002528/http://observer.com/2017/05/turkey-wikipedia-ipfs/

[25] S. Khattak, T. Elahi, L. Simon, C. M. Swanson, S. J. Murdoch, and I. Goldberg, "SoK: Making Sense of Censorship Resistance Systems," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 37–61, 2016.

[26] D. Fifield, N. Hardison, J. Ellithorpe, E. Stark, D. Boneh, R. Dingledine, and P. A. Porras, "Evading Censorship with Browser-Based Proxies." *Privacy Enhancing Technologies*, vol. 7384, no. Chapter 13, pp. 239–258, 2012.

[27] P. Lincoln, I. Mason, P. A. Porras, V. Yegneswaran, Z. Weinberg, J. Massar, W. A. Simpson, P. Vixie, and D. Boneh, "Bootstrapping Communications into an Anti-Censorship System." *FOCI*, 2012.

[28] Tor Bridges Protocol. [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/attic/bridges-spec.txt

[29] E. Y. Vasserman, N. Hopper, and J. Tyra, "SilentKnock: practical, provably undetectable authentication," *International Journal of Information Security*, vol. 8, no. 2, pp. 121–135, Nov. 2008.

[30] P. Winter, T. Pulls, and J. Fuss, "ScrambleSuit," in *the 12th ACM workshop*. New York, New York, USA: ACM Press, 2013, pp. 213–224.

[31] D. McCoy, J. A. Morales, and K. Levchenko, "Proximax: Measurement-Driven Proxy Dissemination (Short Paper)," in *Information Security and Privacy*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 260–267.

[32] D. Barradas, N. Santos, and L. Rodrigues, "DeltaShaper: Enabling Unobservable Censorship-resistant TCP Tunneling over Videoconferencing Streams," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, p. 367, 2017.

[33] A. Houmansadr, T. Riedl, N. Borisov, and A. Singer, "IP over Voice-over-IP for censorship circumvention," *arXiv.org*, Jul. 2012.

[34] S. Burnett, N. Feamster, and S. Vempala, “Chipping Away at Censorship Firewalls with User-Generated Content.” *USENIX Security Symposium*, 2010.

[35] R. Dingledine, N. Mathewson, and P. F. Syverson, “Tor - The Second-Generation Onion Router.” *USENIX Security Symposium*, 2004.

[36] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet - A Distributed Anonymous Information Storage and Retrieval System.” *Workshop on Design Issues in Anonymity and Unobservability*, vol. 2009, no. Chapter 4, pp. 46–66, 2000.

[37] M. Waldman and D. Mazières, “Tangler - a censorship-resistant publishing system based on document entanglements.” *ACM Conference on Computer and Communications Security*, p. 126, 2001.

[38] M. Waldman, A. D. Rubin, and L. F. Cranor, “Publius - A Robust, Tamper-Evident, Censorship-Resistant, and Source-Anonymous Web Publishing System.” *USENIX Security Symposium*, 2000.

[39] A. Serjantov, “Anonymizing Censorship Resistant Systems.” *IPTPS*, vol. 2429, no. Chapter 11, pp. 111–120, 2002.

[40] V. Buterin, “Ethereum white paper,” *self-published paper*, 2013.

[41] G. Wood, “Ethereum Yellow Paper,” *self-published paper*, 2014.

[42] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, “On the Security and Performance of Proof of Work Blockchains,” in *the 2016 ACM SIGSAC Conference*. New York, New York, USA: ACM Press, 2016, pp. 3–16.

[43] S. King and S. Nadal, “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake,” *self-published paper*, 2012.

[44] E. Wustrow and B. VanderSloot, “DDoSCoin: cryptocurrency with a malicious proof-of-work,” *of the 10th USENIX Conference on*, 2016.

[45] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, “SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies,” in *2015 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2015, pp. 104–121.

[46] S. Bowe and et al., “Zcash Protocol Specification,” *self-published paper*, pp. 1–53, Mar. 2017.

[47] “Permacoin: Repurposing Bitcoin Work for Data Preservation,” pp. 1–17, Jun. 2014.

[48] “Blockstack: A Global Naming and Storage System Secured by Blockchains,” pp. 1–15, Jun. 2016.

[49] S. Park, “Spacemint: A Cryptocurrency Based on Proofs of Space,” *MIT*, pp. 1–18, Nov. 2015.

[50] G. Danezis and S. Meiklejohn, “Centrally Banked Cryptocurrencies,” in *Network and Distributed System Security Symposium*. Reston, VA: Internet Society, Dec. 2015, pp. 1–14.

[51] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, “Proofs of Space,” in *Advances in Cryptology – CRYPTO 2015*. Berlin, Heidelberg: Springer, Berlin, Heidelberg, Aug. 2015, pp. 585–605.

[52] e. a. Wilkinson, “Storj - A Peer-to-Peer Cloud Storage Network,” *self-published paper*, Dec. 2016.

[53] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, “Blockstack: A global naming and storage system secured by blockchains,” *2016 USENIX Annual Technical*, 2016.

[54] M. Ghosh, M. Richardson, B. Ford, and R. Jansen, “A TorPath to TorCoin: Proof-of-Bandwidth Altcoins for Compensating Relays,” *Yale University*, Jul. 2014.

[55] M. Milutinovic, W. He, H. Wu, and M. Kanwal, “Proof of Luck,” in *the 1st Workshop*. New York, New York, USA: ACM Press, 2016, pp. 1–6.

[56] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, “A Secure Sharding Protocol For Open Blockchains.” *ACM Conference on Computer and Communications Security*, pp. 17–30, 2016.

[57] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse, “Bitcoin-NG - A Scalable Blockchain Protocol.” *NSDI*, 2016.

[58] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts,” in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, Apr. 2016, pp. 839–858.

[59] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, “Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs,” in *2015 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2015, pp. 287–304.

[60] V. Buterin, “The Meaning of Decentralization – Vitalik Buterin – Medium.”

[61] J. Benet, “IPFS-content addressed, versioned, P2P file system,” *arXiv.org*, 2014.

[62] J. Zittrain and K. Albert, “Perma: Scoping and Addressing the Problem of Link and Reference Rot in Legal Citations,” *SSRN Electronic Journal*, 2013.

[63] I. Marty, I. Miers, and E. Wustrow, “Proof-of-Censorship: Enabling centralized censorship-resistant content providers,” *Financial Cryptography and Data Security*, pp. 1–18, Jan. 2018.

[64] P. Labs, “Filecoin: A Decentralized Storage Network,” *self-published paper*, Aug. 2017.

[65] D. Ryan and C. Liang. (2018, Apr.) Casper Specification. [Online]. Available: https://web.archive.org/web/20180510123524/https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1011.md

[66] Ethereum Foundation. (2018) Sharding introduction R&D compendium. [Online]. Available: https://web.archive.org/web/20180510125204/https://github.com/ethereum/wiki/wiki/Sharding-introduction-R&D-compendium

[67] J. Poon and V. Buterin. (2018) Plasma: Scalable Autonomous Smart Contracts. [Online]. Available: https://web.archive.org/web/20180510130446/http://plasma.io/

[68] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," *University of Chicago*, Nov. 2017.

[69] R. Network. (2018) Raiden Network 0.3.0 Documentation. [Online]. Available: https://web.archive.org/web/20180510124401/http://raiden-network.readthedocs.io/en/stable/