DClaims: A Censorship Resistant Web Annotations System

(extended abstract of the MSc dissertation)

João Ricardo Marques dos Santos
Departamento de Engenharia Informática
Instituto Superior Técnico

Advisors: Professors Nuno Santos and David Dias

I. Introduction

The web plays a critical role in informing modern democracies. A study [1] 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 aged 18-44. 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 [2], influence government elections [3], endanger human lives [4] and even jeopardise the future of the human species [5]. 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 has become one of the most active areas of research[6], [7], [8], [9].

There have been several attempts to improve the reliability of information on the Web. Fact-checking platforms and social networks are two. The problem with these platforms is that they too can be censored. There are multiple instances of state censorship of information. 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 [10], [11]. Google, Facebook and Twitter publish annual reports of the number of government requests of remove content [1] [2] [3].

Besides being vulnerable to censorship, the information created by these platforms is scattered across multiple profile pages and websites. An alternative way to achieve the goal of informing would be to display the information directly on the source webpage. Web annotations[4] 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 empowers 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 shows 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. Currently, there are several web annotation services available to the general public [12], but they have a centralised nature. 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. Other platforms such as Hypothes.is improve over 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.

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.

Our system must satisfy the following requirements:

- **Authenticity and Integrity Assurances:** Ensure that the end-users know who created the web annotations, and that they have not been tampered with.
- **Data Permanence and Portability:** Links to 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 voluntary institutions, and should not be higher than one of current platforms which provide similar services.
- **Scalability:** The system should handle workloads similar to the ones on Facebook's news pages.
- **Compatibility with standards:** The data model used should be compatible with current standards. This is important to ensure interoperability between applications.

This paper 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[5] and the Inter-Planetary File System(IPFS)[6] . To

[1]Google Transparency Report: https://transparencyreport.google.com/government-removals/overview

[2]Facebook Transparency Report: https://web.archive.org/web/20180501211320/https://transparency.facebook.com/government/

[3]Twitter Transparency Report: https://web.archive.org/web/20180501211317/https://transparency.twitter.com/en/gov-tos-reports.html

[4]https://web.archive.org/web/20180502111019/https://www.w3.org/annotation/

[5]https://www.ethereum.org/

[6]https://ipfs.io/

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.

In summary, the contributions of this paper 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.
- An experimental evaluation of the DClaims platform, comparing its performance with other widely adopted platforms.

The rest of this document is structured as follows: Section II is the related work where we overview Web Annotations, blockchain and IPFS. In Section III we present the architecture of the DClaims protocol. After, in Section IV we offer our implementation, followed in Section V by the evaluation of the system. Finally in Section VI we present the conclusion and discussion.

II. Related Work

In this section, we analyse the current systems for Web Annotations and explain how they can be a useful tool for fact-checking information on the web. To understand the general model of web annotations services, 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*.

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 self-verifiability (not be dependent on third parties to be verified), permanency, revocability, portability and resistant to censorship.

Unfortunately, the most used platforms, such as Medium and Genius have a siloed architecture. Medium[7] is an online platform for publishing blog posts. Anyone can create blog posts and share them with the community. Genius allows users to generate annotations about any website. 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. Despite their benefits, these platforms still fall short of attaining all the desirable web annotations properties, more precisely: lack of interoperability, no data permanence assurances and are vulnerable to censorship.

Table I: 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**

With the goal of increasing interoperability between web annotations services, the World Wide Web Consortium (W3C) created a standard[8] for web annotations. Following the standard, a new web annotation service, called Hypothes.is, was created. 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 the ones produced by any other platform that implements the same standard. Furthermore, Hypothes.is has 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.

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 I 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. 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 existing 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.

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 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 sections are dedicated to an overview of it.

A. Background on Ethereum

Ethereum[13], [14], 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. Ethereum's purpose is to run a global virtual machine, on the blockchain, that anyone can use by

[7]https://medium.com/

[8]https://www.w3.org/annotation/

paying a small fee. 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, called smart-contracts, 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.

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[9] which provides enormous flexibility and allows for the creation of arbitrary programs.

Ethereum's censorship resistance features are a product of its decentralised architecture. Ethereum nodes are geographically spread and controlled by different parties, and no trust needs to be placed in a single entity. The Ethereum Virtual Machine assures that all the nodes reach the same state in the blockchain and that such state is permanent.

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. The first challenge is minimising the data stored on the blockchain, for it is extremely expensive. The second challenge is minimising the number of transactions so that Ethereum's 20 transaction per second limitation does not turn into a bottleneck in the system.

B. Background on the Inter-Planetary File System (IPFS)

As described, storing data on a blockchain is expensive and IPFS a good solution to that problem. IPFS [15] 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.

Figure 1 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 application. 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 application, Bob requests the file(*step 5*). Bob's IPFS node searches for the file in IPFS' Merkle Dag – a data structure distributed across the IPFS network which holds a list of all the files available and routes to those files – then IPFS' peer-to-peer library (`libp2p`[10]) handles the transport from Alice's to Bob's node (*steps 6 and 7*). The `cat.jpg` is now stored on Bob's IPFS repo, and Bob can view it (*steps 8 and 9*).

IPFS links[11] are a hash of the data they represent, and the link is valid independent of the file's location. This grants two important features. First, an IPFS link serves as an integrity check for the file. Second, links are permanent, and unlike HTTP links, they never break [16].

IPFS offers a set of desirable features when it comes to censorship resistance. First, it is logically decentralised, meaning it can work in local area networks, disconnected from the Internet. Second, it does not rely on DNS or Certificate Authorities. Third, even if only one node in the network has the requested file, all the other nodes can access it. Finally, all files are cryptographically verified.

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

III. Architecture

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. The attacker model we are considering 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). As shown in Figure 2 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

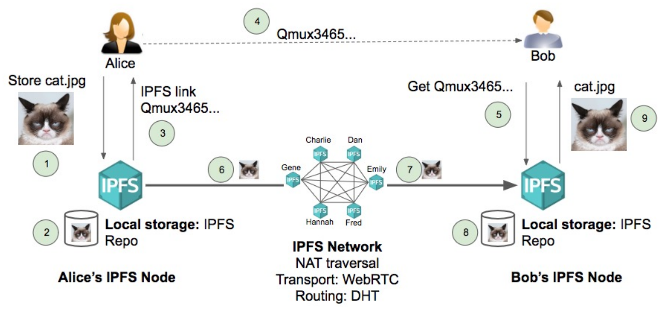


Figure 1: Ipfs Overview. Transfering A Cat's Photograph.

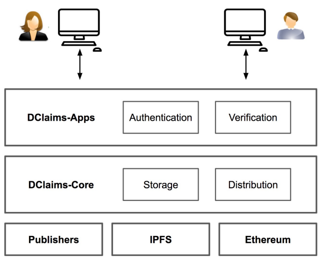


Figure 2: DClaims' Architecture

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

[10]libp2p: https://web.archive.org/web/20180505173610/https://libp2p.io/

[11]IPFS Links:https://github.com/multiformats/multihash

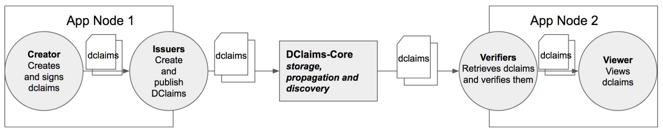

App Node 1
Creator
Creates and signs dclaims
dclaims
Issuers
Create and publish DClaims
dclaims
DClaims-Core
storage, propagation and discovery
dclaims
App Node 2
Verifiers
Retrieves dclaims and verifies them
dclaims
Viewer
Views dclaims

Figure 3: DClaims Entities Interactions.

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.

DClaims' architecture was designed with the requirements discussed in Chapter I. They include revocability, cross-platform support, self-verification, permanence, availability and censorship resistance.

In this section, we present the DClaims system. We start by over-viewing the DClaims-Core layer, in Section III-A. Finally, in Section III-B we analyse DClaims-Apps layer.

A. DClaims-Core

This section describes the DClaims-Core architecture.

1) Data Format: DClaims encapsulates web annotations inside a new data model, called *dclaim*. A dclaim is a data structure, based on the Verifiable Claims Data Model[12] , which supports digital signatures and user identification.

2) Entities: Figure 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.

3) Storage and Discovery: The IPFS and Ethereum Hybrid: 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 4 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. 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

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

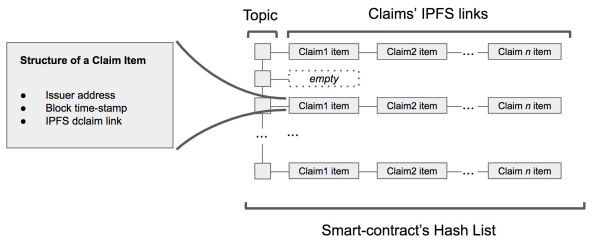


Figure 4: Smart-contract's Hash List That Keeps Track Of The Ipfs Link Of The Issued Dclaims

in the smart-contract under the same hashlist's key (which is the topic).

A simple way of implementing this architecture would be to have users interacting directly with the smart-contract, and pay for their transactions. However, this approach would have several problems. First, the costs would be extremely high, because there would be one transaction per dclaim created. Second, the system would not scale. As the rate of dclaims creation increased, the number of transactions the Ethereum network can process would be reached, acting as a bottleneck in the system. Last, users would be required to run dedicated software to interact with the Ethereum blockchain, which would pose an adoption barrier. In the next section, we present a better way for how this hybrid storage mechanism can be implemented.

4) The Publishers Network: The problems pointed in the previous section (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. Publishers act as a proxy between a dclaims-app who wants to issue or read a dclaim and IPFS and Ethereum.

Figure 5 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).

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 6 illustrates the batch issuance mechanism. 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 explained in Section III-A4). Each buffer has a threshold,

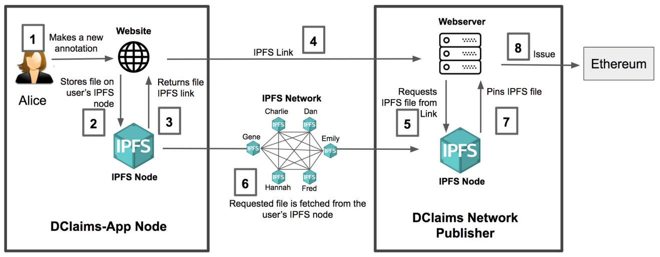


Figure 5: How A Publisher Makes A Copy Of A Dclaim

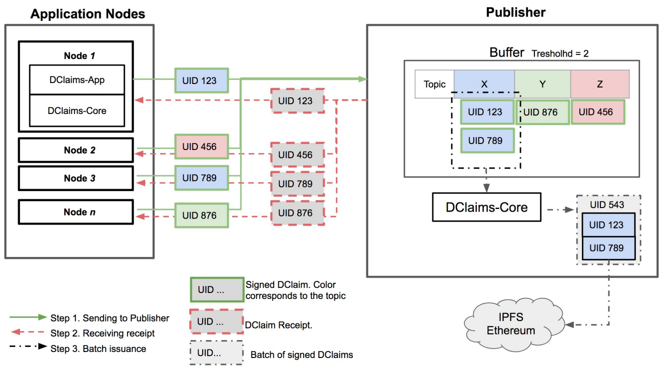


Figure 6: Publisher's Batch Issuance And Receipt Issuance Mechanism

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 issued for a specific topic. 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. 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.

Receipts: A potential concern is that the publisher network can be seen as a point of centralisation in the system architecture. 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 6. 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. If a publisher fails to publish the dclaim, or denies access to it, the user can make a complaint, and warn other users to stop using that publisher. 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.

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 V-B2, the cost of running a full-scale deployment of DClaims would have a cost 5 to 6 times lower than the one of Wikipedia[13]. 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 may justify the cost in several cases. 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[14] offer a service where websites can run a Captcha[15] which is running a proof-of-work algorithm to mine Monero. Ultimately, publishers could also run ads on their websites.

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. The first is requiring a proof-of-work from individual users, issuing individual dclaims, the time spent performing this action would be minimal and cause little to no impact on their user experience. However, for malicious users engaging in heavy activity, the proof-of-work would become a strong disincentive to attack. The second, is to require strong authentication and identification from users, such as requiring a photo of a government-issued ID. In the next section, we present the DClaims-Apps layer.

B. *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. The developer of an application that wants to use DClaims starts by customising DClaims-Apps to its needs. First, it defines the data model to its needs, then chooses the cryptographic package for digital signatures. After that, it can make use of the DClaims-Core and Publisher software, to start the system.

1) 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 7 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

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

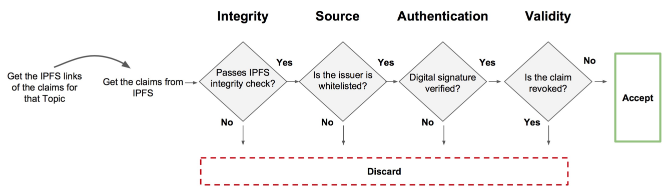

Integrity
Source
Authentication
Validity
Get the IPFS links of the claims for that Topic
Get the claims from IPFS
Passes IPFS integrity check?
Is the issuer is whitelisted?
Digital signature verified?
Is the claim revoked?
Yes
Yes
Yes
No
No
No
No
Yes
Accept
Discard

Figure 7: The Verification Process Of A Dclaim.

Section III-B4, 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.

2) User Authentication and Identification: Dclaims Issuers are authenticated by a digital signature in the dclaims they issue. The reference implementation of a DClaims-App (revealed in IV-C) uses an Ethereum library to sign dclaims using the Ethereum address.

3) Revocation of dclaims: 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.

4) 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. This way if a user goes rogue and starts posting spam, other users need only to remove him from their list.

IV. Implementation

In this section, we present the developed software stack, as described in Figure 8. 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[16]. Section IV-A focuses on the implementation of the DClaims-Core library while Section Section IV-B focuses on the Ethereum smart-contract. Finally, Section IV-C covers the details of the DClaims-News application.

A. 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.

[16]https://github.com/inesc-id/dclaims-pm

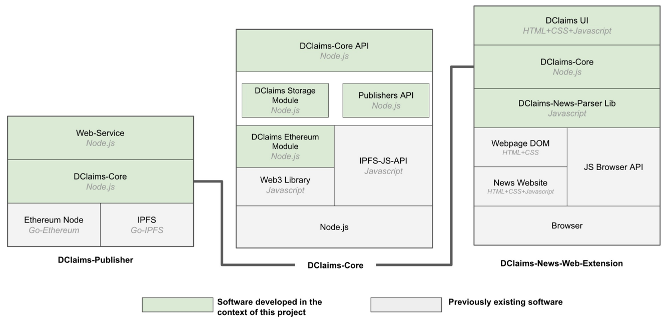


Figure 8: DClaims Core, Publisher And Web Extension Software Stacks.

B. 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*[17] 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*.

C. 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. The primary goal of developing the application was to provide a reference implementation to dclaims-apps verification, authentication and revocation mechanisms.

Figure 9 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 and 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.

1) DClaims-News Implementation: 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 Bootstrap[18] and jQuery[19] libraries) via a

[17]http://solidity.readthedocs.io/en/develop/types.html?highlight=data\%20type\#mappings

[18]https://getbootstrap.com/docs/3.3/

[19]https://jquery.com/

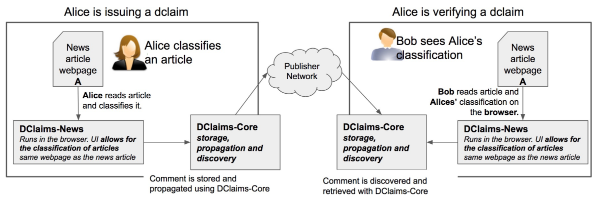

Alice is issuing a dclaim
News article webpage A
Alice classifies an article
Alice reads article and classifies it.
DClaims-News
Runs in the browser. UI allows for the classification of articles same webpage as the news article
DClaims-Core storage, propagation and discovery
Comment is stored and propagated using DClaims-Core
Publisher Network
Alice is verifying a dclaim
Bob sees Alice's classification
News article webpage A
Bob reads article and Alices' classification on the browser.
DClaims-Core storage, propagation and discovery
DClaims-News
Runs in the browser. UI allows for the classification of articles same webpage as the news article
Comment is discovered and retrieved with DClaims-Core

Figure 9: DClaims-News Basic Architecture.

Chrome browser extension[20], that changed the HTML of the web pages. The Chrome extension was configured using the `manifest.json` file.

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.

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.

V. Evaluation

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 V-A, 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 V-B 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.

A. 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 correspond 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 a 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. 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 have 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.

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.

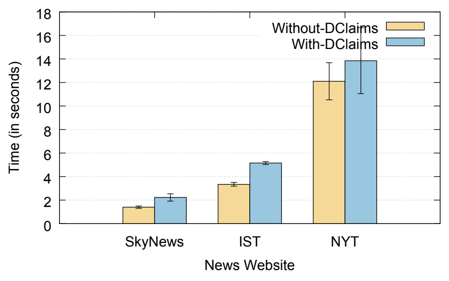
Figure 10a shows the time that the three news websites take to load, with and without DClaims. The overhead DClaims introduces is expected since the web extension needs to connect to an IPFS node, to an Ethereum node and then, for each article, it needs to generate the news article ID (which is the SHA-3 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 showed us that the overhead introduced by DClaims increased linearly with the number of articles each page had. In Figure 10a Sky News' website was expected to take 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 high due to the substandard JavaScript code it has.

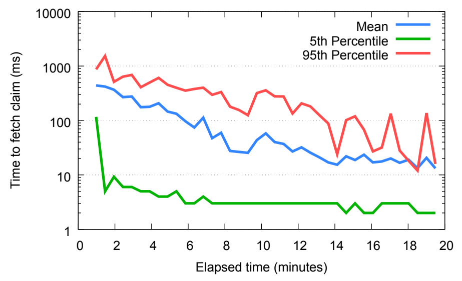
Furthermore, the latency introduced by DClaims does not affect the user experience, as the original elements of the website (news titles, images, among others) appear just as fast as they did before, only the elements introduced by DClaims (view dclaims button, dclaims counter) take longer to appear. In conclusion, the performance overhead introduced by running DClaims-News is negligible in the sense that it does not impact the user experience.

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 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

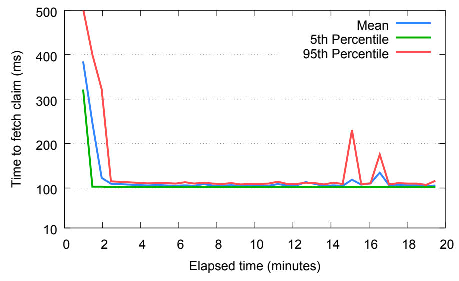
[20]https://developer.chrome.com/extensions



(a) Website Loading Time Comparison With And Without Dclaims Running.



(b) Fetching Individual Claims From IPFS



(c) Time To Retrieve The List Of Claims From Ethereum.

Figure 10: DClaims Test Results

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 dclaims from IPFS.

Figures 10c and 10b 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 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.

B. *Evaluating DClaims Costs*

In this section we determine the cost of a full-scale deployment of DClaims. We start, in Section V-B1, by estimating the level of activity our system would have to endure, using Facebook data as a proxy. Next, in Section V-B2, we calculate the costs based on the considered activity level. Finally, in Section V-B3, 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.

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 main 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.

Table II: 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

Methodology: We used the Facebook Graph API[21] 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, we obtained all the comments and a count of the number of likes and reactions[22]. From that dataset we were able to calculate the values presented in Table II.

Results: Observing Table II, 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.

2) Estimating DClaims' Cost and Performance: We used the activity values presented in Table II 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. Next, we present the costs in terms of storage, computation and Ethereum transactions.

Storage: To calculate the necessary storage of our system, we started by measuring the storage cost of individual annotations. We used the data format from the W3C Web

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

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

Table III: 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 IV: 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

Annotation standard, and placed a string with 148 characters (average length of Facebook comment) inside each annotation. We calculated that each claim occupied 30KB of storage. From that point, we were able to calculate the storage cost per year (using AWS pricing[23]), assuming the activity levels listed in Table II. 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.

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. Our server was able to handle 25 requests per second. We used Table II'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. We used Table II'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 II, 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[24].

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[25] , 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 III.

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[26]). We chose the standard value, 3 Gwei[27] , 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. Looking at Table II's *Average number of interactions (per post, per day)* we know that the number of interactions per article 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. It is important to note that this is the cost of the entire system, the cost per user is very small. In Section V-B3 we provide an analysis of the cost.

3) Cost Summary: : 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 IV.

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, DClaims' costs are still significantly smaller than the ones for real-world systems with a donation based financial model, such as Wikipedia.

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. 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

[23]EBS General Purpose SSD: https://web.archive.org/web/20180503104411/https://aws.amazon.com/ebs/pricing/

[24]https://web.archive.org/web/20180503113204/https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/

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

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

[27]1 Gwei = 1 nanoEther

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 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.

C. Summary

In this section, 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 throughout 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 donations financial model. We also noted that a subscription-based model would be a viable option. Next, in Chapter VI we present the conclusions of this work.

VI. Conclusion

This paper presents 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 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).

Once some of those solutions are deployed, the cost of DClaims' operation will lower dramatically.

References

[1] 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

[2] (2007) Official's Key Report On Iraq Is Faulted. [Online]. Available: https://web.archive.org/wp/web/20171023180238/http://www.washingtonpost.com/wp-dyn/content/article/2007/02/08/AR2007020802387.html

[3] 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.

[4] (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

[5] (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/

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] V. Buterin, "Ethereum white paper," *self-published paper*, 2013.

[14] G. Wood, "Ethereum Yellow Paper," *self-published paper*, 2014.

[15] J. Benet, "IPFS-content addressed, versioned, P2P file system," *arXiv.org*, 2014.

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