CCFS CRYPTOGRAPHICALLY CURATED FILE SYSTEM

A Thesis Proposal
Presented to
The Academic Faculty

by

Aaron David Goldman

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
DECEMBER 2015

CCFS CRYPTOGRAPHICALLY CURATED FILE SYSTEM

Approved by:

Dr. John A. Copeland, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Manos Antonakakis, Committee chair
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. George F Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: August 6, 2014

To Mrs. Hackworth: Thank you for teaching me how to read in 4th grade and believing that I could be taught.

ACKNOWLEDGEMENTS

I would like to thank all the students who helped contribute to the CCFS code base: Priya Bajaj, Mallika Sen, Holly K Parrish, Thomas W Saekao, Natchaphon Ruengsakulrach, Ikenna Uzoije, and Chris Beaulieu. Their countless hours, including nights and weekends; enforcement of good software practices; and dedication to the project were invaluable. I have these students to thank for ensuring I practiced some of what I preached. Together, we brought CCFS from theory to reality.

Next, I am grateful to my co-authors Selcuk Uluagac, John A Copeland, and Raheem Beyah and the many conference reviewers who helped me form vague ideas into actionable material and refine the concepts and the design to a viable architecture.

I am sincerely grateful to George Macon for introducing me to GTRI. Without this experience, CCFS would not exist. George taught me good coding practices, appropriate work ethic, and the day-to-day of PhD life. His mentorship enabled me to develop the discipline and skills necessary to complete this degree and will serve me throughout my career.

I would like to express my heartfelt appreciation to Jerusha Barton for bringing to bear her years of experience in the writing field, thus imposing professionalism and coherence on my writing.

I am grateful to my committee members Drs. Emmanouil K Antonakakis, George F Riley, and John A Copeland for their time, effort, and careful consideration of all the

material that went into this project. I would especially like to thank Dr. Antonakakis for giving me in-depth feedback on the proposal and drafts of this manuscript.

Last and most importantly, my deepest appreciation belongs to John A Copeland, who was my undergraduate, graduate, and thesis advisor. I am thankful to have been welcomed into his lab, be given the freedom pursue my own research interests, and trusted me with the education of his undergraduate students, even allowing me to apply their time to the development of CCFS. I am grateful that he believed I could meaningfully contribute to his students' education and research experience. I will always carry the confidence, support, and advice Dr. Copeland has given me during my time in the lab; these experiences enabled me to grow.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF NOMENCLATURE

HID ...…….…… Hash identifier (either an HCID or an HKID)

HKID ...…….Hash of public key identifier

HCID ..……...……Hash of content identifier

Content ……An exact sequence of digital bytes

Blob ...…… Array bytes referenced by HCID

List …...…….……...File that maps names to HIDs and Type Strings

Commit ....……..Certificate for defining a specific version of a repository

Tag ……...…….....Certificate for defining a specific version of a domain

Collection ……. ….……A group of files and Curators named cryptographically

Repository .…….A collection that is versioned as a single entity

Domain ..……A collection that is versioned independently

Folder …….A collection that cannot be versioned

Name Segment ……Name of a file or directory in a directory

Type String ……..The string that identifies the type of the target of an HID
(“blob”, “list”, “tag”, “commit”,”NaB”)

Content service …….A method of retrieving content by
HCID, HKID, or HKID+namesegment

Interface ……..…...…A map across sundry APIs and the CCFS-named object store

Namespace …...……...The set of all valid names in a context

Curator ……..……Entity with the ability to edit a Collection

SUMMARY

The Internet was originally designed to be a next-generation phone system that could withstand a Soviet attack. Today, we ask the Internet to perform tasks that no longer resemble phone calls in the face of threats that no longer resemble Soviet bombardment. However, we have come to rely on names that can be subverted at every level of the stack or simply be allowed to rot by their original creators. It is possible for us to build networks of content that serve the content distribution needs of today while withstanding the hostile environment that all modern systems face.

This dissertation presents the Cryptographically Curated File System (CCFS), which offers five properties that we feel a modern content distribution system should provide. The first property is Strong Links, which maintains that only the owner of a link can change the content to which it points. The second property, Permissionless Distribution, allows anyone to become a curator without dependence on a naming or numbering authority. Third, Independent Validation arises from the fact that the object seeking affirmation need not choose the source of trust. Connectivity, the fourth property, allows any curator to delegate and curate the right to alter links. Each curator can delegate the control of a link and that designee can do the same, leaving a chain of trust from the original curator to the one who assigned the content. Lastly, with the property of Collective Confidence, trust does not need to come from a single source, but can instead be an aggregate affirmation. Since CCFS embodies all five of these properties, it can serve as the foundational technology for a more robust Web.

CCFS can serve as the base of a web that performs the tasks of today's Web, but also may outperform it. In the third chapter, we present a number of scenarios that demonstrate the capacity and potential of CCFS. The system can be used as a publication platform that has been re-optimized within the constraints of the modern Internet, but not the constraints of decades past. The curated links can still be organized into a hierarchical namespace (e.g., a Domain Naming System (DNS)) and de jure verifications (e.g., a Certificate Authority (CA) system), but also support social, professional, and reputational graphs. This data can be distributed, versioned, and archived more efficiently. Although communication systems were not designed for such a content-centric system, the combination of broadcasts and point-to-point communications are perfectly suited for scaling the distribution, while allowing communities to share the burdens of hosting and maintenance. CCFS even supports the privacy of friend-to-friend networks without sacrificing the ability to interoperate with the wider world. Finally, CCFS does all of this without damaging the ability to operate search engines or alert systems, providing a discovery mechanism, which is vital to a usable, useful web.

To demonstrate the viability of this model, we built a research prototype. The results of these tests demonstrate that while the CCFS prototype is not ready to be used as a drop-in replacement for all file system use cases, the system is feasible. CCFS is fast enough to be usable and can be used to publish, version, archive, and search data. Even in this crude form, CCFS already demonstrates advantages over previous state-of-the-art systems.

When the Internet was designed, there were relatively fewer computers that were far weaker than the computers we have now. They were largely connected to each other

over reliable connections. When the Internet was first created, computing was expensive and propagation delay was negligible. Since then, the propagation delay has not improved on a Moore's Law Curve. Now, latency has come to dominate all other costs of retrieving content; specifically, the propagation time has come to dominate the latency. In order to improve the latency, we are paying more for storage, processing, and bandwidth. The only way to improve propagation delay is to move the content closer to the destination. In order to have the content close to the demand, we store multiple copies and search multiple locations, thus trading off storage, bandwidth, and processing for lower propagation delay.

The computing world should re-evaluate these trade-offs because the situation has changed. We need an Internet that is designed for the technologies used today, rather than the tools of the 20th century. CCFS, which regards the trade-off for lower propagation delay, will be better suited for 21st-century technologies. Although CCFS is not preferable in all situations, it can still offer tremendous value.

Better robustness, performance, and democracy make CCFS a contribution to the field. Robustness comes from the cryptographic assurances provided by the five properties of CCFS. Performance comes from the locality of content. Democracy arises from the lack of a centralized authority that may grant the right of Free Speech only to those who espouse rhetoric compatible with their ideals. Combined, this model for a cryptographically secure, content-centric system provides a novel contribution to the state of communications technology and information security.

CHAPTER 1

INTRODUCTION

The objective of this research is to contribute to the development of a more robust Internet by implementing and testing a single storage system architecture that incorporates the five principles listed and discussed in Subsection 1.1. Throughout this thesis, these five principles are assumed to be true statements. Methods will include leveraging the cryptographic functions of hashing, signing, and verifying.

1.1 Goals of the Dissertation

The goal of the thesis is to implement and test a storage system that incorporates the following five core principles into a single unified architecture: Strong Links, Permissionless Distribution, Independent Validation, Connectivity, and Collective Confidence. The use of cryptographic names provides Strong Links and Permissionless Distribution. The use of delegable namespaces provides Independent Validation, Connectivity, and Collective Confidence. The rest of the thesis is organized as follows: Chapter 1 introduces CCFS; Chapter 2 delineates the architecture of CCSF; Chapter 3 describes the potential use cases for CCFS; Chapter 4 provides the analysis of the system; and Chapter 5 gives the conclusion.

1.2 Properties

A unified architecture that incorporates the following five principles will result in a more robust Internet:

1. **[Strong Links]** References should identify precise content at any one point in time.

2. **[Permissionless Distribution]** Content maintenance and content distribution should not require permission.
3. **[Independent Validation]** The object seeking affirmation should not choose the source of trust.
4. **[Connectivity]** Collections should be able to embed other curators' collections.
5. **[Collective Confidence]** Trust should be an aggregate function of many individual reputations.

The remainder of this section describes these principles in more detail and their role in increasing the robustness of the Internet.

Strong Links

The principle of Strong Links states that *references should identify unique content at any one point in time.* A Strong Link is one that refers to content in such a way that two different followers of the link can confidently know that each is receiving the same content as the other.

Links on the Web are used to refer from one page to another. The reader has no guarantee, however, that what he sees on the referenced page is the same content that the author saw at the time she referenced the page, an error called link rot. Link rot occurs when a link does not point to the content that was intended because the original host no longer provides that content [1]. With Strong Links, the readers can be assured that the content they are viewing is in fact the content that was originally referenced.

The technology to create Strong Links already exists in the form of cryptographic hash functions. A cryptographic "hash" of static content can be used as the name of that content for retrieval and verification purposes and a public key can be used as the name of signed content for both retrieval and verification purposes. The use of cryptography to

create Strong Links increases the Internet's robustness by ensuring the content a user discovered is the identical content that was originally referenced.

Permissionless Distribution

A second way in which to increase the Internet's robustness is to ensure Permissionless Distribution. The principle of Permissionless Distribution states that *content distribution and maintenance should not require permission*. Anyone who possesses content should have the capability to maintain and/or distribute that content without the permission or involvement of the authoritative host.

This low barrier to entry creates an increased supply of distributors that can compete to distribute the content. Following the law of supply and demand, the increased supply will drive down the cost of content storage and distribution.

A retriever could request content in parallel from many different distributors than retrieve from the distributor that offers the lowest cost or best performance. Continuous, real-time auctions would push distribution loads to those that are able to satisfy the demand at the lowest price.

Independent Validation

Third, a storage system that incorporates the principle of Independent Validation would increase the Internet's robustness. The principle of Independent Validation states that *the source of the validation should not originate from the validated.* Cost and convenience are major aspects of consideration when customers choose which Certificate Authority (CA) to hire. An auditor that is chosen and funded by the domain owners has an incentive to provide a seamless certification process. Any additional effort taken by the auditors to perform more extensive validation than that performed by their peers has a competitive disadvantage in the marketplace on both price and convenience.

In the current Secure Sockets Layer (SSL) certificate system, the CAs are chosen by the sites that they are certifying. CAs may have average security or have the potential to perpetrate corrupt business practices. At the same time, a malicious party attempting to obtain a fraudulent certificate can hire a different CA to avoid the additional scrutiny. Large companies, such as Google, have recognized this issue and have proposed methods to mitigate the ongoing problem [2].

A Cryptographically Curated File System (CCFS) can eliminate the incentive conflict by inverting the trust model. Each time one site links to another, trust is established. The reference from one site to another both indicates relevance and establishes security. This method reduces market pressure that can lead to bad security; sites link to other sites as an intentional reference to content. Trust comes not from a CA, which is chosen by the site, but from the party who is referencing the named content.

Trust flows from those who trust to those who are trusted. The signal of trust occurs when one document links to another. In other words, the link serves as a self-certifying name of the content. This reference, therefore, vouches for the identity of the content at the very moment that the content is retrieved.

Connectivity

Connectivity is also very important in improving the Internet's robustness. The principle of Connectivity states that *collections should be able to embed other curators' collections.* If this crawlable web of links is dense enough, it will become a connected graph of content. By delegating namespace, CCFS allows one curator to connect to another curator. By this embedding, we can use each collection to discover the next. This will form a web that tends toward a small-world graph. Once a critical mass of curators delegates enough namespace to each other, the graph will reach the threshold wherein connectivity is established. Once connectivity is established, the relying party can crawl the entire graph

from any point on the graph. Leveraging this property allows the knowledge of a single or a few references to lead to a very large graph of content.

Each newly discovered piece of content would have a relative path back to a known curator. These paths can be used to not only discover new content, but also learn about the relationships between the content objects. We can discover the relationship between curators by the names one curator gives another. For example, placing a curator in a "Friends" or "Vendors" folder would signify trust; discovering a curator in a "Bookmarks" or "History" folder would signal a neutral relationship; yet, a folder called "Blacklist" or "Beta" would signify distrust. Assigning meaningful paths to collections with established relationships communicates the reliability of the curator.

Collective Confidence

Finally, the principle of Collective Confidence would contribute to the Internet's robustness by improving storage systems. This principle states that *trust should be a function of popular reputation.* When the properties of Independent Validation and Connectivity are combined, a new property of CCFS emerges: the ability to aggregate the trust signals of many different sources. When multiple sources all refer to a single entity, their endorsements can be joined. These joins could be functions, which map trust from individuals to communities, and could be as simple as any, majority, all, or numerous complex schemes of trust. For example, if the majority of folders containing a curator are named "Vendor," we can assume "Vendor" is a meaningful label. Together, the endorsers have greater power to grant trust than any one of them would possess individually. This collective endorsement allows for a trust that goes beyond the single-site trust and leads to a more general confidence in the role of a curator.

One of the first demonstrations of this principle was PageRank, which gave Google the ability to sort results by relevance. This application confirmed that a population could collectively vouch for a site. While this approach has proven useful, it has also proven

vulnerable to search engine optimization (SEO) (e.g., Google bombing). As a result, Google has invested significant effort in updating its collective confidence algorithms. CCFS would provide the tools for search engines to establish collective confidence.

Furthermore, PageRank demonstrated that having different rankings for different communities could also lead to significant utility. This method enables community-specific—and even activity-specific—rankings. PageRank has been used on different countries' versions of the various region-specific sites, where the PageRank algorithm was seeded with locally trusted sites. This enabled Google to rank sites according to the relevance within the country in question. Whereas an individual curator would be a dubious source of trust, these aggregated trust values can prove far more robust in a world with ever-shifting trust relationships.

1.3 Background Literature Survey

The traditional Internet was designed to accommodate a body of users who wanted to visit specific network locations. Today, the majority of users are more interested in obtaining specific content than in visiting a specific website. This demand for information has led to a proliferation of proposed alternative-network architectures that emphasize content on the network, as opposed to the location of that content. A recent proliferation of articles refers to these proposals as "name-oriented networking" [3]—[6], "information-centric networking" [7]—[10], and "content-centric networking" [9]–[14], as well as other similar names that emphasize the content on the network, as opposed to the location of that content [3].

Pioneer efforts in information-centric networking (ICN) can be found in Van Jacobson's work [4]. Other models have taken a variety of approaches to ICN, but none have been widely embraced. This section briefly discusses the related work in this domain of research and suggests how CCFS offers advantages over, or complements, other approaches.

CCNx [3], [4], a content-centric networking project, uses hierarchical naming that includes the publisher and a content identifier, along with other information. Consumers request content based on its identifier, and the request is routed to the closest host that has the content. No conversion from the content identifier to a location address exists; and although cryptography can be used, it is not mandatory.

Data-Oriented Network Architecture (DONA) [3], [5] uses names that contain a cryptographic hash of the publisher's public key and a content identifier. The content names and their location addresses are registered with a name resolution handler, which relates content to the authorized points that store the content. When available, resolution handlers can be contacted and can provide the information necessary for deliveries.

Network of Information (NetInf) [3] does not use cryptographic keys, but is similar to DONA in that it registers content called information objects (IO). IOs are registered along with their location addresses to a name resolution service (NRS). When content, consumers, or providers move, the NRS must be notified. This saves the NRS from having to search for the content. The producers have to be tracked because they are hosting the content; the content has to be tracked so NetInf knows which producer has it; and the consumer has to be tracked to ensure the delivery of content with long-standing interests. Because the system is not dynamic, the NRS must be updated as providers, the content provided, and consumers can move to new locations.

Juno [3], [6] uses flat, self-certifying names that identify content and uses the Juno Content Discovery Service (JCDS) and other third-party indexing services. Location addresses are used for routing as in NetInf and DONA. The Juno system also allows for choosing content hosts by multiple criteria, as opposed to simply looking for the nearest location of the content. The JCDS must be updated to show changes in provider locations.

Logical Address Space Network (LANES) [7] uses cryptographic naming for name identifiers (NI) and scope identifiers (SI). Subscribers use application identifiers (AI), which are human-readable and are mapped to rendezvous identifiers (RI). A number of

distributed rendezvous services translate the RI into a forwarding identifier (FI) that leads the consumer to the location address of the content. The SI is used to restrict certain access to content. The local rendezvous point and service must be accessible to retrieve the content.

Mobility First [8], [9] is a significant multiphase effort among multiple universities. Mobility First uses a ground-up, comprehensive approach to develop a clean-slate, content-centric network architecture. This project utilizes many of the concepts examined by others, such as cryptographic self-certifying names and global naming conventions with human-readable names, which are converted to globally unique identifiers (GUIDs). Global name resolution services then convert these GUIDs into network addresses. The Mobility First project is a work-in-progress. The system requires global cooperation to adopt a global common naming convention, as well as to use a global name resolution service.

The other area of research that relates to CCFS is work done on self-certifying file systems. Two examples that are similar to CCFS are the Least Authority File System (Tahoe-LAFS) [10] and Content-Addressable Multi-Layer Indexed Storage (Camlistore) [11].

Although similar to CCFS, Tahoe-LAFS has different goals. Its reliance on cryptographic capabilities is effective for maintaining the principle of least privilege. In contrast, the focus of CCFS is on aspects of digital curation, such as authenticity, non-repudiation, and human-readable names. LAFS has chosen to include the global and secure aspects of Zooko's triangle [12], which holds that no naming system can be global, secure, and memorable (human-meaningful.) CCFS adds the aspect of human-recognizable names translated to global and secure names. By remembering one's own globally secure ID and using the tool of digital curation, the user can collect the global IDs of others and assign them memorable names within the user's own collection.

Camlistore is another content-addressable storage system that incorporates indexing and searching for content. Camlistore focuses on putting users in control of their data (i.e., personal content management over sharing and/or publication needs) and encrypts all content—an attribute not provided by most other storage systems. Unlike LAFS, Camlistore emphasizes user control, whereas LAFS prefers a decentralized model. Like CCFS, Camlistore accomplishes privacy and control through self-certifying names by utilizing content addressability as much as possible. Although Camlistore is content-centric, CCFS differs from this model in its architecture and protocols.

The projects above show that work in self-certifying file systems and content-addressable networks has been done. These projects set the stage for CCFS, which is the first work that has taken the simplified approach of requiring the signing of content in order to name that content. The requirement that all content be signed into and out of collections leads to a simpler system architecture. This simpler, more secure approach to content organization, retrieval, verification, and versioning constitutes a contribution to the field.

CHAPTER 2

CCFS ARCHITECTURE

For the purposes of this thesis, a functional prototype of CCFS was built. This prototype contains a core of CCFS that can currently read from (i.e., retrieve) and write to (i.e., publish) many sources. The prototype also contains a partially implemented file system interface and a web interface capable of performing Hypertext Transfer Protocol (HTTP) GETs, HTTP PUTs, and search.

2.1 Objects and Primitives

The architecture of CCFS consists of interfaces, objects, and services. Each of these layers is discussed in this section.

Cryptographic Identifiers

CCFS uses two different kinds of Hash Identifiers (HIDs): (1) Hash of Content Identifier (HCID) and (2) Hash of Public Key Identifier (HKID). This hashing is performed using the SHA256 cryptographic hash function. The public key signature algorithm is the elliptic curve digital signal algorithm (ECDSA) over the National Institutes of Standards and Technology (NIST) P-521 curve. The HKID is the hash of these public keys.

The role of the HCID is to identify static content and, due to its immutable nature, provides the property of Strong Links. The HCID is a self-certifying name that is used for both content retrieval and verification. The HCID is based on the principle that any two computers that run a hash function over the same content will produce the same hash. This prevents both false-positives and false-negatives in content retrieval. A false negative can occur when a host has the content you are looking for by a different name and you are unable to retrieve it. A false positive can occur when a host has content by that name, but the retrieval does not yield the content that was referenced. Because hashing the content

always produces the same hash, any two computers will have identical names for identical content, thus removing the chance of yielding a false negative. By hashing content upon receipt, verifying the HCID will also eliminate the risk of false positives. An HCID cannot become invalid over time for any reason including to failure to maintain the domain because HCIDs are ownerless.

Not only does the use of HCIDs prevent the generation of false positives and negatives, it also aids in preventing data corruption. HCIDs are consistent from one device to the next and are stable over time. Therefore, an individual can protect himself from the loss of data, and he can retrieve his content, which passed verification, from an external source.

HCIDs provide many advantages, but cannot be used to identify collections that may change over time. Yet, system designers need dynamic content. With the introduction of HKIDs, it is possible to have an identifier that will become meaningful. For example, data may not be available at the time that the reference was created (e.g., tomorrow's temperature data). The curator (Alice) of this data can say, "Tomorrow, I will release this data." Using an HCID would not be possible in this case because she has no way of knowing what the data will be. Without the data, a hash can neither be produced nor could it be updated to reflect subsequent data. There is a need to gain dynamism without sacrificing the benefits of HCIDs. This is where HKIDs become useful. With the further introduction of version numbers, a more robust system can be implemented that allows for repeated content updates. For example, in an update-distribution application, the latest version of the software will change over time. This future data can become a continually updating stream.

CCFS introduces an object that is identified with an HKID, contains an HCID, and becomes available once the content is created. The HKID is known in advance, can be distributed as an identifier, and is used to find the object. This object is signed with a secret key and verified with a public key. Since the same key is needed to identify and verify

these objects, CCFS can ensure that the data provided is valid if the HKID of the curator is known. This allows a curator (Alice) to promise to sign an object at some time in the future with her secret key. The retriever (Bob) can verify whether or not an object presented by any party fulfills this promise. With the further introduction of a version number, this fulfilled promise can become a reference that can be indefinitely updated. Only Alice, the owner of the HKID, knows the secret key that is necessary to update these references.

While the HCID identifies content (i.e., what Bob is seeking), the HKID identifies curators (i.e., Alice). HKIDs are effectively strings of random numbers, created by an individual and, thus, if sufficiently long, unlikely to match any other individual's HKID. Much like if you were to think of a 20-digit number, you are almost certainly the only human who has ever or will ever conceive of this string. In this way, the HKID serves to uniquely identify curators. Alice generates a random, 157-digit (521 bit) number; this is her secret key and, from it, CCFS can calculate her public key. The hash of Alice's public key is the HKID that identifies Alice. CCFS relies on the uniqueness of this large number to identify Alice.

If Alice tells others her secret key, they are also identified by that HKID, since it identifies anybody who knows the secret key. As the secret key can be used to sign content and the public key can be used to verify those signatures, anyone who knows Alice's public key can verify Alice's objects. The HKID is the hash of and certifies the public key. If CCFS knows Alice's HKID, it can verify Alice's collections. Whether the curator is an individual, a group, or an organization, an HKID simply refers to the keeper of the secret key.

Content is named by signing it with a secret key that is unique to each curator. These signatures can be verified using the corresponding public keys. These public keys can be verified because they hash to the HKID, which is known because it is part of the

content name. The act of signing (i.e., cryptography), naming (i.e., a file system), and curating (i.e., gathering and maintaining content) are the same and comprise the act of saving a file in CCFS. Because a saved file in CCFS has been signed, CCFS ensures that the content, however changed, still belongs to the same curator.

Content Objects

CCFS makes use of four content object types: Blobs, Lists, Commits, and Tags (Figure 1). Each serves a distinct role in the organization of CCFS. The purpose and format of the content objects are summarized in Table 1.

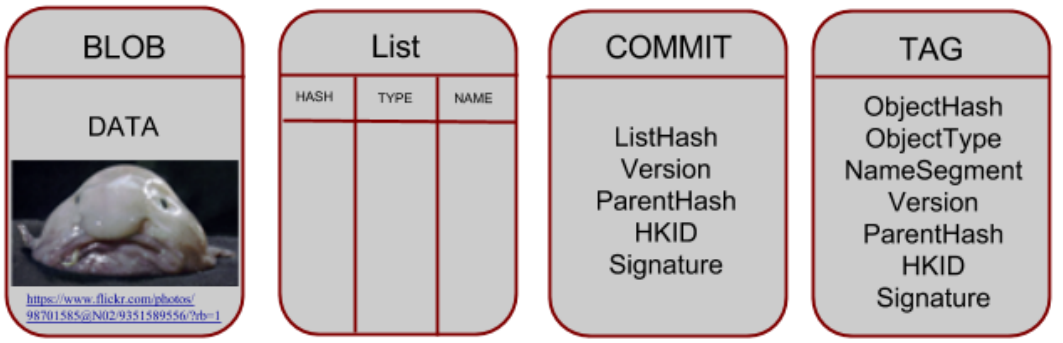

Figure 1. Content object schema.

TABLE 1.

CCFS CONTENT OBJECT TYPES

TYPE	PURPOSE	CONTENT FORMAT
Blob	Static; contains data Collection type: File	{ByteArray}
List	Static; Exclusive map map from: next name segment to: HID, Type Collection type: Folder	{ [ObjectHash, ObjectType, NameSegment] }
Commit	Versioned; Exclusive map from: HKID to: List's HCID Collection type: Repository	{ListHCID, Version Number, ParentHCID, HKID, Signature}
Tag	Versioned; Non-exclusive map from: HKID, next name segment to: HID, Type Collection type: Domain	{ObjectHash, ObjectType, NameSegment, Version Number, ParentHCID, HKID, Signature}

Blobs, the first object type, are referenced by HCID; are the underlying objects for a file; have no inherent structure; and contain arbitrary byte arrays. The purpose of CCFS is to retrieve the correct blob at the correct time.

A List, the second object type, is the underlying object for a folder. Lists are structured objects containing a series of triples. The List is a flat Uniform Character Set and Transformation Format, 8-bit (UTF-8)-encoded text file with each row containing an HID, type string, and a name segment string; each are separated by commas and the line is terminated with a newline character. The HID is represented as a hexadecimal number, and

the type string and name segment are URL-encoded to remove commas and newlines. This List object is then referred to by its own HCID. Referring to the List object using the HCID validates this object, making it as trusted as the source of its HCID. Since Lists can point not only to Blobs, but also to another List, a nested-folders paradigm is enabled. If the List points to a Commit or Tag, the List can reference another curator's dynamic collection. This reference is referred to as sub-namespace delegation and will be further discussed in the description of Tags.

Commits, the third object type, are the underlying object for a repository at a moment in time. A Commit is a UTF-8 encoded flat text file that contains the following: the HCID of a List; the version number expressed as a decimal integer; the hexadecimal encoding of the HCID(s) of the Commit's parent(s) Commits (i.e., if there is more than one parent, the field will be separated by commas); the hexadecimal encoding of the HKID of the curator that signed the Commit; and the hexadecimal encoding of the signature created when the curator signs the previous four fields. Each of these fields serves a purpose within the Commit and is separated by a comma and a new line. The List HCID is the reference that enables the Commit to point to a List and define a collection. The version number will serve as a way to distinguish two different Commits that were signed by the same curator. The Commit with the higher (i.e., newer) version number is the state of the curator's collection at this moment and the Commit with the lower number is simply a previous state that is now obsolete. The parent HCID refers to the Commit that defined the state immediately before the current one was published. If two or more states were merged, the Commit will have multiple parents.

By leveraging parent fields, diverged branches of the repository can be detected and merged. The HKID of the curator identifies the key that must be retrieved to verify the signature. This field will need to be indexed in order to make retrieving the Commit efficient. Finally, the Commit can be verified through the signature. If the signature is valid, then the other fields in the Commit are valid. If the signature fails to verify, the Commit is

either corrupted or malicious and, therefore, should be discarded. Since a Commit must point to a List, any update to a file in the collection will require generating the List and pointing to a new List. Therefore, all of the files in the repository must be versioned together under a single version number. The retriever (Bob) cannot accidentally get an older version of one file and a newer version of a different file in Alice's repository. For example, if Alice's repository defined a website, Bob would not receive a version of the html, javascript, and css from different points in time. Either all files in the repository update, or none of them update.

Tags are the fourth and final type of content object that underlie domains. The Tag combines one row of a List (i.e., the type and the name segment) and all fields of a Commit into a single object. The Tag is a UTF-8-encoded flat text file that contains the hexadecimal encoding of the HID of the referenced content object; the URL-encoded type string; the URL-encoded name segment; the decimal-encoded integer version number, the hexadecimal encoding of the HCID(s) of the Tag's parent(s); the hexadecimal-encoded HKID of the curator that signed the Tag; and, finally, the signature of the previous five fields. The HID could be either an HCID, if the object being referenced by the tag is a Blob or a List; or an HKID, if the object being referenced by the tag is a Tag or Commit. The type string disambiguates the HID so it can be used to retrieve content. The name segment is the name of the resource that is being referenced, such as a file, folder, or even a different curator's collection. The version number enables the determination of the most up-to-date version of the resource. Commas separate the HKID of the parent Tag. Like the Commit, the parent field of the Tag allows CCFS to detect and merge diverged domains. The curator's HKID identifies the key that must be used in verifying the Tag, and the final signature ensures that all of the fields are the same as they were when the curator signed the Tag.

2.2 Content Storage/Retrieval

Content Retrieval

Content retrieval is a recursive process that begins with an HID and a path (Figure 2). The path is a series of name segments (Figure 3). With each iteration, the HID and the first name segment are used to retrieve the next HID. The name segments that make up the path can be utilized in two ways: (1) by a name segment being used to look up the next HID and type from a List object; and (2) by looking up a Tag within the content service. This is repeated until all of the name segments are consumed. The last piece of content is returned to the user.

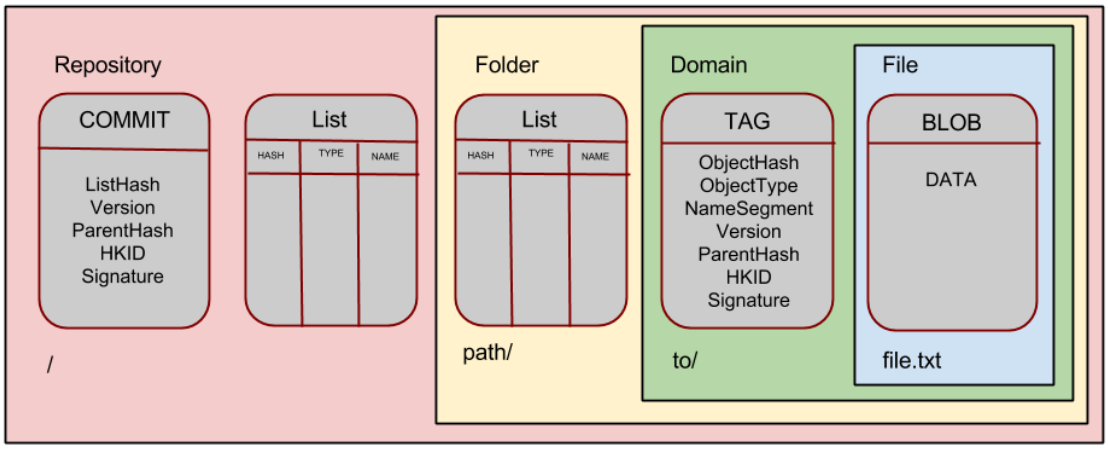

Figure 2. Content objects defining collections.

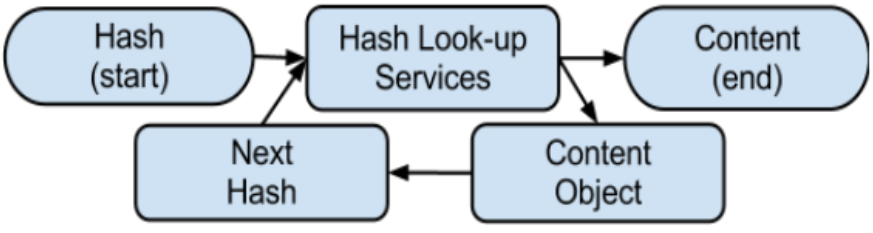

Figure 3. Utilizing name segments.

Figure 2 demonstrates the nested collections within a repository. Table 2 is an example of the recursive sequence for retrieving a blob from a repository (these objects have been placed in Appendix A). A repository contains a folder "path" that includes a domain "to" incorporating a file "file.txt." The system retrieves objects using their HIDs and then uses those objects, in conjunction with the name, to uncover the next HID. This process continues until the name is consumed and the final blob has been retrieved. This blob gives the content as the curator named it.

TABLE 2.

RECURSIVE RETRIEVAL STATE

HID	Type	Path
`880b5c…384db2`	Commit	"/path/to/file.txt"
`89e7de…e0ee50`	List	"/path/to/file.txt"
`8d8915…a124ae`	List	"to/file.txt"
`4448d9…e535c6`	Tag	"file.txt"
`9914ab…c1d860`	Blob	""

This model presents objects pointing to other, retrievable objects and was inherited from the Git version control system. Figure 4 has been reproduced from the Pro Git book (http://git-scm.com/book/en/v2). Git traces the name as a path through a sequence of objects and serves as the inspiration for CCFS' use of names as paths through objects.

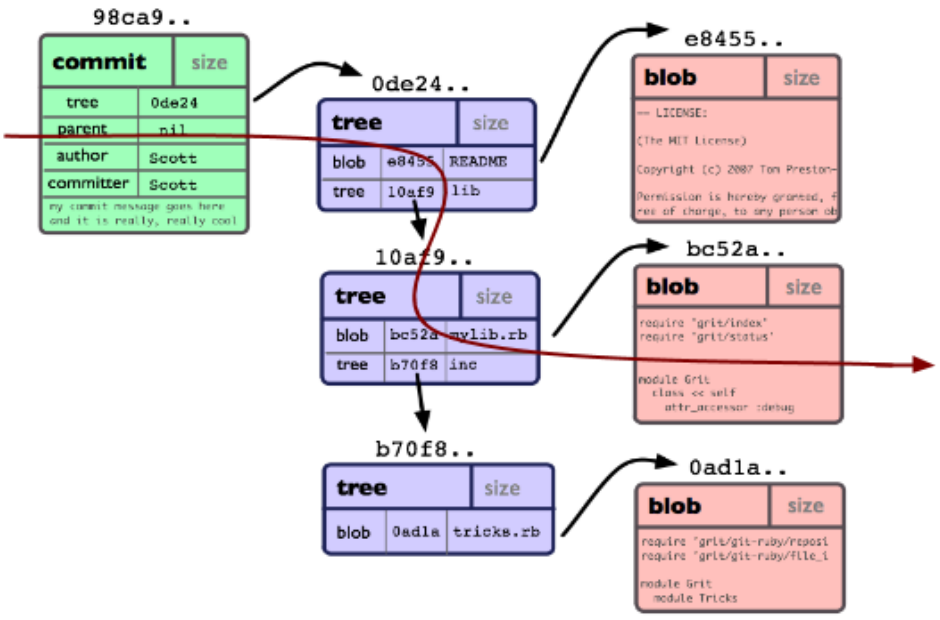

Figure 4. Content retrieval process.

Content Curation

The process of content curation is closely related to the retrieval process. This process begins with the same recursive process of retrieving objects along the aforementioned path. When the retrieval process reaches a part of the path that can't be found, the curator creates new content objects to represent new parts of the path and then publishes these new objects. If a previous version of the content object is found, an updated set of content objects are created and published.

2.3 Services

Content Services

The value of having a cryptographically strong identifier is the ability to verify content. CCFS allows the user to verify content objects, but this verification can only take place once the objects have been retrieved. These content candidates are provided by services that are referred to as content services. The content service is provided with an HID and, if necessary, a name segment. In return, the service provides the retriever with a

content object that may satisfy the query, depending on whether or not the signature is valid.

Five content services have been implemented in order to test the prototype implementation of CCFS. As new, underlying storage technologies come into prominence, the task of implementing new content service wrappers around the technologies will be straightforward, and CCFS will improve with the continuing innovation in storage technologies.

Local File System

The local file system service stores and retrieves the objects from the local computer's file system. Unlike the web-caching of today, the local file service has all the authoritative power of a direct connection to the publisher. If a link is followed a second time or the same content appears in multiple collections, the local file system will supply the content, freeing the retriever from needing to use the network for the same content over and over again.

Multicast

The multicast service uses the local network to send a request to nearby computers within the same broadcast domain to discover content objects. It then uses a direct connection to retrieve them. By using local network discovery, a popular file will likely be discovered on the network. For example, if a computer wishes to update Windows, the first computer on a network will retrieve the content from the Internet. All subsequent computers will then discover that there is already a local copy of the content and use the local network to retrieve that content. This will reduce the amount of traffic exchanged with the Internet Service Provider (ISP).

Direct Http

The direct http service allows for one CCFS instance to query another for objects. If a user had more than one computer or a dedicated CCFS server for his or her organization (e.g. business or school), then he could query the server. The server would frequently have the content the user wants because others in his organization would use the server for similar queries. In this case, the organizational resources would likely overlap well. If an ISP wished to reduce the amount of peering bandwidth needed and improve the experience of their users, it could offer a direct http CCFS service to its customers.

Google Drive

The third service is the Google Drive service. This service uses the Google Drive Application Programming Interface (API) to store and retrieve objects from the Google Drive account of the service user. This is useful for personal files. Objects created by the user himself would need to be retrieved frequently. The use of personal file lockers creates a cost-effective method of hosting personal content that would not otherwise be widely available.

Kademlia DHT

The fourth service is the Kademlia Distributed Hash Table (DHT) service. This service allows for the content objects to be stored and retrieved from Kademlia, which is the same DHT used by BitTorrent. The Kademlia allows a potentially large community (e.g., millions of users) to pool their resources into a distributed content community. By sharing the burden of hosting content, this could enable a community to support very long tail requests at low cost. If authoritative hosts decide to no longer host content, members

of this community could preserve this content, allowing it to remain available long after the original host has ceased hosting. This service was implemented in a single semester, demonstrating the power that comes from the simplicity of the content service model.

2.4 Interfaces

Interfaces

The interface is an important part of any system. IPv4, the network layer protocol for the Internet, was able to radically alter the computing and communication landscape because it built technologies both on top of and below this layer. IPv4 created a narrow waist, as shown in Figure 5. The decoupling of the innovation in the transport layer from the innovation in the link layer accelerated each technology. A goal of CCFS is to keep the interfaces separate from the content services. This goal allows any application to store and retrieve content, independent of the underlying storage and retrieval technology. CCFS has three interfaces: the file system, http, and search.

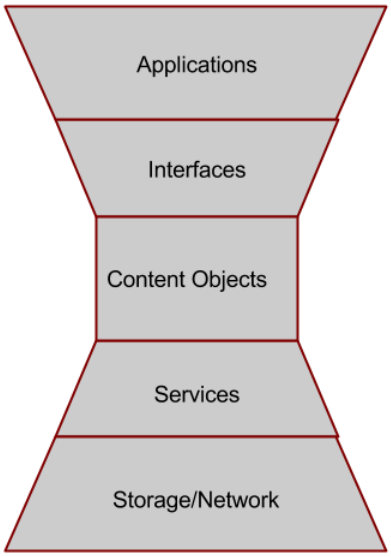

Figure 5. The hourglass nature of CCFS

Users will not adopt a cryptographic file system if they need to understand public and secret key cryptography to use the system. CCFS has two methods of interacting with

the user. The first is the file system, and the second is the web interface. With a minimal understanding of CCFS, these interfaces allow for the use of applications without modification; however, a CCFS developer can create additional interfaces that interact with the aforementioned services, if desired.

FUSE

The file system interface is displayed in Figure 6. The seamless exposure of CCFS is provided using a file system in Userspace (FUSE) [13]. The file system is the primary user interface of CCFS (Figure 6). The name comes from the fact that the file system is not running as part of the operating system (i.e., kernel space), but as an application run by the user (i.e., user space). FUSE enables the implementation of a fully integrated file system without the need to modify the operating system, as illustrated in Figure 7. If CCFS were run as an unprivileged user, then only an attack involving privilege escalation vulnerability would be able to compromise the kernel space.

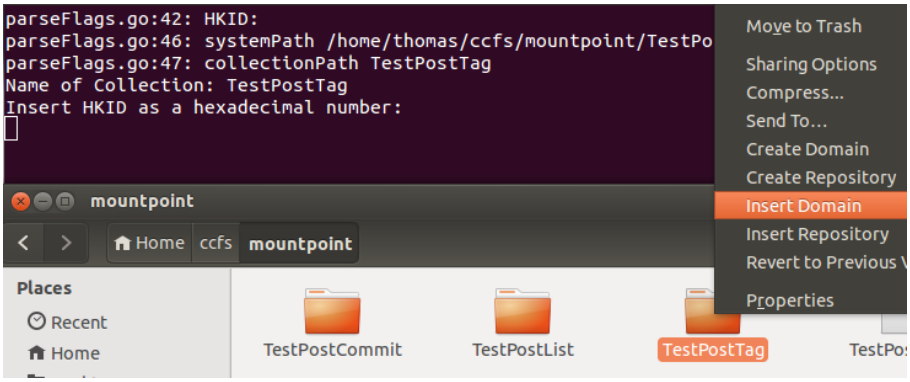

Figure 6. Mounted file interface.

The seamless exposure of CCFS is provided using a file system in Userspace (FUSE). The file system is the primary user interface of CCFS. The name comes from the fact that the file system is not running as part of the operating system (i.e., kernel space), but as an application run by the user (i.e., user space). FUSE enables the implementation

of a fully integrated file system without the need to modify the operating system, as illustrated in Figure 7. If CCFS were run as an unprivileged user, then only an attack involving privilege escalation vulnerability would be able to compromise the kernel space.

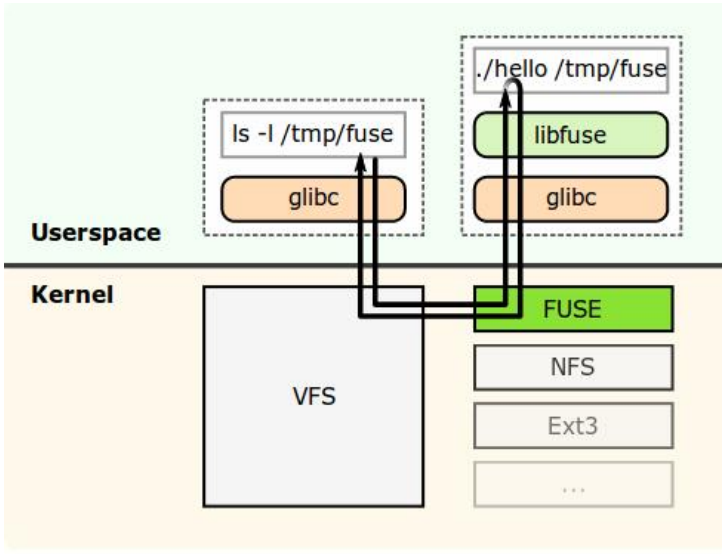

Figure 7. FUSE Architecture.

The user selects a mount point and the user's repository is mounted at that point, allowing the user to browse and interact with all of the collections she has curated as files and folders. Browsing those collections leads to still more collections and so on.

Http

The second interface is the web interface that is necessary for the gradual adoption of CCFS. By making the system accessible through standard http requests, a host who has no knowledge of CCFS will simply retrieve the content as a standard web get on a URL. At the same time, if a client recognizes the URL prefix and uses CCFS for the retrieve, he will gain all of the benefits of CCFS without the need to trust a third party.

There are two sets of functions provided in this interface. The first is the low-level function that enables the retrieval of the content objects using the object's HIDs. This is

useful when one CCFS instance talks to another or for a custom interface to interact with CCFS (e.g., merge, versioning). Low-level functions require the user to perform his own recursion. The second class is the high-level function that uses the CCFS paths to perform the complete recursive retrieval process in order to return the final named content. For example, if an application wanted access to older versions of content, the low-level function would be required. Otherwise, the high-level function can be used if the application wants to store and retrieve current content only. Low-level functions are known as plumbing, and the high-level functions are referred to as porcelain. These terms are borrowed from the Git Distributed Version Control System.

This interface allows browsers, and any other application that can open http connections, to have access to CCFS collections without needing any modification. This modification-less integration enables low-cost adoption and broad usability. CCFS can be placed in one of three locations: (1) the operating system's (OS) host file; (2) a multicast Domain Name System (mDNS) on a local network; and (3) DNS. OS places an entry in the host's file in order to redirect to the local CCFS host instance. The web interface is a server running on the local system, and an mDNS broadcasts an advertisement for that CCFS interface. In this way, CCFS's web interface is not meaningfully distinguishable from any other network resource. Lastly, in the event that the authoritative DNS is hit, DNS will redirect the user to a CCFS instance. These connection options provide a smooth ramp for a paradigmatic shift to content-centric networking without a single switchover point that would otherwise need to be synchronized across the Internet community.

Search

The third interface offered by CCFS is the search engine. This is presented in the form a website with a search box. Entering a search query will return a list of content

objects that contain the words from the string either in the content itself or the name of that content. CCFS ranks search results by the number of references that pertain to each object. In order to accomplish this, CCFS has a crawler, an indexer, and a search backend. The crawler is seated with one HID, pulls all the objects in that collection, and then adds those objects to the queue. CCFS repeats this process of crawling each discovered HID until all reachable collections are found, queued, and presented to the user. As the crawler encounters each object, it places information about that object into indices, and when a query is received from the user, CCFS searches these indices to generate a search result in an efficient manner. CCFS provides a more robust Internet without costing searchability, thus increasing its utility.

For the most part, the user does not type URLs into the address bar. Instead, URLs come from social media, search engines, and links found on webpages. Even if HIDs are neither memorable nor guessable, CCFS still supports content discovery. CCFS inherently has the ability to embed other curators' namespaces in the user's namespace. Because the CCFS names are strings, they can easily be shared through social media. However, the search function was not an innate part of the system. Search, therefore, became the first proof-of-concept application for CCFS. Although CCFS loses readability, the system still supports the most relevant name-distribution methods currently utilized by the Internet community. Finding and linking to content is still supported. The successful implementation of a search engine, crawler, and indexer demonstrates that it is possible to build utilities (i.e., applications) on top of CCFS.

CHAPTER 3

CCFS USE CASE STUDIES

Internet architects can use CCFS to re-implement the essential infrastructure of online naming and trust with all the robustness of a content-oriented system. Here, we describe how an Internet architect would solve common problems using CCFS; compare these solutions to the way in which the Internet's issues are presently resolved; and discuss how CCFS offers Internet users additional security.

3.1 Using Interfaces

CCFS is meant to provide an underlying key value store that can be exposed as a variety of interfaces, such as file systems (i.e., FUSE) or uniform resource identifiers (URIs; i.e., http and search engines). This section provides several use cases for cryptographically curated collections.

Link rot and the case law of the highest court in the land: Strong Links

Link rot is the condition of Internet links that are no longer working. This can occur when the link now points to nothing, or worse, when the link points to some resource that the author never intended. Link rot is one if the primary flaws in the Internet today [14].

When a document that is meant to be used for an extended period—if not indefinitely—links provided within the document should point to external content. One such example is Supreme Court case law. If you can change the meaning of a supreme

court opinion after the fact by altering the target of links that appear in the document, ***that is bad***. In the event that a Supreme Court document provides guidelines that are external to the document itself and the domain is allowed to expire, anyone can register that domain and place their content in the guidelines. Alarmingly, this new content would be U.S. case law, despite the fact that the justice/panel never intended it to be so. According to Liebler and Liebert, as many as 29% of websites cited by Supreme Court opinions are rotten [14]. Links within Supreme Court documents are a reasonable proxy for content that needs to be preserved.

To do this with today's technology, the best alternative is to copy any resource that we wish to cite to a file archive operated by the same organization that supports the perennial documents. This solution has two disadvantages: (1) these links can still rot if the organization fails to maintain the archive; and (2) archiving has severed the external documents from their authors. Any trust that those authors inspired is now lost because the source of the referenced document is the archive, not the original authors.

CCFS offers a better alternative for preserving vital links over the current Internet infrastructure. Instead of putting a URI in a document, we should use a CCFS name to refer to the external link. We embed the HCID of a specific collection (or version) and the path to the content in the document's bibliography. A specific commit refers to that collection within a specific moment in time. This version is still signed by the original author. Once the document is published, neither the justice nor the author of the referenced content can alter the external link since it is cryptographically curated. Because the object is signed, it maintains all of the authority of the original organization. The reader of the document can check the current state of the document because the specific commit/tag contains the

HKID. With the Internet today, the author of the long-lived document is forced to choose between archiving the resource and risking link rot. With CCFS, however, users get the best of both worlds: a user can look at the document as it was originally created. CCFS can cryptographically prove the document's authenticity. The user can also retrieve a specific version of this document. In the event that the document has been updated, a user would be able to pull the current version. With this method, even content like Wikipedia—that today cannot be cited because there is no way to know the status of the article—could now be cited without fear, since the strong link would refer to an exact version of the content.

The scientific world has long struggled with this challenge. We publish our work in journals, which cannot distribute our work nearly as effectively as the Internet could, because we need a way to cite each other's work. In the situation of long-lived links the link should be the hash of a commit/tag. Journals no longer need to be distribution channels if users can distribute their own work. Instead, journals become curators of collections of content that has been accepted. Journals would then publish lists of accepted articles by their HCIDs, bestowing these identifiers with the trust the public holds for the journal.

CCFS as a personal publication platform: Strong Links and Permissionless Distribution

If Alice, a curator wishes to publish a continually up-to-date portfolio of her work, she would name each new article into a repository. Bob, a subscriber, would only need to know Alice's HKID and could periodically poll for it. If he received a commit pointing to an HCID he has seen before, then Bob has up-to-date content; otherwise, Bob would download Alice's new lists and blobs that includes only the file and folders that have

changed. Since Bob does not need to get the content from Alice directly, he can receive content from any host that has it. All Alice needs to do is name the article into her repository and send the objects to any of her subscribers. As subscribers like Bob begin to poll for this content, they serve it to each other. More and more copies will be generated, making content increasingly available. In this way, the popularity of the content provides for its own hosting so the publisher need not fear that excessive demand will render the content unavailable to its intended users.

Domain Name System model with CCFS: Strong Links and Connectivity

The domain name system is a list of top-level domains (TLDs), such as .com, .edu, and .net or .ru, .dl, and .uk. Each of these TLDs, in turn, assigns authoritative domains, such as Google.com, gatech.edu, and Amazon.co.uk. These authoritative domains can then assign subdomains, such as ece.gatech.edu. Since these lists are essentially collections, this model maps well to CCFS.

The DNS root is a collection of TLDs delegated to the registry by creating a tag. The Internet Corporation for Assigning Names and Numbers (ICANN) places the registry's HKID into the root collection. Registries curate a collection of the authoritative domains for individual TLDs, curating the HKIDs of the registrants The authoritative domains are free to use their collections as they see fit, and the registrars must all agree on some multi-party signing algorithm to curate their shared TLD.

If Alice, a registrant, wishes to have her collection listed in the DNS system, then Alice could purchase a domain from a register and send her HKID to the registry. The same collection could be listed in multiple domains or accessed directly by its HKID. If Bob

wishes to maintain access to Alice's domain, despite the possibility that Alice might give up her domain name, Bob would simply name Alice's collection into his own. For example, if Bob discovered Geocities and wanted to preserve the data, he could name the Geocities HKID into his collection. When Yahoo pulls it from their collections for Bob, Bob still maintains access to the content via the link in his own collection and with the full authority of the original host.

Certificate Authority (CA) model with CCFS: Strong Links and Connectivity

Currently, the CA system consists of a list of root CAs that browsers trust. These root CAs sign intermediate CAs who, in turn, sign the certificates for domains. CCFS mimics this process with minimal alterations. This system could be implemented in CCFS by having each root CA be a curator. This curator names the intermediate CAs into its collection and issues a tag that refers to the HKID of the customers by their domain name. Certificates come in three flavors: domain validation (DV), organization validation (OV), and extended validation (EV). Different folders within the intermediate CA's collection can differentiate these types of certificates. Because certificates are chains of signatures, CCFS could enable users to have a CA system without needing to build and maintain a separate infrastructure.

CA, bond rating agency analogy: Independent Validation and Collective Consequence

In 2008, the global financial system collapsed due, in no small part, to the bond rating agencies that rated a series of extremely risky, mortgage-backed securities with AAA (i.e., riskless). The bond rating agencies worked not for the buyers, but for the brokers. In

this way, those agencies are hired and fired by the brokers who create the securities. A more diligent agency would lose billions of dollars of business to an agency that grants more desirable ratings [15].

Analogously, the person who is attempting to acquire an illegitimate certificate chooses the CA most likely to grant the certificate. Meanwhile, there is no advantage for the legitimate site owner in using an average CA. Because Bob can embed Alice in his name space in CCFS, Bob does not need Alice's permission to do so. This gives the curator Independent Validation, since curators do not control who references them, setting the stage for Collective Confidence.

Friend relationship graph: Connectivity, Independent Validation, and Collective Confidence

The theory of six degrees of separation states that an individual is six or fewer steps away from any other person on the planet. If curators maintain a public contacts collection with folders of friends, family, colleagues, and acquaintances, then the whole world can be connected and crawled via personal relationships. Each curator who is found can, in turn, be used to discover more curators. For example, a curator (Alice) who has announced her HKID to her contacts becomes a part of the web as soon as another curator (Bob), who is already in the web, places Alice's HKID in his collection. This is not only a channel for discovery, but also a context for trust. Discovered curators could be identified by the contact of a contact chain or by listing a number of contact paths from the user to the target.

Commercial relationship graph: Connectivity, Independent Validation, and Collective Confidence

A signal of trust is established when Alice, a curator, enters into a relationship with Bob, Inc., a vendor. Alice's folder of vendors contains the list of her vendors. Inclusion in Alice's list is a trust signal for all the vendors Alice has included. A second collection belonging Bob, Inc. could contain Alice if she agrees to be listed as Bob, Inc.'s customers. The same is true of collections containing the HKIDs of business partners. This network of commercial relationships differs in nature from the personal networks, but the same network crawling and reasoning techniques to propagate trust still apply.

Expert references graph: Connectivity, Independent Validation, and Collective Confidence

Many organizations are deeply trusted, but only with regard to their specific areas of expertise. These agencies can produce lists of the curators whom they trust. The curators the public would trust for food and drug safety are different from those trusted to protect civil liberties or make video game recommendations. Expert curators are named into collections, and, while complex systems of genre-specific trust are not yet a part of the system, the foundation has already been laid. A new interface is needed to establish genre-specific trust, but CCFS is versatile. In the event a user wants to build a genre-specific search engine, one can be built as an application on top of CCFS without needing to modify CCFS's structure.

Longevity and cancelled products: Strong Links

One selling point of CCFS is its propensity for long-lived objects. In today's Internet, the longevity of content is limited to the longevity of the hosting provider. In CCFS, anyone can host the data, since hosting does not require knowledge of any private

key. In this way, any entity that is willing and able to provide hosting can host the content. As new content hosts come online, they can retrieve the content from existing hosts. By passing content from one generation to the next, it can be perpetuated long after the original host has lost interest in the content. Today's Internet archives do not maintain a cryptographic assurance that the archive is the genuine reproduction of the site. With CCFS, the content is identified cryptographically anywhere the content persists. If found, content can be recognized, indexed, and reassembled into a web.

Reputation of HIDs: Independent Validation and Collective Confidence

An HID is an identifier whose authenticity can be proven. Not only can the holder of the HID make statements, but also others can use this HID to make statements about the holder. By including an HID in an attribute statement, the HID holder cannot prevent others from sharing his or her reputation data. In order for an HID to build a reputation, it must be exposed to the public. As a scam HKID grows in popularity, it will grow in infamy, eventually catching the attention of the blacklist administrator. The administrator may even terminate the scam HKID's activity before it reaches sufficient trust so as to deceive users. Reputation can also be community-specific. An HID could have a strong reputation for video game reviews, but the same HID may not apply in the arena of political punditry.

Shared cost of content maintenance: Permissionless Distribution

In CCFS, content refresher must maintain the content. This maintenance is accomplished by checking if there are a tolerable number of copies of the content available for download. If there the stock is insufficient, then a CCFS user can create a new copy by asking a new host to store an additional copy.

Content duplication is performed only after a content refresher notices that the available stock has dropped below a threshold. If one refresher demands three copies, but another demands seven, the burden of duplication falls to the refresher with the higher demand (i.e., the refresher who demanded seven copies), at no cost to the less aggressive refresher (i.e., the refresher who demanded three copies). Each time the stock falls to six or fewer copies, the more aggressive refresher will perform the duplication and restore the stock to seven copies. In this way, the cost of maintenance and storage is pushed to the most aggressive refreshers. Since it is not costly to be a less aggressive refresher, each content refresher would be willing to disclose his or her to set true demand for the redundancy.

In the event of a tie in which two different content refreshers have the same demand, replication will be mitigated by the refresher who first notices the deficiency. The randomness of this discovery will lead to an uncoordinated sharing of the replication burden between the refreshers with the same demand. One will discover approximately half of the regressions and the other maintainer will discover the rest.

In the end, CCFS acts as an auction system that incentivizes each refresher to bid his/her true hosting demand and share the cost of hosting across refreshers with similar demands.

Update 10 billion light bulbs: Permissionless Distribution and Strong Links

In the traditional model, a traffic spike of Distributed Denial of Service attack (DDOS) proportions will happen when a commercial device that has hundreds of millions of instances attempts to update its software. Each light bulb would need to establish an SSL

connection to a dedicated host to ensure that the light bulb installs only legitimate updates. This would require a hosting infrastructure capable of withstanding the barrage of update traffic. Using CCFS, the first light bulb would download the update from the authoritative host, and a subsequent update in the same autonomous system, or local network, will pull the update from a neighboring light bulb. If the original manufacturer of the bulbs goes out of business, each new bulb that is connected to the network will be updated to the latest software by the existing bulbs already on the network.

The benefits of a content-oriented system that retrieves trusted content from untrusted hosts could be demonstrated through the need to update the devices that comprise the Internet of Things. With such a system, the developer can work with his own, personal device, and, when satisfied with an update, name it into his collections and send it out to a small number of devices. These devices will spread the updates to other devices and, eventually, the whole network. In stark contrast to needing significant infrastructure for hosting updates, the developer, using CCFS, would need no dedicated hosting.

Broadcast channels: Permissionless Distribution

CCFS and broadcast technologies are complementary. If a piece of content is widely demanded, then a broadcast network could transmit it, and the objects could be verified. Some of the receivers will have errors, leading to objects that will fail to be verified. However, since the broadcast went to many receivers, the corrupted objects can be recuperated from a nearby receiver who has an uncorrupted object. The broadcast of the objects can have much lower redundancy, since receivers can repair each other's receptions, mitigating the consequences of errors. In CCFS, object availability tends to

grow exponentially (Figure 8). This growth starts slowly, but will eventually scale to many subscribers. By using a broadcast technology, CCFS can quickly get a number of hosts in the network to act as seeds, accelerating the speed of the initial publication of the content. Broadcast technologies help CCFS mitigate the slow start; CCFS helps broadcast technologies mitigate errors made during transmission.

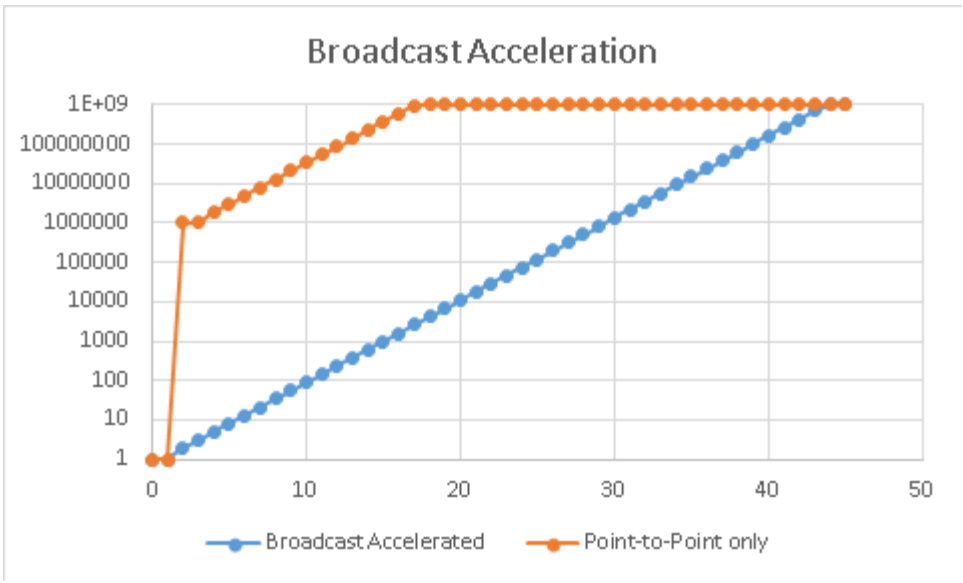

Figure 8. Broadcast accelerating speed to demand saturation.

Archiving Material: Permissionless Distribution

Archival remains a difficult task in today's Internet. Once a page has been downloaded, the source of the content cannot be rigorously verified. For example, Alice and Bob are having a dispute. Bob is using a picture from Alice's website that Bob claims was released under a creative commons license. Alice claims that the picture was never released under such a license. Bob goes to Charlie's web archive to find what Alice's webpage contained at the disputed time. In today's Internet, adjudicators rely on the trustworthiness of a third party (i.e., Charlie). Three issues emerge with this system: (1) the

Internet's trustworthy archives may not contain that website; (2) Bob cannot archive it himself because he is not an impartial party; and (3) the adjudicator may not trust the third party web archive. CCFS provides non-repudiation for all curated content because the curated content is all signed. Under CCFS, Bob can manage his own archive if he knows he is going to need the material, which would contain Alice's signature. If the content under dispute can be found anywhere, then it is just as trustworthy coming from Alice, Bob, or Charlie. In other words, if the content can be found, it can be verified, regardless of the chain of custody.

Archive aggregation and long-running queries

The HKIDs and HCIDs are the same, no matter which archive is being searched. Trust is cryptographically asserted and stored with the content. A search engine could index many different web archives, and these aggregated indices enable long-running searches. If Bob is looking for a copy of Alice's collection within a certain range (e.g., time frame), the archive aggregator could then notify Bob where content meeting his criteria has been found. In cases where content does not have a consistent host, thus appearing, disappearing, or moving unpredictably, CCFS can still retrieve and verify the content.

Pem encoding and web form distribution channels: Permissionless Distribution

Some content that, for whatever reason, has trouble getting or maintaining conventional hosting can be distributed by moving it around sites that allow user-generated text content. By encoding blobs as an ASCII armored output, they can be converted to text, posted to forums that accept text, and then sought with regular search engines. Once found, the blobs can be parsed, cryptographically verified, and indexed. Even if sites with user-

generated content frequently remove the content, it will be accessible, as long as the search engines and Internet archive sites cache the content.

One method for converting data into ASCII is Privacy Enhanced Mail (PEM) encoding. PEM encoding transforms a sequence of bytes into Base64 so that it can be sent through protocols like Simple Mail Transfer Protocol (SMTP), as well as fields labeled with ASCII text for searchability.

For example, a service could be written that performs a Google search for the HCID, HKID, or HKID and NameSegment of objects. The service could then follow the first thousand links and parse for any PEM-encoded CCFS objects. This use of forums anywhere on the Internet that accept text and are indexed by a search engine would lend a degree of anonymity to the publication of the objects. PEM encoding enables the distribution of objects in mediums meant for text, like IMs, chat rooms, or even books, magazines, and theses, increasing CCFS's versatility and robustness.

Multiplexing and first-come, first-serve: Permissionless Distribution

CCFS takes advantage of an array of performance characteristics by multiplexing these services against each other. Multiplexing is choosing among sources based on some variable (e.g., time or speed). CCFS relies on a variety of different content services to retrieve content objects, ranging from local file systems to globally distributed hash tables (DHT). The local file system is not dependent on the retrieval speed of the other services; the system will return the answer as soon as it has it (i.e., first-come, first-serve). The slow services cannot slow down the fast services, and the unreliable services can't tarnish the reliability of other services.

On the other extreme, the content distributed by the DHT is not influenced by the retrieval rates of other services (e.g., retrieval rates would be very low if the service has far fewer objects). When CCFS searches, it does so in parallel (e.g., local drive, local network, and DHT can all search simultaneously) so that slow services cannot slow down fast ones and shallow services that often fail do not reduce the robustness of wider searches. CCFS will pull content from the first service to respond with a successful find; otherwise, if the content is not found, the search will not come back at all. CCFS will be as reliable as the most reliable service and as fast as the fastest service. If the fast services fail, the slow services attempt to compensate. Fast services will eclipse any slow services that responded first to the query. CCFS will degrade gracefully as services that fail fall back to more reliable services.

Crawl Index and Search: Strong Links, Collective Confidence, Connectivity, and Permissionless Distribution

CCFS enables the creation of more efficient crawlers, indexers, and search engines. Crawlers pull content, search that content for links, and then follow those links, repeating this process until no new links are discovered. In the Internet today, the primary way to determine whether content has changed is to revisit that site periodically and download all the content. The content referred to by an HCID can never change. Therefore, once the user has discovered the HKIDs and HCIDs, the crawler has no need to repeatedly visit content referred to by this HCID for updates. Once indices of this content have been built, the indexer does not need to be rerun over this content, either.

While HCIDs are static, HKIDs are not. In other words, the crawler will need to subsequently revisit HKIDs. In the event that the content in the target collection has not changed, the new tag or commit will be downloaded and will have an already known HCID. The HCID will not need to be pulled again, keeping the cost (i.e., the effort required to crawl; bandwidth processing and storage) of static material low. Whether all, some, or none of the content has changed at the target, the crawler will only pull new material. This effort is cheaper than the traditional system, where all content is pulled in case any of it has changed.

Because there are version numbers and parents, a crawler can track how often a collection has been updating recently. By looking at the most recent version and how long ago the parent (and its parent, etc.) was created, the time between collection updates can be predicted. Conventionally, the history of collection updates has not been available. A search engine that has been crawling the same site can guess and may learn the frequencies over time. In CCFS, however, the history is readily available, so even with the first time an HKID is discovered, the user can examine the history to determine update frequencies. Knowledge of update frequencies allows the crawler to more intelligently schedule how frequently collections are revisited.

The primary goal of a search engine is to provide the user with the most relevant content in the least amount of time. Search engines do not intend to provide hijacked content, retrieve queries slowly, or yield broken links. Because the search engine embeds CCFS names in the response page rather than URLs, it certifies all the content that it is sending to the user. The user knows the retrieval he is seeing is the same page the indexer saw, and thus hijacking is no longer a concern. The search engine could give the user the

content directly through Permissionless Distribution, guaranteeing quality of service. If search engines can cache and serve the content without loss of authenticity, it can increase the performance for its customers by caching and serving.

In addition, the Internet under CCFS will not lose the ability to use graph algorithms for relevance (e.g., Page Rank). Instead, graph algorithms become arguably more democratic because a curator does not need to purchase a domain name or hosting resources to garner a reputation. Anyone can generate an HKID and start publishing under it. However, these graph algorithms are still susceptible to the same problems as previously occurred (e.g., Google bombing).

Content-caching communities: Permissionless Distribution

When caching content, difficulty tends to come from two places: some content is so popular that the demand will tend to disrupt the host attempting to serve the content; and there is a such a large variety of content that it is not cost-effective to have dedicated hosting resources for all the content. Most of this variety is contained within the long tail. The long tail is content that has lots of variety, but very little demand. Peer-to-peer technologies and half-life caches can mitigate both of these problems.

Peer-to-peer content distribution is becoming an important method of moving large, popular files around the Internet. The most notable example today is the BitTorent protocol. Modern BitTorent clients use a DHT to connect to peers, even if the tracker is not responding. This DHT is Kademlia and it has been shown to scale to millions of BitTorent users. CCFS also utilizes a Kademlia DHT to find peers with the content.

This peer-to-peer network content caching community will sever the long tail with slow, but very broad storage. Storing content potentially long after the original host has lost interest in hosting the content. Each retriever makes a copy of the content from someone who has it; therefore, each user who has retrieved content has increased the availability by himself. This increased availability enables more users to acquire the content and further increase the availability; such a feedback cycle is exponential in nature.

Half-life caches are defined as follows: When one cache runs out of space, the cache randomly deletes an item. The more copies of an item contained in half-life caches around the world, the more likely any one of these copies will be deleted. As the storage of these copies occupies a smaller and smaller percentage of the total storage volume, the odds that one of these copies will get deleted decreases proportionally to the number of copies in the caches. The rate at which content is deleted is sub-linear, since each removal reduces the commonness of that content and the probability of subsequent removals. This gives the content a half-life.

Satisfying increased demand necessitates generating a large number of copies, yielding an exponential growth curve that rises to meet demand, much like a charging capacitor (see Figure 9). If demand begins to drop, an exponential decline (i.e., half-life) results from the half-life cache until the number of copies reflects the new demand, like a discharging capacitor. Once the spike in demand ceases, the amount of time the content will be preserved will be proportional to its peak popularity and the availability will fall with a half-life. Content will only be lost if there is no demand for an extended period of time. If there is a persistent background demand for older content, these requests will serve

to maintain the content. If a refresher is maintaining a minimum demand, the content will stabilize at the demand of the refresher, rather than falling to zero.

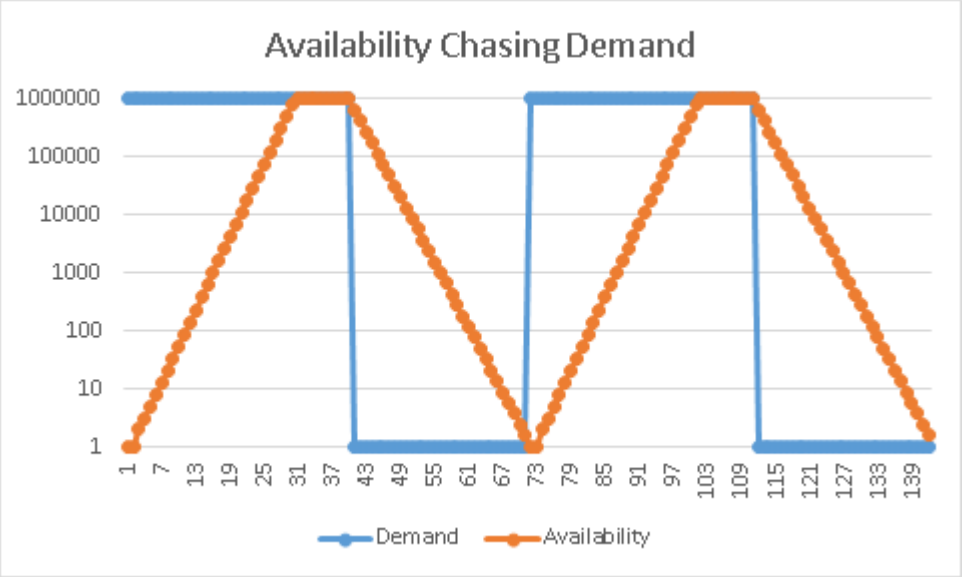

Figure 9: Availability chasing demand in the presence of a half-life cache.

Kademlia gives CCFS an efficient search capacity of many nodes. The peer-to-peer duplication gives CCFS scaled-up quantities of any content that will be stored in the content-caching community; the half-life caches yield scaled-down quantities; and the Kademlia search functions enable discovery of this content, no matter where this content is hosted. In this way, CCFS naturally scales the availability of content, both up and down, to meet the demand.

CCFS and friend-to-friend networks: Permissionless Distribution

Friend-to-friend networks are similar to peer-to-peer networks. In this configuration, the nodes connect not to just any node, but only to nodes run by people who trust each other. In a friend-to-friend network, each node sends a query to trusted nodes.

The answer follows this path of personally trusted nodes back to the original inquirer. A curator can publish to a friend-to-friend network through The Onion Router (TOR) in order to maintain anonymity [16]. Although these paths are not as efficient as the more general peer-to-peer networks, content caching communities that value privacy above performance would consider this a reasonable tradeoff.

3.2 Services

In order to function, CCFS will need services to be provided. It would not take many service providers for CCFS to have value, but as the number of providers grows, so will the utility of CCFS. Below are rationales supporting the operation of CCFS services.

Rational interest in re-hosting

There are many reasons why a rational actor might decide to re-host content here is an enumeration of just a few

- Reduce ISP bandwidth charges: Large organizations pay for Internet as a metered connection. If a lot of CCFS traffic crosses this metered connection, adding a CCFS server to the network will save the customer money, since any objects served by this server will not need to be sent across the metered ISP connection.
- Reduce peering costs: ISPs pay each other based on the amount of bandwidth transferred between them. An ISP that set up a group of servers to cache the most popular content in CCFS will not need to fetch that content from its peers. This will save the cost of peering, as well as improved performance for customers, since the content is cached more locally.

- Hosting own content: If a curator wishes to guarantee the availability of her own content, that curator could host her own content.
- Hosting as a service (commercial): A commercial entity could offer to host content for curators in exchange for compensation (e.g., Amazon web services, Google Compute Engine, Microsoft Azure).
- Content-caching community: In order to get higher priority for downloads, members of a content-caching community would be incentivized to cache and host content themselves. By prioritizing downloads of community members who have taken on the most hosting, content-caching communities could incentivize their members to host more content.
- Hosting to improve quality of experience (QoE): Content that is nearby can be retrieved more quickly, whether by an individual computer, a local network, an autonomous system, or a content delivery network. By providing increased hosting, the quality of the experience of one's customers can be enhanced.
- Historical content archival: Some data online is sufficiently important enough to warrant preservation. Many organizations (e.g., governmental, corporations, and small NGOs) have dedicated resources toward archiving and preserving information over time. These entities may archive information for business, legal, or personal reasons.
- Speculative content storage: If there is a company that believes they can identify content that someone will want at a later date, they can store that content in the hopes of being able to sell it in the future.

- e-grainery for limited connections: In some circumstances, some communities may not have a lot of bandwidth with the outside world. By operating as their own host, they can keep complete copies of their collections that they think are most important, thus enabling them to have access to content, even when the limited connection is down[17].
- Pigeons: One can use a CCFS host to move a large amount of data on a physical medium. For example, if a bus equipped with a CCFS servers goes between a town and a city, that bus can take all of the requests from the town's computers, download all of the relevant (i.e., requested) archives while it is in the city, and deliver this content to the inquirers once the bus returns to the town. This technique of storing content in mobile objects is called "pigeons," named for carrier pigeons that were used to deliver messages.
- Fans supporting a curator: If a curator has a fan base, that fan base would likely be willing to host the content named by that curator, especially if being one of the hosts meant that they would be the first to receive new content when the curator completed it and uploaded it to them.
- Altruism for free speech advocacy: If there is a group of freedom fighters, outsiders might be willing to host their content in order to help provide the publishers a greater degree of anonymity than if they had to arrange hosting through commercial means. Individuals or organizations that have some excess hosting resources could dedicate these idle resources to supporting causes in which they believe, such as making educational materials available to the public or providing hosting support through the belief that everyone has a right to be heard.

- Enhanced local performance: By operating a CCFS server on a personal computer, re-accessing content will be highly efficient and cost-effective. If the local node would like to have its own cryptographic assurances, CCFS enable personal verification of content.

CHAPTER 4

METHODS AND ANALYSIS

In addition to the conceptual architecture described in Chapter 2 and the case studies given in Chapter 3, this CCFS prototype was tested to demonstrate functionality, with benchmarks that exhibit its performance. This successful implementation of the prototype demonstrates the feasibility of the architecture. Because the prototype works, the CCFS architecture is a contribution to the field of information security. This chapter describes the aforementioned tests and prototype; and, therefore, serves as the proof-of-concept for CCFS.

4.1 Methods

The work on CCFS was intended to be a proof-of-concept implementation. This included both low-level service tests, as well as the high-level use of interfaces. While the performance was not the focus of the analysis, the test suite included benchmarks.

The low-level tests created and retrieved objects from the storage servers. These included the local file system, local network, Google Drive, direct HTTP, and the Kademlia distributed hash table (DHT). The prototype implementation [18] was written in the Go programming language (golang.org), with the exception of the Kademlia DHT, which was written in the Python v2.7 programming language. The tests and benchmarks were written using the Go testing library that is distributed with the language. The code for the prototype, the tests, and the benchmarks can be found at
github.com/AaronGoldman/ccfs/.

In order to test the CCFS prototype, two test beds were constructed. The first was for testing CCFS over local-area networks (LAN), and the second was created to test over wide-area networks (WAN).

The LAN test bed consisted of a laptop running Ubuntu 14.10 with an i7 and 8 gb of RAM. For the local network tests, a second laptop with Ubuntu 14.10an i7 and 8gb of RAM was added to the same local area network. The local tests consisted of the local file system and the local multicast DNS service. The local file system can be benchmarked on any computer, but only computers on the same LAN can test the multicast.

The WAN test bed included Google Cloud VM. The machine type was an f1-micro (1 vCPU, 0.6 GB memory) machine type, the zone was US-central1-a, and the type was a Standard Persistent Disk with a size of 10GB. The source image was a backports-debian-7-wheezy-v20141021. The wide-area tests consisted of the HTTP, Google Drive, and Kademlia services. The Google Drive tests were completed by running Google Drive on the Georgia Institute of Technology LAWN network. The Direct HTTP and Kademlia tests were performed over a WAN. These tests were accomplished by running instances of CCFS on a Google compute engine. Google provided $300 in compute engine credits for this experiment.

The tests consisted of directing each content object type to point to all other content object types, as illustrated in Figure 10. These tests can be run for each of the content services, as well as the multiplexer of all the content services. Once these low-level services were tested, high-level, end-to-end tests of the store-and-retrieve functions were performed over the CCFS interface. All tests were run three times: Once with LAN, once with WAN, and once with both network types (referred to as MUX).

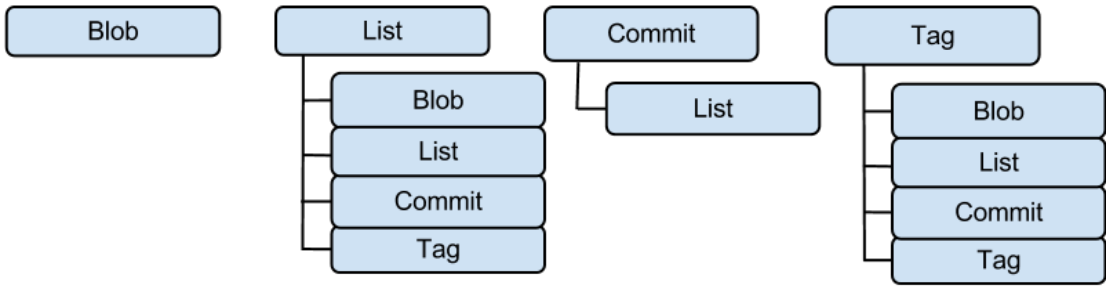

Figure 10. Type reference map.

4.2 Analysis and Results

The analysis of CCFS centers on successful and failed retrievals. Once a successful retrieval is achieved, then performance can be measured in terms of latency. Because CCFS differs from most other file systems that preceded it, it is unclear which file system benchmarks would be the most useful in evaluating the performance of CCFS. CCFS would perform poorly on traditional, stream-oriented file system benchmarks because CCFS is a versioned key-value store. The majority of file system benchmarks are stream-oriented; these benchmarks evaluate systems by making small changes quickly. CCFS is better equipped to make infrequent (e.g., once per second), large changes and, consequently, would compare poorly on traditional benchmarks. For this reason, human-perceivable latencies were the focus of the analysis.

Because traditional benchmarks are not well suited to evaluate CCFS on its five properties, we created a suite of tests from the proposed use cases described in Chapter 3. The tests were derived from tasks needed to solve the problems that arise in the case studies. This suite runs through the tasks necessary to curate all three types of collections: folders, repositories, and domains. Since all of the tests were successful, the suite demonstrates that the tools contained within CCFS can address a multitude of issues. They were run on an early CCFS prototype successfully, albeit slowly by modern file system standards. The suite of tests can be found in Appendix B, and the execution time (in milliseconds) of the tests running over the LAN, WAN, and MUX are displayed in Figure 11.

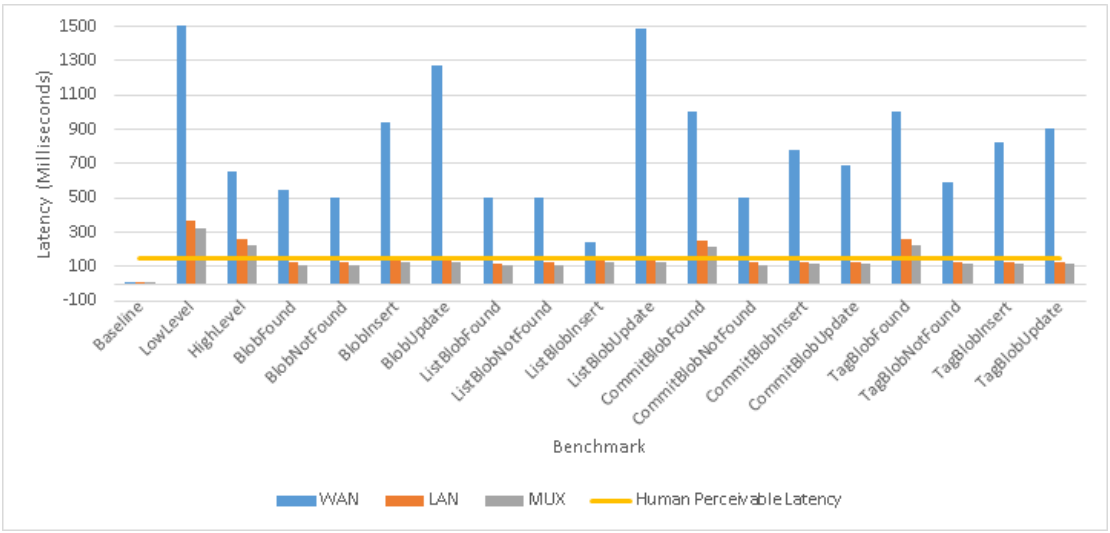

Figure 11. Benchmark results.

As illustrated in Figure 11, the LAN is predictably much faster than the WAN, since it is not limited by ISP performance. However, it is interesting to note that the MUX is the fastest of the three network types. Because CCFS is a first-come, first-serve system, a slow network request does not inhibit the speed of more local requests. Likewise, the small cache size of local requests does not hinder the long tail retrieval capabilities of the more global requests. The fact that the MUX shows the best performance serves as an affirmation of the CCFS architecture. By having multiple, competing providers of content, each provider can contribute its own strength to a whole that outperforms any of its components.

While this prototype show a lot of promise, the results in Figure 11 indicate clear limitations. The test suite included baseline tests to determine how long it would take a conventional file system operating on a local hard drive to perform similar tasks. The results cannot be seen on the graph, as it took less than one millisecond to complete. Yet, the speed at which the prototype operates is not comparable to that of a native file system. This gap in performance demonstrates that CCFS is not suitable as a drop-in replacement for a traditional file system. Therefore, CCFS is not suitable for high-speed, automated uses of the file system that would rely on significantly greater speed and parts of the

Portable Operating System Interface (POSIX) application-programming interface (API). All told, CCFS will not take a very long time to perform the tasks typical of a file system and will be improved in future versions. Despite this limitation, CCFS trades off speed for the assurances of the five properties: Strong Links, Permissionless Distribution, Independent Validation, Connectivity, and Collective Confidence.

CHAPTER 5

CONCLUSION

This work demonstrates a layered approach that enables the creation of a Cryptographically Curated File System. A focus on Strong Links, Permissionless Distribution, Independent Validation, Connectivity, and Collective Confidence lead to a cryptography-based design. While the techniques are not new, their application as the references within a file system is a novel contribution to the field.

Strong Links

CCFS provides Strong Links because content can only be removed from a collection by the action of a curator, as compared to link rot, where links "go bad." Link rot occurs over time, not through deliberate alteration or removal of the referenced link or material. CCFS ensures non-repudiation of material, so long as users retain the material, regardless of whether the curator decides to remove the material from his or her collection.

Permissionless Distribution

CCFS provides Permissionless Distribution because of its other properties. Each curator can make his or her own collections if he or she has access to any device that can perform these computations. Rather than having a central naming system, each curator is responsible for naming within his or her collections and does not need external permission to be a curator.

In a system with a centralized, global namespace, curators must coordinate in order to guarantee that names remain unique. This is done by dividing up a top-level namespace and giving each of those domains the ability to assign subdomains. While this enables individuals to have control over their own domains, it leaves the parent domain with the

ability to seize the subdomain namespace. Parent domains can use this ability to reclaim names as leverage to impose control over the subdomains. Meanwhile, the parent domain is subject to similar pressure from its respective parent, all the way up to the root. This leaves the root, a point of control, vulnerable and thus irresistible to those who would wish to coerce influence over the Internet.

Alternatively, in CCFS, each curator is his or her own root. Once Alice discovers Bob, another curator of interest, she can name Bob's HKID into one of her collections. Conversely, both Mobility First and DONA, competing technologies with nicely distributed software architectures, require a centralized namespace that stems from a root in order to be useful. In centralized systems, a domain can put prerequisites on namespace. Since all namespace flows from the root and subdomains require the permission of parent domains to distribute content, the root has the power to impose conditions for being a part of the Web. Without unique names, content-centric networking cannot be done. However, with a decentralized system like CCFS, a curator does not require the permission of a parent domain to distribute his or her content. As such, there is no global namespace, only local names. Since Bob is the only one who knows his private key, no other curator can take away his ability to distribute his content or Alice's ability to point to it. In this way, CCFS has created a global naming convention that does not impinge on an individual curator's freedom to distribute content without permission.

Independent Validation

CCFS provides Independent Validation because any curator can include any collection within his or her own collection and choose its name. This allows for many parties to vouch for a particular collection and, thus, the curator of the collection. It is a delegation of trust: Because Alice trusts Bob and Bob vouches for Charlie, Alice will also trust Charlie.

Unlike the CA, which only enables positive trust, CCFS also enables blacklists. Because Charlie cannot prevent Bob from telling Alice about Charlie, Charlie is not solely

responsible for how Alice views Charlie. Charlie's validation is independent because he cannot control into whose collections he has been placed (e.g., Candy Mountain) or what name his content is given (e.g., Alice refers to Charlie as "Unicorn"). This shifts power from the curated to the curator, enabling richer systems of trust.

Connectivity

CCFS provides Connectivity because not only is each curator able to embed others through naming content, but embedded curators also have the same power. A crawler flows from collection to collection, discovering more collections and curators along the way. In this way, this web of connections could grow to a truly global scale, granting us global Connectivity without a managed global namespace and using only local names. Therefore, there is no competition for global names because there is no global namespace. CCFS supports Global Connectivity without the cost of maintaining a global state.

Collective Confidence

CCFS provides Collective Confidence as it arises from two other CCFS principles: Independent Validation and Connectivity. When Connectivity is used to discover a large number of curators that all point to the same collection, collective knowledge can be used to make judgments about this collection. Reputation can be built based on the fact that curators can become well known. In this way, the highest levels of trust are achieved only by collective consensus.

Conclusion

Our recommended use is to mount CCFS to a folder and only save content there when the content is ready to be published. Streaming data is not suitable for CCFS; data

that can be batched and copied, however, would interact far better with CCFS' performance characteristics. Although CCFS does not have drop-in replacement capabilities, we feel that this use of the OS's copy command as CCFS' commit-and-publish makes the system both easy enough to use and fast enough to be useful.

The valuable use of CCFS comes from the use cases of backing up one's own content, publishing content, and versioning content that changes over time. Using CCFS to back up files gives the user cryptographic assurance that the back-ups have not become corrupted. By publishing content with it, consumers can fulfill each other's demands for the content once a single copy has been published. Since CCFS has references to previous versions of content, it can be used to go back in time, if necessary, and recover older versions of the content. Likewise, two diverged copies of content can be merged together. CCFS can detect instances of overwriting and merge the diverged histories into a single, current version, resolving the discrepancies. Presently, we tend to use very different technologies for personal file storage, publication and distribution of files, and versioning of our work. However, CCFS can serve all of these roles while providing far stronger security guarantees.

The security properties of Strong Links, Permissionless Distribution, Independent Validation, Connectivity, and Collective Confidence support the use case scenarios, resulting in a freer and more robust Internet. We believe the CCFS architecture proposed in this thesis successfully provides these theoretical security properties in a practical way. These properties, while valuable, do not come without a cost. CCFS uses more storage because content is stored in more than one place; uses more bandwidth because it requests content objects from multiple local locations; and requires more processing because hashes must be check and signatures must be verified. However, despite these limitations, CCFS offers the advantage of lower round-trip delay. This advantage comes from a content-centric architecture that provides maximal locality (i.e., content is stored as close to the users as possible) without sacrificing the security properties. This resource consumption is

a reasonable trade-off between storage, bandwidth, and processing, which are all becoming cheaper, and the speed of light, which remains constant.

CCFS decentralizes both storage and naming, allowing anyone to create and maintain collections of content. This enables the architecture to be more democratic [19] than its predecessors. The absence of a global namespace leads to a namespace that does not impose the arbitrary privilege held by a namespace manager. Any curator can organize his or her own content, as long as any community is willing to keep it accessible. Communal trust, hosting, and reputation allow for a democratic cacophony of voices that goes beyond a mere tyranny of the majority to provide a democratic public forum. This curation system provides the tools for sub-communities to organize content in accordance with whatever ideals they hold, while at the same time preserving the global namespace for free expression. In conclusion, the authors believe that the CCFS architecture [20] is a valuable contribution to the field of information security because the Cryptographically Curated File System offers democracy, performance, and robustness.

APPENDIX A

CONTENT OBJECTS

Blob 9914ab23f1ce1974f3de7976529b2534f473def11c5bc829aa2d72afc8c1d860

-----BEGIN BLOB--------

Contents of the file

-------END BLOB--------

Tag 9fa649180b7432ed9af0c3d2edba3d5b881decbf5386af629a7c952c3b95ac28

-----BEGIN TAG--------

9914ab23f1ce1974f3de7976529b2534f473def11c5bc829aa2d72afc8c1d860,

blob,

file,

1418139493730903105,

e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855,

4448d9b9116395012934705067b92aecbe983b7ee349f872575c6ef21fe535c6,

0400a53f37830d53a6da974ec7a37a5fdaf4af099e7e4ee9d95b81b45cc38511dfa7431

89d33e26264661e93bf6be95cbd56445f859d71529983446c92d4d7c35594d0016e4abd

196e66ff170b23605359872d636f9720839e43626d4d8099614a98b0b14a369b73020d6

8286664fe2b7bdd482273d72988a17ad2367697c1b0d670d3efb8

-------END TAG--------

```
List 8d89150c5d53a769d09548a6a2536a1f8b5ccdfb3761218a6a1ba482e2a124ae
-----BEGIN LIST-------
4448d9b9116395012934705067b92aecbe983b7ee349f872575c6ef21fe535c6,tag,to
-------END LIST-------


List 89e7de6393b270190ec3becb911c3bee640b11df908820793276318661e0ee50
-----BEGIN LIST-------
8d89150c5d53a769d09548a6a2536a1f8b5ccdfb3761218a6a1ba482e2a124ae,list,p
ath
-------END LIST-------


Commit 5165140a59d7abb6fa24c60866bee987c25ce4ece7bd87cf023a3f01600d6b96
-----BEGIN COMMIT-------
89e7de6393b270190ec3becb911c3bee640b11df908820793276318661e0ee50,
1418139493751374464,
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855,
880b5cbb8e788e549f5830ab145e98478817c1d8d8ff76a6e46845e741384db2,
0400a8127e7b17cfa9cde02fa2598d788e6e6d302778e5244062aa6580ea06b0ebcd085
faf926b5ef981511852e01853c61fe10cdbc0b376681cc4bf6b2eacd633b93c011d376b
35b0bdd4528112a8c44b72ef04d6df15a21ffe638cf3ea4942ebdf740308012432602f5
03ed220b7f087f4793cee0172d3bff9a763ef56e988b52fff228e
-------END COMMIT-------
```

taghkid:
4448d9b91163950129347050 67b92aecbe983b7ee349f872575c6ef21fe535c6
tagD:
30523977409793671218517668231153153538928929565895175942831440520
38270206976946542302868839710993215915089139582451818876185479053803765
155029790903622401367

```
hexdump -C 4448d9b91163950129347050 67b92aecbe983b7ee349f872575c6ef21fe535c6
00000000  04 01 5b d6 1f d7 c0 6e  66 d2 0a 20 97 36 ff 73  |..[....nf.. .6.s|
00000010  36 c1 fd 2e 04 2f 0e 03  35 0f 9c db ab d7 46 ad  |6..../..5.....F.|
00000020  2a 6d dc 24 f6 6b d3 4a  3b 16 d2 37 5a a4 c1 0e  |*m.$.k.J;..7Z...|
00000030  76 fa a5 62 b5 3e 1e 5a  60 64 bb 28 71 87 07 8a  |v..b.>.Z`d.(q...|
00000040  3d 3b 83 00 d8 3e 18 fa  21 04 7e f8 a5 9f 41 e7  |=;...>..!.~...A.|
00000050  e7 0c 53 9f a5 24 8a 19  2e 70 ac 40 73 96 32 d1  |..S..$...p.@s.2.|
00000060  25 c0 72 3e d0 44 34 77  c2 e1 3c fd 49 23 c0 bf  |%.r>.D4w..<.I#..|
00000070  16 c8 58 89 92 fe fc a4  e3 e4 02 f2 8b 27 b7 fd  |..X..........'..|
00000080  80 73 8f 55 ae                                    |.s.U.|
00000085
```

commithkid:
880b5cbb8e788e549f5830ab145e98478817c1d8d8ff76a6e46845e741384db2
commitD:
44527274704768423153382349147334223795196483682735383217006839495
04181241890147539847759883020942643821462168277085472099350155347325000
234047557001442840709

```
hexdump -C 880b5cbb8e788e549f5830ab145e98478817c1d8d8ff76a6e46845e741384db2
00000000  04 01 0c 7f 75 b4 f2 b1  fd 82 3e 52 b9 d4 ee 16  |....u.....>R....|
00000010  d3 5d e6 4f 49 9c 94 e1  2f 6d 6a b8 de af df 24  |.].OI.../mj....$|
00000020  3d b2 59 4c 4e 05 24 03  c2 bc 52 a0 86 c9 ef 53  |=.YLN.$...R....S|
00000030  8d c0 2f ad 58 a1 e2 6e  43 4b 7f 70 4a 69 de e2  |../.X..nCK.pJi..|
00000040  0c 7e b0 00 d4 dc b0 6c  18 d3 ef df e2 fb 41 f7  |.~.....l......A.|
00000050  0c 20 86 28 76 32 1c 81  9f ea 91 6f f8 c5 4c f1  |. .(v2.....o..L.|
00000060  38 a9 b0 f2 81 f3 62 b5  ad 8f 19 5d 91 aa b6 84  |8.....b....]....|
00000070  b9 78 ac 3d be 62 24 40  f2 d6 42 68 9e 96 00 07  |.x.=.b$@..Bh....|
00000080  87 a7 83 4b 16                                    |...K.|
00000085
```

APPENDIX B

BENCHMARKS

- Baseline: serves as a baseline and stores a file to the conventional file system
- LowLevelPath: calls the plumbing functions directly; test the object-creation system without having to run the safety or processing features of CCFS (i.e., the services of CCFS: calls post blob, get blob, post list).
- HighLevelPath: generates a series of example objects [see Appendix A] that serve as a proof of the [functionality of the system].
- BlobFound: retrieves a blob that has been named.
- BlobNotFound: attempts to retrieve a name that is undefined.
- BlobInsert: assigns a blob to a previously undefined name.
- BlobUpdate: assigns a new blob to a name that was previously defined.
- The following four functions can be found within a folder:
    - ListBlobFound: retrieves a blob from a name within a folder.
    - ListBlobNotFound: attempts to retrieve a name that is undefined within a folder.
    - ListBlobInsert: assigns a blob to a previously undefined name within in a folder.
    - ListBlobUpdate: assigns a new blob to a name that was previously defined within a folder.
- The following four functions can be found within a repository:
    - CommitBlobFound: retrieves a blob from within a repository.
    - CommitBlobNotFound: attempts to retrieve a name that is undefined within a repository.
    - CommitBlobInsert: assigns a blob to a previously undefined name within in a repository.

    - CommitBlobUpdate: assigns a new blob to a name that was previously defined within a repository.

- The following four functions can be found within a domain:
    - TagBlobFound: retrieves a blob from within a domain.
    - TagBlobNotFound: attempts to retrieve a name that is undefined within a domain.
    - TagBlobInsert: assigns a blob to a previously undefined name within in a domain.

TagBlobUpdate: assigns a new blob to a name that was previously defined within a domain.

REFERENCES

[1] Raizel Liebler & June Liebert, “Something rotten in the state of legal citation: the life span of a United States Supreme Court citation containing an internet link,” 2010.

[2] Laurie, B., A. Langley, and E. Kasper. "RFC 6962 - Certificate Transparency." *RFC 6962 - Certificate Transparency*. Internet Engineering Task Force (IETF), June 2013. Web. 17 May 2015. <https://tools.ietf.org/html/rfc6962>.

[3] G. Tyson, N. Sastry, R. Cuevas, I. Rimac, and A. Mauthe, “A survey of mobility in information-centric networks,” Commun. ACM, vol. 56, pp. 90–98, Dec. 2013.

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, “Networking named content,” in Proceedings of the 5th international conference on Emerging networking experiments and technologies, CoNEXT ’09, pp. 1–12, 2009.

[5] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A data-oriented (and beyond) network architecture,” SIGCOMM Comput. Commun. Rev., vol. 37, pp. 181–192, Aug. 2007.

[6] G. Tyson, A. Mauthe, S. Kaune, P. Grace, and T. Plagemann, “Juno: An adaptive delivery-centric middleware,” in Consumer Communications and Networking Conference (CCNC), 2012 IEEE, pp. 587–591, Jan. 2012.

[7] K. Visala, D. Lagutin, and S. Tarkoma, “Lanes: An inter-domain data oriented routing architecture,” in Proceedings of the 2009 Workshop on Re-architecting the Internet, ReArch ’09, (New York, NY, USA), pp. 55–60, ACM, 2009.

[8] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, “Mobilityfirst: A robust and trustworthy mobility-centric architecture for the future internet,” SIGMOBILE Mob. Comput. Commun. Rev., vol. 16, pp. 2–13, Dec. 2012.

[9] F. Zhang, K. Nagaraja, Y. Zhang, and D. Raychaudhuri, “Content delivery in the mobilityfirst future internet architecture,” in Sarnoff Symposium (SARNOFF), 2012 35th IEEE, pp. 1–5, May 2012.

[10] Tahoe-LAFS project, “Tahoe-lafs.” https://tahoe-lafs.org, Mar. 2013.

[11] Brad Fitzpatrick, “Camlistore content-addressable multi-layer indexed storage.” http://camlistore.org/, Mar. 2014.

[12] Marc Stiegler, “An introduction to petname systems.” http://www.skyhunter.com/marcs/petnames/IntroPetNames.html, Jun. 2010.

[13] FUSE Project, “Filesystem in userspace.” http://fuse.sourceforge.net/, Mar. 2014.

[14] Liebler, Liebert “Something rotten in the state of legal citation: the life span of a United States Supreme Court citation containing an internet link” http://yjolt.org/sites/default/files/Something_Rotten_in_Legal_Citation.pdf, 2013

[15] Lewis, Michael. *The Big Short: Inside the Doomsday Machine*. New York: W. W. Norton, 2010. Print.

[16] The Tor Project, Inc., “The solution: a distributed, anonymous network.” https://www.torproject.org/about/overview.html.en#thesolution, Mar. 2013.

[17] Goldman, A.D.; Uluagac, A.S.; Beyah, R.; Copeland, J.A., "Plugging the leaks without unplugging your network in the midst of Disaster," Local Computer Networks (LCN), 2012 IEEE 37th Conference on , vol., no., pp.248,251, 22-25 Oct. 2012, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6423620&isnumber=6423567

[18] Aaron Goldman, “Cryptographically curated file system.” https://github.com/AaronGoldman/ccfs, Mar. 2014.

[19] "Democracy." Def. 6. *Democracy*. Springfield: Merriam-Webster, Incorporated, 1997. 207. Print.

[20] Goldman, A.D.; Uluagac, A.S.; Copeland, J.A., "Cryptographically-Curated File System (CCFS): Secure, inter-operable, and easily implementable Information-Centric Networking," Local Computer Networks (LCN), 2014 IEEE 39th Conference on , vol., no., pp.142,149, 8-11 Sept. 2014, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6925766&isnumber=6925725