Cryptographically-Curated File System (CCFS): Secure, Inter-operable, and Easily Implementable Information-Centric Networking

Aaron D Goldman*, A. Selcuk Uluagac†, John A Copeland*
*Communications Systems Center (CSC)
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
{goldman}@gatech.edu, {john.copeland}@ece.gatech.edu
†Electrical and Computer Engineering Dept.
Florida International University
Miami, FL 33174
{arifselcuk.uluagac}@fiu.edu

***Abstract*—Cryptographically-Curated File System (CCFS) proposed in this work supports the adoption of Information-Centric Networking. CCFS utilizes content names that span trust boundaries, verify integrity, tolerate disruption, authenticate content, and provide non-repudiation. Irrespective of the ability to reach an authoritative host, CCFS provides secure access by binding a chain of trust into the content name itself. Curators cryptographically bind content to a name, which is a path through a series of *objects* that map human meaningful names to cryptographically strong content identifiers. CCFS serves as a network layer for storage systems unifying currently disparate storage technologies. The power of CCFS derives from file hashes and public keys used as a name with which to retrieve content and as a method of verifying that content. We present results from our prototype implementation. Our results show that the overhead associated with CCFS is not negligible, but also is not prohibitive.**

***Index Terms*—Inter-operable Heterogeneous Storage, Content Centric Networking (CCN), Delay Tolerant Networking (DTN), Information Centric Networks (ICN), Name Orientated Networking (NON), Cryptographically Curated File System (CCFS), Self Certifying File Systems**

I. Introduction

The traditional Internet was designed to accommodate a body of users who desired to visit specific network locations. Today, the majority of users are more interested in obtaining a specific content of knowledge 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 [1], [2]. In the literature, this work is variably referred to as information-centric, name-oriented, content-centric, and other similar names. Although irrefutable advantages have been soundly defended by the researchers of Information-Centric Networks (ICNs), there has been relatively slow movement toward any mass adoption of ICNs. We believe this resistance rises from a combination of concerns about security, interoperability with legacy networks, and difficulty of implementation.

In this paper, we propose a secure, inter-operable, easily implementable first-step toward adopting ICNs in the form of a *Cryptographically Curated File System (CCFS.)* In this file system, any individual or entity serves as a *curator*, or collector of content objects. The curator of content binds a name to the content by means of a cryptographic signature. This content name is the path through a series of *Content Objects* that map human readable names to cryptographically strong content identifiers (CID). The names are based on cryptographic operations to enable multiple independent, but inter-operable, namespaces. By relying on cryptographic properties of content, rather than location-relative identifiers, the content can be moved between systems without the need to translate content names from one naming model to another. The dual usage of cryptographic hashes and public keys, as both a name with which to retrieve content, as well as, a method of verifying that content, adds security to the system. CCFS serves as a network layer for storage systems and can be used with any ICN or Content Delivery Network (CDN). CCFS can unify currently disparate storage technologies into one universal storage pool. CCFS is, therefore, a system of file hierarchies that is both storage-method and delivery-method agnostic, making it inter-operable and easily implementable. Additionally, CCFS can be readily used by consumers of content because of its familiar file system interface. Furthermore, Global adoption is not required for local use of CCFS. Any local community can adopt CCFS protocols and immediately reap the benefits of ICN.

Moreover, CCFS simplifies the current dominant networking paradigm which includes three distinct hierarchies for naming, trust, and retrieval into only one naming hierarchy. The dominant naming model is the DNS; the trust model is the X.509 certificate chain; and the retrieval model is CDN (content delivery networks). In CCFS, the naming convention serves as all three: a unique identifier to reference content, as a trust model to verify the integrity of the content and its inclusion in a curated collection, and as a sequence to be used to retrieve the content.

CCFS maps between two types of names. One is the hierarchical arbitrarily-chosen name that is required by humans and by many applications. The other is a flat self-certifying unique name that is capable of being used by a name oriented network. CCFS provides this mapping in a cryptographically secure way by using hashes and signatures. This mapping will cause name oriented networks to be compatible with existing file-systems as well as with the DNS paradigm. This will allow for wide adoption, making the benefit of name oriented networking economically leverageable.

Finally, under the current Internet paradigm, content is trusted because the receiver trusts the deliverer of the content, the content host. Consider the X.509 for the name

978-1-4799-3780-6/14/$31.00 ©2014 IEEE

”mail.google.com”. The name ”mail.google.com” is signed by ”Google Internet Authority G2”, which is signed by ”GeoTrust Global CA”, which is signed by ”Equifax Secure CA”. This X.509 chain of trust is then used to bind the DNS name to Google’s CDN servers. The end user trusts the front-end server and that server authenticates the back-end servers. This creates a large number of sources of authority. The compromise of any one of these sources of authority will lead to content integrity becoming vulnerable. If any Certificate Authority (CA) in the X.509 chain, or any domain along the dotted URL path, or any front end server is compromised, then the whole system fails. Even when these systems are working, moving content or services between domain names or certificate authorities can be difficult and costly.

The necessity of highly trusted CA’s and content hosts, however, is a construct of the current architectural design, not a fundamental necessity. With CCFS, only one hierarchy is necessary. A curator, an individual or entity that has the private key to a collection of content, can add or remove content to the collection. The curator adds content to the collection by naming the content and signing the new version of the collection. The content name is a consistent pseudonym, capable of being associated with a level of trust. The receiver of content can verify that the content is identical to the content of interest and was named by the claimed curator. Curation is accomplished by using cryptographically strong hashes and signatures. The hashes and signatures are references connecting component parts of a human readable name. Under CCFS, anyone who possesses specific content can deliver that content. The content is trusted because the receiver trusts the curator. Any specific content could be contained within any number of collections, under any number of names. The content host does not have to be the content curator.

With CCFS, the trust hierarchy is inherent in the naming hierarchy. Whereas, in the dominant public key infrastructure, certificate chains certify connections, as described above; in CCFS, object chains certify content. With CCFS a chain of trust persists within the content name, rather than with the chain of Certificate Authorities. The content is independently verifiable, irrespective of the host that provides the content. Content durability, therefore, is no longer limited by the longevity of authoritative hosting. This limitation is a problem colloquially referred to as link rot.

In Section II of this paper, we discuss the related work. Section III includes a description of the CCFS model. An evaluation of the prototype implementation along with the results is presented in Section IV. Lastly, Section V presents our conclusions.

II. Related Work

The pioneer efforts in information centric networking can be found in Van Jacobson’s work [2]. Other models have taken a variety of approaches to ICN, but none have been widely embraced. In this section, we briefly discuss the related work in this domain of research and suggest how CCFS offers advantages or compliments other approaches. CCNx [1], [2] uses hierarchical naming that includes the publisher and a content identifier along with other information. Consumers of content make requests for content based on the content identifier and the publisher directly, and the request is routed to the closest host that has the content. There is no conversion from the content identifier to a location address, and there is no use of cryptography. DONA [1], [3] uses names that contain a cryptographic hash of the publisher’s public key and a content identifier. The content names are registered along with a location address with a Name Resolution Handler, which relates content to the authorized points that store the content. The Resolution Handler must be available and contacted before deliveries can be made. Net-Inf [1] does not use cryptographic keys directly as identities, but can be included in an identification object. This is similar to DONA in that it registers content that it calls Information Objects along with their location addresses to a Name Resolution Service (NRS). The system is not dynamic and the NRS must be updated as providers, and often consumers, move to new locations. Juno [1], [4] uses flat self-certifying names that identify content and utilizes a Juno Content Discovery Service (JCDF), as well as, other third party indexing services. Location Addresses are used for routing as in Net-Inf and DONA. Juno also allows for choosing content hosts by multiple criteria, as opposed to simply looking for the nearest location of the content. The JCDF must be updated to show changes in provider locations. LANES, Logical Address space Network [5], uses cryptographic naming for Name Identifiers and Scope Identifiers (SI). Subscribers use Application Identifiers, 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) which leads the consumer to the location address of the content. The SI is used to restrict certain access to content. Note that the local Rendezvous point and service must be accessible to retrieve the content. Mobility First [6], [7] is a multiphase, significant, joint effort of multiple universities, that is taking a more comprehensive approach to developing a clean slate content-centric network architecture from a ground up perspective. This project utilizes many of the concepts examined by others, such as cryptographic self-certifying names, Global Naming Conventions with Human Readable Names that are converted to Globally Unique Identifiers with a Global Name Resolution Service (GNRS), and converted into Network Addresses. The Mobility First project is a work in process, however, the system will require cooperation in the adoption of a global common naming convention and the use of a GNRS.

The other area of research that relates to CCFS is work done on self-certifying file systems. Two examples that are similar to our work are the Least Authority File System (Tahoe-LAFS) [8] and Content-Addressable Multi-Layer Indexed Storage (Camlistore) [9]. Although similar to the authors’ proposal, Tahoe-LAFS has different goals from our work. Its reliance on cryptographic capabilities is effective for maintaining the principle of least privilege. In contrast, our focus is on curation aspects such as authenticity, and non-repudiation, as well as

on human readable names. LAFS has chosen to include the global and secure aspects of Zooko's triangle [10], 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 only her own globally secure ID, and using the tool of curation, the user can collect the global IDs of others and assign them memorable names within the curator's own collection. Camlistore is another content-addressable storage which incorporates indexing and searching for content. Although Camlistore is content-centric, CCFS differs from this model in its architecture and protocols.

III. Cryptographically Curated File System (CCFS)

CCFS is a trust model based on a distributed system of content objects that can be cryptographically referenced and authenticated by their hashes and signatures in a hierarchy defined by chains of trust. The chain of trust in CCFS results from the curator choosing which other curators to include in their namespace as opposed to X.509 where content hosts choose the authority that will certify their site. Currently, validating a message means authenticating the channel. Under CCFS to validate a message means to authenticate the content and the curator.

CCFS is a system of file hierarchies that is both storage-method and delivery-method agnostic. In the strictest sense, CCFS is an abstraction on top of storage, just as IP is an abstraction on top of the existing packet forwarding system. Just as IP has revolutionized packet-switched networks, CCFS could revolutionize storage and delivery. The value derived from the adoption of IP was due to the ability of developers to build both above and below the IP layer. As with IP, the value of CCFS derives from the ability to build upon the layer, both from above and from below. In CCFS, the upper layer consists of interfaces to the content that is stored in CCFS, such as filesystems or URIs. The lower layer contains technologies that provide for the storage and retrieval of the content objects. This layered approach led to the explosion of innovation that followed IP adoption, and could have a similar effect following the adoption of CCFS.

A. Components of CCFS

The CCFS model requires the use of two cryptographic primitives, four types of content objects, and two lookup services. The construction of these components is described below. The first primitive is a *cryptographic hash function* which converts binary data of any length to a fixed length binary string (Hash) such that the following three conditions hold:

- Given a Hash h it should be difficult to find any message m such that $h = Hash(m)$.
- Given an input $m1$ it should be difficult to find another input $m2$ such that $m1 \neq m2$ and $Hash(m1) = Hash(m2)$.
- It should be difficult to find two different messages $m1$ and $m2$ such that $m1 \neq m2$ and such that $Hash(m1) = Hash(m2)$.

Because of these properties, a resultant hash can be unequivocally used to identify the initial data. Collision avoidance is inherent because the identifier keys are uniformly randomly generated. The probability that any two will be identical for a 256 bit number is infinitesimally small, sometimes approximated at 2^{-128}, a well known calculation referred to as the birthday problem. The second primitive is a *digital signature* used to sign data, for which two conditions must hold:

- The signature requires knowledge of the data and a private key in order to be generated.
- The signature requires knowledge of the data and the corresponding public key to be verified.

In this way, the digital signature is proof that the signed content was signed by the holder of the private key.

TABLE I
CCFS Content Object Types

Type	Purpose	Content Format
Blob	Static; contains data	{ByteArray}
List	Static; Exclusive map from: next name segment to: HID, Type	{ [NameSegment, ObjectHash, ObjectType] }
Commit	Versioned; Exclusive map from:HKID to:List's HCID	{ListHCID, Version Number, ParentHCID, Signature, HKID}
Tag	Versioned; Non-exclusive map from:HKID, next name segment to: HID,Type	{ObjectHash, ObjectType, NameSegment, Version Number, ParentHCID, Signature, HKID}

1) Content Object Types: CCFS utilizes four types of Content Objects, as described in Table I. The "BLOB" object is a data container with *no inherent structure*. The BLOB is the content of interest requested by the user. A BLOB is referred to by the hash of its content, HCID (Hash of Content IDentifier). The "LIST" object is a table used to look up a human readable name to obtain the type and hash (HID) of the content to which it refers. A LIST is a flat text file, each line of which contains a *name (Name Segment), a type (Object Type), and an HID (Object Hash)*. The first field is the next Name Segment in the name path. The second field indicates the Type of content object to which the Name Segment refers. The HID is the cryptographic name of the Content Object. A LIST is therefore a specific type of blob and, like other blobs, can be referred to by its Hash of Content Identifier (HCID). The exclusive use of the BLOB and LIST types, would allow users of CCFS to identify static content only. Any modification of the BLOB or LIST creates a new HCID and requires updating all references to the content. To accommodate identifying collections of dynamic content (content that changes over time) we define two additional Content Objects. A "COMMIT" is used to indicate a specific version of the dynamic content of a Repository, a collection that explicitly versions all items in the collection as a whole.

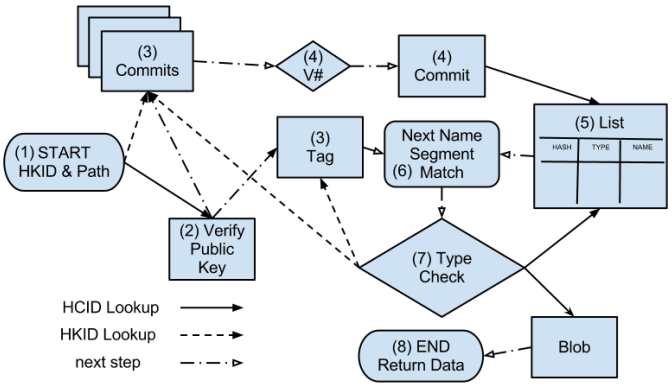

Fig. 1. CCFS Retrieval Flow Chart

A COMMIT is a flat text file, containing *the Hash of a List object* (HCID), a *Version Number*, a *Parent Hash* (HCID), a *Signature*, and the *Hash of the public Key that verifies the signature* (HKID). The first field in the COMMIT is the HCID which always points to a LIST. The next field indicates the version number used to determine the latest version of the dynamic content. The Parent Hash HCID indicates the ancestor content that was used to create a new version of the content. The Parent HCID allows the merger of versions when two or more users are making modifications to the content independently. This represents the exception where a COMMIT is referenced by an HCID, as opposed to an HKID. The Signature is created by signing the previous fields in the COMMIT with the Curator's private key. The last field in the COMMIT contains the Hash of the public key that verifies the signature (HKID). The identifier in this field is the HKID that is used to identify this COMMIT, along with all of its previous and future versions. A TAG is similar to a COMMIT, but is used to indicate a specific version of a single item within a Domain, a collection that versions items in the collection independent of other items in the collection. A COMMIT points to a LIST, whereas a TAG includes the Name Segment, Object Type, and Object Hash directly within the Tag, allowing for items to be version-ed independently. A "TAG" is a flat text file containing the *Hash of a Content Object* (HID), an *Object Type*, a *Name Segment*, a *Version Number*, a *Parent HCID*, a *Signature*, and also the *Hash of the public key that verifies the signature* (HKID).

2) Lookup Services: CCFS requires that the underlying storage system provide two data lookup services. The Hash of Content Identifier (HCID) lookup service uses HCIDs as inputs, and outputs BLOBs, LISTs, or Keys. The Hash of Public-Key Identifier (HKID) lookup service uses HKIDs as inputs, and outputs COMMITs or TAGs. These services can be provided by a combination of underlying systems that are as simple as a classical file system, or as complicated as a global scale information centric network. An example folder structure implementation is as follows:

\blobs\<HCID>
\commits\<HKID>\<Version>
\tags\<HKID>\<Name Segment>\<Version>

B. CCFS Procedures

The main operations performed with CCFS are to retrieve and verify content and to curate content. The steps to perform these operations are in Figure 1 and listed below:

- R1 START: Input HKID and Human Readable Path to Content
- R2 PUBLIC KEY VERIFICATION: Obtain public key for HKID; If key hashes to HKID, then key is valid
- R3 LOOK UP COMMIT/TAG: Retrieve latest version Commit and Tag that contains HKID
- R4 VERSION CHECK: Verify signature on most recent version
- R5 LOOK UP LIST: Extract HCID from the Commit or Tag; Retrieve List for HCID
- R6 NAME SEGMENT MATCH: From List, extract HID and Type associated with next Name Segment
- R7 TYPE CHECK: If Type is Commit, use HKID to find next Commit; Check version
  If Type is Tag, use HKID to find Tag with next Name Segment; Check Version
  If Type is List, use HCID to find List, then if HCID verifies Content Object, go to next Name Segment
  If Type is Blob, use HCID to find Blob, then if HCID verifies Content Object, return content
- R8 END: If verification has succeeded, return data with successful status code

The curation process is an extension of the retrieval process and the associated procedures are listed below:

- C1 Curator selects content and chooses a Human Readable Name, for example, path\to\file.txt
- C2 Follow the Name path as if retrieving the content
- C3 Keep track of Commits and Tags to determine last Commit or Tag (closest to the end of the name), and the Lists that follow the Commit or Tag
- C4 Publish the content as a Blob to storage
- C5 Generate or Update List Object with HCID of the new Blob
- C6 Repeat step 5 until a Commit or Tag precedes the revised list
- C7 Update Commit or Tag with the new List, signing with Private Key associated with the HKID
- C8 Publish the Commit or Tag Object to storage
- C9 Perform a Retrieve operation to obtain Content to verify successful curation

The power of CCFS derives from the dual usage of the file hashes and the public keys, as both a name with which to retrieve the content, as well as, a method of verifying the content.

Content retrieval under CCFS assumes four conditions. First, CCFS software is present on users' devices. Second, users have stored the HKID of a curator that they trust. Third, there exists a communication path from the requesting node to some node containing the content and the content objects leading to it. Fourth, the retriever knows a human readable path from the curator root to the desired content, for example, <HKID>/path/to/file.txt. HCIDs (Hashes of Content Objects) and HKIDs (Hashes of Curator's Public key) will be used to retrieve the Content Object, and also to insure its validity. The HCID checks the validity of a Blob or List. The HKID verifies the public key, which in turn, is used to verify the Signature on Commits and Tags. To retrieve the requested content, the system follows the iterative steps as shown in Figure 1. and simplified in Figure 2.

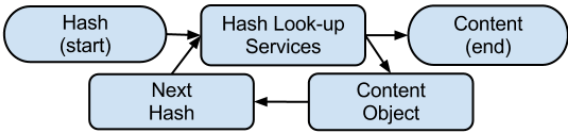

Fig. 2. CCFS Iterative Retrieval Process

IV. Performance Evaluation

In this section, we present CCFS performance evaluation results. We include empirical results of the prototype implementation, a discussion of the enhancements that CCFS adds to previous proposals for ICN, and finally, a comparison of the security advantages of CCFS over X.509 + SSL, the currently dominant on-line architecture for secure content delivery.

A. Proof-of-Concept Implementation

To prove the viability of CCFS, we built a prototype. We developed an implementation of CCFS that used FUSE (File System in Userspace [11]) in order to export CCFS to a Linux operating system as a file system. The cryptographic hash function that we used is SHA256 and the digital signature algorithm is ECDSA over Nist's curve P521. This implementation was benchmarked on a laptop with an Intel Core i7 CPU and 8 GB of RAM running Ubuntu 13.10.

In Figure 3, we show the time to access files stored in the form of each of the following content object types: a blob that is accessed directly ($blob$), a commit that points to a blob ($commit \rightarrow blob$), a tag that points to a blob ($tag \rightarrow blob$), and a list that points to a blob ($list \rightarrow blob$). Each type of retrieve takes approximately the same amount of time. This similarity is evidence that the cryptographic overhead is dwarfed by the other overhead in the system. Therefore, there is not a high price paid for the cryptographic security. The FUSE overhead dwarfs the cryptographic overhead which is significantly different for each content type. For comparison, access time is shown for Ext4, the native file system in Ubuntu kernel. The difference between these two measures is the overhead of CCFS and is approximately one millisecond which is non-negligible, but not prohibitive.

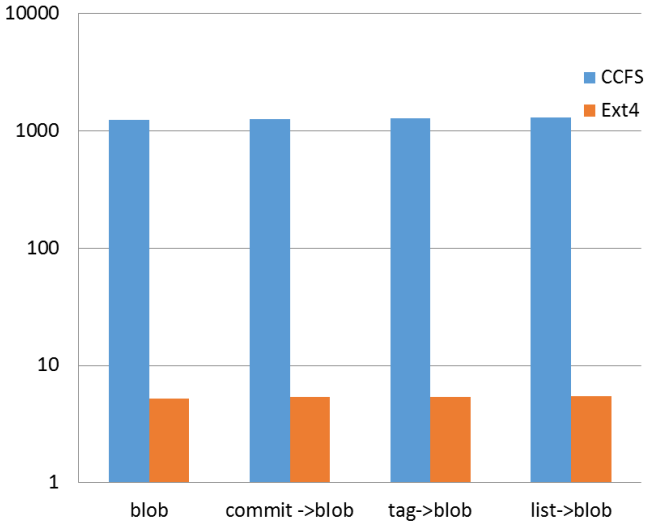

Fig. 3. File read times in microseconds

The second item we measured is the time that the core logic of CCFS takes to perform its main functions of storage, retrieval, and verification under a variety of conditions. These times are shown in Figure 4. The experiment was set up as follows: a web server was run on the local host for serving the Content Objects. In addition, Content Objects were stored in Google Drive and accessed through the Google Drive web API. Of the operations performed in the benchmark, the most common time for the completion of operations was approximately $150ms$. This is a 682 fold increase over the $220\mu s$ use of the local file system. The local-only bar graphs refer to CCFS, running with only the local hard-drive as a content source, whereas, the multi-source bars indicate that Google Drive was also available as a content source. The multi-source outperformed the local-only in all operations. This outcome is consistent with theory: As the number of sources are increased, the multi-source performance (i.e. the performance of CCFS) will also continue to increase. The broader the adoption of CCFS, the better CCFS will perform. Adding more sources will not degrade performance as shown in Equation 1 below:

$$mean(min(\sigma_i, \nu_i)) \leq min(mean(\sigma), mean(\nu)) \quad (1)$$

where σ and ν are vectors of the latencies of requests for content and where $\sigma \cup \nu \in \mathbb{R}$ and $|\sigma| = |\nu|$.

The CCFS measurements are comparable to a sample of measurements of Web performance. These measurements are included here to provide anecdotal evidence of a rough order of magnitude of latency on the Internet today for purposes of comparison. The sample of Web times include the following: $217ms$ average ping time to www.com.au in Australia, $48.667ms$ average ping time to googleapis.com, and $359ms$ average wget time to googleapis.com. The code for these observations is publicly available at github.com/AaronGoldman/ccfs [12].

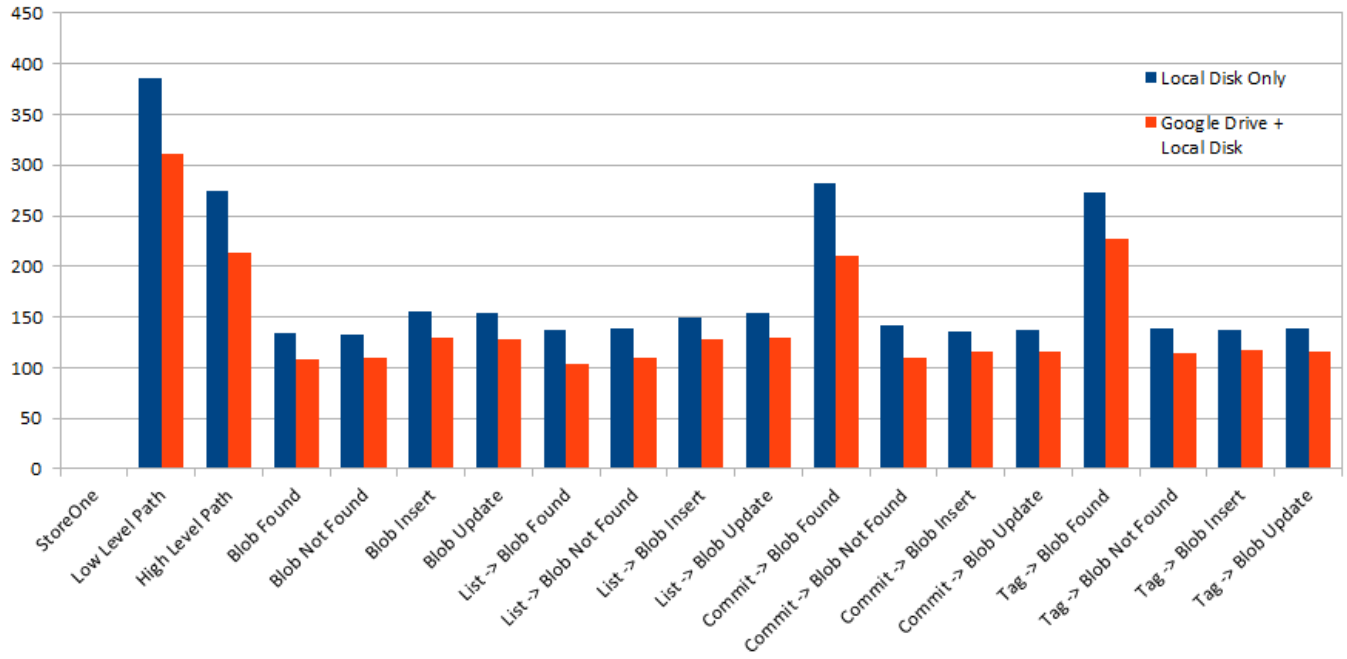

Fig. 4. Time per operation in Milliseconds

B. Enhancements to Information-Centric Networking

CCFS possesses unique attributes that can leverage the advantages of previously proposed ICN models.

1) Delegatable Namespace: CCFS introduces a unique distributed naming model in which each curator has control over her own namespace, independent of a central namespace arbiter. A curator can delegate a segment of that namespace to any other curator, who in turn can delegate a segment of his namespace to yet another curator. This creates a nesting doll type of embedding that allows the consumer who trusts one curator to trust a series of subsequent curators, each included within an earlier curator's namespace. The CCFS software would be distributed with a default list of HKIDs of trusted curators. The users would add those they choose to their own curated space. They would also add lists of curators they know and trust through personal connections. Through CCFS, curators that are trusted would then point to additional curators that once discovered could be added to their own collection. Once a users collection is populated, he only needs to remember his own HKID to use the system.

2) Independent Control: Other proposed content-centric models [1] rely on a central authority, on global adoption of an explicit name structure, or some other variety of name resolution arbitrator. This reliance on a third party is subject to external control with the potential of undesired filtering or interference with the free flow of information. CCFS does not depend on any central arbiter because each curator has control over her own namespace and each name is relative to the curator. There is no global namespace to be arbitrated, therefore CCFS offers arbitration-free content distribution.

3) Durable Content: CCFS creates content permanence by separating the retrieval of content from the ability to reach the original curator, eliminating the problem of link rot. CCFS certifies that the content returned is exactly identical to the content when it was named (added to the collection) by the curator. Since cryptographic names are tied forever to the content, the trust model is inherent in the name of the content, *itself*, and does not rely upon contacting the curator or any other trusted authority. The content persists long after the curator may no longer exist. In addition to this content object permanence, CCFS is disruption tolerant. Content remains available from any host that has the content even when central authorities are disrupted by natural, technical, or malicious disasters.

4) Inter-operable: As previously stated, CCFS is an overlay network layer that can be built upon from above or below. It will operate with any ICN and any Content Delivery Network (CDN). CCFS provides a system of file hierarchies that is both storage-method and delivery-method agnostic, making it inter-operable with legacy technologies. The familiar file system interface adds to the ease of usability by all consumers.

5) Implementable: CCFS is a convenient first step to take towards the introduction of ICN. The building blocks of the architecture and elements of technology are simple and have already been tested and applied in other contexts. Moreover, CCFS does not require a global adoption nor a major clean slate restructuring of the Internet. A local community such as a university, an organization, or a geographic neighborhood can implement CCFS and have a functional ICN with little investment of resources. CCFS can be easily implemented on a local community basis and then as communities interact with each other, CCFS will be applied on a more global basis. This in essence is the roadmap that can lead to widespread adoption of CCFS.

6) Secure: Through the use of well-established cryptographic principles to create hashes of content and signatures, CCFS protects privacy and provides a method of verifying the identity of both curators and content. For a discussion of CCFS and its role in establishing the fundamental attributes of security, see the discussion comparing CCFS to traditional Internet technology.

C. Security Attributes of CCFS

CCFS offers positive contributions in all areas of information security: availability, integrity, authenticity, non-repudiation, and confidentiality. Each of these is included in Table II and discussed below, and comparisons are drawn with the currently dominant X.509 and SSL systems.

1) Availability and Scalability: With CCFS, multiple content sources offer improvements in availability and scalability. Currently, content is retrieved from a single authoritative source, for example, a Web server with an X.509 certificate. CCFS retrieves content from any source where it resides and it derives the authoritative certification from the chain of objects in its name. Anyone can store and forward content without loss of authenticity. Caching content closer to the requester will reduce the load for the delivery of popular content. Many techniques that improve availability and scalability require the content publisher to coordinate with the content delivery network (CDN), often at great cost. Under CCFS, these services can be provided by anyone who has the means and motivation to cache and re-host the content. Rational actors choosing to host content may include subscribers wanting to support a publisher, ISPs trying to redu ce their peering costs, members of a peer-to-peer community sharing the costs of hosting, Internet archives such as archive.org, or for-profit content hosting services. All stakeholders contribute to availability and scalability.

The availability of content from any location where it resides mitigates the chance of the content becoming irretrievable upon disruption to any one source location. The content continues to exist and can be accessed without reliance upon either a centralized authority or the original host. Disruption to these entities will not necessarily hinder the accessibility of the content. Content can also be retrieved if either the authoritative host or the content consumer is mobile. With CCFS, as content consumers move across geographic or domain boundaries, they can continue to access content from the nearest source. The content host can be mobile without eliminating access to the content. CCFS will not only scale to accommodate proliferation of base stations and mobile hosts, but will actually improve performance with densification due to cooperative caching.

2) Integrity, Authenticity, Non-repudiation: Integrity, authenticity, and non-repudiation, under the current paradigm, are handled by a chain of custody. The channel, and not the content, is authenticated under the currently dominant X.509 + SSL system. The creator provides the content to the content host through an SSL channel and the host provides the content to the CDN through a subsequent SSL channel. The subscriber retrieves the content from the CDN using a third SSL channel. If the CDN and the host can not be reached, no alternative host can deliver the content because the chain of custody can not be verified. Currently, caching is subject to time-to-live issues, as well as, to accidental or intentional cache poisoning.

In stark contrast, the CCFS model allows for content to be identified by a Name that serves as the chain of trust. The CCFS Content Objects are certified by the content, itself, as opposed to the delivery channels. Under CCFS, as long as a single hash (HID) can be stored securely, this seed of trust can be used to verify the content being delivered. Once verified, the delivered content can be used to retrieve and verify subsequent content. The integrity under CCFS is assured by the content hashes (HCIDs). The authenticity and non-repudiation are provided by Signatures that associate content with curators.

3) Confidentiality, Privacy: With CCFS, requests for content will reach a greater audience as a broader network is searched. This increases the probability of successfully retrieving content in spite of network disruptions, but results in less privacy. In practice, however, CCFS increases the inconvenience of tracking who is accessing content. This provides a level of increased privacy to the retriever of the content data. In exchange for performance, trade-offs can be made that will diminish the loss of privacy. Two techniques for this purpose are onion-routing and compartmentalization [13]. Onion-routing helps disguise the identity of the requester of content by using multiple layers of encryption and will operate easily with a CCFS network. Compartmentalization, which allows the requester to break the request into multiple segments which improves privacy, can be used with CCFS, but not with current content delivery networks.

V. Conclusion and Future Work

The Cryptographically-Curated File System, which we propose in this paper, offers a secure, inter-operable, and easily implementable first step toward Information Centric Networking. The benefits of CCFS include improvements in availability, authenticity, integrity, non-repudiation, and scalability. The advantages of CCFS, extend not only to content consumers, but also to content producers. Content consumers will seamlessly access content from the closest source, even as the closest source continues to vary. CCFS allows content producers to share the burden of hosting and distributing content with others who also host the content. It allows the producer to leverage others content distribution infrastructure to distribute content. CCFS performance will continue to improve with the proliferation of mobile hosts, base stations, content hosts, and software optimizations. The CCFS user will experience greater performance, and with greater privacy, than is now experienced under current networks.

In this work, we presented the conceptual groundwork and fundamental elements of CCFS. We also implemented a prototype of the CCFS design on real systems using Google Drive. Results show the feasibility of CCFS. We evaluated its performance with respect to the timing of file operations

TABLE II
EVALUATION OF CCFS

	LEGACY X.509 + SSL	PROPOSED CCFS
Confidentiality (privacy)	Interactive communications Key exchange and on public key cryptography(PKC)	store/forward communication Public key cryptography(PKC)
Integrity	Cyclical Redundancy Check (CRC) Error resistant but easily attacked	Hash is tamper proof Integrity built into system
Availability (reliability)	URL & Cache (identical URL) Cache content expires Synonyms - no cache hit Local caching	Cache (by Content Hash) Content never expires No synonyms, No homonyms, one to one Hash Distributed caching
Authenticity	Authoritative source trusted Chain of custody required Server authenticated	Content self authenticating Custody is irrelevant Content authenticated
Non-Repudiation	Uses public key cryptography (PKC) and symmetric key cryptography Non-repudiation is not imposed	Uses public key cryptography (PKC) and cryptographic hashing Imposes non-repudiation
Scalability	Scales at great cost Large scale CDNs High cost bandwidth High cost for popular content Fragile for rare content Limited by central authority	Scales with popularity Distributed resources Proximity related queries Maximizes local retrieval Low cost Unlimited distributed authority

and network operations. In both cases the results revealed that the overhead associated with CCFS was not negligible, but also was not prohibitive. In future work, we will expand evaluation of efficiency and scalability, for both local and global networks, as well as focus on the development of self-assembling networks to use in conjunction with CCFS. CCFS can unify currently disparate storage technologies into one universal storage pool. CCFS is a welcome direction for the developing Internet, leveraging the benefits of ICN, changing the query for information from "what host to reach" to the simplified, "what data to fetch" [2].

ACKNOWLEDGMENTS

We thank Priya Bajaj, Mallika Sen, Holly Parrish, and Thomas Saekao of the Opportunity Research Scholars Program within the School of Electrical and Computer Engineering at Georgia Institute of Technology for their contributions to the CCFS code base [12].

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] Tahoe-LAFS project, "tahoe-lafs." https://tahoe-lafs.org, March 2013.

[9] Brad Fitzpatrick, "Camlistore content-addressable multi-layer indexed storage." http://camlistore.org/, March 2014.

[10] Marc Stiegler, "An introduction to petname systems." http://www.skyhunter.com/marcs/petnames/IntroPetNames.html, June 2010.

[11] FUSE Project, "Filesystem in userspace." http://fuse.sourceforge.net/, March 2014.

[12] Aaron Goldman, "Cryptographically curated file system." https://github.com/AaronGoldman/ccfs, March 2014.

[13] The Tor Project, Inc., "The solution: a distributed, anonymous network." https://www.torproject.org/about/overview.html.en#thesolution, March 2013.