NDN Technical Report NDN-0012, March 2013

FileSync/NDN: Peer-to-Peer File Sync over Named Data Networking

Jared Lindblöm[1], Ming-Chun Huang[1], Jeff Burke[2] and Lixia Zhang[1]
1. Computer Science Department, University of California, Los Angeles
2. REMAP, University of California, Los Angeles

***Abstract*—FileSync/NDN is a distributed application that implements file synchronization using Named Data Networking. Its purpose is to provide a service similar to the commercial DropBox platform, with improvements. Unlike DropBox, FileSync/NDN adopts a peer-to-peer (P2P) model that does not require centralization of data, and it employs NDN to reduce the traffic that would be present in a TCP/IP-based P2P approach. FileSync/NDN takes advantage of several features of NDN to improve the distribution and security of the content it provides. It employs the CCNx Synchronization Protocol (CCNx-SYNC) to maintain consistency of its content across multiple nodes on the network. In this technical report, the design considerations of NDN and SYNC are discussed, and a novel application design for distributed file synchronization is introduced.**

I. Introduction

File synchronization is a useful network application that provides data replication over a variety of hosts to support sharing, backup, and remote access. The commercial service DropBox uses a client-server (C-S) model to implement such synchronization. It is simple and popular, but has drawbacks including high traffic concentration at the server and concentrated points of failure. To enforce its security model, clients on the same local area network are required to coordinate with a remote central server before they may exchange information locally to achieve consistency. In addition to these drawbacks, client-server models also may offer easy attack targets because the addresses of the servers are well-known.

Peer-to-peer (P2P) alternatives use a distributed file sharing model, which has different drawbacks, such as a large number of duplicated packets when using the current Internet Protocol (IP), which is due to the lack of efficient control in data distribution, and limited resilience to failure when there are a small number of participants. Also, since data can be disseminated by every peer, the integrity and relevance of the data are difficult to assure.

File synchronization applications reconcile all distributed copies of a given file, which is usually identified to the user by its path and filename. Typically, applications attempt to ensure every participating node has the latest version. In the C-S model, all clients must be informed of an update to a file on any other client either through a callback from, or periodic polling of, a server. When there is an update to a file on one client, the server must reflect this quickly so that all other clients are able to access the latest copy from the server. At the time of replication, high traffic at the server results, because each client must ask for a copy of the file from the server and cannot request it from another client. In a P2P file sharing system, there isn't a central server to mandate the most up-to-date copy of a file. Therefore, propagation of synchronization updates must be well-designed, otherwise synchronized files may develop inconsistencies across hosts.

This report explores how Named Data Networking (NDN) can be used to support file synchronization without centralization, while also mitigating the significant drawbacks of P2P strategies used on the current Internet. This application seeks to harness NDN's efficient multicast dissemination, intrinsic content signatures and straightforward incorporation of storage at every node. FileSync/NDN uses the CCNx Synchronization Protocol to maintain consistency of files being synchronized and relies on NDN's intrinsic content caching and multicast to reduce traffic when compared to IP-based P2P approaches.

II. Background

A. Named Data Networking

Named Data Networking (NDN) [10], [11] is a data consumer-driven network architecture in which data flows through the network when it is requested by a consumer. In NDN, named content is identified with, and requested by, its unique name; additionally, every segment of content is signed by the producer, binding it to its name. To retrieve such a *content object* from the network, a consumer specifies the name of the content object in an *interest* packet which it issues to the network. Each upstream node in the network attempts to satisfy the interest by the providing the corresponding content, if available, or else forwards the interest on to other nodes advertising an appropriate prefix.

When an interest is satisfied, the corresponding data follows the path of the interest back to the requester. Each router along the path traveled by an interest receives the interest's corresponding data packet, and it checks its *forwarding information base* (FIB) to determine where the content should be forwarded. At each node, pending interests are consumed, and the data packet is forwarded via all registered interfaces to requesting consumers. If no data has arrived to fulfill an interest, the interest is kept in the router's *pending interest table* (PIT) until it times out. Content objects are cached in the router's memory for future use, enabling a router to satisfy repeated interests for the same content directly from its cache.

B. CCNx-SYNC protocol in NDN

In addition to the *content store* of each node, which caches the content objects that pass through the node *repositories* are used within the network to provide persistent storage of published content objects. In the CCNx reference implementation, the Synchronization Protocol [3] provides a mechanism to efficiently synchronize namespaces and their corresponding content across repositories in an NDN network. CCNx-SYNC synchronizes named data saved to repositories under a given name prefix, enabling applications to synchronize content by simply saving it to their repository under that prefix. The list below outlines important concepts and terminology used by CCNx-SYNC.

1) A *collection* is a set of content to be synchronized, as identified by its namespace, the *content prefix*.
2) The *topological prefix* defines the namespace used for the broadcast messages needed to synchronize a given collection.
3) The definition of a collection is called a *slice* and includes (1) the collection prefix, (2) a set of filters that limit the content to be synchronized under that prefix, and (3) the topological prefix under which broadcast messages used for synchronization can be issued for the collection. In order to synchronize content objects among peers, each peer's repository must define the same slice.
4) After a collection has been defined in a repository, *sync agents* associated with that repository build a *sync tree* based on the collection's content. The combined hash for each node in the tree is the sum of the hashes of individual content names in that node and of the combined hashes of all its child nodes. The root hash is compared by sync agents across repositories to determine if the content objects represented in the collection are consistent.
5) *Root advise interests* are periodically broadcast to sync peers using the topological prefix. By default, the protocol limits such peers to those reachable in two hops or less. Each root advise interest contains the peer's current hash digest of the local sync tree and is used to check the consistency of sync trees for a given collection across repositories within the network.

CCNx-SYNC's process can be summarized follows: First, an appropriate slice with valid topological and content prefixes is established for a group of repositories to be synchronized. Then the local sync agent for each repository generates its own sync tree based on the stored collection of content objects. Periodically it sends out a root advise interest containing the tree's hash digest, which serves to inquire about updates to the collection it is synchronizing. All peers that receive a root advise interest corresponding to a slice they share compare the digest in the interest with their own. If they are inconsistent, the trees are reconciled using the interest/data exchange in the topological prefix described in [3].

C. Routing considerations

Before proceeding to describe the architecture of FileSync/NDN, it is also important to note some routing-related considerations resulting from the use of CCNx-SYNC.

Each peer that participates in synchronizing a repository must be able to publish content and receive interests in the topological prefix, where root advise interests and responses are shared, and the content prefix, where the actual data is published. This requires an approach somewhere along the range of two extremes: (a) providing routing table entries for both prefixes per collection (e.g., FileSync/NDN shared directory) *per peer*, or, (b) using a common "broadcast" namespace for all synchronized repositories. For simplicity, we take the latter approach in our testbed implementation, and assume a single prefix routed with broadcast semantics, e.g., `/ndn/broadcast`. The scaling limitations of both approaches must be addressed in future work.

CCNx-SYNC, by default, limits interests to "scope 2", which enables only directly-connected peers to exchange content using this mechanism. While this default can be changed,[1] the preferred method is to use the standard implementation of CCNx to promote easy adoption of the application. To enable use on the NDN testbed, where most users' nodes are not directly connected to each other, requires running the Filesync/NDN application on each hub, which is connected via scope 2. These hubs are configured to synchronize content with the nodes that use them for connectivity. These nodes are typically end users' hosts and are also connected with scope 2.

III. Architecture and System Design

CCNx-SYNC on its own does not support file synchronization as performed by popular applications like DropBox because (1) it does not yet provide an API or convention for the removal or modification of content that has previously been added to a collection; (2) it is embedded inside the CCNx C-based repository; and (3) it cannot be used directly to synchronize the local filesystem. FileSync/NDN is designed to work in conjunction with CCNx-SYNC to enable the necessary operations - add, delete, move, modify and rename - to be performed directly on the local filesystem and to allow synchronization to occur automatically in the background without user intervention.

FileSync/NDN is designed to synchronize files between multiple hosts by maintaining a consistent view of the common shared directory. Files within the designated local directory on each host are mapped to a collection of versioned content objects within a repository and synchronized between machines using CCNx-SYNC. Because the synchronization protocol only supports the addition of content into a collection, FileSync/NDN extends the functionality of CCNx-SYNC to maintain consistency between the collection in the repository and the local files the collection represents. The following sections describe the application's design.

[1] This can be changed by setting the `CCNS_SYNC_SCOPE` environment variable and modifying the code itself to accept larger scope values.

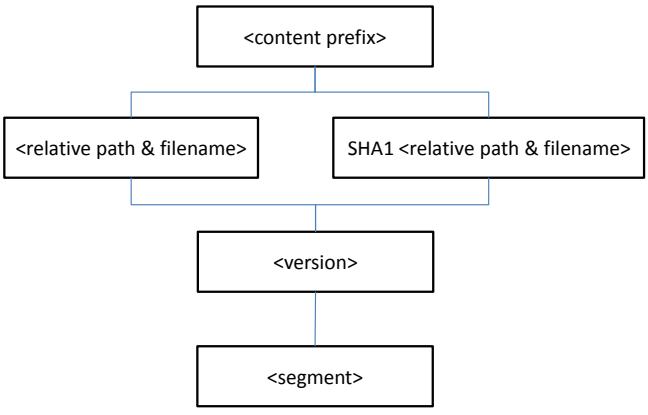
A. Namespace Design

As previously mentioned, in CCNx-SYNC a slice defines a collection as a combination of a topological prefix, a content prefix, and filters.[2] To configure FileSync/NDN to synchronize a directory with other peers, a user provides (1) a local directory to hold the files to be shared; (2) the topological and content prefixes associated with the desired shared directory, thus defining the slice for CCNx-SYNC. In the current design, the prefixes are the same for all participating peers. Other users wishing to share this folder must similarly provide their local instance of FileSync/NDN with the identical topological and content prefixes so that the same slice can be saved to each of their local repositories. Figure 1 depicts an example slice definition.[3]

	Topological Prefix	Content Prefix
Slice	/ndn/broadcast/apps/filesync	/ndn/broadcast/apps/filesync/uclaClass217

Fig. 1: Example Slice in FileSync/NDN

Every file in the shared directory is written by FileSync/NDN to a local CCNx repository in the form of segmented, versioned content objects adhering to the repository naming conventions. The file data is stored under the content prefix, using the file's filename and relative path as its name, in UTF-8 text, followed by a version which corresponds to the timestamp of when the object was created, followed by segment numbers, if needed. Additionally, the file is also made available in a flat namespace under the content prefix, where it is named by the SHA-1 hash of its name and relative path, followed by version and segment numbering.


<content prefix>
<relative path & filename>
SHA1 <relative path & filename>
<version>

/ndn/broadcast/apps/filesync/uclaClass217/Document.txt/%FD%05%0B0%CFf%21/00

Fig. 2: Shared File Namespace

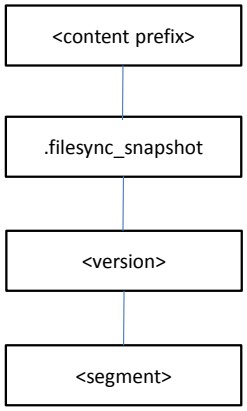
[2]Filters are not used in the current implementation of FileSync/NDN.

[3]These prefixes could be different, but both require all peers to be able to publish content and receive interests for both the topological and the content prefix. As discussed above, the example uses the `/ndn/broadcast` namespace for both, because that will be the most straightforward deployment on the NDN testbed.

The naming convention is shown in Figure 2, along with an example for the first segment of the shared file `Document.txt` in the root of the shared directory published under `/ndn/broadcast/apps/filesync/uclaClass217`.

B. Shared Directory Reconciliation

Each host informs all other hosts of local changes to the synchronized collection by generating a *snapshot* representing the complete state of the local shared directory, as well as a record of deletions, and adding it to the collection using a unique name. This snapshot is a plaintext, CSV-format file that lists both the files that exist in the shared directory and those that have been deleted from it. Its content is synchronized in the same manner as the other files in the shared directory by the underlying CCNx-SYNC mechanism. Figure 3 shows the naming convention for the snapshot which is also stored using standard CCNx repository conventions. In FileSync/NDN, there is one "latest" snapshot for every collection that is reconciled across all participating peers.


<content prefix>
.filesync_snapshot
<version>

/ndn/broadcast/apps/filesync/uclaClass217/.filesync_snapshot/%FD%05%0B0%F5%B6f/00

Fig. 3: Snapshot Naming

Figure 4 depicts the information which a snapshot holds on each file: relative path, existence, content name and digest. Relative path specifies the file's name and path relative to the local shared directory root.

Relative Path	Existence	Content Name	Digest
Document.txt	True	/ndn/broadcast/apps/filesync/uclaClass217/Document.txt/%FD%05%0B0%CFf%21	2205e48de5f93c784733ffcca841d2b5
Reports/Report.pdf	True	/ndn/broadcast/apps/filesync/uclaClass217/Reports/Report.pdf/%FD%05%0B0%F5%B6f	d41d8cd98f00b204e9800998ecf8427e

Fig. 4: File Metadata in the Snapshot

The existence entry indicates whether that file should exist in the shared directory and it is used by FileSync/NDN to overcome the add-only nature of CCNx-SYNC. [4] The content

[4]This approach is in contrast to the approach proposed by [12], [13], whereby every content object that should no longer exist is overwritten with an empty content object. Although both approaches overcome this limitation, the approach proposed by [12], [13] produces, in the worst case, twice as many content objects in a collection than the approach used by this application.

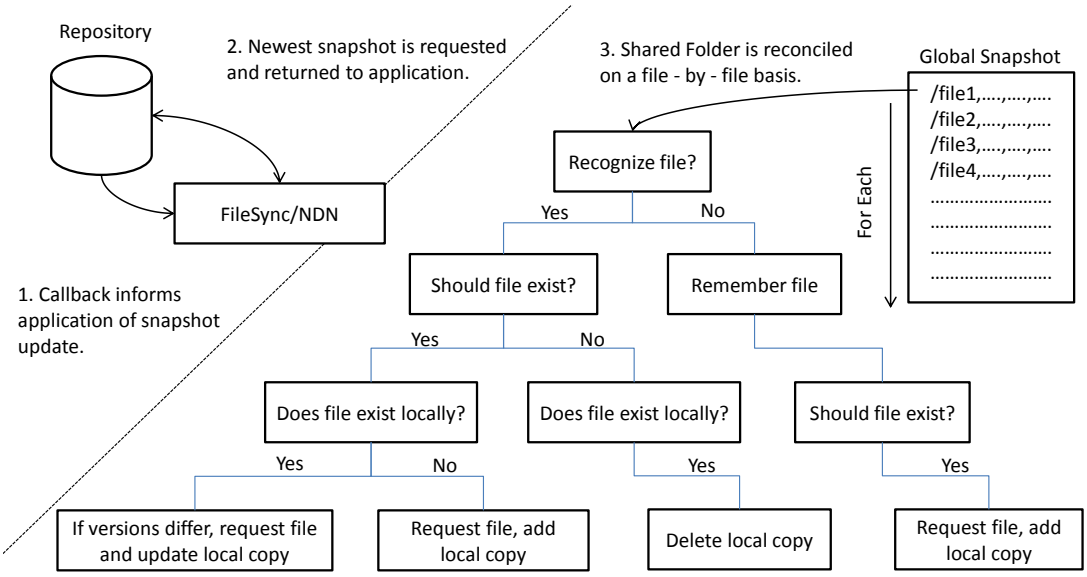


Fig. 5: Shared Directory Reconciliation

name is included as an absolute path, including the content prefix, and follows the naming convention described above. The digest of a file is used by FileSync/NDN to verify file integrity before it is placed in the shared directory, and it is calculated by taking the MD5 summation of its contents.

Hosts reconcile the differences between a local hashtable, which tracks file additions, deletions, and modifications in the local directory, and the state of the shared directory as specified in the synchronized snapshot file. Each host reconciles the differences between the local shared directory and the state from the latest snapshot using the algorithm shown in Figure 5, which is applied whenever a snapshot is received that is newer, i.e., has a higher version number, than the previous local copy. In the worst case scenario, the algorithm compares n number of files in the snapshot to n number of local file copies, which occurs when every file entry is compared with a local copy in the local shared directory.

An event-based approach is used to detect and respond to changes in the local file system and corresponding repository. A file system watcher generates an event (callback) when changes to the local shared directory occur. The local hashtable is updated and a new snapshot is written to the repository reflecting those changes. Asynchronously, the sync tree is updated by CCNx-SYNC, and the synchronization process occurs as discussed in the previous section. Similarly, when a new snapshot file coming from another peer is synchronized with the collection, a callback is received from CCNx-SYNC. The snapshot is compared with the local directory, and, if necessary, interests are issued to get new content objects from the repository, which are then written to the local file system.

IV. Implementation

In this section, the approach to implementation of the application is briefly outlined.

A. Cross Platform Support

FileSync/NDN was developed in Java so that it is platform independent. This application will run on any platform supported by the CCNx library. It has been tested in the Linux and Mac environments.

B. File System Monitoring

Jnotify[4] is used for file system monitoring. It is a Java library that enables Java applications to listen to file system events such as creation, modification, renaming, and deletion. Jnotify informs the application of file system events by invoking a registered callback function. Jnotify was incorporated into FileSync/NDN to monitor changes made to the local shared directory.

C. Multi-threading

FileSync/NDN is a multi-threaded application. File system events, content publication and content retrieval are all handled by threads to improve the efficiency of the application. The current application dispatches threads to a thread pool that is allocated to run up to twenty threads simultaneously.

D. Batch-processing

To improve shared directory reconciliation efficiency, snapshots are published no faster than every two seconds and reflect all changes that have been made to the local shared directory over that period.

V. Preliminary Evaluation

A. Speed of Reconciliation

An experiment was conducted to determine the relationship between the time it takes the application to reconcile a shared directory and the number of file updates contained in a snapshot.

Two virtual machines running Ubuntu 12.04 were used. The Eclipse application profiling tool, TPTP [5], was employed to measure the time it took FileSync/NDN to reconcile a local directory after first receiving a new snapshot. The results from this experiment suggest that the shared directory reconciliation algorithm that was developed for this application performs linearly. Figure 6 shows the relationship between the number of files in a snapshot and the time it takes the application to process that snapshot.

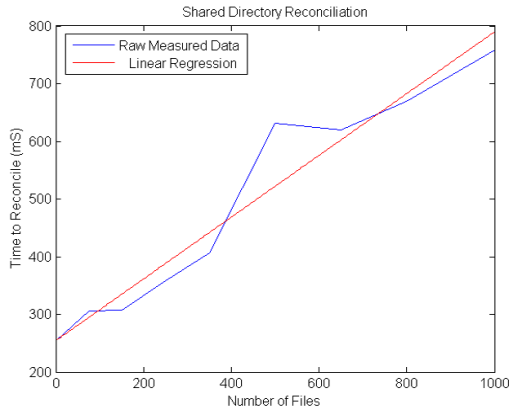


Fig. 6: Reconciliation Experiment Result

B. Speed of Propagation

A second experiment was conducted to determine the relationship between the number of peers sharing a synchronized collection and the time it takes a snapshot update to be propagated to all peers.

Six virtual machines running Ubuntu 12.04 were used. Each VM was time-synchronized using the network time protocol (NTP) over IP. The application was tested using groups of two, three, four, five and six peers. From these experiments, it was observed that, on average, it took the application 0.35 seconds to propagate the global snapshot to all peers, with a standard deviation of 97 milliseconds. This measurement was taken from the time just before a peer wrote a new global snapshot to the time each peer received word of the update. There was no observed correlation between the number of peers sharing the synchronized collection and the average time it took each peer to receive the snapshot update.

VI. Discussion and Future Work

A. Use of CCNx-SYNC

CCNx-SYNC provides the ability to effectively synchronize content across hosts over NDN. However, it has limitations for supporting a "DropBox-style" file synchronization application like this one. The sections above describe how FileSync/NDN builds on the current CCNx approach to support the file operations necessary for a distributed file sharing service. Other design decisions in CCNx-SYNC require tradeoffs for this application: (1) As discussed, CCNx-SYNC limits the scope of the root advise interest to two hops or less, limiting the application's reach to neighboring repositories. (2) It requires that each participating repository agree on the definition of a synchronized collection, i.e. uses the same content prefix, in order to synchronize the content objects associated with that collection. (3) The order in which names and content are synchronized is arbitrary, which requires the application to track the order of file system events itself, if it chooses to perform reconciliation. [2].

B. Security

Future versions of FileSync/NDN will provide users with the ability to control access to the files they are sharing. Users will have the ability to decide who can read from, and/or write to, a collection that will be synchronized across machines. We aim to achieve this by leveraging the group-based access controls available in the CCNx Java API and obfuscating the content names of shared files.

1) Group-Based Access Controls: In the CCNx Java API, an AccessControlManager can be used to enforce an access control list on a particular namespace in a repository. We plan to adapt this for subsequent versions of FileSync/NDN. The AccessControlManager implements read, write, or manage permissions for named users or groups. The permissions of each user are tied to the user's published public key. Permissions are supersets of one another, allowing writers to read, and managers to read and write. Managers have the additional capability to create and edit the access control lists [1]. The repository enforces the access policy by requiring that the AccessControlManager of the namespace in question be consulted before any content is read from, or written to, the repository. Content stored under the namespace is encrypted; if access is granted, the AccessControlManager provides the appropriate content encryption/decryption keys to encrypt/decrypt the content. An access control policy will be created by allowing the user to define the managers, writers and readers of a namespace. A manager of a namespace would have the option to add or revoke permissions for any user at any time.

2) Preventing File Name Leakage: Even though access to the shared files in a namespace can be controlled by an access policy, an outsider could glean information about what files are being shared by listening to the interests and data packets flowing between repositories. To prevent snooping, users will be given the option to obfuscate the content names of each file in the shared directory. This will be accomplished by replacing the relative path of each file with its secure hash and mapping that name to the file's relative path in the snapshot. An initial version of this feature has already been implemented, as mentioned above.

C. Version Control and Conflict Resolution

FileSync/NDN handles conflicts between snapshots of a common shared directory by employing a last-writer-wins policy. When a snapshot is created, it is versioned using a network timestamp. If two snapshots with conflicting views are created and added to the collection, the snapshot with the

latest timestamp is considered, and the other one is ignored. In CCNx, network timestamps are accurate to 1/4096 second, making it very unlikely that two snapshots could be created with identical timestamps. By employing this form of conflict resolution, changes proposed by the ignored snapshot are discarded and never reflected in the state of the shared directory. Work is presently being performed to incorporate more desirable forms of conflict resolution into this application, which could provide version control features.

VII. Related Work

H. Choi and J. Kang proposed a home-networking file storage system that runs over NDN [6]. They designed and implemented an application that provided a file sharing service using the client-server model. They determined that their design alleviated traffic concentration at the server by taking advantage of NDN's in-network content caching, but did not prevent single-point failure problems inherent in a client-server design. Z. Qu and J. Burke proposed a framework to support a car racing game, Egal Car, over NDN [13]. They used the CCNx Synchronization Protocol to synchronize gaming messages between players. V. Jacobson, et. al proposed the concept of custodian-based information sharing [12]. To support the removal of content from a synchronized collection, they proposed adding a content object to the collection to flag certain content as having been removed. Chronos, proposed by Zhu, et. al, is a multi-user chat application that runs over NDN [14]. The authors of this application bridged existing XMPP protocol-enabled, client-end software with a synchronization protocol that they designed to provide scalable data dissemination of chat messages over NDN. In their design, special namespace rules were defined to guarantee reliable delivery of chat messages in chronological order to each client. In Securing Network Content by V. Jacobson, et. al, general security considerations and designs in NDN are discussed [8] and two implementations of a security-centric design are proposed in D. Kulinski and J. Burke's NDN Video technical report [7] and Securing Instrumented Environments by J. Burke, et. al[9].

VIII. Conclusion

FileSync/NDN is a distributed file sharing application implemented using the CCNx reference implementation. An initial evaluation suggests synchronization for modest numbers of files to be linear in time. The application's design takes advantage of several features of NDN to improve the distribution and consistency of the content it provides. FileSync/NDN extends CCNx-SYNC to support the file operations necessary for a distributed file sharing service that is transparent to the user. This approach holds promise for redefining the way applications share information across multiple hosts on a network. Current limitations include routing requirements dictating that each peer be able to publish content in the same prefix. Future work is underway to improve the application's synchronization efficiency, routing requirements, robustness and security with the goal that it will be a popular application within the NDN community.

References

[1] Access control manager class reference. https://www.ccnx.org/releases/latest/doc/javacode/html/classorg_1_1ccnx_1_1ccnx_1_1profiles_1_1security_1_1access_1_1_access_control_manager.html.

[2] Ccnsyncslice manual page. https://www.ccnx.org/releases/ccnx-0.7.0rc1/doc/manpages/ccnsyncslice.1.html.

[3] Ccnx synchronization protocol. http://www.ccnx.org/releases/latest/doc/technical/SynchronizationProtocol.html.

[4] Jnotify-file system event library for java. http://jnotify.sourceforge.net/.

[5] Test and performance tools platform project. http://www.eclipse.org/projects/project.php?id=tptp.

[6] H. Choi and J. Kang. NDN-based Smart Digital Home Storage. UCLA CS217 Class Project, 2011.

[7] D. Kulinski and J. Burke. Ndnvideo: Random-access live and pre-recorded streaming using ndn. Technical report, University of California, Los Angeles, REMAP, 2012.

[8] D. Smetters and V. Jacobson. Securing network content. Technical report, PARC, 2009.

[9] J. Burke and P. Gasti and N. Nathan and G. Tsudik. Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control. In *CoRR*, 2012.

[10] L. Zhang and D. Estrin and J. Burke. Named data networking (ndn) project. Technical report, PARC, 2010.

[11] V. Jacobson and D. Smetters and J. Thornton and M. Plass and N. Briggs. Networking Named Content. In *CoNEXT '09*, pages 1–12. ACM, 2009.

[12] V. Jacobson and R. Braynard and T. Diebert and P. Mahadevan and M. Mosko and N. Briggs and S. Barber and M. Plass and I. Solis and E. Uzun and B. Lee and M. Jang and D. Byun and D. Smetters and J. Thornton. Custodian-based information sharing. *Comunications Magazine*, 50(7):38–43, 2012.

[13] Z. Qu and J. Burke. Egal car: A peer-to-peer car racing game synchronized over named data networking. Technical report, University of California, Los Angeles, REMAP, 2012.

[14] Z. Zhu and C. Bian and A. Afanasyev and V. Jacobson and L. Zhang. Chronos: Severless multi-user chat over ndn. Technical report, PARC, 2012.