The Story of ChronoShare, or How NDN Brought Distributed Secure File Sharing Back

Alexander Afanasyev, Zhenkai Zhu, Yingdi Yu, Lijing Wang[†], and Lixia Zhang

University of California, Los Angeles
{afanasev, zhenkai, yingdi, lixia}@cs.ucla.edu

[†]Tsinghua University, China
wanglj11@mails.tsinghua.edu.cn

***Abstract*—Information sharing among a group of friends or colleagues in real life is usually a distributed process: we tell each other interesting or important news without any mandatory assistance or approval from a third party. Surprisingly, this is not what happens when sharing files among a group of friends over the Internet. While the goal of file sharing is to disseminate files among multiple parties, due to the constraints imposed by IP's point-to-point communication model, most of today's file sharing applications, such as Dropbox, Google Drive, etc., resort to a centralized design paradigm: a user first uploads files to the server (cloud), and the server (cloud) re-distributes these files to other users, resulting in unnecessary tussles and inefficient data distribution paths. To bring the truly distributed file sharing back into the cyberspace, this paper presents ChronoShare, a distributed file sharing application built on top of the Named Data Networking (NDN) architecture. By walking through ChronoShare design details, we show how file sharing, as well as many other similar applications, can be effectively implemented over NDN in a truly distributed and secure manner.**

I. Prologue

In the real world, information sharing among a group of friends or colleagues is usually done in a distributed fashion: we discover something interesting and tell it to our friends; our friends tell their friends, and so on. However, when a group of friends share files using today's popular applications, such as DropBox, Box, Google Drive, etc., the sharing process does not follow this natural information distribution pattern. Instead, people have to first upload the files to the server (cloud), which then re-distributes the uploaded files to the designated recipients. In the real world, this would be almost equivalent to living in a dystopian society where a designated officer keeps records on who is communicating with whom and relays information from the originator to the recipients.

Why cannot we just replicate the real world information distribution process in the cyberspace? We believe that the reasons behind this phenomenon are the shortcomings of today's networking technology: IP was designed to solve the problem of supporting a point-to-point conversation between two entities. However, the success of the Internet has resulted in IP being used to serve completely different communication patterns, such as social networking, content distribution, information sharing, etc., where users are mostly interested in obtaining desired data rather than reaching a specific node. One can even observe an ongoing trend toward reducing dependency on centralized servers. For example, in popular peer-to-peer file sharing applications such as BitTorrent [1], [2], one can request desired content from *any* peer and download from multiple peers in parallel. However instead of simply requesting the desired data, these P2P applications still have to consider selection of specific peers and setting up connections between peers since they are implemented as overlays on top of TCP/IP's point-to-point communication channels, which may take penalties of inefficient file distribution paths and redundant data transmissions over the same links.

Named Data Networking (NDN) [3], [4], [5] proposed a new Internet architecture design. NDN incorporates basic principles that have made the IP protocol suite widely adopted and globally scaled, including the hourglass protocol architecture model and end-to-end principle. At the same time, NDN completely changes the narrow waist layer to make it better suited for the emerging communication patterns, focusing on data instead of data containers (hosts). Each data packet in NDN is named, cryptographically secured, and delivered to consumer when, and only when, the consumer requests it with an Interest packet. Moreover, since each piece of the data is named and secured, it can be cached anywhere in the network to satisfy future requests; and as the routers maintain per-request states, Interests for the same data piece can be aggregated and the returning data packet are multicast to all requesting consumers.

These and a number of other features from the NDN architecture enabled us to design and implement **ChronoShare**, a *completely distributed* file sharing application. ChronoShare design is based on a new communication primitive, ChronoSync [6], which is briefly introduced in Section II. Being completely distributed, ChronoShare is *agnostic to the network infrastructure support* and is *mobile-friendly*: users can seamlessly share files among their devices, regardless of whether these devices are stationary and connected to the Internet, or constantly moving and have only intermittent or ad hoc connectivity. ChronoShare uses NDN's data-centric security to keep the *provenance* of each file and enable *access control* to the data.

The rest of the paper tells the story of ChronoShare through a description of its design decisions and implementation details, exemplifying how the right architecture can enable "good old" ways of efficient and truly distributed information sharing among friends. Our freely available prototype implementation [7] showed promising results, even though it is still an open-ended story at the time of this writing, with a number of research questions yet to be fully answered, which we will elaborate as we walk through the design of ChronoShare.

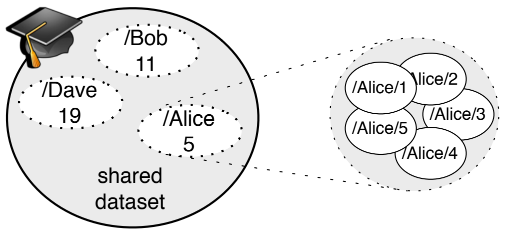


Fig. 1: Maintaining knowledge of a shared dataset

II. A Few Words About ChronoSync

As a new group communication primitive in NDN, ChronoSync [6] enables dataset synchronization among a group of participants in a completely distributed way. This primitive is the most efficient when (1) the synchronized dataset consists of a number of sub-datasets, each produced by an individual participant in the sharing group, and (2) each piece of data is named sequentially in the participant's namespace (Figure 1). This assumption of sequentially numbered datasets significantly simplifies the task of maintaining the up-to-date knowledge about the dataset, since each sub-dataset can be completely represented by the participant's name prefix plus the latest sequence number under that name. For example in Figure 1, a complete knowledge about what dataset was produced by Alice, which consists of five items `/Alice/1`, ..., `/Alice/5`, can be compactly represented as a pair of name prefix `/Alice` plus the sequence number 5.

ChronoSync encodes the knowledge about the whole dataset into a compact crypto digest form (e.g., using SHA256), which we call the *state digest*. This digest is carried in a special "sync" Interest which reaches all participants in a group. Semantically, a sync Interest is a request for any dataset state changes that happen since the state represented by the digest. To reach all the participants in a group, the sync Interests can be multicasted to all participants directly in small networks or via multicast overlays in large networks.[1] When an incoming sync Interest carries a digest that is the same as the locally computed one, it indicates the local dataset is identical to the dataset of the sync Interest's sender. If the state digest carried in a received Interest is different but has been seen before, ChronoSync uses a historical log to find out which of the recently produced data pieces is missed by the digest sender, and replies with the missing data; otherwise, ChronoSync uses a state reconciliation method to resolve the differences. Eventually, sync Interests from all participants will carry an identical state digest, indicating that knowledge about the dataset has been synchronized, and there will be a pending sync Interest at each participant, awaiting for the new data to be generated.

ChronoSync assures that participants will be notified about the changes to the dataset as soon as possible. Note that ChronoSync focuses solely on facilitating the synchronization of the *knowledge* about the dataset, leaving the decision of what to do after new items are discovered to the application's discretion: fetch all missing data items, fetch some of the items, or fetch at a later time when better connectivity becomes available.

[1]The specific design of efficient global-scale participant rendezvous is one of the open-ended parts in ChronoShare story and is under active research.

III. ChronoShare Design

A. Design Decisions

There are many ways to implement file sharing in NDN, with varying level of design complexity and communication efficiency. A straightforward way is to directly synchronize the file set of the shared folder [8], in a way similar to many conventional synchronization applications such as rsync [9]. However, this may not be the best design choice, especially for file sharing among devices with varying connectivity. In a typical scenario, when one user modifies multiple files in a shared folder, it may not be desirable to synchronize all the modified files with all the group members right away—a member may be connected via cellular network, it has no rush to get all the updated files, or may only need one specific file to edit right away.

ChronoShare chose a different approach, which is to treat individual user operations on files as streams of "actions," where each action specifies which file has been modified and what changes has been made (new file, updated content, changed file system permissions, or removing a file).[2] By applying actions from all participants in a deterministic order, in combination with the conflict resolution process described (see Section III-D2), each ChronoShare user can build the consistent up-to-date view of the shared folder and, when desired, fetch all missing files. The main advantage of this action-based approach is that in typical shared folder usage scenarios, no matter how many changes a user might have made to the shared folder, there is a straightforward way to propagate changes to other participants: others just need to fetch all the actions from the user and apply these actions to their folder. Actions by each user form a "stream" of data items, and the streams from all users of a shared folder form a dataset that can be synchronized using ChronoSync primitive.

B. System Entities, Roles, and Assumptions

ChronoShare is a general-purpose file sharing application, where *users* create or join shared folders on one or more of their *devices* (Figure 2). Each *shared folder* is composed of a set of files, which were either locally created, or discovered (and subsequently fetched) via user *actions*. Locally generated actions update the *knowledge about the actions*, which is then synchronized with remote participants using ChronoSync.

Each of the five types of entities in ChronoShare plays a different role. A ChronoShare user is the basic security entity. Each user owns a specific namespace,[3] and a public/private

[2]Actions are carried in NDN data packets, thus they are named and signed, automatically adding ownership information for each operation.

[3]The issue of Internet namespace governance is beyond the scope of this paper; for now we assume the name ownership is supported in the same way as DNS/DNSSEC.

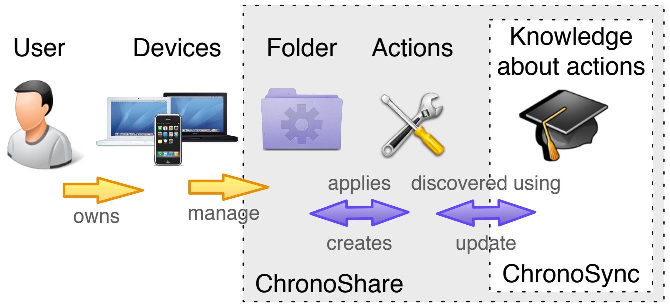


Fig. 2: ChronoShare entities

key pair, which has been certified via an appropriate trust model (e.g., hierarchical trust model certificate deployed on NDN testbed [10]). For example, if Alice owns `/ucla/Alice` namespace, and UCLA authorizes this ownership, then Alice can use the key to authorize her devices for the shared folder by issuing device and shared-folder specific certificates.[4] This way, each user can participate in multiple different shared folders using the same or different devices.

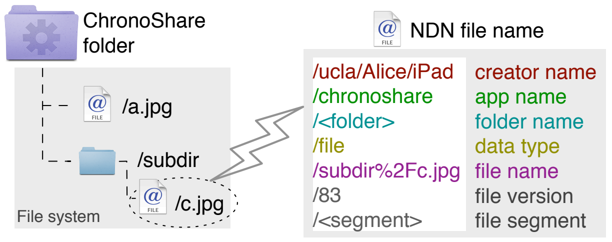
User devices are actual holders of user-authorized shared folders. The ChronoShare design assumes that each device can move from one network to another, having only intermittent Internet connectivity or even only ad hoc connectivity to other devices, getting disconnected any time, or being completely shut down at some point.

A user can create per-device sub-namespaces under his/her own namespace and publishes data. Assuming Alice owns `/ucla/Alice` namespace, her desktop can publish data using `/ucla/Alice` prefix, while her laptop and iPad can be publishing data using more specific `/ucla/Alice/laptop` and `/ucla/Alice/iPad` prefixes. In this way, we leverage the unique feature of NDN to specify authority and provenance of the data in the data name and secure this authority using cryptographic signatures, while all cryptographic operations can be completely automated using a trust schema [11].

C. Data Naming

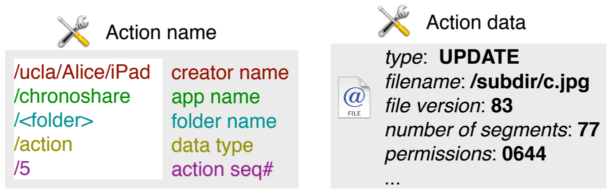
NDN "smashes" communication layers by directly using application-defined names to deliver packets across the network. Therefore, data naming in an NDN application needs to be cross-layer cautious. In other words, because an NDN network forwards each Interest packet using the data name carried in the Interest, the name should contain some "routable" (unicast or multicast) prefix (i.e., a longest match lookup of the name against the router FIB should be able to find a matching entry). At the same time, when/if an Interest reaches the data producer, the Interest name should then identify the appropriate application process to which this Interest should be delivered. Finally, when the application receives the Interest, it needs to use the Interest name internally to "route" this Interest to the right method to return the requested data.

[4]Trust model and specific tools for certificate handling and verification is currently under active research; see [11] for our recent results.



/ucla/Alice/iPad	creator name
/chronoshare	app name
/<folder>	folder name
/file	data type
/subdir%2Fc.jpg	file name
/83	file version
/<segment>	file segment

Fig. 3: Cross-layer design for NDN file names



/ucla/Alice/iPad	creator name
/chronoshare	app name
/<folder>	folder name
/action	data type
/5	action seq#

type: **UPDATE**
filename: **/subdir/c.jpg**
file version: **83**
number of segments: **77**
permissions: **0644**
...

Fig. 4: Cross-layer design of action names

ChronoShare uses three types of inter-dependent data and the corresponding naming schemes for files, actions, and ChronoSync knowledge about the actions. Below we introduce each of these datasets, their naming design, and their relations.

1) Files: Files in ChronoShare's shared folder are split into segments, each segment forming an NDN data packet that is named, secured, and can be stored in the database and effectively transmitted over the network. NDN name of the file segment consists of seven components as illustrated in Figure 3: creator's prefix, application name, folder name, file data type, file name, file version, and file segment. The creator prefix guides Interests towards the specified creator and his device of the desired data, serving also as a permanent file provenance identifier. The next components identify the application and application instance on the host with the role similar to port number in IP, while the rest are used internally by the application to identify a specific file, its version, and segment of the file.

2) Actions: Similar to the cross-layer consideration of the file name structure design, ChronoShare defines action names consisting of creator's prefix, application name, folder name, action data type, and action sequence number (Figure 4). Each user (or each of his or her device) publishes a "stream" of actions under user's (or the device's) namespace, which identifies the corresponding ChronoShare application and specific shared folder instance.

Note that the components of the creator's name, application name, and folder name are exactly the same in both the action and the corresponding file names (compare Figure 3 and Figure 4): the file modification by a user in a shared folder is always paired with the corresponding action item from this user in the shared folder. Therefore, the content of the action does not need to include the full NDN name of the file in the shared folder, only a relative name of the file in the shared folder and additional meta information about the revision of this file (version, size, permissions, etc.). Given an action, the

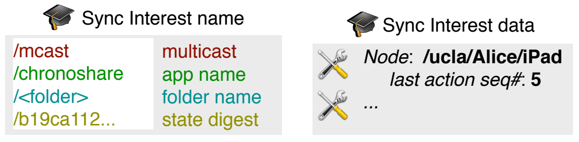


Fig. 5: Cross-layer design of sync Interest names

implicit relationship between action content and the file name can be used to construct a full NDN name for data retrieval.

3) Knowledge about actions (ChronoSync): The structure of the action namespace, with the last component being the action sequence number, is designed specifically to meet the data naming requirements of ChronoSync [6]. The remaining question is the proper naming model for ChronoSync Interests: a sync Interest should fully represent the state of a *specific* shared folder, and it should be able to reach every participant.

Following the guidelines [6], ChronoShare defines a four-component structure for sync Interests: multicast prefix, application and folder name, and dataset digest (Figure 5). The first component is to ensure that every participant receives the sync Interest; the middle two ensure that the right instances of ChronoShare will get the Interest; and the last component is computed over all user names plus their sequence numbers. Sync Interests should be small in size and volume, multicasting them directly (or via multicast overlays in large networks) is a simple way to achieve the goal of completely distributed propagation of sync Interests to all participants and synchronization of the knowledge about actions on the shared folder. As mentioned earlier, after the actions are discovered, actions and corresponding files can be directly pulled from the data producers, leveraging the benefits of NDN's built-in multicast delivery.

Similar to actions, the content of sync Data does not need to specify the full names of actions. Instead, sync Data specifies only action creator's prefix and action sequence number; the application and folder fields of the action name can be inferred from the sync name.

D. Folder and File Operations

ChronoShare assumes that each device runs a local file manager, which detects file system changes and notifies ChronoShare to create new actions and distribute them to remote folders via ChronoSync. On the other direction, actions discovered by ChronoSync are routed to the file manager, who applies them to the shared folder and resolves any potential conflicts, e.g., when different users modify the same file.

1) Creating actions from local changes: To detect local changes to the shared folder, ChronoShare relies on platform-dependent file watching mechanisms.[5] Since the same file can be updated multiple times within very short time periods, ChronoShare delays (e.g., for 0.5 second) reaction for each notification from the file watcher. If the same file is being repeatedly modified within this delay, processing is postponed further, under the assumption that only the "stable" file version is of interest to the remote users.

During the notification processing, ChronoShare compares meta-information (permissions, file size, content digest, etc.) of the file on disk and information stored internally. If the file was indeed changed (not just re-saved or created by applying remote action), ChronoShare creates a new **UPDATE** or **DELETE** action, specifying new sequence number of the action as part of the name (+1 to the last user's action or 1 for the first action), a new version number of file (+1 to the locally known version or 0 for a new file) and other meta information as part of the content.

After the action is created, ChronoShare signs, encrypts, and publishes it in the local database, to be fetched upon requests. Similarly, content of the modified file is chopped into segments, each receiving a unique name within a user-device-shared-folder namespace, signed, encrypted, and published. As a final operation, ChronoShare notifies ChronoSync module about the newly available local action, which in turn notifies the remote participants of the new action.

2) Applying actions and conflict resolution: Whenever ChronoSync discovers new actions, it immediately passes the actions to ChronoShare if they are piggybacked in sync Data packets, otherwise informs ChronoShare to fetch them. If the fetched actions specify file updates, ChronoShare also fetches the corresponding segments of the updated file. After this process, ChronoShare determines how to apply the fetched action to the shared folder on the local file system. To have a consistent view of the shared folder, actions from different participants must be applied in a consistent order. ChronoShare defines this order based on a per-file versioning: *the larger file version number of a file supersedes any other version*. If two users were working on the same file while being physically disconnected and assigned the same version, then version of a user with lexicographically "larger" name prefix is defined to be the "winner." This way, whenever a conflict happens, ChronoShare automatically determines which is the "correct" version of the file to be present in the folder.

ChronoShare stores all actions and all file versions permanently or within a user-configured time period. As a result, if the user determines that the automatic action process made a mistake, he or she can always instruct ChronoShare to check out an "older" version, in a way similar to how it is done in many version control systems. In our prototype, we implemented this functionality leveraging another feature of NDN: simple name-based inter-process communication. We defined a REST-like protocol to request lists of all available actions and file versions, as well as a web browser-based interface (implemented NDN-JS) [12] to "restore" the specified version of a file.

Conceptually, ChronoShare allows action aggregation and periodic snapshots of the shared folder, which would speed up shared folder initialization, especially when new users join a shared folder with a long history of changes. However, this is another open-ended part of our story: the specific design is on our future work agenda (RepoSync [13] is an example of

[5]Our prototype implementation uses QT framework which provides these platform-dependent mechanisms as part of QFileSystemWatcher class.

our exploration of the design space).

E. Security Considerations

To secure file sharing, ChronoShare relies on several aspects of NDN architecture. First, since all NDN data packets, including sync data, action data, and file segments are uniquely named and signed, it is impossible for unauthorized third parties to tamper with the communication: false data packets would be simply discarded. In the existing NDN implementations, although malicious users may be able to disrupt ChronoShare functions using content poisoning [14] or Interest flooding attacks [14], [15], NDN as a whole provides all necessary components—per-packet state and two-way symmetric Interest/Data flows—to effectively mitigate such attacks [15].

Each data packet in private data folders, including content and application-specific name components, can be effectively protected using, for example, a well know shared key or group-based encryption [16] technologies. In this case, the creator (or an elected moderator) can authorize new users to join in and share the folder. All data exchanges among ChronoShare users can be encrypted for privacy protection, however we are yet to consider whether user and folder names also need protection by encryption. Again, ChronoShare privacy is another branch of our story that does not yet have a concrete denouement.

F. Action and File Retrieval Strategy

1) Ordering and selective retrieval: ChronoSync notifies ChronoShare ASAP about newly occurred actions, and it is the responsibility of each specific ChronoShare instance to decide when and which actions to fetch. Since the action with the largest sequence number is more likely operating on the latest version of a file, in our prototype we fetch (and if necessary apply) new actions for each participant in the order from the largest sequence number to the smallest. This way, multiple changes of the file by the same participant will be quickly reflected in the shared folder (even for changes for the same file by multiple participants, the latest action(s) from each participant most likely represent the latest version of the file in the shared folder). Although our prototype implementation fetches all versions of all discovered files, this behavior is not required in general; a mobile ChronoShare instance may elect to fetch files only when it is WiFi connected or when requested by the user.

2) Special cases: An action or a file segment can be fetched directly from the creator, with or without infrastructure support—this can happen when a requester and the creator are both connected to the Internet, or both connected to the same WiFi access point, or simply have an ad hoc connectivity in between. ChronoShare also leverages NDN ability to get data by name from any node that has the data. When the creator device of the requested data moves to a different network, or when the device is powered off, a requester may still be able to retrieve the data from router caches.

However, if the data is not in router cache, the data cannot be retrieved using the original data name, even if other online ChronoShare participants have the requested data. This is because they will not receive the Interest packets as NDN routers forward interests based on the data creator's name carried in the interest packets (towards UCLA in `/ucla/Alice` case). To solve this problem, ChronoShare employs the concept of a *link* that is included in Interests, alongside to the requested name [17]. The link is defined to include one or more name prefixes, which can guide Interest forwarding to the place it can be satisfied. However, since the link is yet to be supported by NDN forwarder implementation [18], our ChronoShare prototype simply prepends the link in front of the requested data. As a response, the data producer will encapsulates the requested data inside of the data packet with the concatenated name.

Mobility: When user's device moves, it can notify others through ChronoSync, about its current link, which can be used to reach it. For example, when Alice's laptop moves from UCLA to home, it can inform others about its link {`/Alice`→`/comcast/westwood/Alice`}, so that others can continue to fetch effectively data with `/Alice` prefix, attaching this link to the interests.

Device Offline: When a device that created an action of a file segment goes offline, the data it created can still be available on other devices that share the same folder. Thus, when another ChronoShare device experiences multiple Interest timeouts, it can simply use a link of some other participant, e.g., selecting this user at random, to request the desired data. If the folder is shared inside a local environment, it also possible to use a multicast link {`/Alice`→`/mcast/Alice`} (this feature is implemented in our prototype) to request data from anybody who may have it.

IV. Related Work

Many cloud-based services like Dropbox (https://www.dropbox.com/), SugarSync (https://www.sugarsync.com/) and SpiderOak (https://spideroak.com/) provide their users with software that is capable of running on most platforms and maintaining a consistent view of shared files across multiple devices. However, these services rely on a central server to coordinate updates to the files across devices, and each device must consult only this server for the latest information.

Dropbox includes a local area network synchronization feature to their client software that allows devices to request the latest copies of shared files from other devices within the same subnet, bypassing the need to upload and download the files from the Dropbox cloud. This synchronization feature takes advantage of the high bandwidth connections that exist between local devices to speed up the synchronization process. However, in order to take advantage of Dropbox LAN synchronization, all devices involved must be connected to *both* the LAN and the Internet at the same time, because the Dropbox cloud is responsible for negotiating the connections between the local devices to maintain their security model.

SugarSync is a cloud-based file synchronization service that behaves much in the same way as Dropbox. However, where Dropbox prefers to create its own dedicated folder on a device, SugarSync provides its users with the ability to synchronize any folder on that device. It also provides support for platforms Dropbox does not currently support including Windows Mobile, Symbian and Kindle Fire.

SpiderOak is another cloud-based file synchronization service that behaves much in the same way as SugarSync. SpiderOak provides users with the ability to choose which folders on the device should be synchronized. SpiderOak boasts a "zero knowledge" security model where the password used to encrypt and decrypt shared files is only known to the user. Not only this makes it impossible to recover data if one forgets the password, but it also makes sharing folders and files more difficult. To share folders and files with others, the user must create a "ShareRoom" with a unique name and password and securely pass this information to those the user wishes to share the files with.

In addition to the above mentioned services, other services like Google Drive and Microsoft Skydrive provide their users with the ability to edit documents in the cloud directly using in-browser tools. In addition, these services also offer real-time collaborative editing for certain document types. However, an Internet connection is required to access and edit the documents; the files would not be accessible even if the sharing devices can directly reach each other. Needless to say that the files will also be unavailable if the cloud-based services experience any downtime.

BitTorrent Sync (https://www.getsync.com/) seems resembling ChronoShare most closely. BT Sync discovers peer devices that have access to a shared folder through one of the following means: a special BitTorrent tracker, or contacting a user-defined IP address (serving the same role as the tracker), or DHT, or searching local network. The peer devices then inform each other of their own folder content, so that each can decides what to fetch and from whom. File transfers are done by the BitTorrent protocol, allowing devices to receive file pieces from several peers simultaneously. When the contents of the shared folder change, Sync immediately informs peers of the changes. Besides no mentioning of mobility or ad hoc connectivity support, BT Sync differs from ChronoShare in that it does not eliminate the need for a rendezvous server (tracker), and that it lacks strong data provenance support.

V. Epilogue

We have built a working prototype of ChronoShare and put it through trial usage. On several occasions, ChronoShare already helped us to sync up files among nearby laptops, when other cross-platform file sharing applications failed due to the failure of WiFi-based Internet connections. However, the story of ChronoShare, as it can be seen from several remarks in the design sections, is still at its beginning. We plan to continue refining the design and implementation, and we would like to invite all interested parties to help us resolve the identified research challenges, so that the story will reach its finishing line.

At the same time, we hope that our story has fulfilled another, no less important goal of exemplifying the data-centric application design process in NDN. If we look back at the ChronoShare design, we can see that all of the design decisions are about data, data naming, data provenance, and data fetching. There is no notion about addresses, sessions, or maintaining connections between hosts. This focus on data makes mobility support easy. Although device mobility or disconnectivity leads to special case handling, yet the solution is still about the data, with the link suggesting the direction of where the data may be found. Thus, ChronoShare is a good example of not only how NDN brings back truly distributed file sharing, but also how NDN ensures that application developers do not have to waste their time on developing and debugging various connection-oriented tricks, but focus primarily on data: how to name the data, who is authorized to produce the data, who should have access to the data, and what one wants to do with the data.

References

[1] B. Cohen, "The BitTorrent protocol specification," 2008.

[2] BitTorrent Labs, "BitTorrent Sync," http://labs.bittorrent.com/experiments/sync.html.

[3] V. Jacobson *et al.*, "Networking named content," in *Proc. of CoNEXT*, 2009.

[4] L. Zhang *et al.*, "Named data networking (NDN) project," NDN, TR 0001, 2010.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM Computer Communication Reviews*, June 2014.

[6] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking," in *Proceedings of IEEE ICNP*, 2013.

[7] A. Afanasyev, Z. Zhu, and L. Wang, "ChronoShare prototype implementation," http://github.com/named-data/ChronoShare, 2013.

[8] J. Lindblom, M.-C. Huang, J. Burke, and L. Zhang, "FileSync/NDN: Peer-to-peer file sync over Named Data Networking," NDN, TR 0012, 2013.

[9] A. Tridgell and P. Mackerras, "The rsync algorithm," TR-CS-96-05, 1996.

[10] C. Bian, Z. Zhu, A. Afanasyev, E. Uzun, and L. Zhang, "Deploying key management on NDN testbed," NDN, TR 0009, Rev. 2, February 2013.

[11] Y. Yu, A. Afanasyev, D. Clark, kc claffy, V. Jacobson, and L. Zhang, "Schematizing and Automating Trust in Named Data Networking," in *Proceedings of ACM Information Centric Networking Conference*, September 2015.

[12] W. Shang, J. Thompson, M. Cherkaoui, J. Burke, and L. Zhang, "NDN.JS: A javascript client library for Named Data Networking," in *Proc. of INFOCOMM NOMEN Workshop*, 2013.

[13] W. Shi and A. Afanasyev, "RepoSync: Combined action-based and data-based synchronization model in Named Data Network," in *Proceedings of 18th IEEE Global Internet Symposium (GI 2015)*, April 2015.

[14] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang, "DoS & DDoS in Named-Data Networking," *arXiv preprint arXiv:1208.0952*, 2012.

[15] A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and L. Zhang, "Interest flooding attack and countermeasures in Named Data Networking," in *Proc. of IFIP Networking*, 2013.

[16] A. Kiayias, Y. Tsiounis, and M. Yung, "Group encryption," in *Proc. of ASIACRYPT*, 2007.

[17] A. Afanasyev, C. Yi, L. Wang, B. Zhang, and L. Zhang, "SNAMP: Secure namespace mapping to scale NDN forwarding," in *Proceedings of 18th IEEE Global Internet Symposium (GI 2015)*, April 2015.

[18] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Li, S. Mastorakis, Y. Huang, J. P. Abraham, S. DiBenedetto, C. Fan, C. Papadopoulos, D. Pesavento, G. Grassi, G. Pau, H. Zhang, T. Song, H. Yuan, H. B. Abraham, P. Crowley, S. O. Amin, V. Lehman, and L. Wang, "NFD Developer's Guide," NDN, Technical Report NDN-0021, Revision 4, May 2015.