Named Data Networking Next Phase (NDN-NP) Project
May 2016 - April 2017 Annual Report

Principal Investigators

Van Jacobson, Jeffrey Burke, and Lixia Zhang
University of California, Los Angeles

Tarek Abdelzaher
University of Illinois at Urbana-Champaign

Beichuan Zhang
University of Arizona

kc claffy
University of California, San Diego

Patrick Crowley
Washington University

J. Alex Halderman
University of Michigan

Christos Papadopoulos
Colorado State University

Lan Wang
University of Memphis

Contents

FIA-NP: Collaborative Research: Named Data Networking Next Phase 2016 – 2017 Report

Executive Summary

The heart of the current Internet architecture is a simple, universal network layer (IP) which implements all the functionality necessary for global interconnectivity. This *thin waist* was the key enabler of the Internet's explosive growth, but its design choice of naming communication endpoints is also the cause of many of today's persistently unsolved problems. NDN retains the Internet's hourglass architecture but evolves the thin waist to enable the creation of completely general distribution networks. The core element of this evolution is removing the restriction that packets can only name communication endpoints. As far as the network is concerned, the name in an NDN packet can name anything – an endpoint, a data chunk in a movie or a book, a service, or a command to turn on some lights. This conceptually simple change allows NDN networks to use almost all of the Internet's well-tested engineering properties to solve not only communication problems but also digital distribution and control problems.

Our seven years of NDN design and development efforts tackled the challenge of turning this vision into an architectural framework capable of solving real problems. Our *application-driven architecture development* approach forces us to fill in details and to verify and shape the architecture design. We translated our vision into a simple and elegant packet format design, a modular and extensible NDN forwarding daemon, and a set of supporting libraries. We illustrated the utility of the architecture for four environments: building automation management systems, mobile health, multimedia conferencing tools, and scientific data applications. The implementation and testing of pilot applications in these network environments demonstrated our research progress in namespace design, distributed dataset synchronization, trust management, encryption-based access control, and stateful forwarding.

Highlights from this year's achievement include the following.

- NDN-based applications to enable IoT systems and multimedia applications that work without reliance on cloud infrastructure.
- Developments in flexibile NDN certificate management to address security usability.
- Lessons learned from a comparative analysis of data synchronization approaches in NDN, culminating in a development of a new VectorSync protocol design responsive to these lessons.
- Progress in NDN hyperbolic routing protocol designs, and improvements to efficiency of conventional name-basead forwarding performance.
- Evolution of NDN congestion control, and NDN deployability in LANs and ad hoc scenarios, including name-based packet filtering at network interface cards, NDN over WiFi Direct, and a secure and efficient self-learning strategy for switched Ethernet.
- Multi-faceted evaluation platforms of the architecture, from a well-instrumented NDN testbed to the Mini-NDN emulator environment and ndnSIM.
- The third NDN Community meeting hosted by University of Memphis with sponsorship from NDN Consortium members to promote a vibrant open source ecosystem of research and experimentation around NDN.

NDN has gained increasing attention in the broader networking community, as evidenced by the NSF/Intel's new program on Information Centric Networking at Wireless Edge Networks (ICN-WEN) and DARPA's Secure Handhelds on Assured Resilient networks at the tactical Edge (SHARE) program.

NSF's FIA and FIA-NP programs nurtured NDN from an architecture vision to an operational prototype. The conclusion of NSF's FIA-NP program marks the end of our beginning of an exciting journey in architecture research. Most NDN project team members have received funding to carry on NDN design and development efforts through other programs, including NSF CRI (Univ. Arizona, Univ, Memphis, UCLA, Washington Univ.), NSF CC* Integration (Colorado State Univ.), and two expected awards from the NSF/Intel ICN-WEN program. The NDN team will continue to expand our collaborative efforts to refine the NDN architecture and demonstrate its capabilities to support 21st century communication scenarios.

Chapter 1

Introduction

This report summarizes our accomplishments during the third year of the "Named Data Networking Next Phase (NDN-NP)" project. This phase of the project focuses on deploying and evaluating the NDN architecture in four environments: building automation management systems, mobile health, multimedia real-time conferencing tools, and scientific data applications.

Our NDN application efforts over the last year have included the framework development of IoT systems that can bootstrap autonomously with no dependency on cloud services (Section 2.1.1); completion of the NDNfit application whose design touches upon naming, trust management, data confidentiality control, mobility support, and distributed data storage (Section 2.1.2); and continued development of the Flume application which enables users to participate in group conversations (text, images, files, etc.), as well as live media streams, e.g., video and/or audio (Section 2.1.3). Especially exciting is our new collaboration with Operant Solar, a DOE funded startup, to help them develop NDN-based solar networking applications. These new application developments drive evolution of the NDN architecture.

Our efforts in the security area focused on the usability of data-centric security support. To that end we made substantial progress in developing the NDN certificate management protocol NDNCERT [7] (Section 3.1). NDNCERT provides flexible mechanisms to establish certificate authorities (CA) for different NDN namespaces. To facilitate data validation, we developed NDN certificate bundling functionality (Section 3.2), which leverages naming conventions to enable applications to automatically retrieve batches of certificates needed for data verification.

Some of the most interesting architecture and protocol advances over the last year were in distributed dataset synchronization, an important NDN communication primitive. We completed the design and evaluation of PartialSync (pSync) [5]. We also conducted a thorough comparative study of the different sync protocols we have developed over the last few years, including CCNx Sync [1], iSync [6], ChronoSync [8], RoundSync [2], and pSync. We summarize their commonalities, differences, and design tradeoffs (Section 4). Building upon what we have learned, we started developing a new Sync protocol, VectorSync, to meet the needs of different classes of applications.

We also made significant progress in the area of routing and forwarding, including routing protocols, scalable forwarding, congestion control, and NDN in local area networks. The routing protocol development continues to progress in two parallel directions: conventional routing (Named-data Link State Routing, NLSR [4]) and update-less greedy routing (Hyperbolic Routing, HR [3]) (Section 5.1). In scalable forwarding, we enhanced name-based forwarding performance with memory- and time-efficient data structures and deepened our understanding of the role of forwarding strategy and its interplay with applications. In parallel we continued implementing new congestion control techniques. We also implemented functionality to support LAN-based NDN deployments, including name-based packet filtering on network interface cards, a secure and efficient self-learning strategy for switched Ethernet, and consideration of scenarios to support incremental deployment paths for NDN (Section 5.4).

This past year has seen an intensified effort in moving NDN forward. We organized an NDN routing and forwarding meeting at University of Memphis on May 8-9, 2016 and two NDN project retreats, one hosted

by Colorado State University in November 3-4, 2016, and one by University of Memphis on March 22, 2017. These meetings helped resolve pressing issues in the NDN design and development, including hyperbolic routing, sync design, NDN certificate management, and application supports.

In parallel, we continued our efforts in expanding the NDN research and development community. The National Institute of Standards and Technology (NIST) and NDN team jointly organized an NDN workshop in June 2016.[1] The NDN team organized the third NDN Community Meeting on March 23-24, 2017 at the University of Memphis.[2] During the first two months of 2017 we also collaborated with multiple companies in their proposal development for DARPA's SHARE program. As we finish this report, we are starting the preparation for the first NDN tutorial to be given at ACM SIGCOMM 2017 in August.

The NSF's Future Internet Architecture program has provided us with a tremendously valuable seven years to gain deeper insights into name-based data networking architectures and associated research challenges. The conclusion of this program marks the end of our beginning on this exciting journey – the intellectual depth of reflection on lessons learned from the last forty years of network architecture research and development, and the broad impact possible from practical experimentation with architectural innovations based on these lessons learned.

References

[1] CCNx Project. CCNx Synchronization Protocol. `https://github.com/ProjectCCNx/ccnx/blob/master/doc/technical/SynchronizationProtocol.txt`, 2012.

[2] Pedro de las Heras Quirós et al. The Design of RoundSync Protocol. Technical Report NDN-0048, NDN Project, March 2017.

[3] Vince Lehman, Ashlesh Gawande, Rodrigo Aldecoa, Dmitri Krioukov, Lan Wang, Beichuan Zhang, and Lixia Zhang. An Experimental Investigation of Hyperbolic Routing with a Smart Forwarding Plane in NDN. In *Proceedings of the IEEE IWQoS Symposium*, June 2016.

[4] Vince Lehman, A K M Mahmudul Hoque, Yingdi Yu, Lan Wang, Beichuan Zhang, and Lixia Zhang. A secure link state routing protocol for NDN. Technical Report NDN-0037, NDN Project, January 2016.

[5] Lan Wang Minsheng Zhang, Vince Lehman. Scalable Name-based Data Synchronization for Named Data Networking. In *IEEE INFOCOM*, April 2017.

[6] Fu Wenliang, Hila Ben Abraham, and Patrick Crowley. Synchronizing namespaces with invertible bloom filters. In *To appear in ANCS 2015*, 2015.

[7] Zhiyi Zhang, Yingdi Yu, Alex Afanasyev, and Lixia Zhang. NDN certificate management protocol (NDNCERT). Technical Report NDN-0054, NDN, April 2017.

[8] Zhenkai Zhu and Alexander Afanasyev. Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking. In *IEEE ICNP*, 2013.

[1] `https://www.nist.gov/news-events/events/2016/05/workshop-named-data-networking`.
[2] `https://www.caida.org/workshops/ndn/1703/`.

Chapter 2

Network Environments / Applications

Contributors	
PIs	Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Lixia Zhang (UCLA), Tarek Abdelzaher (UIUC), Christos Papadopoulos (Colorado State), Beichuan Zhang (University of Arizona), Lan Wang (University of Memphis)
Grad Students	Dustin O'Hara, Wentao Shang, Haitao Zhang, Zhiyi Zhang, Zhehao Wang, Yukai Tu, Yumin Xia, Spyridon Mastorakis (UCLA); Nick Gordon (University of Memphis); Jongdeog Lee (UIUC); Junxiao Shi, Eric Newberry, Teng Liang (University of Arizona); Nick Gordon, Muktadir R Chowdhury (University of Memphis).
Staff	Peter Gusev, Jeff Thompson, Zoe Sandoval (UCLA)

This section of the report covers work done on these network environments and other applications performed by the team over the course of the last year, during the supplement period.

2.1 Network Environments

As part of the NDN "Next Phase" research, the NDN project team proposed two network environments, **Enterprise Building Automation & Management** and **Open mHealth**, and one application cluster, **Mobile Multimedia**, to drive our research, verify the architecture design, and ground evaluation of the next phase of our project. The two environments represent critical areas in the design space for next-generation Health IT and Cyberphysical Systems, respectively. They also extend work started in the previous NDN FIA project on participatory sensing and instrumented environments to focus on specific application ecosystems where we believe NDN can address fundamental challenges that are unmet by IP.

2.1.1 Enterprise Building Automation & Management

Our objectives for the supplement period were to reconnect the campus data to the Mini-BMS testbed, to complete and document security and aggregation approaches, and to demonstrate interaction with IoT devices communicating natively over NDN.

Mini-BMS

Mini-BMS is an extension of the NDN Building Automation and Management system from 2015 to 2016, discussed in previous reports. It incorporates data collection from the UCLA campus Siemens systems, data aggregation, schematized trust [16], and a visualization unit. The system was reconnected with the UCLA campus data; real UCLA campus data (raw and aggregated) are now made available on the NDN testbed

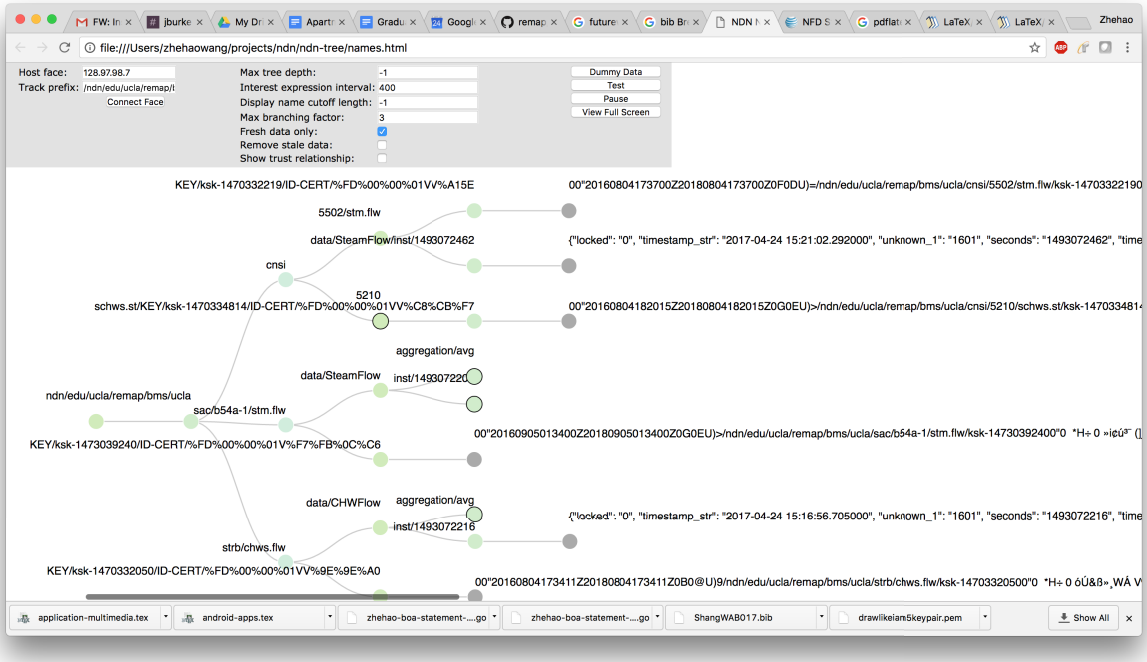


Figure 2.1: Example of Mini-BMS UCLA campus data namespace tree visualization.

for researchers from other campuses. We have receive a few requests for access to this sample BMS data and are considering the policies and process for handling them.

To make the system easier to understand, we have developed an in-browser namespace tree visualization tool, built using NDN-JS and D3.js libraries. The tool is now used to visualize data from the running BMS system. Fig. 2.1 shows an example of UCLA Mini-BMS namespace tree visualized using data from the testbed. The same visualization tool has been demonstrated to work with the IoT demonstration discussed below, enabling illustration of how IoT and traditional BMS systems can be integrated.

We did not make further progress on name-based access control for this application during the supplement period as we originally planned; instead, we focused the effort to get the NDN-IoT framework not only up and running, but also progressed further than originally envisioned, as described below.

NDN-IoT Framework

This year, we extended consideration of EBAMS environments to include home IoT devices, a consistent research interest of many in the ICN community. Most, if not all, emerging IoT approaches depend on cloud services to facilitate interoperation of devices and services within them, even when all communicating entities reside in the same local environment, as in many smart home applications. While cloud-based designs offer a path of least resistance to implement IoT applications using today's TCP/IP protocol stack, they also introduce dependencies on external connectivity and services that are unnecessary and and often brittle, as evidenced by the never ending news series on cloud services failures. This motivated us to achieve a cloud-optional design, a problem not addressed yet in the EBAMS environment. The power and scale of cloud services can offer a rich set of advanced functions such as voice recognition, machine learning, purchasing, etc., however we do not want NDN applications to *depend* on the cloud to carry out local functions.[1]

We identified two core framework-level services for IoT applications, security bootstrapping and device discovery/rendezvous, that today's home IoT solutions depend on the cloud to provide. We used the design

[1] "Smart feeder outage left pets hungry for 10 hours," `http://www.telegraph.co.uk/technology/2016/07/28/smart-feeder-outage-left-pets-hungry-for-10-hours/`.

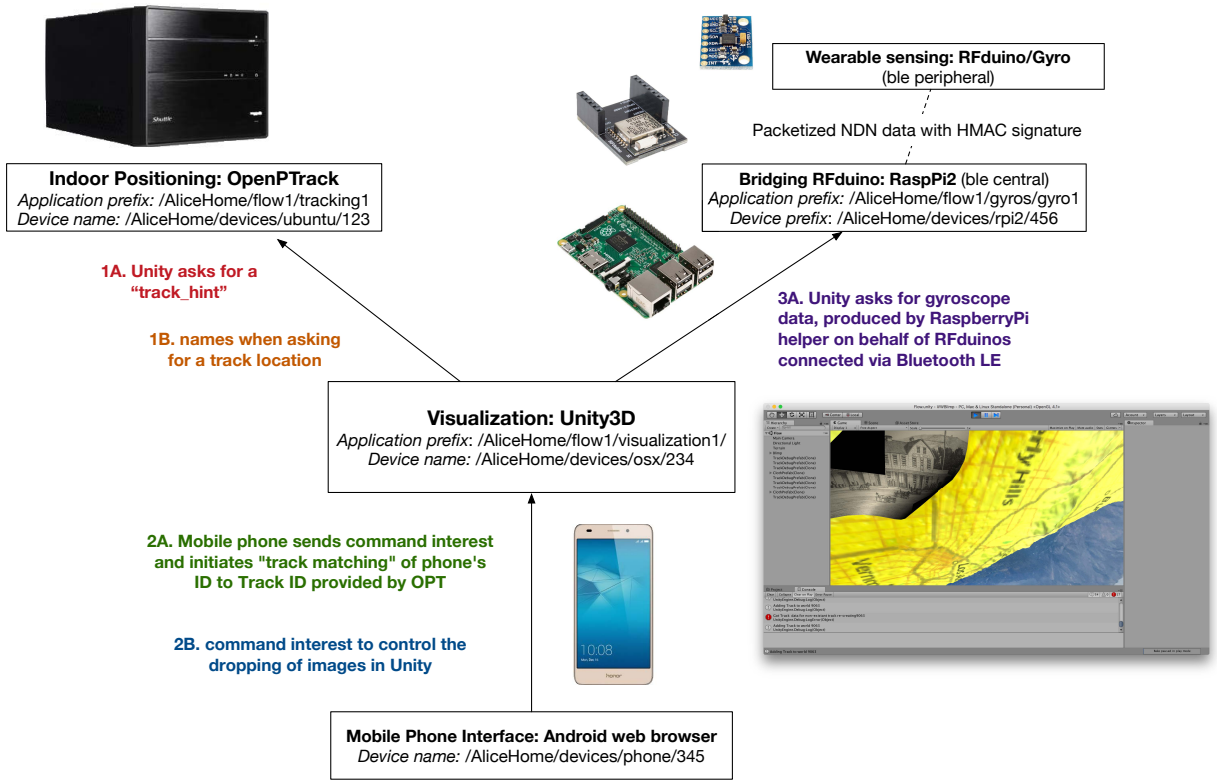


Figure 2.2: Application components and message flows in Flow.

of an IoT-enabled home entertainment application—dubbed *Flow* –to demonstrate how the NDN architecture enables cloud-independent IoT applications, including those with low-latency requirements. NDN enables local trust management and rendezvous service, which play a foundational role in realizing other IoT services. We completed the development of Flow through a combination of NDN-NP effort and an industry gift funding. It provides an exploratory experience in which a player navigates and interacts with a virtual environment via a combination of infrastructure-based sensing (person tracking), wearable devices (gyroscope), and their mobile phone, all communicating using NDN.

To develop Flow, we designed and implemented the Named Data Networking of Things (NDN-IoT) framework based on the high-level ideas the team described in a paper at IOTDI 2016 [11]. This work included generalized libraries in C#, JavaScript, Python and C++ languages that implement naming, trust and bootstrap, discovery / rendezvous and application level publish/subscribe, to explore IoT-based application development in a home environment. Fig 2.2 shows an example of messages exchanged between different components of the application in the installation. Our latest results were presented at IOTDI 2017 [12], at the NDN 2017 Community Meeting, and installed at Futurewei Technologies in January 2017.

Fig. 2.3 shows an example of a namespace in the Flow application. Expanding on the data hierarchy in the EBAMS system, the application uses three levels of names—manufacturer-level (e.g. "`/com/RF-digital`"), device-level (e.g. "`/AliceHome/devices/rfduinos/123`"), and application-level (e.g. "`/AliceHome/flow1/gyro1/data`")—which differentiate the naming of devices, things and their data in an IoT environment. The project used the team's work on schematized trust [16] to implement a hierarchical trust relationship, in which an authorization server in the home environment acts as the trust anchor, and grants fine-grained control, specifying which devices can publish what data for chosen application instances. It also demonstrates a name-based, distributed rendezvous mechanism for devices and applications to discover each other over NDN. The synchronization process utilizes the decentralized and serverless ChronoSync [20] protocol to effectively synchronize prefixes of active devices and applications under the "`/AliceHome/discovery`" sub-namespace. In this way the application brought together a number of different key concepts in NDN.

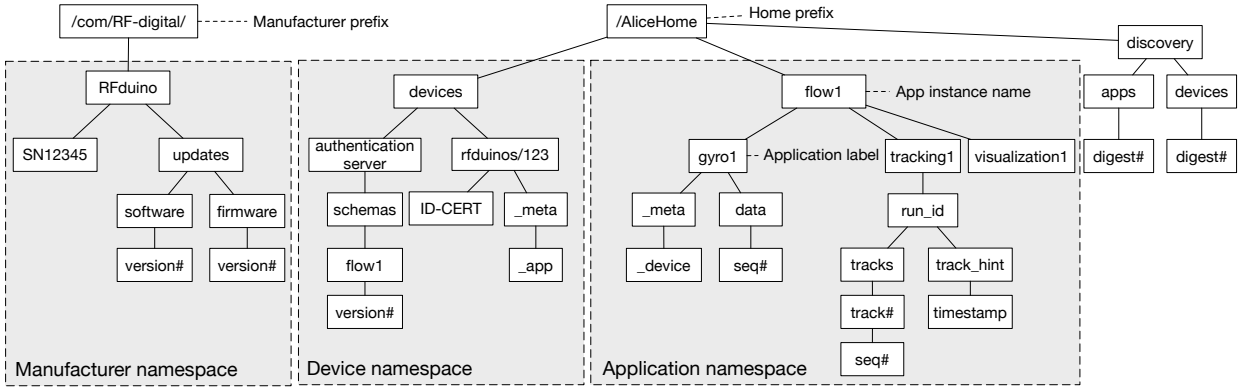


Figure 2.3: Example namespace within the home environment where Flow is deployed.

Through the design and development of the Flow application and NDN-IoT framework, we explored a cloud-independent approach to IoT applications, which we believe is a fundamental contribution to the "enterprise" part of the EBAMS network environment, although the work did not start with this explicit focus. We also made a side-by-side comparison between NDN's cloud-independent solutions and cloud-centric approaches [12].

Challenges and future work remain for this application, including designs for encryption-based access control in networks with heterogeneous computational capabilities, which we expect to explore as part of IoT work in the NSF CRI award recently received by some team members. Also, further evaluation of our approach and empirical study of the framework interface and mechanism designs using more applications is needed. Given the high level of both research and industry interest in this area, we are hopeful this will be pursued by other researchers as well as our team.

2.1.2 Open mHealth

Our overarching task for the Open mHealth network environment during the supplement period was to finish and document the NDNFit driver application, which has many constituent components. We achieved these objectives. We designed and implemented a mobility support mechanism, *data visualization unit (DVU)*, and incorporated a namespace visualizer based on the one designed for Mini-BMS. We also extended the name-based access control mechanism to better support NDNFit. We integrated certificate management and autoconfiguration mechanisms into NDN-Android, and pushed out the LINK object support mechanism on the testbed. We also added an NDNFit trust anchor as an additional trust root on the testbed, demonstrating an independent root for the application. With these new designs, we revised the demo system running on the testbed—supporting a capture application, a *data storage unit (DSU)*, two *data processing units (DPUs)*, one NFN DPU and one native NDN DPU, the DVU and namespace visualizer.

We published our results from this work [18] at ACM ICN 2016; we are currently working on a technical report (NDN-0052) to document the design and implementation specifics as well as the lessons learned; we expect to finish this report by the end of the academic year.

Mobility support. To support mobility (specifically, producer mobility), we employed the *data depot+mapping* solution described in [19]. In NDNFit, DSUs work as the data depot, and a *forwarding-hint* (represented as a Link Object) is used to implement mapping. To report a mapping to the DSU, the NDNFit data capture application running on mobile devices actively acquires the routable prefixes of the connected edge NFD, creates a LINK object and sends it to the DSU. To keep the mapping up to date, whenever the device moves to a new network, it provides a new forwarding-hint to the DSU, which enables the DSU to send an interest to fetch data from the mobile data capture application. Figure 2.4 summarizes the process of supporting producer mobility.

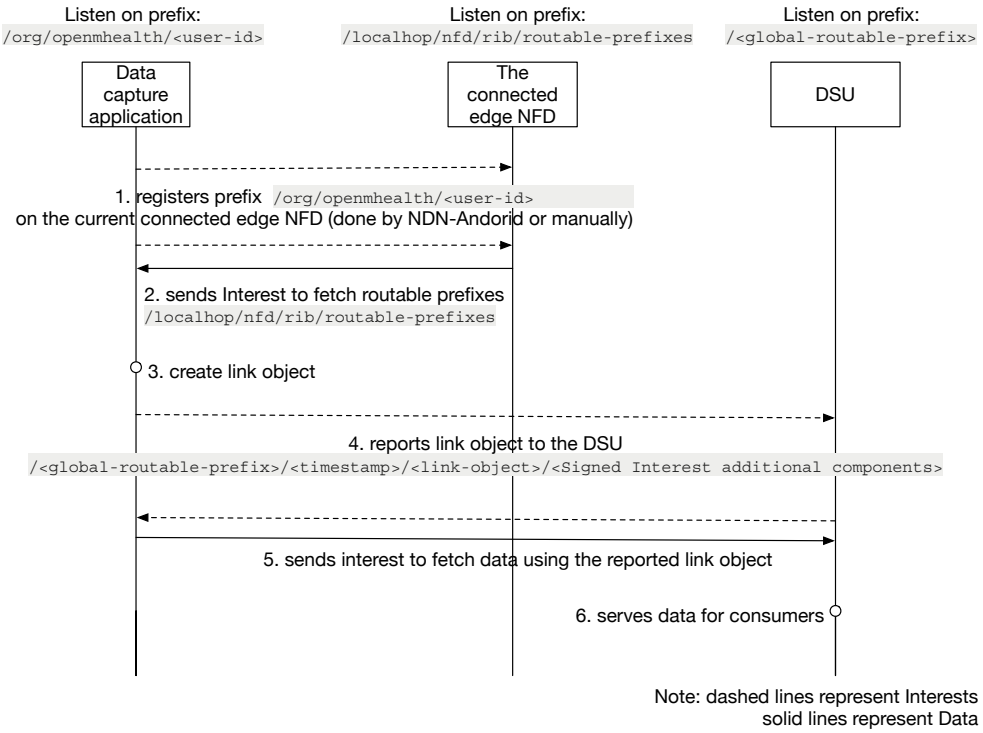


Figure 2.4: Producer mobility supported in NDNFit

DVU. We built example DVUs using NDN-JS (the NDN-CCL Javascript library) to visualize raw and processed data. DVUs are authorized by the end-user, who is the data owner, to access encrypted data using the name-based access control mechanism. After being authorized, they fetch, decrypt, and visualize data on an ongoing basis. DVUs can also trigger DPUs to generate processed data for display, "pulling" data through a distributed processing chain. Notably, in this year, we have demonstrated such chained processing in which a DVU requests processing of encrypted data by an authorized DPU. This meets our objective of designing and demonstrating the use of encrypted data throughout the NDNFit ecosystem.

An example DVU is shown in figure 2.5, which uses a DPU to calculate the bounding box around a walking path. The simple UI displays a log of Interest-Data exchange, the namespace in a tree representation, and a 2D visualization of both the raw data and the DPU-calculated bounding box.

NDNFit has been a challenging driver application to develop because it represents a complete end-to-end NDN application that uses: 1) location-independent data naming; 2) schematized trust with an app-specific root; 3) name-based access control for personal data stored in servers; 4) distributed data processing; and

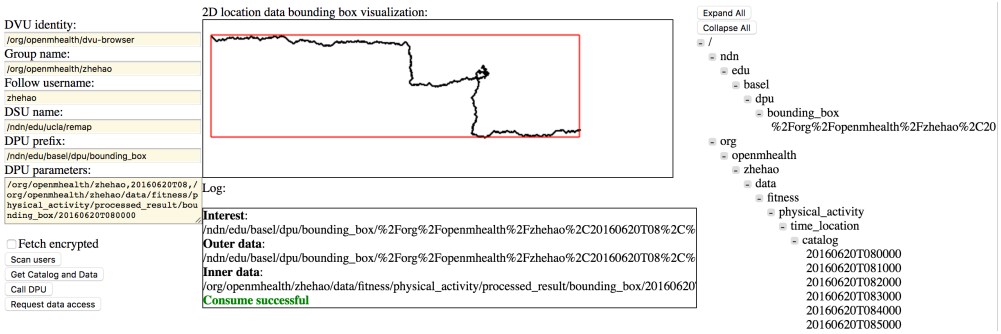


Figure 2.5: A DVU that requests and displaces the bounding box of a walking path, which is calculated by the DPU asynchronously.

5) publisher mobility support. Developing a modestly operational prototype of NDNfit is a significant achievement.

Open challenges remain in the best solutions for producer mobility in the NDN network infrastructure. An ideal solution is to make mobility transparent to applications, so that applications do not need to consider mobility as NDNFit did. Additionally, we need effective solutions to minimize information leakage in data names, especially for high privacy applications such as NDNFit; how to achieve the goal of name confidentiality, but without hurting routing performance/scalability, is an area that requires more work. Finally, the design tussle of data naming remains an open challenge to which NDNFit only provides one example solution of how to balance the requirements that applications, security, and the network place on data names.

2.1.3 Mobile Multimedia

Our two objectives for the supplement period were to finalize the NDN-RTC application as a usable videoconferencing tool and to develop an initial solution to just-in-time selection and navigation of multidimensional content, which is uniquely well-supported by NDN and a building block of next generation mobile augmented reality applications–a major interest of collaborators and partners. The latter objective is reflected in the proposal submitted to the NSF ICN-WEN program by some members of our team, and the solution sketch will be documented in [6]. The former is discussed below; we have nearly reached completion and expect to do so by the early summer 2017.

NDN-RTC and Flume

This past year, our work on the NDN-RTC library and *ndncon* application evolved into a project designing and implementing a new videoconferencing application, Flume, around the second generation of our library, which was updated to include schematized trust. The Flume application enables users to participate in conversation channels where they can publish group chat messages (text, images, files, etc.), as well as live media streams (video and/or audio captured from web cameras). Users can scrub through all content published in a channel, and playback historical data from earlier discussions. This work was supported through a combination of gift funds and NDN-NP effort.

The development of this application broadened our original objective to include two high-level goals: a) to demonstrate more efficient bandwidth usage over NDN compared to IP; and b) to target users with no special technical knowledge, in order to trigger wider adoption of NDN.

For the first goal, we focused on intermittent connectivity scenarios, where IP-based applications generally perform poorly. In order to support such use cases in NDN, we identified several challenges in collaboration with the architecture team:

- **Ad hoc connectivity.** To enable Flume among nearby devices, we implemented NDN over WiFi-Direct on Android. Support for other platforms is in progress.
- **Producer mobility.** To enable producer mobility, we implemented route re-advertisement and deployed it with the latest NFD release. This feature enables updates of RIB entries on the testbed when end hosts connect to non-home hubs, thus making them reachable by consumers' interests. While this solution is feasible on a small scale; we may need a different approach identified in [19] for larger scale scenarios.
- **Peer (re)publishing.** To provide a better end-user experience and resilient communication, channel participants should serve data they have already fetched from others, i.e, act as peer publishers. This challenge motivated the use of forwarding hints to route interests toward data mule peers, which may also provide a simpler solution to the producer mobility challenge.

For the second goal of easing the engagement of users with no special technical knowledge, we worked with the architecture team to develop an NDN Control Center (NDN-CC) which provides support for autonomous application identification and local connectivity establishment.

- **Application and instance identities.** The application requests identity signing from the NDN-CC,

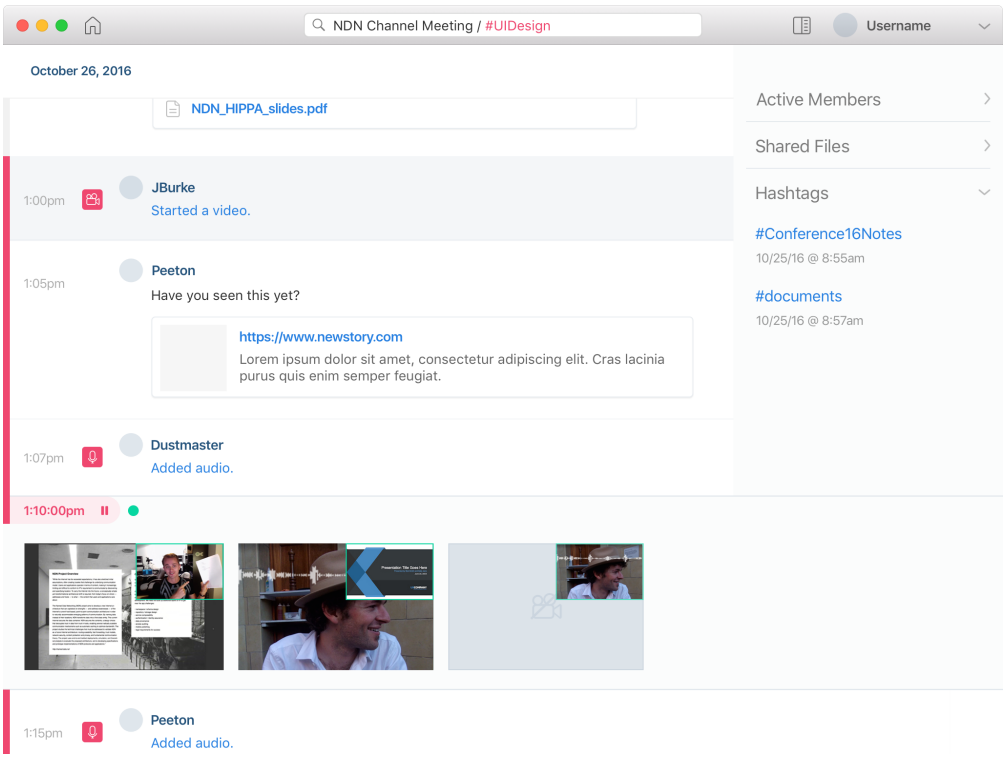


Figure 2.6: Flume application user interface design

which prompts the user with available options: use an existing identity or request a new one from the list of available CAs. The application then generates a long-lived identity, which is stored and served by the NDN-CC, and a short-lived in-memory instance identity, used for signing data (see Figure 2.7 and Section 3.1).

- **Automation of Local connectivity.** To support bootstrapping and ad-hoc scenarios, NDN-CC is responsible for establishing NDN connectivity with both the testbed and nearby peers.
- **Self-contained bundle design.** Both the application and NFD Control Center need to be designed as self-contained applications without requiring additional technical knowledge from users.

Development of the Flume application began in September 2016; we presented a prototype at the 6th NDN Retreat in November 2016. We completed a headless version of the underlying library and ported it to Linux for use by WUSTL and others at UCLA in testing in early July 2017; we expect the first version of the end-user facing application to be completed in August 2017. During the implementation, we have identified design patterns that can be useful for other applications. These concepts are being implemented as a shim library, which may eventually become part of NDN-CCL:

- **Generalized content fetching.** We designed a sub-namespace, which allows publishing arbitrary content and defines universal fetching rules for consumers (Figure 2.8).
- **Channel discovery.** In order to address the discovery problem in Flume, we designed a library module based on ChronoSync that allows for discovering arbitrary data (conversation channels in the case of Flume).
- **Persistent in-app storage.** To support the peer publishing capability of Flume requires in-app persistent data storage also capable of answering high-frequency interests for media data. The storage is designed as an augmentation of existing memory content cache; every incoming or outgoing interest is checked against the storage.

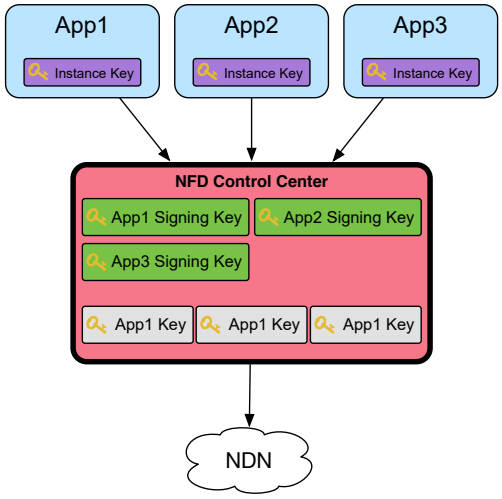


Figure 2.7: Application identity setup and NFD Control Center role in it.

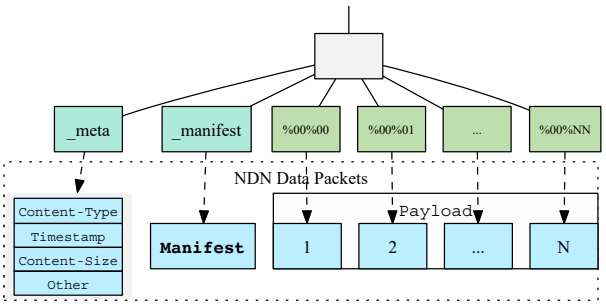


Figure 2.8: Generalized content namespace: The top box represents the last component of an application-level object prefix; the second row of green boxes shows standard naming for metadata, manifest, and segment data.

2.2 Applications

In addition to work in the network environments, the team continued work on a number of different applications, ranging from network support to analytics and file sharing.

2.2.1 Scientific Big Data

For the supplement, we proposed two tasks in this area: 1) Investigate the NDN-based data catalog's applicability to existing software widely used in climate and physics communities to distribute scientific data, such as ESGF and xrootd; and 2) Investigate integration of the catalog with layer 2 reservation systems such as OSCARS and potentially GENI. Since NDN can run directly over layer 2, integrating NDN with these reservation mechanisms can potentially leverage NDN routing and forwarding strategies.

This period we worked on the user interface of the catalog for ESGF data, specifically CMIP5 (available at `http://atmos-sac.es.net/catalog/`). The application supports search, selection and retrieval of scientific data. The data currently is limited to what we can store on the testbed, which is about 50TB. The entire CMIP5 dataset is over 3PB, so we cannot store it in our experimental testbed. The UI offers three different views of the dataset collection, whereas the original ESGF UI offers only one, a filter-based approach. The NDN UI offers a prefix and tree search capability. Moreover, for the datasets that exist on the testbed, the NDN UI can display the metadata and also allow a subset operation based on the metadata.

We also created a brief tutorial and a feedback form for the UI.[2] We had several users take the tutorial and provide feedback about the UI. We plan to incorporate that feedback into the system before releasing it to the broader community for alpha testing.

The next area we worked on was to integrate existing reservation systems such as OSCARS, a layer-2 system offered by ESnet, to NDN retrieval. We showed how NDN can improve bulk scientific data transfer for both best-effort and resource-reserved traffic using intelligent strategies. Using real access logs, we investigated access patterns in scientific data. We then proposed a network query protocol that was able to provide an estimate of how long a transfer might take and the best time to start retrieval. When best effort was not fast enough, a strategy can help to interact with lower layer protocols for automatically setting up reserved paths using OSCARS, a system for reserving guaranteed bandwidth-on-demand paths. Finally, using our strategy, we showed how much network bandwidth we can save while ensuring fast data transfers.

[2] `https://docs.google.com/document/d/1LTy156opP2A68OsuYOCAbXHilhwwPG_B_SakKd1QITg`

2.2.2 Namespace design for summarization

Work at UIUC in the current reporting period focused on exploiting information-centric networks in the age of data overload, produced by sensors, social media, and IoT devices. Specifically, the work explored a new type of network transport protocols that offers *representative summaries* of requested data, retrieved at a consumer-controlled degree of granularity. Given the over-abundance of data, consumers will seldom need all data on a topic, but rather will increasingly favor an appropriate sampling for summarization purposes. We explored such sampling as a novel service enabled by NDN. By naming data objects, it becomes possible to selectively retrieve them, but the properties of the resulting sampling depend on the naming scheme. We developed an automated object naming service, called Espresso, that facilitates content sampling over information-centric networks. We demonstrated how Espresso, combined with a trivial retrieval policy, translates the sampling problem into a naming problem, and customizes the naming to different applications sampling needs. Experimental results showed that the computational overhead of automated naming is affordable. The service was evaluated in simulation, demonstrating a higher sampled-data utility to the consumer, while balancing retrieved data importance and diversity. Social network applications were then introduced, where naming was produced by Espresso. Results demonstrated the advantages of Espresso, compared to baselines, in terms of retrieving meaningful media data summaries.

Specifically, we introduced a tweet-based newsfeed summary service running on a named data network (NDN) stack, called iApollo. This application offered a customized newsfeed to individual readers based on their interests. Data sampling was essential in iApollo because of the large volume of tweets. Espresso, the automatic naming agent, translated this sampling problem into a hierarchical namespace for the given set of tweets. It allowed readers to quickly achieve the best semantic understanding of the news topic with the minimal number of data retrievals. The application demonstrated not only a tweet-based newsfeed service, but also the synergy between Espresso and NDN.

2.2.3 NDNS

Our NDN design and development efforts over the last few years identified the need for a DNS-like lookup service in an NDN network to support three functions: (1) the secure namespace mapping component of our recently proposed solution to the challenge of NDN routing scalability [1]; (2) rendezvous for mobile data producers [19], i.e., to announce reachability of a given data prefix; and (23) persistent storage of critical data, such as public key certificates needed for data authentication in an NDN network [5]. We began developing NDNS (NDN DNS) a few years ago. NDNS is a distributed, always-on global distributed lookup service. The NDNS design leverages the experiences of successfully designing the Domain Name System (DNS) and its security extensions (DNSSEC) [2–4, 7, 8], but addresses fundamental differences in design requirements of NDN and IP-based applications. We have prototyped a few NDN implementations based on the evolving NDN codebase and security development. Over the last year we adapted our design to the new security framework.

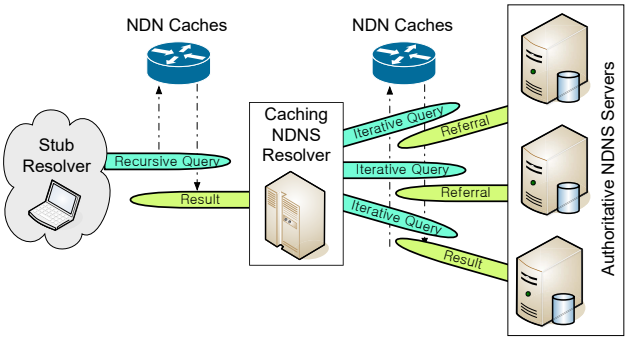


Figure 2.9: NDNS operation overview

Figure 2.9 shows an overview picture of NDNS operations, resembling DNS/DNSSEC operations but with several notable differences. The query process starts with stub resolvers sending recursive queries *toward* local NDNS caching resolver(s); we use "toward" instead of "to" here, because the query (an Interest packet) may trigger the requested answer from a router's cache (network level caching) before it reaches the local caching resolver. If the query does reach a caching resolver, the resolver either finds the answer from its internal caches (application level caching), or otherwise issues a set of top-down iterative queries to find the answer.

To facilitate information discovery, we have designed the following naming conventions for the iterative

query interests:

- a query Interest carrying a name starting with "`/NDNS/...`" prefix is requesting data from the root zone;
- a query Interest carrying a name starting with "`/net/NDNS/...`" prefix is requesting data from the "`/net`" zone;
- a query Interest carrying a name starting with "`/net/ndnsim/NDNS/...`" prefix means requesting data from the "`/net/ndnsim`" zone;

and so forth. With these conventions, the local and global bootstrap of NDNS infrastructure can be accomplished simply by announcing the NDNS recursive resolver service prefix "`/NDNS-R`" and the root zone prefix "`/NDNS`" into the local and global routing systems, respectively. In other words, the stub and caching resolvers do not need to be configured (or re-configured) with information about NDNS servers. All they need to do is generate query names by following proper naming conventions; the network can then take care of forwarding the query interests towards the right servers.

In cases where some TLD names (e.g., "`/net`") or second-level site names (e.g., "`/net/ndnsim`") are not "reachable" names (not announced to the routing system due to routing scalability constraints), then parent zones' referral responses must contain a delegation link object, so that a query interest for "`/net`" zone or "`/net/ndnsim`" zone can carry the delegation link object to be forwarded accordingly.

By running on top of NDN, NDNS takes advantage of all built-in NDN features, including (1) query interest forwarding based on server and network availability information at the network layer, avoiding resolvers having to select a specific name server for each query; (2) aggregation of same queries and multicasting query results; (3) in-network caching of query results; and (4) built-in authentication for query results carried in NDN data packets.

2.2.4 ChronoShare

During the past year, we have revamped our previous effort to enable distributed file sharing over NDN. This included the adaptation of the previously developed ChronoShare application to the new codebase and the latest version of the NDN protocol.

There are many ways to implement file sharing in NDN, with varying levels of design complexity and communication efficiency. The implemented design of ChronoShare uses an approach that treats individual user operations on files as streams of "actions," where each action specifies which file has been modified and what changes have been made, e.g., new file, updated content, changed file system permissions, or removed file (Figure 2.10). Actions are carried in NDN data packets, thus they are named and signed, automatically adding ownership information for each operation. By applying actions from all participants in a deterministic order, in combination with the conflict resolution process described, each ChronoShare user can build her 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 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 the ChronoSync primitive.

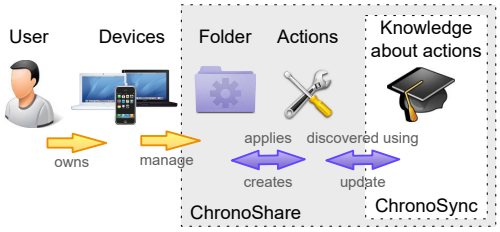


Figure 2.10: ChronoShare entities

We have built a working prototype of ChronoShare, and put it through trial usage. During the latest trial—at the NDN community meeting (NDNcomm 2017)—ChronoShare already helped us sync up files among nearby laptops utilizing just ad hoc WiFi connectivity. At the same time, ChronoShare is still in active research and development along several dimensions, including adding shared folder confidentiality and

enabling use of name-based access control.

2.2.5 nTorrent

Another direction of file sharing over NDN is exploration of existing peer-to-peer file sharing protocols. As a proof-of-concept, we have developed nTorrent, a BitTorrent-like NDN application, and extensively studied this application using the ndnSIM simulation framework. Our main goal was to study the impact of implementing the data-centric logic at different layers of the network architecture and to understand fundamental similarities and differences between application-layer data-centric protocol and applications developed on top of the data-centric networking architecture.

Through the nTorrent design process, we learned:

- TCP/IP imposes hurdles to data-centric applications, such as file-sharing, which have to implement efficient data retrieval logic on top of point-to-point connections by explicitly selecting locations from which to fetch data. NDN allows applications to focus exclusively on data fetching, leaving to the network the task of determining where to fetch data.
- NDN simplifies the design of data-centric applications, since the network is able to provide the desired functionalities of data fetching from the nearest locations, via multiple paths, quickly adapting to failures as well as data source changes, and obeying routing policies. However, these gains come at the cost of routing scalability challenges.
- Embedding data digests in NDN packet names enables in-network data integrity checking for applications exchanging static contents with minimal router processing overhead. This is distinct from BitTorrent, where bogus content is delivered to peers before being discarded, in nTorrent, bogus contentis dropped by the first router they cross.

2.2.6 ChronoChat-Android

Recently, we have created ChronoChat-Android, a port of the ChronoChat instant message application [13] for Android mobile devices. Similar to the existing ChronoChat application for desktop systems reported in previous years, ChronoChat-Android allows completely distributed communication in a chatroom, where participants can be connected to the NDN testbed or connected directly via Wi-Fi direct. During this development, we have created a new example of "Android-style" the NDN programming model on Android devices, as unlike desktop applications, Android applications can be stopped at any point of time, e.g., the application is completely recreated when user switches phone orientation.

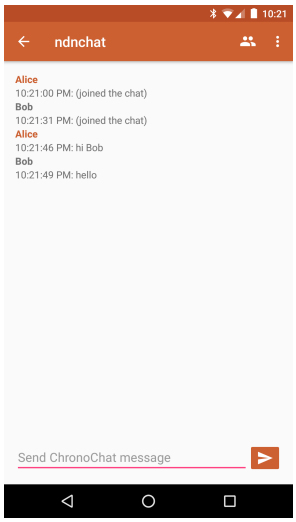


Figure 2.11: ChronoChat-Android

ChronoChat-android consists of two Android *Activities* and one *Service* as its primary components. Activities are the user-facing components, providing general coordination when application is launched. To handle network operations, ChronoChat-Android uses the *ChronoChatService* component, communicating with Activities through Intents, the interprocess communication mechanism on Android platform. When the Activity signals ChronoChatService to send a message, the service starts up, establishes a connection to the chatroom, and publishes the data via ChronoSync. Moreover, the ChronoChatService remains running as a *foreground service*—indicated by the ongoing notification shown in Fig. 4—to prevent the Android OS from killing the ChronoChat-Android process whenever the user is not interacting with the app. The service maintains its NDN connectivity relays any received messages to Activity via Intents, while also keeping track of the chatroom roster and periodically sending heartbeat messages to the chatroom. Activity may also send Intents to retrieve the current roster from the service, or

direct it to shut down.

The previous implementation examples, e.g., NDN Whiteboard reported last year, keep ownership of the "network thread" equivalent at the Activity. Because Activity objects are routinely destroyed by the Android OS, e.g., when switching an Android device from portrait to landscape orientation, reliable maintenance of the state at the Activity level is normally tedious and sometimes impossible. However, services can be configured to remain resident, and if run as a foreground service—as ChronoChatService is—the service should only be destroyed if the enclosing process is terminated by the system under extreme memory pressure from a foreground process. Moreover, previously used chains of so-called *AsyncTasks* to perform network-related initialization provide a useful mechanism to prevent blocking the UI thread, but re-creation of the activity would destroy the whole chain state and require complete re-initialization. Performing the same tasks sequentially in the service-provided network thread avoids these problems, ensuring also the serialization of tasks on a single thread, as required by jNDN library [14].

The current version of ChronoChat-Android serves as a proof-of-concept and is limited to a single chatroom at a time, lacks discovery of available chatrooms, and does not yet include proper security mechanisms. We would also like to improve the UI, including adding the ability to silence or disable new message notifications and the ability to save chatroom history to persistent storage.

This development exercise highlighted that the conventional development model is not directly applicable on mobile platforms, at least not in a robust form. Interaction between applications and local NFD on Android device is more efficient to accomplish through Intents, instead of managed persistent Face object provided by the jNDN library. An Intent-based "Android-style" asynchronous programming model simplifies development of NDN applications, which tends to increase their robustness. We also found a better solution to ensure aliveness of the NDN service: running it as a foreground service, which constraints how the service can be terminated by the OS, instead leaving this decision with the user.

2.3 Libraries

To support the applications and network environments, and to promote research and experimentation with NDN across a variety of application domains and platforms, the team continued its work on the core ndn-cxx and NDN-CCL libraries, as well as starting work on a few new efforts.

2.3.1 ndn-cxx: NDN C++ Library with eXperimental eXtensions

This past year, we have continued development of the NDN C++ library with eXperimental eXtensions (ndn-cxx), featuring one major and one minor release. The new development covers multiple areas, including features driven by the development of the NDN Forwarding Daemon, usability enhancement, and security:

- added new and updated data structures and operational interfaces for NFD management;
- enabled formatted output for management data structures;
- enabled validator to fetch certificates directly from the signed/command interest sender;
- added extended abilities to control local communication behavior;
- added new transformation API to simplify cryptographic operations and reduce library dependencies;
- introduced basic support of the refactored NDN Security toolkit (NDN security version 2).

We have also been addressing discovered bugs, incorrect or undefined behavior, and performance. The list of resolved issues includes ensuring consistency of the library and NFD state when network NACKs are returned, optimization of name comparisons, fixing memory leaks in NDN regular expression implementation, fixing SegmentFetcher helper class behavior, fixing support for the ImplicitSha256Digest name component in Exclude selectors, making the trust schema validator (ValidatorConfig) evaluate all checkers inside a rule, and others.

Some of the development features in the library received a significant boost from the NDN Hackathons. In particular, the universal logging feature that now powers the ndn-cxx library, NFD, NDN Essential Tools,

ChronoShare, and several other software projects originated as hackathon projects. Similarly, a recently released version of the NDN Control Center is a successful extension of a hackathon project.

2.3.2 NDN-CCL: Common Client Libraries

During the supplement period, work continued on the NDN Common Client Libraries (NDN-CCL), motivated by a variety of different applications listed at the end of this section. NDN-CCL provides a common application programming interface (API) across multiple languages for building applications that communicate using NDN. They incorporate features often first introduced in ndn-cxx. Currently, CCL is implemented in C++, Python, JavaScript, Java and C# .NET, with support for pure C and Squirrel.

Since May 2016, our library development has incorporated new features motivated by application development, new platforms, and improving performance. Many of the following additions were motivated by our goal of provide usable security libraries during the supplement period:

1. To allow developers to experiment with application-specific signature algorithms, added the flexible GenericSignature type, which complements standard SignatureInfo types.
2. To allow developers to experiment with application-specific Data packet types beyond the standard BLOB, KEY, etc., added support for any ContentType.
3. Generalized handling of name component types, including ImplicitSha256Digest.
4. To support application debugging, added reason strings for validation failure.
5. Added support for the new Protobuf Version 3.0, used by newer applications.
6. Adapted a lightweight RSA implementation for NDN-CPP Lite, for the constrained Arduino IoT devices, needed for key exchange in the Flow augmented reality application.
7. Working with a group using jNDN in an IoT application with intermittent connectivity, adapted the asynchronous transport to handle reconnection attempts.
8. Completed the C# .NET library, now passing all unit tests with the full CCL API.
9. A hackathon project adapted the NDN-JS in-browser Micro Forwarder to use LAN multicast and a content store so that one computer could serve as a bridge between computers on a local network and a single-user slow connection to the Internet.

NDN-Squirrel and FleetLink

Additionally, preliminary NDN-CCL support for the Squirrel language was developed, through a combination of support from a DOE SunShot award and NDN-NP effort. Squirrel is similar to JavaScript, used on IoT devices and as a scripting language for game engines. The Electric Imp IoT device is used by Operant Solar in their FleetLink neighborhood wireless solar device network, which chose NDN as the network protocol. To support this application, we developed the open source CCL library, NDN-Squirrel, including an adaptation of the JavaScript in-browser Micro Forwarder to forward packets among the IoT devices. NDN-Squirrel supports HMAC and RSA signatures, AES and RSA encryption, transport over a wireless radio via a serial port, and unit testing. The Micro Forwarder supports configurable faces and routing strategies with logging for debugging. Since the IoT devices have fixed physical locations, we plan to implement a geo forwarding strategy based on [15].

The NDN-Squirrel library is powering communications in the first generation commercial prototype shown in Figure 2.12, which includes both WiFi and LoRa radios.

Future plans for NDN-CCL under the NSF CRI award include:

1. Porting security library Version 2 from ndn-cxx, including certificate format 2.0.

Figure 2.12: Operant Solar first generation Fleetlink prototype, running NDN-CCL.

2. Integrating new autoconfiguration protocols, including discovering local forwarders.
3. Supporting namespace syncing as an alternative to Interest selectors. (See CNL below.)
4. Add Protobuf, ChronoSync and ECDSA signature support to the C# .NET library.
5. Use conditional compilation in Squirrel to reduce code size on a constrained device.

2.3.3 New & Experimental Libraries

SwiftNDN and NDN Support for iOS

SwiftNDN is an experimental client library developed for iOS and macOS platforms using Apple's latest Swift programming language. The main objectives of this work are to provide better support for developing NDN applications on Apple devices using the official programming language created by Apple, and easier integration with existing libraries (called *frameworks*) on those platforms, such as Cocoa and Cocoa Touch GUI.

SwiftNDN offers an application development interface similar to those in the ndn-cxx and NDN-CCL libraries. It also implements the NFD management protocol, allowing applications to register prefixes with NFD and retrieve the forwarder's status (e.g., list of Faces and FIB entries). It provides basic data signing and verification support using the native KeyChain service on iOS and macOS. It further integrates the consumer-producer API defined in [9] to provide different data fetching strategies for consumer applications. The library is currently used by the Infomax project to develop a demo app that runs on iPhones and iPads. The app is available as a free download from the App Store.[3]

[3] `https://itunes.apple.com/us/app/visual-tourism-on-ndn/id1199129737`

While developing the SwiftNDN library, we also explored the feasibility of porting the existing implementation of NFD to the iOS platform, so that apps on iPhones or iPads could talk to a local NDN forwarder running on the same device without having to connect to remote forwarders over TCP/UDP tunnels. While we managed to cross-compile NFD for iOS, the iOS platform prohibits any third-party app (using the standard API) from actively running in the background mode, which prevents the user from running an NDN app in the foreground talking to the local forwarder daemon in the background. Eventually, we decided not to pursue this direction, and shifted our focus on mobile platforms to the Android platform.

2.3.4 NDN-CNL: Common Name Library

Finally, the Common Name Library (CNL) is a new experimental API for NDN applications motivated by the desire to offer higher-level primitives than Interest-Data exchange to application developers, an area pursued in previous work such as [10] and reflected in the supplement's goal to disseminate NDN design patterns. Built on top of the lower-level Interest/Data exchange primitives of the Common Client Libraries (CCL), CNL maintains an in-memory abstraction of the application's namespace. The application can attach specialized handlers to nodes of the CNL namespace object, e.g., to treat part of the name tree as segmented content, or to do data encryption/decryption. CNL can also alert the application when new names are added to the namespace or when content is attached to a namespace node, whether by receiving a Data packet from the network, retrieving from a repo, transforming a packet, or combining multiple packets. Development has included the following:

1. While CCL has a simple utility to assemble segments, one API call can attach a CNL segment stream handler to a namespace node to process segments automatically, using pre-fetching, out-of-order reassembly, stream mode and progress updates.

2. A call can attach a CNL handler to a namespace node to decrypt packets in that namespace automatically, using Name-based Access Control [17]. This can be combined, for example, with the segment handler, to decrypt, automatically, segments being processed.

3. A hackathon project developed a handler that, attached to a namespace node, uses Sync to sync that namespace automatically. When a name is added to one users's CNL hamespace, the namespace of other users is updated, alerting the application. This discovery method can be an alternative to "name probing" with Interest selectors.

4. We plan a CNL handler to verify signatures of packets in a namespace automatically, using schematized trust [16].

5. Currently, there are implementations of CNL in C++ and Python. We plan to add Java, JavaScript and C# .NET.

References

[1] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. SNAMP: Secure namespace mapping to scale NDN forwarding. In *Proceedings of 18th IEEE Global Internet Symposium*, 2015.

[2] Roy Arends, Rob Austein, M Larson, D Massey, and Scott Rose. Protocol modifications for the dns security extensions. Technical report, RFC 4035, March, 2005.

[3] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Dns security introduction and requirements. Technical report, RFC 4033, March, 2005.

[4] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Resource records for the dns security extensions. Technical report, RFC 4034, March, 2005.

[5] Chaoyi Bian, Zhenkai Zhu, Alexander Afanasyev, Ersin Uzun, and Lixia Zhang. Deploying key management on NDN testbed. Technical Report NDN-0009, Revision 2, NDN, February 2013.

[6] Jeff Burke. Opportunities in augmented reality over named data networking (ndn), *Invited Paper*. In *The 26th International Conference on Computer Communications and Networks (ICCCN 2017)*. IEEE, 2017.

[7] P. Mockapetris. Domain names: concepts and facilities. RFC 1034, 1987.

[8] Paul Mockapetris. Domain names, implementation and specification, november 1987. RFC 1035, 1987.

[9] Ilya Moiseenko and Lixia Zhang. Consumer-producer api for named data networking. In *Proc. 1st ACM Conference on Information-Centric Networking (ICN-2014)*, Paris, France, September 2014.

[10] Ilya Moiseenko and Lixia Zhang. Consumer-producer api for named data networking. Technical Report NDN-0017, Revision 1, NDN, February 2014.

[11] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang. Named Data Networking of Things (Invited Paper). In *Proceedings of the 1st IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 117–128, Apr. 2016.

[12] Wentao Shang, Zhehao Wang, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. Breaking out of the cloud: Local trust management and rendezvous in named data networking of things. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, pages 3–13, New York, NY, USA, 2017. ACM.

[13] Tyler Vernon Smith, Alexander Afanasyev, and Lixia Zhang. ChronoChat on Android. Technical Report NDN-0059, Revision 1, NDN, April 2017.

[14] Jeff Thompson and Jeff Burke. Ndn common client libraries. Technical Report NDN-0024, Revision 1, NDN, September 2014.

[15] L. Wang, A. Afanasyev, R. Kuntz, R. Vuyyuru, R. Wakikawa, and L. Zhang. Rapid traffic information dissemination using named data. *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications (NoM 12)*, page 712, 2012.

[16] Yingdi Yu, Alexander Afanasyev, David Clark, kc claffy, Van Jacobson, and Lixia Zhang. Schematizing Trust in Named Data Networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, September 2015.

[17] Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. Name-based access control. Technical Report NDN-0034, Revision 2, NDN, January 2016.

[18] Haitao Zhang, Zhehao Wang, Christopher Scherb, Claudio Marxer, Jeff Burke, and Lixia Zhang. Sharing mhealth data via named data networking. In *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*, pages 142–147. ACM, 2016.

[19] Yu Zhang, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. A survey of mobility support in named data networking. In *Proceedings of the third Workshop on Name-Oriented Mobility (NOM 2016)*, 2016.

[20] Zhenkai Zhu and Alexander Afanasyev. Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking. In *IEEE ICNP*, 2013.

Chapter 3

Security

Contributors	
PIs	J. Alex Halderman (UMich), Alexander Afanasyev, Jeff Burke, Van Jacobson, Lixia Zhang (UCLA)
Grad Students ..	Zhiyi Zhang, Manika Mittal (UCLA)

Security remains as one of the most challenging areas in NDN research, as NDN's mandate of securing every Data packet departs in fundamental ways from today's practice of securing end-to-end connections. Over the last year our focus has been on addressing two specific issues: certificate management, and ease of data verification through certificate bundling.

3.1 NDN Certificate Management Protocol (NDNCERT)

In Named Data Networking (NDN), every entity must have an identity (namespace) and the corresponding certificate for this namespace. Moreover, to support granular security management, each entity also needs simple mechanisms to manage its sub-identities and their certificates. To that end, we have developed the NDN Certificate Management protocol (NDNCERT) [4], an adaptation of the Automatic Certificate Management Engine protocol [1] originally designed for Let's Encrypt Internet Certificate Authority.

NDNCERT is specifically designed to foster simple yet secure NDN certificate issuance and management on the NDN testbed, client machines connected to the testbed, and any other computers that speak NDN. In particular, NDNCERT provides flexible mechanisms to establish certificate authorities (CA) for different NDN namespaces (e.g., NDN Testbed CA for certificates in "`/ndn`" namespace, NDN OpenMHealth authority for "`/org/openmhealth`" namespace, and a few others), and to request certificates from the established authorities. Note that the NDNCERT protocol does not impose any specific trust model or trust anchors; instead we let individual applications make their choices to best fit their own needs. We aim to understand the commonalities and differences across application security requirements, and then to extract patterns to put into standard libraries while retaining as much flexbility as possible.

Our NDNCERT design allows any node to become a certificate authority for a namespace, which is either delegated to this node by a higher-level CA (e.g. "`/ndn`" ⇒ "`/ndn/edu/ucla`") or self-claimed (self-signed trust anchor), e.g., when being used in local environments such as a smart home. *Hierarchical naming structure* and well established *naming conventions* are the main contributors to simplicity in NDN certificate management. To request a certificate from the NDN Testbed CA, one sends an Interest packet that starts with "`/ndn/CA`"; for a certificate from UCLA's node of the NDN Testbed, "`/ndn/edu/ucla/CA`", etc. Similarly, to become a CA for "`/<prefix>`", a node N registers "`/<prefix>/CA`" with its local NFD. For N to become a *recognized* CA, its own certificate must become trusted by other parties in the network. Such trust may derive from a higher-level authority that issued the certificate, or by others' endorsements of N's self-signed

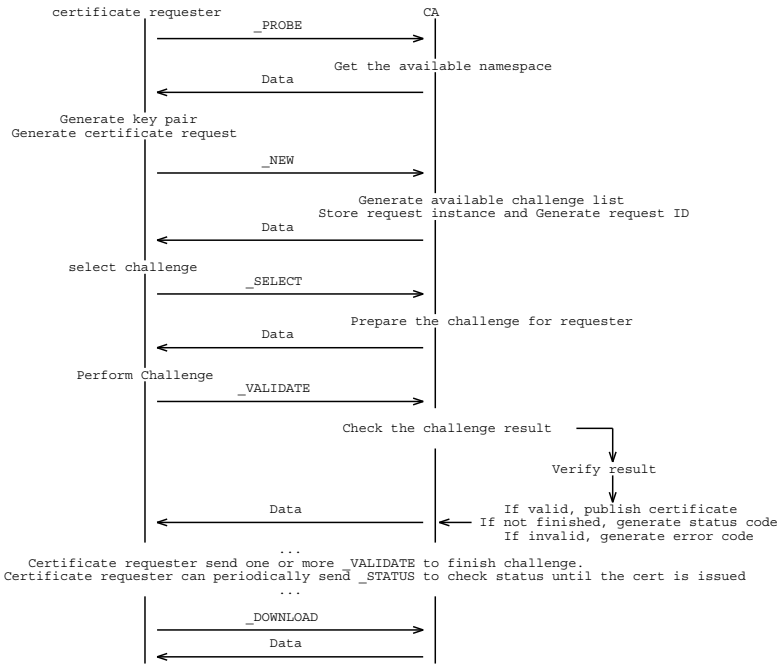


Figure 3.1: NDNCERT Protocol Overview

certificate.

Figure 3.1 illustrates the interactions between certificate requesters and CAs, including five steps by the requester:

1. discover available sub-namespace ("`_PROBE`"),
2. apply for certificate for selected (assigned) namespace ("`_NEW`"),
3. select an out-of-band challenge to prove legitimacy/ownerships of the namespace,
4. satisfy the selected challenge, and
5. download the issued certificate.

Note that the security properties of the NDN certificates obtained using the NDNCERT protocol come from the out-of-band validation mechanisms, through so-called *authentication challenges*. With these challenges, a requester proves to a CA, and others who trust the CA's judgement, that it is a legitimate party to request certificate in a given namespace. One supported challenge is proof of control of an email account. This challenge provides a way for the CA to ensure that the requester possesses the email address it provided with its certificate request, and this email address will be directly tied to the NDN certificate namespace ("`/ndn/edu/ucla/<email-address>/...`"). We have also implemented a shared secret challenge, which assumes execution of the email-based challenge and control of the namespace assignment, perhaps via direct agreement between requester and CA.

Future Work We are looking into several aspects of advancing the NDNCERT system. First, we are investigating other types of authentication challenges. Constrained IoT controllers usually have no user interfaces and require special handling in identity assignment, e.g., leveraging physical possessions of the devices, or physical proximities through light/audio sensors, accelerometers, etc.

Second, up to now we have excluded certificate revocation as part of the NDNCERT design. Our automated mechanisms issue short-lived certificates, where certificate revocation eeffectively occurs by blocking the re-issuing of certificates without further verification. In other words, our protocol assumes two separate timelines: a timespan during which the CA assumes the validity of the out-of-band challenge (for email-based certificates this can be a year) and another timespan for a certificate instance (on the order of days or weeks). While the previous challenge remains valid, the CA is allowed to re-issue a certificate with an

updated validity period without going through the challenge process. If there is any indication of a problematic certificate, NDNCERT will repeat authentication validations. The effectiveness and the cost of this design requires validation through a wide deployment.

Third, we plan to integrate NDNCERT into our trust schemas [3] to automatically, when needed, generate public/private keys and request certificates to sign data packets. In other words, when the trust schema receives a Data packet for signing, the trust schema engine will automatically determine the name of the key to sign (certificate to validate); if such a key is not available, the engine will determine the proper CA for the namespace and initiate a request for the certificate using NDNCERT protocol.

3.2 Certificate Bundling

In order to authenticate a received NDN Data packet, the consumer needs to verify the signature against the trust model (trust schema). Unless the data packet is directly signed by a known trust anchor key, the authenticator will need to retrieve additional certificates, as indicated by the "`KeyLocator`" field in the data packet(s). While conceptually simple, solely relying on using "`KeyLocator`" to fetch missing certificates poses several challenges, including (1) the delay from multiple non-parallelizable data retrievals and (2) the need to ensure availability of all certificates along the trust chain.

Over the last year, we designed and implemented NDN Certificate Bundling [2] to address these challenges (Figure 3.2). The idea is to let data publisher publish all certificates needed to authenticate the published Data packet in a bundle, which has been an established practice in serving TLS certificates. This way, instead of retrieving every single certificate in the trust chain one by one, one can retrieve the whole chain as a bundle (which may be contained in one or multiple Data packets). Bundling improves certificate availability as the bundle can be published/provisioned together with the data packets they authenticate.

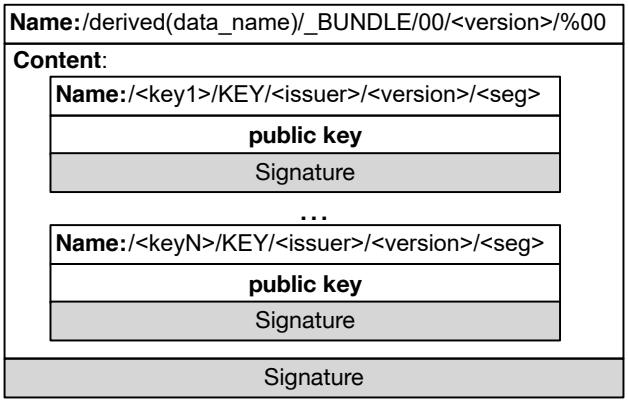


Figure 3.2: NDN Certificate Bundle

Naming Conventions Although the basic concept of certificate bundling is not new, the design of NDN certificate bundle raises a unique question: how to name a key bundle. There are three design considerations. First, in order to enable a consuming application A_c to automatically retrieve the certificate bundle when it starts retrieving data, A_c must be able to *construct* the name for retrieving the bundle. Second, to facilitate storage and retrieval of certificates (i.e., to ensure that certificates can be stored in the same cache/managed storage as the data and can be retrieved along the same routes), the bundle's name should share the same prefix with the corresponding data. Third, we need to be able to clearly distinguish data and certificate bundles under the same prefix.

We have defined the following naming convention for the certificate bundle: "`/<derived(data_name)>/_BUNDLE/<trust-model>/<version>/<seg>`", where "`derived(data_name)`" is a functional derivation of the prefix based on the name of the data, and "`_BUNDLE`" is a designated marker for certificate bundles. Currently, we have two defined rules for data names, names that end with a segment number and those that do not. In the first case, we assume that different segments will be signed by the same key, therefore "`derived(data_name) = data_name.getPrefix(-1)`". The second case is currently the basic case, which does not make any assumptions about datasets and their signing keys and simply uses "`derived(data_name) = data_name`".

As a next step, we plan to define more rules for bundle name derivations, including ones based on explicit definitions in the extended version of the trust schema. Another issue to address is the support for multiple trust chains in data verification. Up to now we have defined only a single convention for the "`<trust-model>`"

component: "00" to represent a trust model with a single certificate chain. We plan to define additional conventions over time as we gain further experience with certificate bundle usage.

References

[1] Richard Barnes, Peter Eckersley, Seth Schoen, J. Alex Halderman, and James Kasten. Automatic Certificate Management Environment (ACME). `https://github.com/letsencrypt/acme-spec`, September 2014.

[2] Manika Mittal, Alexander Afanasyev, and Lixia Zhang. NDN certificate bundle (version 0.1). Technical Report NDN-0054, NDN, March 2017.

[3] Yingdi Yu, Alexander Afanasyev, David Clark, kc claffy, Van Jacobson, and Lixia Zhang. Schematizing Trust in Named Data Networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, September 2015.

[4] Zhiyi Zhang, Yingdi Yu, Alex Afanasyev, and Lixia Zhang. NDN certificate management protocol (NDNCERT). Technical Report NDN-0054, NDN, April 2017.

Chapter 4

Distributed Dataset Synchronization

Contributors	
PIs	Alex Afanasyev, Jeff Burke, Van Jacobson & Lixia Zhang (UCLA), Lan Wang (U. Memphis), Patrick Crowley (Washington University)
Grad Students ..	Spyridon Mastorakis, Wentao Shang (UCLA), Minsheng Zhang (U. Memphis), Hila Ben Abraham (Washington University)
Staff	Ashlesh Gawande, Vince Lehman (U. Memphis)

As reported in our first NDN project annual report back in 2011 (See [6], Section 2), one of our early discoveries from the NDN effort is the recognition of a common need among distributed applications: synchronization of shared dataset, or Sync in short. Sync provides a layer of abstraction to support distributed applications on top of NDN's Interest-Data exchange primitives. All collaborating entities (called *nodes*) in an application instance join a *Sync group* to operate over a shared dataset. When any node in the sync group adds or removes data objects from the dataset, those changes are propagated to all other nodes via the Sync protocol to update their local views of the shared dataset accordingly. Sync is viewed as playing a *transport layer* role in the NDN architecture because it bridges the gap between the network layer best effort data-fetching semantics and the application layer distributed synchronization demands, similar to how TCP bridges the gap between IP's point-to-point datagram delivery and application demands for reliable delivery.

Over the last six years we have explored a number of different design approaches to Sync support.

- The first two years of NDN research used CCNx 0.X as the codebase, and we designed and experimented with CCNx 0.X Sync.
- We developed *iSync* [14], which follows the basic design assumptions as CCNx 0.X Sync but utilizes the Invertible Bloom filter (IBF), instead of a tree of hashes, to reconcile data collections among multiple distributed nodes.
- We also developed *ChronoSync* [17] whose design departs from CCNx Sync and iSync in a fundamental way: it adopts a specific data naming convention to simplify the overall design. ChronoSync was initially designed to support a Multi-User Chat application; over time it has evolved to a general library utility used by other NDN applications including NLSR [8], ChronoShare [1], and scientific data management applications [5, 10].
- We further developed *RoundSync* as a revision to the ChronoSync design to address its semantic overload problem identified through extensive simulation studies.
- Recognizing the need for supporting pub-sub applications, during the last reporting period we started the development of *PartialSync (pSync)* [9]. The pSync design adopts the naming convention from ChronoSync and the use of IBF from iSync, and enables individual consumers to synchronize a *subset* of the dataset that may be published by distributed producers.

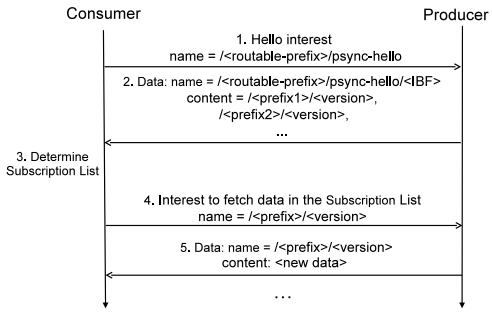


Figure 4.1: Initialization Phase in pSync

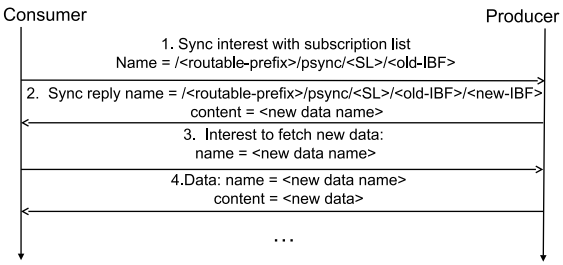


Figure 4.2: Sync Phase in pSync

Over this reporting period, we moved NDN Sync research forwarding on three fronts. First, we completed the design and evaluation of pSync. Second, we conducted a thorough comparative study of the above different designs to summarize their commonalities, differences, and design tradeoffs. Third, building upon what we have learned from the existing work, we started the development of a new Sync protocol, dubbed VectorSync [12], that provides built-in group membership management and enables useful services such as distributed dataset snapshot and data total ordering.

In the rest of this chapter, we first report our results on the design and evaluation of pSync, then summarizez our comparison across all of our previous sync implementations, followed by a report on the initial design of VectorSync (the topic of a PhD dissertation to be finished in summer 2017).

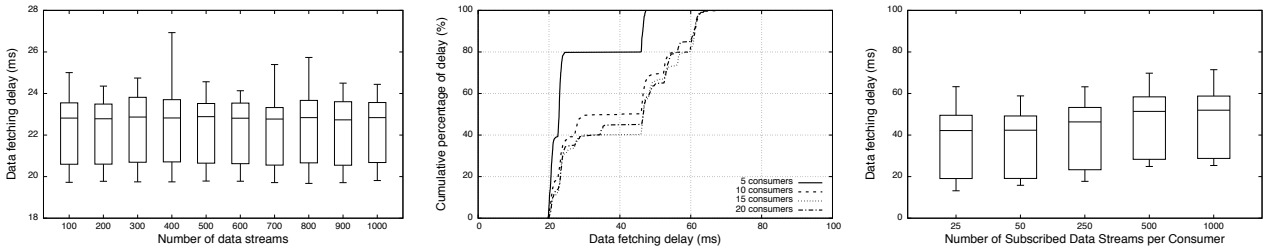
4.1 pSync

The need for pSync [16] arises from the requirement for scalable and efficient partial synchronization in NDN. Some applications such as multi-user chat require full-data set synchronization. On the other hand we have pub-sub applications where a consumer is only interested in a subset of the data set produced by the producer. pSync can support both types of synchronizations. It is designed with scalability, efficiency, and robustness as guiding principles: (a) To achieve *scalability under large number of consumers*, producers do not keep state of each consumer. The interest sent by a consumer contains information about a consumer's subscription list and previously received producer state. (b) *Scalability under a large number of subscriptions* is supported by having efficient data representations such as using a Bloom Filter and ranges to encode consumers' subscriptions to only one interest message. (c) *Robustness to producer failure* is built in as producers do not store the state of the any consumer. Other producers replicating the failed producers can serve consumers. In the past year, we have refined the pSync design, e.g., adding the Initialization Phase and support for multi-party full sync. We also performed more comprehensive evaluation.

4.1.1 Design

There are two phases in pSync. In the *Initialization Phase*, a consumer needs to know what data streams to subscribe to and also get the producer's latest IBF [4] which encodes the producer's state. Assuming the producer is reachable via the name prefix */routable-prefix*, the consumer first sends a Hello Interest to the producer using the name */routable-prefix/pSync-hello*, as shown in Figure 4.1. Upon receiving this Hello Interest, the producer will send a Hello Reply with its IBF as the last component in the name and the set of latest data names (one per data stream) in the content. Based on the Hello reply, the consumer then chooses the data streams to subscribe to, retrieves their latest data items, and enters the Sync Phase.

After receiving the producer's state information, the consumer enters the *Sync Phase* in which it subscribes to some (or all) of the data streams and receives notifications from the producer by sending a *Sync Interest* 4.2. The producer sends a Sync Reply if its IBF is different from the one contained in the consumer's interest and the new names are in the consumer's subscription list. No special case is needed for handling of simultaneous data production by multiple producers. Consumers and producers keep synchronizing until all



(a) Different Number of Data Streams per Producer (b) Different Number of Consumers (c) Different Number of Subscribed Data Streams per Consumer

Figure 4.3: Data Fetching Delay in Partial-Data Synchronization

consumers have obtained all of the new data from different producers. Full-data set synchronization can be achieved by having a consumer and producer on each participant where each participant registers the same sync prefix, subscribes to the entire data set, and sends its sync interests under a multicast strategy.

4.1.2 Implementation and Evaluation

We implemented the proposed pSync protocol in C++ using the ndn-cxx library to ensure compatibility with the NDN Forwarding Daemon (NFD), both the initialization and sync phases.

We performed our evaluation in Mini-NDN. We focused on the data fetching delay, i.e., the time from when a data item is produced to when the data is obtained by a consumer. We used the Sprint point of presence topology [13] with 52 nodes and 94 links to evaluate the performance of pSync. We first evaluated the performance of partial synchronization. We selected one node as a producer serving many data streams, each generating data at a random interval between 1 and 5 minutes. Each consumer subscribed to a random set of 100 data streams.

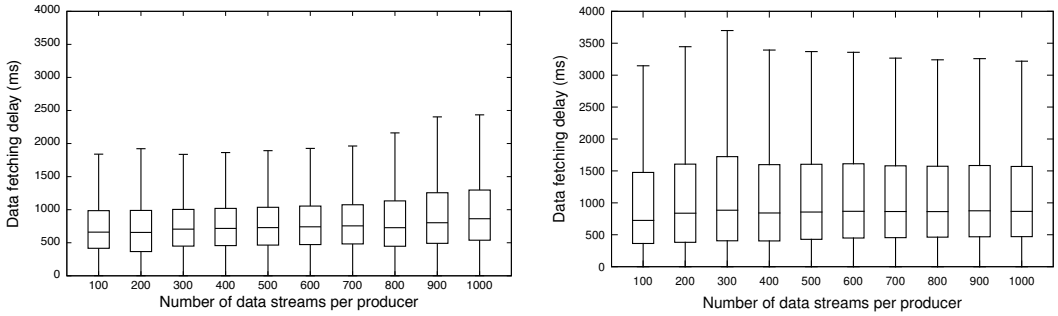
In the first experiment, we randomly selected 5 nodes as consumers and varied the number of data streams served by the producer from 100 to 1000. Figure 4.3a shows that the median data fetching delay remained around 23ms regardless of the number of data streams. In the second experiment, we fixed the number of data streams to be 1000 and randomly chose 5, 10, 15, and 20 nodes as consumers. Figure 4.3b shows that the data fetching delay increased slightly as the number of consumers increases, but the maximum delay was below 100ms and median delay was below 50ms in all the runs. Finally, we randomly chose 20 nodes as consumers and vared the subscription size per consumer from 25 to 1000 data streams. Figure 4.3c shows that the data fetching delay changed little even though the subscription size increased by a factor of 40.

We compared pSync and ChronoSync performance in supporting multi-party full-data synchronization. All 52 nodes synchronized with each other and we varied the number of data streams from 100 to 1000 with each data stream generating data at a random interval between 1 and 5 minutes. Figure 4.4 shows that pSync achieved lower data fetching delay than ChronoSync which is specifically designed for full-data synchronization.

In the future, we hope to reduce the Sync Interest/Reply message size as well as evaluating the protocol using real applications. The initial implementation leveraged some NFD hacks such as the no cache policy for the Hello data. We want to eliminate such hacks, clean up the implementation, and provide better logging. We aim to release a library similar to ChronoSync that other programs can easily use.

4.2 Comparative Evaluation of Sync Protocols

In this section we summarize the lessons and insight from the exploration of different approaches to Sync design. Our goal is to extract common design patterns and identify different design choices and tradeoffs made in different protocols.



(a) pSync (b) ChronoSync

Figure 4.4: Data Fetching Delay in Full-Data Synchronization

By examining all the existing Sync protocols, we identified the following key design questions.

Data naming Thanks to the unique binding between names and immutable data objects in NDN, a shared dataset can be uniquely identified by the namespace containing the names of all data objects in the dataset. Therefore NDN converts the dataset synchronization problem to the synchronization of the corresponding namespace. If the data packets published to the shared dataset are named under the topological prefix of their publishers, then the Interests for the data can be forwarded to the data producers in a straightforward way.

Besides the namespace of the shared dataset, all the sync protocols also require another namespace for group communications. This group namespace is used for the sync nodes to exchange protocol messages regarding the state of, and the updates to, the shared dataset. It is typically a multicast prefix, so that an Interest carrying names under this prefix is forwarded to the entire sync group.

Namespace representation The data structure that represents the state of the shared dataset namespace is referred to as the *sync state*. Every sync node maintains a local sync state and uses the sync protocol to keep this local state synchronized with the changes generated by other nodes in the sync group. The sync state must encode the namespace accurately with no loss of information and facilitate the reconciliation of any differences between distinct states.

State sync mechanism Each node participating in a sync group may publish new data into the shared dataset at any time. The sync protocol should ensure the other nodes in the group can receive the new data and reach agreement on the state of the dataset. To keep all the nodes in a sync group synchronized, whenever a node publishes new data, it needs to *notify* the rest of the group about the sync state changes. Given any packet may fail to reach all intended recipients, one also needs periodic exchanges among nodes that detect inconsistent sync states.

In the rest of this section, we give a high-level overview of each sync protocol by focusing on their design choices on the above three design aspects.

4.2.1 CCNx Sync

The CCNx Sync protocol [2] is the earliest synchronization solution proposed as a service module of the *ccnr* repo daemon, enabling a group of repos to synchronize a shared dataset that contains data with *arbitrary names* under a common *collection prefix*. The sync state is represented as a *sync tree* as shown in Fig. 4.5. The structure of the sync tree is determined by the order in which the data names are added to the shared dataset, which is independent from the canonical ordering of the data names. Each node in the sync tree is associated with a hash value: the value of the leaf node is simply the hash of the data name represented by that node; the value of the non-leaf node is recursively computed as the sum of the hashes of all its children nodes. The root hash (H_0 in Fig. 4.5) provides a summary of the entire namespace (i.e., the sum of all data name hashes).

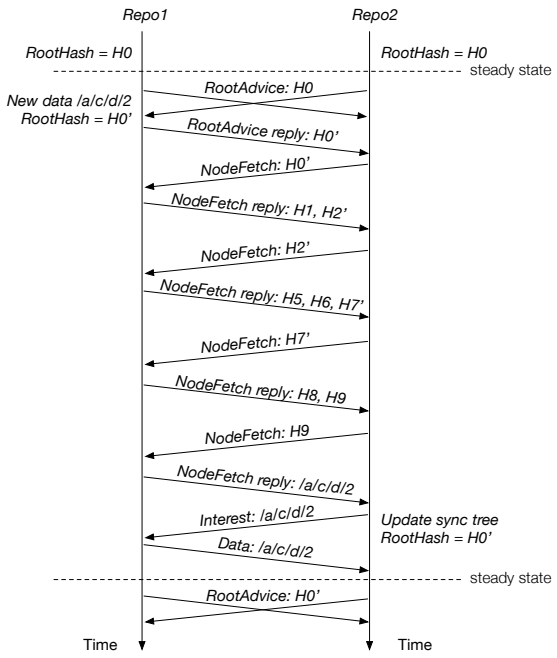


Figure 4.6: Synchronization process in CCNx sync

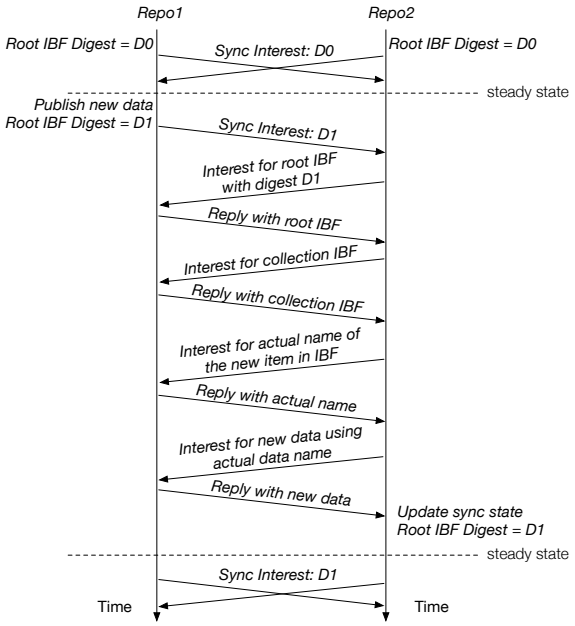


Figure 4.7: Synchronization process in iSync

New data can be published into any repo at any time. The sync module in the repo daemon (called *sync agent*) keeps track of insertions of new data and updates the sync tree accordingly, adjusting the hash values along the path from the new leaf node to the root. For example, in Figure 4.5 the insertion of a new data "`/a/c/d/2`" (marked as the red dashed square at the bottom right) will cause the sync agent to update the node hashes H7 and H2, eventually propagating the change up to the root hash H0.

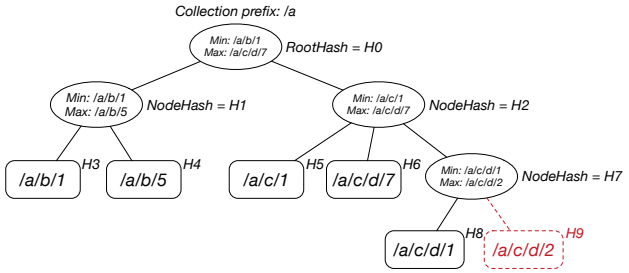


Figure 4.5: Example of a sync tree in CCNx Sync

The sync agent at each node periodically advertises the latest root hash by sending a *RootAdvice* Interest to all the other repos in the sync group. The RootAdvice Interest name starts with a *multicast prefix* shared by every repo in the sync group, followed by the current root hash of the sync tree. When node $N2$ receives a root hash from node $N1$ which is identical to its own, it simply drops it silently; otherwise it replies with an NDN data packet containing its own root hash. Any node which receives the reply from $N2$ will send a *NodeFetch* Interest to $N2$ to retrieve the list of hashes for all the direct children under the root node of $N2$. The NodeFetch process recursively goes through all the nodes in the sync tree, skipping those with the same hash value, until all the leaf nodes that caused the difference in root hash have been identified. The sync agent can then fetch those data from $N2$ via normal Interest-Data exchange and insert new data to its local copy of the shared dataset. An example of the synchronization process in CCNx Sync is illustrated in Fig. 4.6. The number of round-trips to fetch an update is directly proportional to the depth of the naming tree.

When multiple repos in the sync group publish new data simultaneously, there will be more than one reply to a RootAdvice Interest sent by a node $N1$, but only one of them will be returned to $N1$ (one Interest retrieves one Data packet only). Thus $N1$ needs to issue additional Interests to fetch all of the replies. After $N1$ learns multiple new root hashes, it starts the reconciliation process (as we described above) independently with remote repos who have published new data.

Another issue due to the CCNx Sync algorithm, which compares the local and remote sync trees and updates the local state to be the union of the two, is that no data can be deleted from the shared dataset, because the algorithm cannot distinguish the case where a repo intentionally removed a piece of received data from the case where the repo has never received the data before (Section 4.3) describes how VectorSync addresses this gap.)

4.2.2 iSync

iSync [15] is a direct optimization of the CCNx Sync design. Like CCNx Sync, it supports the synchronization of shared data with *arbitrary names*. To represent the *sync state* more efficiently, iSync uses Invertible Bloom Filter (IBF) [4] to store all names in the shared dataset in compressed form. Since the IBF can only store fixed-length items, the data names must be first converted to fixed-length IDs (generated from the hash of the names) before they are added to the IBF. For scalability consideration iSync maintains a 2-level IBF structure. It also maintains a bi-directional mapping table to convert original data names to IDs and vice-versa.

iSync uses "digest broadcast" Interests (equivalent to the RootAdvise Interest in CCNx) as a *notification mechanism* for a node to advertise its current state to other nodes. The notification Interest carries the digest of the current IBF of the sending node. To get around the issue of multiple replies triggered by the same broadcast/multicast notification Interest that CCNx faced, this notification Interest does not expect any reply. Instead, whichever nodes detected difference in digest will send Interest to retrieve the IBF. When a node $N2$ receives a digest sent by node $N1$ that differs from its own, $N2$ sends another Interest to request the corresponding root IBF content. After it receives the root IBF, $N2$ subtracts its own root IBF from the remote IBF and extracts individual IDs from the resulting "diff" IBF to identify the second level IBFs with different digests from $N1$. $N2$ repeats the process to find the IDs for new data. Once $N2$ extracts all the new IDs, it issues Interests to request the original data names corresponding to those IDs, so that it can fetch the new data using those names.

Fig. 4.7 illustrates iSync's synchronization process. With a 2-level IBF structure, iSync takes 3.5 round trips to learn the new data name starting from the first digest broadcast Interest notification. This delay can be significantly shorter than that of CCNx Sync, the delay of which depends on the depth of the data name hierarchy. However the IBF data structure can only losslessly encode up to a certain number of items, beyond which some of the stored items cannot be extracted. iSync provides several ways to control the size of the set difference at multiple levels in the protocol design, as described in [15].

4.2.3 ChronoSync

Different from CCNx Sync and iSync which synchronize datasets made of arbitrary data names, ChronoSync [17] utilizes naming conventions to simplify the sync protocol design. ChronoSync assumes that each node in a sync group has a topologically meaningful name, and publishes data with a name pattern of data names followed by a sequence number.[1] The data name is the concatenation of node name with application name, and the sequence number starts from zero and increments for each new data published by the sync node.

Each sync node maintains a 2-level flat sync tree (Fig. 4.8), with each leaf (immediately under the root) containing the data prefix and the latest sequence number of each member in the sync group. The root of the tree contains the digest of all the information in leaves, summarizing the current state of the sync tree. Since the naming convention is to publish data with continuously increasing sequence numbers, this sync tree is a condensed representation of the namespace containing all Data ever published in the group.

ChronoSync nodes maintain *long-lived Sync Interests* in the network by transmitting a new Sync Interest immediately when the previous one expires or gets satisfied. The long-lived Interest stays in the pending Interest table of the forwarders in the network so that any reply to the Sync Interest can be returned to every node in the group as soon as it is generated. The Sync Interest name starts with the multicast sync

[1] Real-world applications may use complex name structures to encode richer semantics information. ChronoSync supports such applications by "one level of indirection": actual application data names (or the data itself if the size is small) can be carried in the content of the Data packet published by the sync layer.

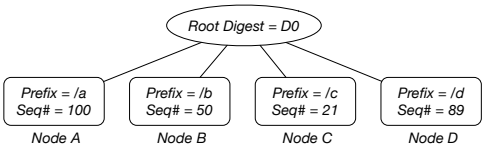


Figure 4.8: Example of a sync tree in ChronoSync

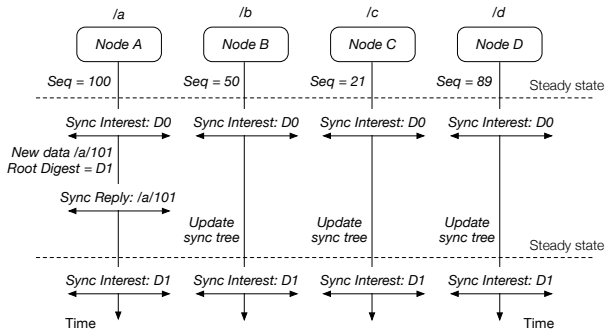


Figure 4.9: Synchronization process in ChronoSync

group prefix and carries the current root digest of the sender's local sync tree. In the steady state, all nodes generate identical state digests and send out the same Sync Interest that is aggregated by the NDN forwarders. When a node publishes new Data, the node replies to the pending Sync Interest with the name of its newly published data (i.e., the node prefix and the sequence number).[2] This *Sync Reply* is efficiently delivered to all the other nodes in the group by following the multicast tree created by the previous Sync Interest. After a node receives the reply, it updates the local sync tree, recomputes the root digest, and then sends out a Sync Interest carrying the new digest. Fig. 4.9 illustrates the process.

ChronoSync combines state change notification and update retrieval into a single NDN Interest-Data exchange. However when multiple nodes produce data at the same time, they will reply to the same Sync Interest carrying a digest D, with each reply containing different updates. Similar to CCNx Sync, at most one can be received by the nodes that sent a Sync Interest with digest D, and depending on the topological connectivity, different nodes may receive different updates. When these nodes send out their next Sync Interests, their digest values will not be the same. Consider a scenario where nodes A and B produced data at the same time, A and B receive each other's update, but node C only receives A's update and node D only B's update. Consequently the four nodes send Sync Interests carrying three different digest values, indicating a dataset state divergence.

Similar to CCNx Sync, ChronoSync handles such divergence by resending the previous Sync Interest with exclude filters that contain the digests of the updates already received. However, in a complex scenario where updates are generated when the nodes are already in a diverged state, the mechanism using exclude filters may not be able to bring the group back to synchronization (Section 4.2.4 provides details). In such a case ChronoSync falls back to a *recovery* mechanism: when a node observes an unknown digest, it will send out a special *Recovery Interest* containing the unknown digest; the nodes who recognize that digest will reply with complete information about its sync tree, rather than the specific changes that led to that digest; when the requesting node gets the reply, it will merge the received sync tree into its local sync tree by taking the higher sequence number from both trees for each sync node.

4.2.4 RoundSync

RoundSync [3] modifies the ChronoSync design based on the following key observation: the Sync Interest in ChronoSync is overloaded with two functions: (1) detecting different states among nodes and (2) retrieving updates from other nodes. As a result, the Sync Replies carrying the updates to the shared dataset will be named after the previous Sync Interest name which contains the digest of the corresponding sync state. If a node generates Sync Replies on top of a diverged state (e.g., in the scenario with partitioned sync group), nodes with different states will not be able to derive the correct name for those Sync Replies and therefore cannot send Interests to retrieve them.[3] In that case ChronoSync must rely on the recovery mechanism to bring the group in sync again.

[2] If multiple data packets are generated at the same time, the Sync Reply carries the latest sequence number of new data.
[3] Note that merging the diverged sync states will only create another sync state with a new digest value.

To address this problem, RoundSync introduces a new type of Interest packet called *Data Interest* in order to decouple state notification from update fetching. In RoundSync, the Sync Interest carrying the state digest merely serves as a notification mechanism (similar to iSync) so that the sync nodes can detect state divergence in the group when it happens. Updates generated by other sync nodes are retrieved via Data Interests whose names do not depend on the state digests. This feature allows the sync nodes to construct Data Interests to fetch the updates even if their states are not fully synchronized. Replies to the Data Interest achieve the same functionality as the Sync Reply in the original ChronoSync design, i.e., carrying node prefix and sequence number of the new Sync Data.

Another major change made by RoundSync is to divide the synchronization process into multiple *rounds*, identified by unique round numbers. A sync node can publish at most one data packet in each round and must move to a new round when it receives new data published by others in the current round. This constraint helps reduce the chances of state divergence caused by simultaneous data production. The names of both Sync Interest and Data Interest carry the round number so that each round is synchronized independently. For example, a sync node may start publishing data at round 11 even though it is still trying to synchronize with other nodes at round 10 or earlier. If multiple nodes publish data in the same round simultaneously, they will detect the inconsistency through Sync Interests and then send Data Interests with exclude filters to retrieve those Data Interest replies. Since there will be at most one reply from each node in a single round, the exclude filter mechanism will allow the nodes to eventually retrieve all updates. A basic example of the synchronization process in RoundSync is shown in Fig. 4.10.

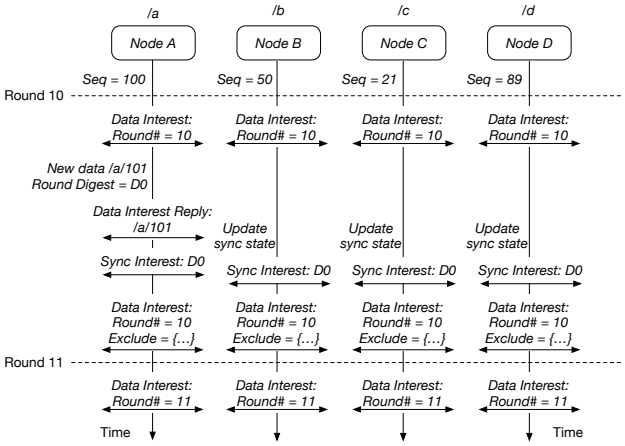


Figure 4.10: Synchronization process in RoundSync

RoundSync maintains a digest for each round in a *rounds log* table. To allow nodes who missed the Sync Interests in earlier rounds to detect and recover the missing data, RoundSync also computes a *cumulative digest* for the previous rounds that have been stable for a long time (which therefore has high probability of remaining stable in the future). The cumulative digest for a round covers the entire dataset as observed in that round and is piggybacked in Data Interest replies of future rounds. Upon receiving a different cumulative digest for some stable round, the sync node sends out a Recovery Interest to fetch the full sync state and the current round number S from the node who generated that cumulative digest. After receiving the reply, the node merges the received sync state with its own, discards the log entries for rounds before round S and resumes normal RoundSync operation for rounds subsequent to S.

4.2.5 pSync

As we described in Section 4.1, pSync adopts iSync's use of IBF to represent the namespace by storing the hash of the names (called *KeyID*) in the fixed-length slot of the IBF, and adopts ChronoSync's naming convention to name data packets using the concatenation of node||application||sequence numbers. Therefore the IBF in pSync only needs to store the latest data name from each data stream, which allows pSync to use a small IBF size that can be transmitted more efficiently over the network.

To support synchronization of a subset of the producer's data (a.k.a., *partial sync*), pSync introduces the *subscription list* to encode the prefixes of data streams of interest to a consumer.[4] The subscription list is a Bloom Filter (BF) that stores the hashes of those stream prefixes. The size of the Bloom Filter is determined by the total number of streams a consumer may subscribe to, and the false positive rate the consumer is willing to accept. Special markers enable encoding special cases such as empty and full subscriptions. During the

[4]pSync allows the consumers to specify their subscription only at the granularity of data streams.

sync process, the consumer keeps a local copy of the producer's IBF which indicates the data it has received so far. To sync up with the producer and retrieve new data, the consumer maintains *long-lived Sync Interest* whose name contains the local IBF copy and the consumer's subscription list. When the producer publishes new data, it first subtracts the IBF in the pending Sync Interest from its new IBF, and extracts the KeyIDs of the new data packets that have not been received by the consumer yet. Then the producer checks whether the stream prefixes of those new data packets are included in the consumer's subscription list (subject to certain false positive rate). Finally the producer generates a sync reply containing the original names of the new data packets in the subscribed streams and also its latest IBF. Upon receiving the sync reply, the consumer updates its local IBF copy with the received IBF, and sends out Interests to fetch the new data.

The pSync design lets each consumer maintain its own data consumption and subscription status, so that producers do not need to maintain per-consumer state. This feature enables scaling the growth of subscribers, and enables consumers to send Sync Interests via *anycast* to reach any producers that serves the same set of (synchronized) data streams. The tradeoff of this *stateless* producer design is two-fold: first, the Sync Interest and Sync Reply must carry both the IBF and the subscription list, which increases the size of the name. Second, the producer needs to generate sync replies in real time for each received Sync Interest because it does not retain the consumption state of each consumer and thus cannot pre-generate the next sync reply.

4.3 The Design of VectorSync

VectorSync is a new sync protocol to run over NDN. The design of VectorSync benefits from the experience we gained from the above mentioned sync protocols, which provides valuable insights into the tradeoffs among design choices. Similar to ChronoSync and pSync, VectorSync adopts the naming convention that each sync node names its data under its own data publishing prefix with continuous sequence numbers. This naming convention enables VectorSync to represent the state of the shared namespace efficiently using *version vectors* [11], hence the name of the protocol, that contain the latest sequence number from each member in the group. Inspired by iSync, VectorSync notifies the group members about data publishing events by announcing the digest of the new sync state via multicast Interests. Nodes that receive the announcement can fetch the corresponding sync state and reconcile the differences using version vector operations.

A key difference between VectorSync and its predecessors is the built-in *group membership management* that tracks membership changes, which allows it to compact the version vector to an array of integers where the order of the nodes' sequence numbers is pre-determined by the membership list. The group membership information also includes each member's security credentials, supporting data authentication and access control. Managed group membership also enables VectorSync to support dataset snapshot and garbage collection services, and to build a data ordering layer on top of VectorSync using the well-known logical clock algorithm [7]. Below we briefly describe the basic operational model of VectorSync.

4.3.1 Data Naming and State Synchronization

Figure 4.11(a) shows the naming convention for data published by sync nodes in the shared dataset. Each sync node publishes under its own data prefix, constructed by concatenating the topological prefix of the node's access network with its unique node ID. VectorSync names data by appending sequence numbers to the end of the data publishing prefixes, so that the entire namespace of the shared dataset is precisely summarized by a list of *(node id, latest sequence number)* pairs. VectorSync further compresses the sync state representation by pre-configuring the order of each node in the version vector via the group membership list, so it can omit node IDs and further reduce the vector to an array of integers. This allows VectorSync to omit the node IDs and reduce the original version vector format to an array of nonnegative integers which can be efficiently encoded and transmitted over the network.

When a sync node $N1$ publishes new data, it updates its own entry in the local version vector to the latest sequence number and sends out a *Sync Interest*, which serves as a notification of the data publishing event and is forwarded via multicast to all other nodes in the group. Figure 4.11(b) shows the naming convention for the Sync Interest name, which starts with a multicast prefix that uniquely identifies the sync

(a) Node data name:
/[unicast-network-prefix]/[node-id]/[app-id]/[sequence-number]

(b) Sync Interest name:
/[multicast-group-prefix]/notify/[node-id]/[view-number]/[leader-id]/[state-digest]

(c) Sync state data name:
/[unicast-network-prefix]/[node-id]/state/[view-number]/[leader-id]/[state-digest]

(d) ViewInfo data name:
/[multicast-group-prefix]/vinfo/[view-number]/[leader-id]/[segment-number]

(e) Snapshot data name:
/[multicast-group-prefix]/snapshot/[view-number]/[leader-id]/[segment-number]

Figure 4.11: VectorSync naming conventions

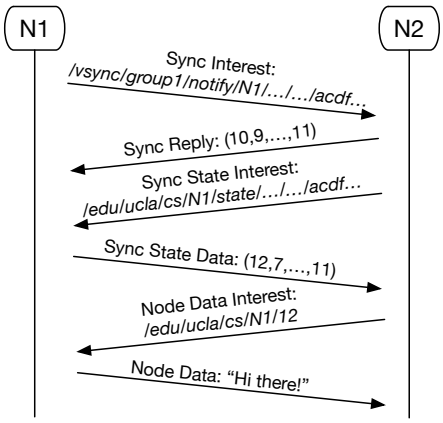


Figure 4.12: VectorSync protocol message exchange between two sync nodes

group, followed by a special marker component "notify" and the publisher's node ID, and ends with the digest value of the publisher's latest sync state (i.e., the version vector).

Upon receiving $N1$'s Sync Interest, a sync node $N2$ sends a *Sync Reply* with a small FreshnessPeriod (e.g., 5ms) to satisfy the pending Sync Interests in the network. The Sync Reply carries $N2$'s own sync state propagating to $N1$. Since $N1$ can receive only one Sync Reply, exchanging sync states via Sync Replies does not guarantee synchronization across the entire sync group.

After sending the Sync Reply, $N2$ immediately issues a Sync State Interest to fetch $N1$'s Sync State Data. Figure 4.11(c) shows the naming convention of the Sync State Data: the name begins with the data publisher's unicast prefix, which can be determined by looking up the group membership information using the publisher's node ID carried in the Sync Interest; it is followed by a special marker component "state", the current view number and leader id, and ends with the digest of the sync state. The data publisher, $N1$ in this case, replies to the Sync State Interest with the corresponding version vector that represents state of $N1$'s dataset *before* the Sync Interest is sent.

After receiving the Sync State Data packet, the sync node, $N2$ in this case, extracts the version vector and performs a $Join$ operation over the received version vector and its local one by taking the entry-wise maximum of the two. The resulting version vector represents the smallest common sync state that subsumes both the local and the remote states. The sync node replaces its local version vector with the output of the $Join$ operation, then checks if any entry in the new version vector contains a higher sequence number than the previous one, in which case it issues Node Data Interests to fetch the new data asynchronously. The node may also keep a temporary log of Sync State Data packets and ignore the Sync Interests if the announced states have been processed in the past.

Figure 4.12 illustrates the message exchange between data publishing and receiving nodes. When there is no packet loss and no cached data in the network, the minimum delay to synchronize the dataset state is $2.5 \times RTT$s. If the group size is small, the data publishing node can attach the signed version vector (instead of the digest) in the name of the Sync Interest.[5] This step saves one RTT in the synchronization process: the sync nodes can start processing the version vector once it receives and authenticates the Sync Interest, making VectorSync more suitable for real-time applications with tight delay budget for data synchronization.

4.3.2 Group Membership Management

VectorSync operates over *managed groups*, where a sync node always keeps information about the current active members in the sync group, a.k.a. the *view* of the group, and synchronizes only with these. To

[5]One can use the ECDSA or HMAC algorithms to reduce the size of the signature in the Interest name.

maintain its membership, each node publishes heartbeat data packets in the shared dataset periodically which propagate to other nodes through the synchronization process. The heartbeat message contains the publisher's current view ID, which the receiving node ignores if it is in a different view. If a node N's heartbeat is missed for M times consecutively ($M = 3$ in our current design), N is considered to have left the group.

To avoid inconsistency in the view of the sync group, a distinguished *leader* node is elected by the group to monitor the status of each node in the current view. Each view can be uniquely identified by a pair of [view#, node ID], where the view# is a monotonic-increasing number and the node ID the ID of the view leader. The view IDs are lexicographically ordered. as follows:

Definition 4.3.2.1. Given $vid_1 = (vn_1, nid_1)$ and $vid_2 = (vn_2, nid_2)$, $vid_1 < vid_2$ if $vn_1 < vn_2$ or ($vn_1 = vn_2$ and $nid_1 < nid_2$).

When the leader detects some node has left, it initiates a *view change* process to move the remaining members to a new view with a higher view number (typically incremented by one). The leader first publishes the membership information of the new view in a *ViewInfo* data packet whose naming is shown in Figure 4.11(d). The ViewInfo contains a list of members with their unique node IDs and data publishing prefixes; the order of the members in the ViewInfo packet determines the position of each node in the version vectors exchanged in the new view.

We have sketched a solution for handling leader departures, making the overall design self-organizing and self-adapting. We hope to finish the design soon and invite application developers to experiment with VectorSync.

4.4 Summary

The series of sync protocols described in this chapter represents the learning process in understanding the design space of sync, which helped gain insights of the different design approaches and their tradeoffs. Table 4.1 compares the existing NDN sync protocols on a few performance metrics that are closely related to the protocol design. For example, the choice of sync state representation directly impacts each protocol's message overhead and update retrieval delay. Table 4.1 shows the minimal possible retrieval delay for each design but not the state sync overhead, which we describe below.

Table 4.1: Comparison of existing sync protocols in NDN

	CCNx Sync	iSync	ChronoSync	RoundSync	pSync	VectorSync
Data dissemination delay	Interest period + tree walk	Interest period + 3.5 RTT	Min is 0.5 RTT; can be long with simultaneous publishing	Min is 1.5 RTT; can be long with simultaneous publishing	1.5 RTT (assuming a single producer)	2.5 RTT
Interest overhead	Periodic	Periodic	Long-lived Interest	Two per update	Long-lived Interest	Two per update
Factors affecting Interest size	Node hash	IBF digest	State digest (+ exclude filter)	Round digest (+ exclude filter)	IBF + subscription list	State digest
Factors affecting Data content size	Number of children under the requested node	IBF size (depending on the number of new data)	Number of names with new seq#	Number of names with new seq# in a round	IBF size + number of names with new seq#	Version vector size

One important performance metrics is the data dissemination delay, i.e., the number of round-trips necessary for propagating new data to other nodes. CCNx Sync usually requires multiple round-trips to

synchronize the data collection between two repos due to the "tree walking" process. iSync improves the CCNx Sync design by using the IBF instead of the sync tree to represent the dataset namespace, allowing the repos to synchronize within a fixed number of round-trips in most cases. Both CCNx Sync and iSync perform periodic synchronization without providing triggered update mechanism, which further adds to the data dissemination delay. ChronoSync and pSync utilize long-lived Sync Interest to retrieve the information about new updates as soon as they are generated. However, if multiple nodes generate sync replies at the same time, the protocols need additional round-trips to retrieve all sync replies using Interests with exclude filters.[6] RoundSync also has the same problem with simultaneous data publishing. VectorSync uses version vector to convey specific information about the new data names, which enables the sync nodes to fetch new data with exact names and therefore avoiding the use of exclude filters.

Another critical metrics is the packet size of the sync protocol messages, which reflects the network bandwidth requirement of the sync communication. CCNx Sync enumerates different name hashes in the sync tree through multiple Interest-Data exchanges. iSync compresses the set of name hashes using the IBF which is encoded in a single Data packet. pSync benefits from the sequential naming convention to simplify the dataset namespace, allowing it to use substantially smaller IBF to encode the namespace and carry the IBF directly in the Sync Interest name. The sequential naming convention also enables ChronoSync and RoundSync to disseminate only the latest data names rather than the whole dataset namespace, which leads to much smaller protocol messages. VectorSync further reduces the namespace representation to a vector of sequence numbers, making it feasible to propagate the whole namespace efficiently over the network.

Following the application-driven design principle, we expect to learn more about VectorSync's utilities and limitations from future application development and to gain further insight into this new "transport area" research. Note that the Sync discussions in this chapter focus on the sync state representation and update dissemination from a transport viewpoint; a related challenge at the network layer is how to achieve group multicast delivery of sync interests in a scalable, resilient, and secure way, a long recognized challenge for which we continue to explore the design space for effective solutions.

References

[1] Alexander Afanasyev, Zhenkai Zhu, Yingdi Yu, Lijing Wang, and Lixia Zhang. The Story of ChronoShare, or How NDN Brought Distributed Secure File Sharing Back. In *Proceedings of IEEE MASS 2015 Workshop on Content-Centric Networks*, October 2015.

[2] CCNx Project. CCNx Synchronization Protocol. `https://github.com/ProjectCCNx/ccnx/blob/master/doc/technical/SynchronizationProtocol.txt`, 2012.

[3] Pedro de las Heras Quirós et al. The Design of RoundSync Protocol. Technical Report NDN-0048, NDN Project, March 2017.

[4] D. Eppstein, M. T. Goodich, F. Uyeda, and G. Varghese. What's the Difference? Efficient Set Reconciliation without Prior Context. In *Proceedings of ACM SIGCOMM*, 2011.

[5] Chengyu Fan, Susmit Shannigrahi, Steve DiBenedetto, Catherine Olschanowsky, Christos Papadopoulos, and Harvey Newman. Managing scientific data with Named Data Networking. In *Proceedings of the Fifth International Workshop on Network-Aware Data Management*, November 2015.

[6] Van Jacobson, Jeffrey Burke, Lixia Zhang, Beichuan Zhang, kc claffy, Dima Krioukov, Christos Papadopoulos, Tarek Abdelzaher, Lan Wang, Edmund Yeh, and Patrick Crowley. Named Data Networking (NDN) Project 2010 - 2011 Report, 2011. `http://named-data.net/project/annual-progress-summaries/project/ndn-ar2011-html/`.

[7] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM (CACM)*, 21(7):558–565, July 1978.

[6] Note that pSync typically synchronizes with a single producer.

[8] Vince Lehman, A K M Mahmudul Hoque, Yingdi Yu, Lan Wang, Beichuan Zhang, and Lixia Zhang. A secure link state routing protocol for NDN. Technical Report NDN-0037, NDN Project, January 2016.

[9] Lan Wang Minsheng Zhang, Vince Lehman. Scalable Name-based Data Synchronization for Named Data Networking. In *IEEE INFOCOM*, April 2017.

[10] C. Olschanowsky, S. Shannigrahi, and C. Papadopoulos. Supporting climate research using named data networking. In *Local Metropolitan Area Networks (LANMAN), 2014 IEEE 20th International Workshop on*, pages 1–6, May 2014.

[11] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of Mutual Inconsistency in Distributed Systems. *IEEE Transactions on Software Engineering*, SE-9(3):240–247, May 1983.

[12] Wentao Shang, Alexander Afanasyev, and Lixia Zhang. VectorSync: Distributed Dataset Synchronization over Named Data Networking), April 2017. submission to ACM ICN 2017.

[13] Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring ISP topologies with rocketfuel. *Networking, IEEE/ACM Transactions on*, 12(1):2–16, 2004.

[14] Fu Wenliang, Hila Ben Abraham, and Patrick Crowley. Synchronizing namespaces with invertible bloom filters. In *To appear in ANCS 2015*, 2015.

[15] Fu Wenliang, Hila Ben Abraham, and Patrick Crowley. Synchronizing namespaces with invertible bloom filters. In *Proceedings of the ACM IEEE Symposium on Architectures for Networking and Communications Systems*, May 2015.

[16] Minsheng Zhang, Vince Lehman, and Lan Wang. PartialSync: Efficient Synchronization of a Partial Namespace in NDN. Technical Report NDN-0039, NDN, June 2016.

[17] Zhenkai Zhu and Alexander Afanasyev. Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking. In *IEEE ICNP*, 2013.

Chapter 5

Networking

Contributors	
PIs	Beichuan Zhang (Arizona), Alex Afanasyev, Van Jacobson & Lixia Zhang (UCLA), Lan Wang (Memphis), Christos Papadopoulos (Colorado State University), Patrick Crowley (Washington University)
Grad Students	Junxiao Shi, Teng Liang, Klaus Schneider (Arizona); Spyridon Mastorakis, Wentao Shang (UCLA), Muktadir R. Chowdhury, Laqin Fan, Lei Pi, Minsheng Zhang (U. Memphis), Chengyu Fan (Colorado State), Hila Ben Abraham, Adam Drescher (Washington University)
Undergrads	Damian Coomes (U. Memphis), Eric Newberry (Arizona), Allen Gong (UCLA)
Staff	Ashlesh Gawande, Nick Gordon, Vince S. Lehman (Memphis), John DeHart, Jyoti Parwatikar (Washington University)

During this reporting period, we continued our research to develop the underlying network layer of the NDN architecture, with an emphasis on how to best meet the needs of the target network environments. We made progress on routing protocols, congestion control, scalable forwarding, and NDN in local area networks.

5.1 Routing Protocols

Our work on NDN routing protocols continued in two parallel directions: conventional link-state routing (Named-data Link State Routing, NLSR [4]) and update-less greedy routing (Hyperbolic Routing, HR [3]).

5.1.1 NLSR

Route Readvertise

We added support in NLSR for mobile producers to inject their name prefixes into the testbed through non-home hubs. The entire process from an application's prefix registration to NLSR's route advertisement involves four steps: (1) a mobile producer's application registers its name prefix with its local NFD; (2) the local NFD uses Auto Prefix Propagation[1] to register the application's name prefix with the NFD at the hub; (3) the hub's NFD passes the prefix to its local NLSR; and (4) the hub's NLSR advertises the prefix to the entire network so that other routers will be able to reach the mobile producer through its new non-home hub. Since Step 1 and 2 are already implemented in NFD, our implementation in NFD and NLSR supports Step 3 and 4. During our design process, we realized that the functionality in Step 3 can generalize to propagate a name prefix from NFD to any other process on the same or a remote node, e.g., to the NFD on

[1] `http://named-data.net/doc/NFD/0.4.0/doxygen/d0/d84/classnfd_1_1rib_1_1_auto_prefix_propagator.html`

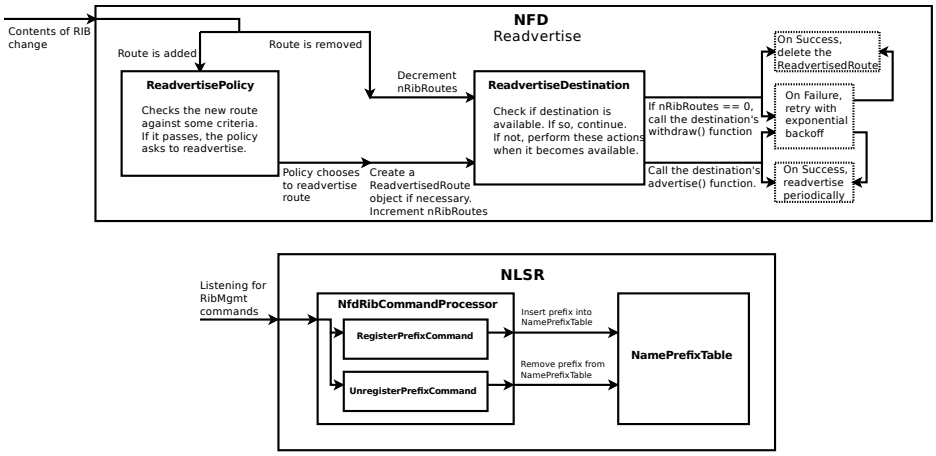


Figure 5.1: The structure of the Readvertise module.

a remote hub (i.e., Auto Prefix Propagation) or to the NLSR process in the local node (the original Route Readvertise). The trust model and policies likely differ across scenarios, so we need to provide a place for such configurations. This modularity provides good code re-use, keeping with NFD's design philosophy.

Figure 5.1 describes the structure of the Readvertise module in NFD, along with the listener implementation in NLSR. Specifically, when a route is added to the RIB, Readvertise consults its ReadvertisePolicy instance. If the policy chooses to readvertise, the module will provide the Readvertise instance with a prefix and some credentials to advertise. On route removal, the policy is not consulted, and Readvertise withdraws the route from the destination. The destination is only an NFD-side interface; the listener must implement the other side of the relationship and process any advertisements or withdrawals itself, depending on what that means for each situation. In the case of NLSR, NFD emits RibMgmt commands on a prefix that NLSR is listening on. When the module in NLSR, called NfdRibUpdateProcessor, receives these RibMgmt commands, it checks the verb in the command. On a registration command, NLSR inserts the prefix into its NamePrefixTable, and NLSR will advertise the prefix to the network. On an unregistration command, NLSR removes the prefix from its NamePrefixTable, and NLSR will no longer advertise the prefix.

We have finished the implementation in NFD and NLSR. We still have to refactor the Auto Prefix Propagation system to use the Readvertise platform instead of the currently-independent module that it is. The implementation effort showed that it was not trivial to integrate the Readvertise platform into the existing NFD framework, mostly because of how NFD manages independent modules compiled with it. During the implementation of the NLSR components, we encountered a significant trust issue: how do you trust that the Readvertise messages you are getting come from NFD? Currently it is not possible to use network channels to establish the identity of NFD, so a different trust model is required.

This work illustrated the need for a message passing system for routing information. Design discussions concluded that rather than the original plan to construct an independent client program to coordinate between NFD and NLSR, which would minimize the coupling between them, NFD should directly communicate with NLSR. This approach reduces complexity in NFD, which is a priority for scalability. We also investigated possible uses of lightweight, signaling systems to coordinate different parts of NFD. The RIB module now emits signals of certain important events, which the Readvertise module listens for. This approach may be useful to decouple other parts of NFD and increase orthogonality. As more applications need access to routing information updates, we will consider how to incorporate code into NFD to support it, and consider scalability implications of this functionality.

NLSR Port to ndnSIM

The motivation of this work was to allow for in-depth experiments on NLSR utilizing ndnSIM, such as packets exchanged, convergence time, etc. ndnSIM allows for accurate measurement through complete simulation

of a network, a difference from the simpler miniNDN, which only emulates networks by providing virtual network interfaces running natively on one or more real machines. We can now more effectively study how changes to NLSR will impact performance of NDN networks. The design consisted of identifying necessary changes to port NLSR to ndnSIM. Fortunately they were few, and the implementation phase went quickly, consisting of compiling and testing ndnSIM with NLSR and packaging the work for redistribution. Both link-state and hyperbolic routing functionality within NLSR are now working. This work benefited from the fact that the ndnSIM system design is highly modular, allowing easy porting of complex applications. In particular, the modularity provided by the Command Interest and more broadly the Interest systems demonstrates the platform agnosticism of NDN in general. As associated challenge (with great power comes great responsiblity!) is keeping the ndnSIM port of NLSR and the mainline brach of NLR in sync.

Face Discovery

Currently NLSR maintains a configured list of neighbors, and will create Faces for those neighbors in NFD as necessary. This is analogous to having OSPF or any other routing protocol set up a link between neighbors. In practice, links are not set up by routing protocols, but instead by network operators. Therefore, NLSR needs mechanisms to discover available physical and logical interfaces upon startup. It should concern itself only with routing, not with the job of creating network-level link infrastructure needed to support routing. Other applications also share this issue, so our work will further inform work on auto-configuration.

In our new design, NLSR obtains a list of Faces from NFD upon startup, and builds its routing procedures around that list. It then listens to the events that NFD emits concerning the change in any Faces's status. In order to guard against missed notifications, NLSR also regularly requests this information from NFD.

The implementation phase is well underway. We still need to complete the routine refetch of the Face list, listening for Face change notifications, and the necessary NFD reconfiguration to accommodate this new functionality.

Documentation

Over the development of NLSR, it became clear that we needed robust documentation describing the software design and implementation. As NLSR grew and components became more sophisticated, complex relationships between modules emerged, which required a reference document to support developers. We decided to provide high-quality documentation, modeled on the NFD Developer's guide due to its thoroughness, completeness, and reputation within the NDN community. We completed a first rough draft, which still lacks sufficient explanations of module interfaces and implementation details needed by developers. Still on our list to document this year are specifications of:

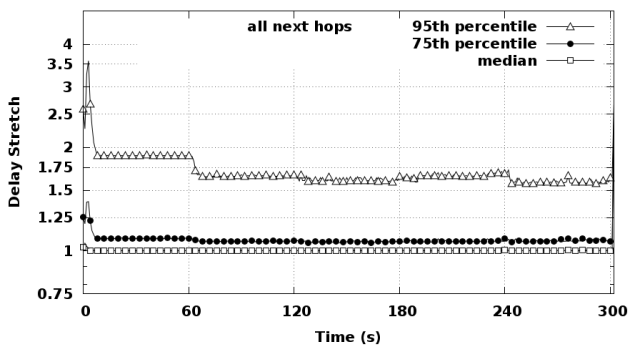
- When a module's functions are called in a significant way by another module.
- A full list of the classes that each module directly interacts with, as caller or callee.
- Each module's *purpose*. The difference between purpose and function are important to form a macroscopic understanding of the code.

This work will also benefit from elaborating in-code using a programmatic API documentation generator known as Doxygen. Doxygen provides a way to "connect the dots" between classes, which will make it easier to maintain the developer's guide, and allow it to focus on design and purpose explanations. We use Doxygen in NLSR, but not to an extent that it can supplement the current developer's guide.

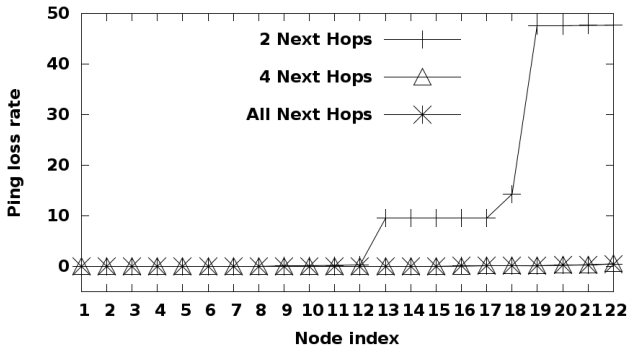
5.1.2 Hyperbolic Routing Evaluation in Mini-NDN and Testbed

Hyperbolic Routing (HR) offers an appealing possible solution to the routing scalability problem in NDN because it does not exchange routing updates upon changes in network topologies[2]. Although HR has the drawbacks of producing suboptimal routes or local minima for some destinations, NDN's intelligent data forwarding plane can mitigate these issues. However, HR's viability still depends on both the quality of the

[2]Current HR implementation does exchange coordinates and refresh them periodically.



(a) HR vs LS delay stretch



(b) HR ping loss rate

Figure 5.2: Mini-NDN experiment results with 22-node topology

routes HR provides and the overhead incurred at the forwarding plane due to HR's sub-optimal behavior. We designed a forwarding strategy called Adaptive Smoothed RTT-based Forwarding (ASF) [3] to mitigate HR's sub-optimal path selection.

For our initial experiments we had used a snapshot of the NDN testbed when it had 22 nodes. We ran a 5-minute emulation experiment in Mini-NDN where we let each node ping every other node. Then we looked at the delay stretch between HR and LS (Figure 5.2a). Delay stretch is the RTT under HR divided by the RTT in LS routing. Under the controlled emulation environment we can see that the HR with ASF has a median stretch of 1 with LS. Next we looked at the ping loss rate of HR (Figure 5.2b). In the emulation environment, LS routing had no losses and HR routing with all next hops enabled had almost no losses.

We continued our research effort as we prepared for the deployment of HR on the NDN testbed. We ran experiments in Mini-NDN with the current 33-node testbed topology to make sure HR converges. Our experiments in Mini-NDN had shown us that there was a critical bug [5] in the ASF strategy that was leading to false measurement that forced ASF to switched paths unnecessarily to high RTT paths. We fixed the bug and deployed the updated code on the testbed as a part of NFD version 0.5.1. Before we turned on HR, we collected statistics with Link State (LS) routing so as to compare with HR later, and configured appropriate HR coordinates in nodes. The experiment we ran was to let each node ping every other node for 5 minutes, i.e., the same as our emulation experiments. In LS routing we used the BestRoute strategy and in HR we used the ASF strategy. We ran the pings every hour for 24 hours for each trace. We had usable data for 26 nodes. Figure 5.3a shows the comparison of aggregate delay stretch between HR and LS. The stretches in the testbed results were higher than those in the emulations, but the median and 75-th percentile were still low, around 1.25 and 1.5, respectively.

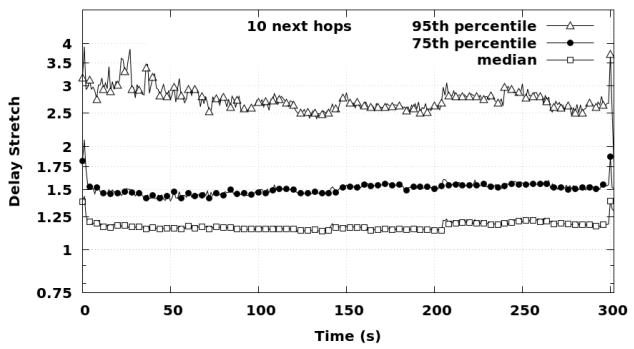
We then compared loss rates (the percentage of pings that timed out) between HR and LS. Note that on the testbed pings under LS routing did suffer some losses unlike the emulation. Figure 5.3b shows the best loss rate observed in 24 traces. HR's loss rate is close to that of LS.

Below are a few important things that we learned:

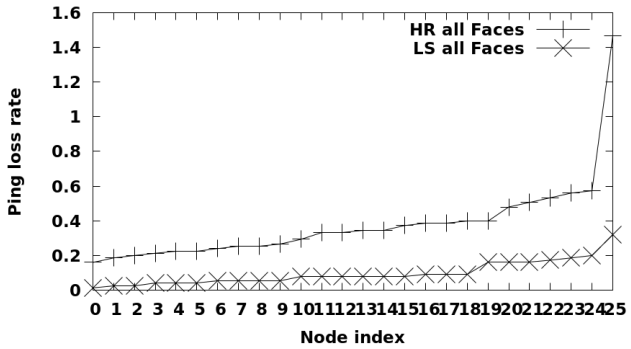
- In emulation we are not considering random delay changes and random packet losses which are prevalent in the real testbed;
- ASF is very sensitive to occasional timeouts and transient changes of RTTs;
- Current HR routing is not very effective in producing loop free routes and things get worse with having multiple routers in the same site;
- Testbed results are very noisy and are hard to compare without aggregation.

There are several open issues with HR. We need to collect more data from the testbed to know exactly how ASF is performing. Currently we cannot experiment with changing parameters of ASF such as probing frequency. We also need to change the ASF design to make it less sensitive.

This year we explored a new geohyperbolic routing scheme that considers centrality of a node along



(a) HR vs LS delay stretch



(b) Best loss rate observed

Figure 5.3: Testbed experiment results (testbed has 33 nodes, but only 26 nodes had ping results)

with its geographical coordinates [9] to improve the delay stretch compared to pure hyperbolic routing. This scheme uses a 3-dimensional hyperbolic space. As part of the 3rd NDN hackathon, we have already implemented n-dimensional HR which generalizes how NLSR handles HR coordinates. The work is almost ready to integrate into the next major release of NLSR. However, before testbed evalujation, we need to analyze the current 2-dimensional HR and collect more data to compare it to 3-dimensional HR.

5.2 Congestion Control

Traditional TCP-like congestion control uses pre-established connections between two fixed endpoints. The sender detects congestion based on round-trip time (RTT) measurements and packet losses, then adjusts its window size or sending rate accordingly. In NDN, the concept of end-to-end connections does not apply. Data chunks of the same content may be retrieved from different repositories or different caches along the paths towards these repositories. Since these different content sources result in varying retrieval delay, and the consumer cannot distinguish between them, traditional RTT-based timeouts become unreliable indicators of congestion.

NDN's stateful forwarding plane enables routers to control congestion at each hop, by either dropping Interest packets or diverting them to alternative paths. However, most existing solutions in this direction assume known or predictable link bandwidths and data chunk sizes, which limits their effectiveness in scenarios where these assumptions do not hold. For example, the largest experimental NDN deployment, the NDN Testbed, runs over UDP tunnels where the underlying bandwidth is unknown and keeps changing. In another example, a video-on-demand provider may respond to congestion by adjusting the video quality (hence data chunk size) dynamically.

This year we proposed **PCON: a practical NDN congestion control scheme** [6] that does not assume known link bandwidth or data chunk sizes. PCON routers detect congestion on their local links by using an active queue management (AQM) scheme extended from CoDel. When a router detects congestion, it signals this state to consumers and downstream routers by explicitly marking data packets. Downstream routers react by partially diverting subsequent Interests to alternative paths and/or passing the signal further downstream; consumers react by reducing their Interest sending rate. Since routers and hosts on each hop participate in avoiding congestion, PCON is a "hop-by-hop" congestion control scheme. Its fundamental advantage over TCP/IP-based router-assisted congestion control is its ability to use multi-path in the network to diffuse congestion. As shown in Figure 5.4, PCON consists of the following components: congestion detection, congestion signaling, consumer rate adaptation, multipath forwarding, and local link loss detection.

Congestion Detection The most reliable place to detect congestion is where it happens, i.e., the outgoing queue of the congested link. This fact is more relevant to NDN than TCP/IP because NDN lacks a clear notion of a baseline RTT, the primitive on which TCP congestion control builds. We adopt a recently

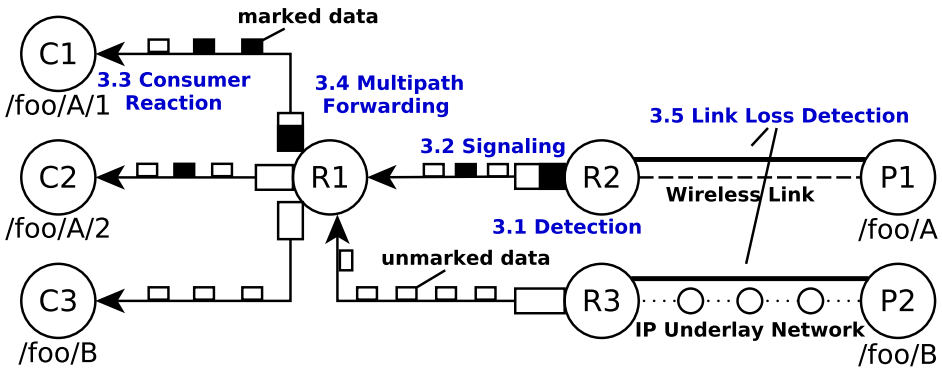


Figure 5.4: PCON Architecture

proposed AQM mechanisms, CoDel, to detect congestion. Instead of monitoring the queue length, CoDel monitors the queuing delay (called "sojourn time") of each packet on its outgoing links. If the minimum sojourn time over a time period (default: 100ms) exceeds a threshold (default: 5ms), it considers this link as congested. This threshold allows temporary buildup of the queue to absorb traffic bursts, while still using a persistent queue to infer evidence of congestion.

PCON monitors congestion in both the downstream (Data) and the upstream (Interest) direction. Interests can cause congestion when they are larger than data packets, on asymmetric links, or when there is cross-traffic in the upstream direction. After detecting congestion in the Interest direction (like R1–R2 in Figure 5.4), R1 will *mark the PIT entry* of that Interest, and consider the corresponding Data for congestion signaling. In the case that the underlying router's queues are not accessible, such as in a UDP tunnel, we will rely on packet loss detected by the NDN Link Layer Protocol as a sign of congestion.

Congestion Signaling After a router has detected that one of its outgoing links is congested, it signals this state to the consumers and to all routers along the path. Signaling is only necessary in the downstream direction, because only downstream routers can reduce the amount of traffic that the signaling router will receive. Thus, we signal congestion by marking NDN *Data packets*, but not Interests. To signal congestion, we mark packets similar to CoDel's marking: The first packet is marked when entering the *congested* state and later packets are marked in an increasing "marking interval" (similar to CoDel's drop spacing); the marking interval starts at 1.1 * the CoDel interval (110 ms) and is decreased inversely proportional to the square root of number of marks, in order to achieve a linear decrease in the consumers sending rate. On congested upstream links, we use the same marking interval for setting the flag in the PIT entry, which then marks the associated data packets.

Consumer Rate Adjustment Once the congestion marks reach the consumer, the consumer needs to decide how to adjust its rate. We use a congestion window (specifying the maximum number of in-flight Interest packets) which is increased on unmarked data packets and decreased on marked ones, negative acknowledgment (NACKs), and timeouts. Data packets act as *selective acknowledgements* of Interest packets, meaning that each data packet acknowledges one specific corresponding Interest. If we perform a multiplicative decrease on each packet loss in a loss burst, the congestion window will be reduced too drastically (often back to 1) after a single congestion event. We prevent this over-adjustment by applying the principles of the TCP SACK-based Conservative Loss Recovery Algorithm to make sure that the consumer decreases the window at most once during one RTT. With the conservative loss adaptation and CoDel marking in place, PCON can implement a number of classical loss-based TCP window adaptation schemes at the end hosts. The difference from traditional TCP is that a window decrease is triggered not only by timeouts, but also by packet marks and NACKs.

Multipath Forwarding In addition to consumer rate adaptation, routers along the path will also see congestion marks and will adjust their traffic split across multiple paths. This feature can take advantage

of the NDN architecture. We designed our multipath forwarding strategy with the objective of *maximizing end-user throughput* while keeping the network cost (as measured by path length) as low as possible. We assume that the strategy has access to a list of next-hops ranked by their path lengths, set by the routing protocol. The optimal strategy decision at each hop depends on the demanded rate (determined by the capacity of downstream links and the consumer) relative to the capacity of the outgoing links. If the best path can satisfy the demand, a node should not split traffic on other paths. If the demand exceeds the capacity of the best path, a node can incrementally distribute traffic on the other path(s).

For each FIB prefix, PCON maintains a forwarding percentage of traffic to each next-hop, which is initialized to 100% for the shortest path and 0% for all other paths. Then for each marked data packet, it reduces the forwarding percentage of the congested face by a fixed percentage, and moves that amount of traffic to other faces evenly:

$reduction = fwPerc(F) * \frac{CHANGE\_PERC}{f(Distance)}$

$fwPerc(F) -= reduction$

$fwPerc(\bar{F}) += \frac{reduction}{NUM\_FACES-1}$

fwPerc(F) is the forwarding percentage of the current face and *CHANGE_PERC* is a fixed parameter, which makes a trade-off between how fast the face adjusts the forwarding ratio to the optimal value (higher is better) and how much it oscillates around that value (lower is better). Through experiments, we found a value of 1%-3% to work well for a number of different bandwidths. *f(Distance)* is a factor to adjust the forwarding ratio more strongly the closer the adjusting router is to the congested link.

We evaluated PCON using ndnSIM in various scenarios to demonstrate its effectiveness in handling caching, multicast, and multipath forwarding. Detailed results are available in the published paper [6].

Performance Evaluation We evaluate PCON using ndnSIM in various scenarios to demonstrate its effectiveness in handling caching, multicast, and multipath forwarding. Here we only include part of the results in multipath forwarding. More results are available in a published paper [6].

We compare PCON against two alternative designs that split traffic over multiple paths based on PIT occupancy: if an outgoing face has higher number of pending interests, this face is less preferred in forwarding future Interests. There are two specific designs: *PI*, which chooses the face with the lowest number of pending Interests, and *CF*, which uses a weighted round-robin scheme : $weight(face, prefix) \leftarrow 1/avgPI(face, prefix)$, where "avgPI" is the exponential moving average of the number of pending Interests.

In each measurement, we run the simulation over a small topology with three different paths, waited until the forwarding split ratio has stabilized and then take its average. We compared the performance under three different topology configurations:

- **Equal:** the three paths have the same bandwidth of 10Mbit/s and RTT of 20ms.
- **Diff_Delay:** the paths have the same bandwidth of 10Mbit/s, but a different RTTs of 100ms (face 257), 50ms (face 258), and 10 ms (face 259).
- **Diff_BW:** the paths have the same RTT of 20ms, but different bandwidth of 22.5Mbps, 6 Mbps, and 1.5Mbps. Here the ideal traffic split ratio should be 75%–20%–5%.

The results (Figure 5.5) show that PIT-based distribution (CF and PI) works well when the bandwidth and delay of available paths are the same: it achieves the optimal split of 33%–33%–33% in this case. However, when delays differ, both of them prefer the lower-delay path at a cost of total throughput (CF achieves a slightly better split ratio and higher bandwidth than PI). When the bandwidth differs, both CF and PI bias against high-bandwidth paths: instead of the optimal ratio of 75%, they only achieves about 54%. This leads to reduced aggregated bandwidth usage. Because PCON responds directly to the congestion state of the involved links, it achieves the expected split ratio in all three cases, which results in a significantly higher overall throughput in the cases of different delay and different bandwidth.

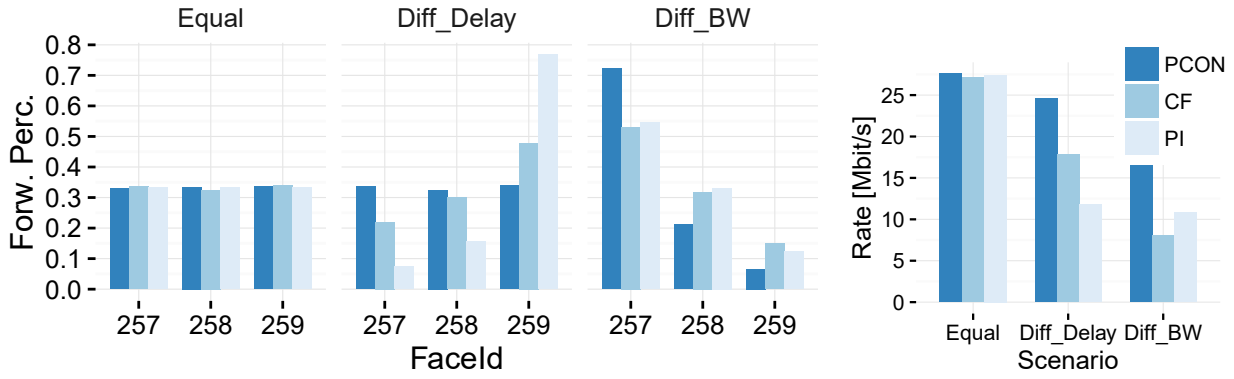


Figure 5.5: Multipath Forwarding: PCON vs. PI vs. CF

5.3 Scalable Forwarding

In the third year of the project, the Washington University team led efforts and made substantive contributions in three primary areas: scalable forwarding, forwarding strategy, and testbed development (discussed in Section 6.3).

Name-based forwarding is a core component in NDN, and designing scalable name-based forwarding solutions is challenging because name prefixes are of variable length and forwarding tables can be much longer than seen with IP. Recently, we proposed a speculative forwarding method, in which the forwarding structure size is proportional to the information-theoretic differences between the name prefixes rather than their lengths. In the past year, our goal has been to enhance name-based forwarding performance with memory- and time-efficient data structures. This work culminated in a 2017 publication [11]. In this work, we first define the string differentiation problem, based on the behavior of speculative forwarding in core networks, and then propose fingerprint-based solutions for both trie-based and hash table-based data structures. We experimentally demonstrate that the proposed solutions reduce the lookup latency and memory requirements. The proposed fingerprint-based Patricia trie decreases the average leaf-node depth and thus reduces the lookup latency. The proposed fingerprint-based hash table design requires only 3.2 GB of memory to store 1 billion names where each name has only one name component, and the measured lookup latency of the software-based single-threaded implementation is 0.29 microseconds. Whats more, the distributed forwarding scheme presented in this paper makes name-based forwarding truly scalable.

Additionally, we studied NDN forwarding dynamics analytically. Prior work has introduced methods for analyzing the theoretical performance of caching systems that rely on aggregating requests. NDN exhibits this behavior, and in recent work (published this year [2]) we studied the application of these analytical methods to NDN forwarding dynamics.

NDN's Pending Interest Table (PIT) can be viewed as a non-reset time-to-live (TTL) based cache. The Content Store (CS) is a content cache placed in front of the PIT on the NDN forwarding path, so they make up a tandem cache network. To investigate the metrics of interest in this network, like the hit probability for the PIT and the CS, the expected PIT size, non-zero download delay (non-ZDD) should be taken into consideration. Caching policies usually assume zero download delay (ZDD), i.e., request and response arrive simultaneously, and numerous analytical methods have been proposed to study the ZDD caching policies. In this work, after dissecting the LRU policy, we for the first time proposed two LRU variants considering non-ZDD by defining separate operations for the request and response arrivals. Further, when the CS adopts the proposed LRU variants, the analysis of the CS-PIT network can still take advantage of the existing models, so the metrics of interest can be computed. Especially, the distribution for the inter-miss time of this network can be derived, which has not been achieved by prior work.

Our efforts also included advances in understanding the role of forwarding strategy, and improving its viability. In NDN, a forwarding strategy decides how to forward an Interest packet. As discussed in past reports, we have surfaced the tension between application developers and network operators when specifying forwarding strategy. An application can pair its namespace to use a specific forwarding strategy in the local

host, but has no control over the strategies used in remote routers. Despite the central role the forwarding strategy plays, its interaction with applications has not been explored or well understood. We have been working to understand forwarding strategy, and to decompose the core mechanisms of a forwarding strategy into pieces sustainable for both application developers and network operators. In our recent publication [1], we illustrate how the correctness of some NDN applications can be affected by the coupling between the application design and the strategy decision to retransmit an unsatisfied Interest. This coupling creates challenges for application developers, who must implement their fixed application logic on a variable forwarding mechanism, and can lead to failure of application correctness and performance. We propose a new retransmission abstraction that decouples this strategy mechanism from the application design, and differentiates application Interests from network retransmissions. This allows every application to determine its own retransmission policy. We show that in some use cases the proposed abstraction can maintain continuous traffic flow regardless of the strategy used.

5.4 NDN in Local Area Networks

While the basic NDN architecture applies to any network environment, local area networks (LANs) are of particular interest because of their prevalence on the Internet and the relatively low barrier to deployment. If running NDN is **easy** and **beneficial** to applications, NDN can be deployed in LANs with no external coordination and much less effort compared to wide-area Internet. In the long run, as more and more LANs are NDN-enabled, the deployment may grow from the network edges towards the core, bringing more benefits to applications. Therefore in the past year we look deeper into various research problems in running NDN in LANs, proposing name-based packet filtering at network interface cards, a secure and efficient self-learning strategy for switched Ethernet, and an incremental deployment path for NDN.

5.4.1 NDN-NIC: Name-based Packet Filtering

On single-hop shared media, such as wireless networks, signals are transmitted to every node within radio range, and then each node filters incoming packets, accepts those of interest, and discards the rest. Traditionally packet filtering is conducted at the network interface cards (NIC) based on the destination MAC address, which is not applicable to NDN traffic because NDN packets do not carry source or destination addresses. The current implementation of NDN over Ethernet is to multicast NDN traffic to all NDN nodes, and let NDN software conduct name-based filtering in user space. Compared to filtering on NIC, user-space filtering not only incurs significant overhead to the main CPU, but also increases system power consumption because irrelevant packets make awaken the system.

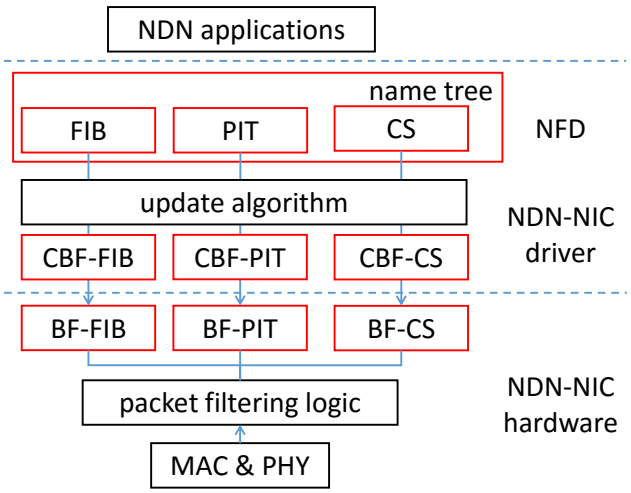


Figure 5.6: NDN-NIC overall architecture

In order to reduce CPU overhead and system power consumption while keeping NDN's benefits, we propose **NDN-NIC** [7], a network interface card that can filter out irrelevant NDN packets based on their names. The main technical challenge is how to support scalable name-based filtering using only a small amount of on-chip memory. We propose to store name-based packet filtering rulesets in Bloom filters (BFs) on NDN-NIC and conduct packet filtering based on these BFs. The NDN-NIC design (Figure 5.6) consists of two components:

- The **NDN-NIC hardware** performs name-based filtering based on the packet filtering logic and three BFs: BF-FIB, BF-PIT, and BF-CS. When a packet arrives, the packet filtering logic queries the

BFs with the packet name. An incoming packet is delivered to NFD if any BF query finds a match, otherwise the NIC drops the packet.

- The **NDN-NIC driver** maintains three Counting Bloom Filters that are updated in response to changes in NFD's FIB, PIT, and CS tables. CBFs can support name removal but need more memory than regular BFs, so they are maintained in software, while their contents are synchronized to the more compact BFs on hardware.

Bloom Filters support exact-match membership tests, but NDN's name matching is more than exact match.

- When comparing an Interest name and a Data name, they are a match if the Interest name is the same as or a prefix of the Data name. This happens when we look in the CS for an incoming Interest, or the PIT for an incoming Data.
- When comparing the names of two Interests or two Data, they match if the names are exactly the same. This happens when we look in the CS for an incoming Data, or PIT for an incoming Interest.
- When looking up the FIB for an incoming Interest, it is a match if the FIB entry is the longest prefix of the Interest name, i.e., longest prefix match.

Therefore, we need a separate BF for each table of CS/PIT/FIB, and use different lookup method on each BF.

A unique property of the Bloom Filter is the guarantee of no false negatives, which means an NDN-NIC will not drop any packet that it should accept. However, as a BF may yield false positives, some unwanted packets may pass the filter and reach the system. The technical challenge is to minimize NDN-NIC's false positive rate while also minimizing memory usage on the NIC and CPU overhead in maintaining the data structures. The naive approach, **Direct Mapping** adds every name in a table to the corresponding BF. While this works for BF-FIB and BF-PIT because they usually only contain hundreds of entries, BF-CS would have hundreds of thousands of names due to the large amount of Data cached in the CS, resulting in high probability of false positives. We propose two optimizations to reduce the number of names in the BFs so as to improve filtering accuracy:

- **Basic CS** exploits the overlap between FIB and CS, and reduces BF-CS usage by not adding names already covered by FIB entries. This optimization does not introduce more false positives, but it is effective only if there is overlap between FIB and CS, which happens only on producer nodes.
- **Active CS** aggregates CS names into fewer shorter prefixes in BF-FIB However, this would cause "prefix match false positives" because these shorter prefixes are less accurate. The extent of name aggregation is a trade-off between two types of false positives.

We evaluate NDN-NIC with different Bloom filter settings and update algorithms using an NFS trace collected from a department network, and observe its filtering accuracy, CPU usage, and BF update overhead on hardware. Simulation shows that, with 65536-bit BF-FIB & BF-CS, and 256-bit BF-PIT (16.03KB total size), NDN-NIC can filter out 96.30% of received packets, and reduces CPU usage by 95.92% compared to a regular NIC. The above accuracy is achieved with a small hardware overhead of 6.72% extra clock cycles for BF updates, compared to clock cycles spent on processing outgoing packets.

Bloom filter size directly affects filtering accuracy of NDN-NIC. Figure 5.7 shows the percentage of delivered packets with different BF sizes using Direct Mapping (DM) algorithms. With DM, having larger BFs significantly improves filtering accuracy, because there are fewer BF false positives with the same number of names added. Since CS is the largest among the three tables, increasing BF-CS size gives the most benefit. We observed a similar trend with the Basic CS algorithm.

Figure 5.8 shows the estimated CPU usage with different BF sizes under different update algorithms, where CPU usage is measured in terms of memory accesses: we count how many name tree nodes are accessed during packet processing and name tree updating, and use that number to represent CPU usage. Basic CS performs only slightly better than DM, because in our traffic trace, few CS entries are covered by FIB entries registered by local producers. On the other hand, Active CS has much less CPU overhead than DM and Basic CS with smaller Bloom filters; the CPU usage of Active CS with 4096-bit BF-FIB/BF-CS is only slightly higher than that of Basic CS with 1048576-bit BFs. However, with larger BFs, Active CS has

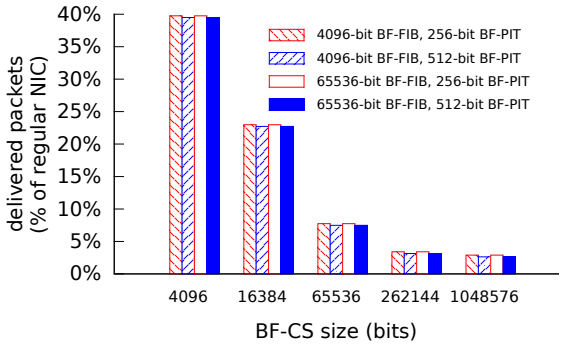


Figure 5.7: Effect of Bloom filter size, Direct Mapping

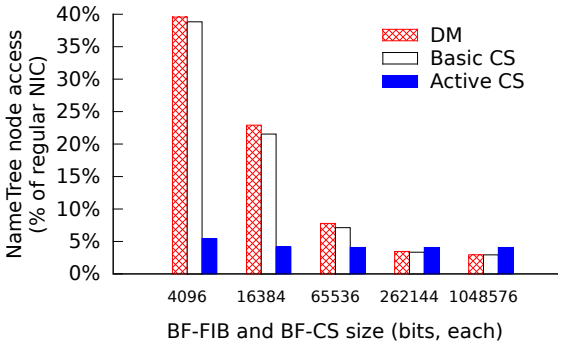


Figure 5.8: Comparison among update algorithms

somewhat higher CPU usage than DM and Basic CS, because BF false positives are already low with little room for further reduction, and as a result, the name tree updating cost exceeds the potential saving on packet processing. Considering the limited memory resources on NIC, Active CS is better than the others.

Our simulation also shows that the impact of Bloom filter updates on hardware packet processing is minor. Assuming a NIC can process one byte of outgoing traffic per clock cycle, and each BF update command consumes 8 clock cycles, we estimate that the overhead of BF updates is between 6.21% and 9.45%. Larger BF sizes tend to require more updates. Active CS incurs less BF update overhead than DM and Basic CS, because many CS entry insertions and deletions occur under BF-FIB prefixes and do not require updates to BF-CS.

5.4.2 NDN Self-learning

Switched Ethernet adopts a flood-and-learn procedure to communicate over multiple hops. An Ethernet switch broadcasts the first packet with an unknown path across the network. When a response packet returns, a forwarding table entry is created toward the destination, so that future packets will only need unicast. Using a similar idea, we proposed NDN Self-learning [8]: NFD floods Interests that do not match an entry in the FIB, and learns forwarding paths through observation of Data return paths, which are then inserted into the FIB. However, this still leaves unresolved the issue of how to determine the name prefix of an inserted FIB entry from the Data name.

Prefix Granularity Having an accurate FIB prefix is critical for the performance of NDN self-learning. Given the names of the broadcast Interest and the returned Data, what prefix granularity should be used in the FIB entry for a learned path? Having an accurate FIB prefix is critical for the performance of NDN self-learning. We have identified three potential solutions to the prefix granularity problem:

- **k-shorter Prefix**: derive the FIB entry prefix from the Data name, removing the last k components. One issue with this solution is that the number of name components to remove (k) is highly dependent on the naming scheme used by the application. In a general-purpose network, it is impossible for forwarders to understand the naming scheme of every application and adjust k accordingly. Therefore, there will always be some FIB entries with prefixes that do not match the producer's prefix, negatively impacting network performance. Since unnecessary flooding does less harm than mis-forwarding Interests to the wrong producer, k is often conservatively set to a small value.
- **FIB Aggregation**: aggregate related prefixes into more generic prefixes over time. In this solution, NFD initially sets FIB prefixes conservatively, such as the Data name minus one component. Then, if most FIB entries under a common prefix point to the same nexthop, they can be aggregated into a single entry at the shorter common prefix. During our investigation of this method, we found that

there is significant computational overhead in implementing the aggregation algorithm, and that the optimal parameters of the algorithm differ with the naming structure of each application.

- **Prefix Announcements**: the producer explicitly informs the network of the prefix it serves. This solution is implemented by having the producer attach a **prefix announcement** as a link-layer header on a Data packet sent in response to a flooded Interest. The prefix announcement is itself a Data packet, containing the prefix served by the producer. This solution is straightforward and allows FIB prefixes to adapt to applications' differing granularities.

We chose prefix announcements as our final solution.

After examining the granularity problem that is common to all networks, we developed **NDN self-learning**, a forwarding scheme applying self-learning to local area networks and switched Ethernet in particular that builds forwarding tables in the data plane with low overhead, recovers quickly from link failures and other path problems, and makes use of off-path caches for Internet contents.

Minimizing Flooding Overhead Compared to switched Ethernet, NDN has built-in loop freedom through the use of nonces. Therefore, NDN self-learning can flood Interests without requiring a protocol like FSTP to prevent bridge loops. However, Interest flooding consumes network bandwidth, and incurs CPU processing overhead at end hosts that receive the flooded Interests but do not have the content. The key to minimizing overhead is to flood less often and in smaller regions. In keeping with this goal, only the consumer can initiate Interest flooding. Flooding should be initiated for any Interest with an unknown path or where the previously learned path has failed. A *discovery tag* field on every Interest indicates whether it is part of an intentional Interest flood.

If multiple consumers concurrently request content under the same name prefix, although each joining consumer will initiate flooding of its first Interest, when NFD receives a discovery Interest but already knows a working path, the Interest will be retagged as *non-discovery* and forwarded along the known path via unicast, effectively absorbing Interest flooding from joining consumers. When a Data packet comes back to the same forwarder, the original prefix announcement that was used to create the FIB entry will be attached onto the Data packet, allowing new downstream nodes to learn the producer's prefix.

Fast Reaction to Link Failures We use a variant of Bidirectional Forwarding Detection (BFD) at the link layer to detect link failures between adjacent nodes. If an Interest arrives and the nexthop in FIB entry has a link failure, NFD can forward the Interest on an alternate path if there is one, or flood the Interest if it is tagged as "discovery", or return a *Nack* against the non-discovery Interest so that the consumer can initiate flooding. This allows NDN self-learning to react to link failures at packet time scale.

Caching of Internet Contents NDN self-learning supports the retrieval of Internet contents via a gateway router, which announces itself as the "producer" of the "`/`" prefix through a prefix announcement attached to the returning Data packet. Since this FIB entry will match every Interest requesting Internet contents, all Interests for Internet content will be forwarded as non-discovery, allowing use of only caches on the path between consumers and the gateway. Although these on-path caches can satisfy some redundant Interests, requiring retrieval of less content from the Internet, the caches in network switches and the gateway router have limited capacity and handle a high volume of traffic, and therefore can only offer limited assistance in this regard. However, abundant cache capacity on other, off-path nodes and end hosts may store data for longer.

To utilize off-path caches, all nodes remember where they forwarded each Data packet they processed, recording the downstream face as a potential off-path cache in the Data's CS entry. When a node must evict a CS entry, instead of deleting it altogether, the node converts it to a stub entry, containing the Data name and the locations of potential off-path caches, but not the Data payload. Since a name is much smaller than a Data payload, a node can store many more stub entries than regular CS entries.

When an incoming Interest matches a CS stub entry, the forwarder examines potential off-path cache locations, and decides whether to divert the Interest by predicting whether the Data is still cached on the off-path cache. We use a simple heuristic for this prediction: a Data is less likely to have been evicted if few

Data packets crossed that downstream in the meantime.

When an off-path node receives a diverted Interest, the forwarder on that node queries its CS to look for a match. If the forwarder finds a regular CS entry, it returns the Data packet to the diverting node, which returns the Data to the downstream. If the forwarder finds a stub CS entry, it passes the Interest to the downstream most likely to still have the Data, chosen with the same heuristic as above. If there is no match in the CS, the off-path node returns a Nack to the diverting node, which then forwards the Interest toward the gateway.

We conducted evaluations of NDN Self-Learning using both real and synthetic traffic, detailed in [8]. We showed that self-learning can more accurately learn FIB prefixes from producers, leading to more efficient bandwidth usage. Additionally, we showed that NDN self-learning can allow recovery from link failures without waiting for FIB entry expiration or incurring the overhead of FSTP convergence, due to the use of Nacks [8].

5.4.3 NDN Incremental Deployment in Ethernet

Existing NDN deployments are overlays using TCP, UDP, or IP tunnels, which usually require manual configuration and maintenance of tunnel endpoints. Tunnels limit NDN's capability to utilize the underlying broadcast media. In order to take full advantage of data-centric communications and ease the deployment process, we advocate to deploy **NDN directly over Ethernet**. We set forth the following design goals:

- **Co-existence with IP Traffic**: The network should be able to support IP traffic and applications. The common mechanisms, such as address-based IP and Ethernet packet forwarding, Ethernet's Spanning Tree Protocol (STP), MAC learning mechanism and the address resolution protocol (ARP), etc. should be able to run without any change or performance penalty.
- **Native NDN Support**: For NDN traffic, the network should provide native name-based forwarding instead of relying on overlays or tunnels. Mechanisms such as in-network caching, multicast, and forwarding strategies should all work natively among deployed NDN nodes.
- **Incremental Deployability**: NDN deployment should be carried out in a gradual fashion. The deployment process may take years, during which both NDN and IP traffic should be supported, and the more NDN deployment, the more benefits to NDN applications.
- **General Applicability**: Given the prevalence of Ethernet, this paper focuses on the deployment of NDN in Ethernet LANs. However, we intend the principle and main approach to be a general solution that, after further research, can be extended to other network environments.

We propose a *Dual-Stack switch* (*D-switch*)[3], which provides name-based forwarding for NDN traffic and address-based forwarding for conventional traffic such as IP. As a contrast, conventional Ethernet switches (*E-switch*) only support address-based forwarding, while pure NDN switches (*N-switch*) only support name-based forwarding. Identifying NDN traffic by the EtherType field in the Ethernet header, D-switches act as layer-3 switches and process these packets based on content names carried in NDN header, providing native NDN features such as in-network caching, loop-free forwarding, and multicast. D-switches do not need to be restrained by Ethernet's spanning tree protocol because NDN can detect and break forwarding loops. NDN traffic can thus utilize all physical links in the LAN. For non-NDN traffic, a D-switch acts as a layer-2 switch and forwards these packets based on destination MAC address.

We focus on three scenarios for NDN deployment in Ethernet LANs [10]: (i) NDN-enabled hosts, which may emit both NDN and IP traffic, and all E-switches, (ii) NDN-enabled hosts and all D-switches, and (iii) a hybrid network with both D-switches and E-switches. In particular, we focus on the hybrid scenario, where the network contains both D-switches and E-switches, because this is the most important stage of NDN deployment and it raises several interesting technical issues in how to orchestrate the two types of switches. In a hybrid LAN, D-switches and E-switches may be placed arbitrarily anywhere in the network. A D-switch may have neighbors of E-switches, D-switches, or a mix of both. The challenge is to design mechanisms to support name-based forwarding while co-existing with address-based forwarding within the same LAN.[4] For

[3]D-switches can be implemented on general programmable platforms and software-based switches.

[4]Note that the NDN traffic frames still carry MAC address at layer-2 in order to travel among E-switches.

example, a D-switch may forward NDN frames to a link considered disabled by Ethernet's STP, because it makes forwarding decisions for NDN traffic only based on the content names without considering layer-2 protocols. If a neighbor E-switch gets the frame from the STP-disabled link, it may drop this frame, which results in conflicts between the two types of switching.

To make sure the two types of switches can co-exist without any conflict, we propose a number of mechanisms when D-switches forward NDN packets. First, D-switches employ NDN self-learning to populate their forwarding tables on demand. This keeps the plug-n-play feature of the traditional Ethernet as well as enhancing the Ethernet by allowing use of non-spanning-tree links. Second, when a D-switch forwards an NDN packet, it will update the source and destination MAC addresses to reflect the current hop. In other words, the MAC addresses of each NDN packet will change at each D-switch hop. Third, when an NDN packet has traversed an STP-disabled link, the receiving D-switch may or may not put this packet back on to STP links, depending on the tradeoff between path length and number of duplicate packets. Fourth, the cache of D-switches should selectively serve Interests to stabilize the content of forwarding tables.

Furthermore, we investigate the problem of how to place D-switches in such a hybrid network so as to maximize the benefits from deploying NDN. We identify two heuristics, pair-based and connectivity-based, to optimize different performance goals. Simulations show that as NDN deployment grows, the network starts to spread traffic over more links, take shorter paths, and experience less traffic load.

5.4.4 NDN Over WiFi Direct

To support NDN applications to effectively run over WiFi-Direct links, during the last year we have designed a dedicated name discovery protocol. This protocol provides a means for WiFi-Direct compatible devices to connect to one another, and to exchange and maintain information about the available NDN Data prefixes. We have also implemented this protocol as part of NFD-Android (Figure 5.9), which relies on UDP tunnels over WiFi-Direct links, because the non-rooted Android platform does not provide direct access to links.

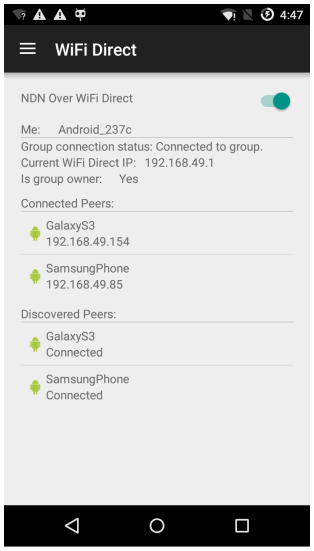


Figure 5.9: WiFi-Direct Support for NFD-Android

The protocol establishes not only connectivity but communication between multiple WiFi Direct enabled devices. Devices connect at the link layer via WiFi-Direct, and communicate using NDN semantics, e.g., face creation, packet exchange, etc. After successful group formation (i.e., after two users initiate WiFi-Direct connectivity towards each other), devices connected via WiFi-Direct will register their local-hop probe routes, e.g., "`/localhop/wifidirect/192.168.49.1`" (The IP address is provided by the Android framework and is unique for each node in the WiFi-Direct group). Whenever a node receives a probe interest, it enumerates all relevant entries in a FIB and returns the discovered set. The set contains irrelevant entries including "`/localhost/*`" prefixes, which cannot be retrieved over the network due to name-based scope control, and entries that point toward the face of the incoming Interest. Whenever a node receives a response, it registers the set of discovered prefixes with local NFD, allowing applications to effectively retrieve data from nearby devices. The probe procedure periodically repeats, ensuring up-to-date information about the available data within the WiFi-Direct network, including transitive knowledge (e.g., in cases where multiple clients connect to the same group owner). Note that locally registered prefixes use a pre-configured timeout (soft state) and require periodic refresh.

With the help of this protocol, we ran successful demos of the NDN-Whiteboard Android application last year, as well as developed and experimented with ChronoChat-Android application (Section 2.2.6). Our experiments revealed aspects of the Android's WiFi-Direct framework that complicate device-to-device communication. Android devices have a big range of quality of WiFi-Direct support: some phones refused to connect, some lost connectivity state soon after connection, others were stable. The WiFi direct framework has hard limitations on the number of connection attempts, which requires the user select an available nearby devices to connect to. This was contrary to our initial expectation of automated connection and communicate

with nearby systems. Other remaining issues include increasing the stability of our implementation and refining the probe response format (currently a simple list of new-line-separated prefixes).

5.5 NDN Forwarding Daemon (NFD)

We continued to develop the core NDN networking software, NFD, and its related library and tools. In the past year we made a major release (version 0.5.0[5]) and a minor release (version 0.5.1[6]) with a number of new features and many bug fixes. NFD now runs on Linux, FreeBSD, Mac OSX, Android, DD-WRT/OpenWRT (home routers), Raspberry Pi, and a couple of other embedded platforms, as well as in virtualized environments. The development of NFD continues to use an open source and distributed model, involving the broader community. We use Redmine for issue tracking, Gerrit for code review, and Jenkins for automatic build and continuous integration. NFD has over 30 contributors from 10 different institutions, as well as several contributions outside the NSF-funded NDN team. To coordinate NFD development, we use the NFD developer mailing list with more than 100 members currently and conference calls twice weekly. We also continually update the NFD Developer's Guide and other related documents to provide implementation details and suggested development practices for new developers and researchers. The major NFD features developed in the past year include the following:

Forwarding

- Introduced configurable policy regarding the admission of unsolicited data packets into the content store. Currently the available policies are:
  - DropAllUnsolicitedDataPolicy (the new default): drop all unsolicited data packets
  - AdmitLocalUnsolicitedDataPolicy (the old default): allow unsolicited data packets from local applications to be cached (e.g., with a lower priority), drop all other unsolicited data.
  - AdmitNetworkUnsolicitedDataPolicy: allow unsolicited data packets from the network to be cached (e.g., with a lower priority), drop all other unsolicited data.
  - AdmitAllUnsolicitedDataPolicy: cache all unsolicited data packets.
- Optimizations of various tables and forwarding pipeline, including reduced usage of shared_ptr, which led to considerable performance improvements.
- Introduced tables.cs_policy config option to configure cache policy.

Forwarding Strategy

- Added Adaptive SRTT-based Forwarding strategy
- Introduced strategy parameters that can be specified when selecting a strategy for a namespace
- Changed Interest pipeline and Strategy API to give strategies an opportunity to pick an outgoing Interest that matches the Interest table entry.
- Refactored localhop scope enforcement in strategies. to improve accuracy and flexibility.

Face System and Link Adaptation Service

- Introduced mechanism to update properties (e.g., flags, persistency) of an existing Face.
- Added configuration options to Ethernet and UDP multicast faces.
- Added support for permanent persistency in TcpTransport.
- Continued development of NDN Link Protocol (NDNLPv2) to support fragmentation and reassembly, as well as specifying cache policies in NFD. Overhauled face management system to allow disabling and enabling NDNLP and other features.

[5] `http://named-data.net/doc/NFD/current/release-notes/release-notes-0.5.0.html`
[6] `http://named-data.net/doc/NFD/current/release-notes/release-notes-0.5.1.html`

- (ongoing) Implementing packet loss detection and limited retransmission mechanism to improve packet delivery reliability on a single link.
- Update GenericLinkService to encode and decode the NDNLPv2 CongestionMark field as the congestion signaling mechanism.

Tools

- Refactored and extended nfdc tool, which now supports a new command-line syntax and retrieval of status datasets. nfdc has become a universal instrument to query and change NFD state.
- In ndn-autoconfig, added the Find (geographically) Closest Hub stage (NDN-FCH) using a deployed service.
- Refactored *ndnchunks*, a file transfer program, to include different ways of adjusting pending Interest window size. We implemented TCP-Reno-like algorithm to give the application a way to respond to congestion. Implementation of other rate adjustment mechanisms are under way.

Routing

- Completed initial implementation of route re-advertise feature.
- Disabled *autoreg* on NDN testbed, replaced it with remote prefix registration via auto prefix propagation.

Management

- Refactored face management system. Split function of the monolithic FaceManager class among newly introduced FaceSystem class and other protocol factories. FaceSystem class is the entry point of NFD's face system and owns the concrete protocol factories, created based on face_system section of the NFD configuration file.
- Introduced FACE_EVENT_UP and FACE_EVENT_DOWN notifications to notify the system about face status changes.

References

[1] Hila Ben Abraham and Patrick Crowley. Controlling strategy retransmissions in named data networking. In *Proceedings of the ACM IEEE Symposium on Architectures for Networking and Communications Systems*, 2017.

[2] Huichen Dai, Huichen Dai, Bin Liu, Haowei Yuan, Patrick Crowley, and Jianyuan Lu. Analysis of tandem pit and cs with non-zero download delay. In *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[3] Vince Lehman, Ashlesh Gawande, Rodrigo Aldecoa, Dmitri Krioukov, Lan Wang, Beichuan Zhang, and Lixia Zhang. An Experimental Investigation of Hyperbolic Routing with a Smart Forwarding Plane in NDN. In *Proceedings of the IEEE IWQoS Symposium*, June 2016.

[4] Vince Lehman, A K M Mahmudul Hoque, Yingdi Yu, Lan Wang, Beichuan Zhang, and Lixia Zhang. A secure link state routing protocol for NDN. Technical Report NDN-0037, NDN Project, January 2016.

[5] NDN Project Team. ASF strategy does not check whether out record exist before calculating RTT. `https://redmine.named-data.net/issues/3829`.

[6] Klaus Schneider, Cheng Yi, Beichuan Zhang, and Lixia Zhang. A practical congestion control scheme for named data networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking (ICN)*, 2016.

[7] Junxiao Shi, Teng Liang, Hao Wu, Bin Liu, and Beichuan Zhang. NDN-NIC: name-based filtering on network interface card. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking (ICN)*, 2016.

[8] Junxiao Shi, Eric Newberry, and Beichuan Zhang. On broadcast-based self-learning in named data networking. In *Proceedings of IFIP Networking*, 2017.

[9] Ivan Voitalov, Rodrigo Aldecoa, Lan Wang, and Dmitri Krioukov. Geohyperbolic Routing and Addressing Schemes. `http://arxiv.org/abs/1703.00520`, 2017.

[10] Hao Wu, Junxiao Shi, Yaxuan Wang, Yilun Wang, Gong Zhang, Yi Wang, Bin Liu, and Beichuan Zhang. On incremental deployment of named data networking in local area networks. In *Proceedings ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, 2017.

[11] Haowei Yuan, Patrick Crowley, and Tian Song. Enhancing scalable name-based forwarding. In *Proceedings of the ACM IEEE Symposium on Architectures for Networking and Communications Systems*, 2017.

Chapter 6

Evaluation Tools

Contributors	
PIs	Beichuan Zhang (Arizona), Van Jacobson & Lixia Zhang (UCLA), Lan Wang (Memphis), Christos Papadopoulos (Colorado State University), Patrick Crowley (Washington University)
Grad Students	Junxiao Shi, Weiwei Liu (Arizona); Yingdi Yu, Wentao Shang, Spyridon Mastorakis (UCLA), Steve DiBenedetto, Chengyu Fan (Colorado State), Haowei Yuan, Hila Ben Abraham, Adam Drescher (Washington University)
Staff	John DeHart, Jyoti Parwatikar (Washington University), Ashlesh Gawande, Vince Lehman (Memphis)
	Researcher: Alex Afanasyev (UCLA)

We continued to refine and enhance our emulation and simulation toolsets to support the demands from ever increasing usage, both by NDN project team members as well as by an increasing number of users from network researchers community at large.

6.1 Mini-NDN

The development and maintenance of Mini-NDN has continued since our last reported version of 0.1.1. We released version 0.2.0 on August 18, 2016; its most major feature was the enabling of NLSR security. Now Mini-NDN automatically configure security for NLSR with a simple command line argument, allowing extensive testing of NLSR before deployment on the testbed (where security is now turned on by default). This release included minor bug fixes and changes to improve usability such as specifying the NLSR's configuration file to correctly identify different NLSR processes.

At the 3rd NDN hackathon, we introduced the next major feature of Mini-NDN i.e. the cluster edition, to support unprecedented scale of emulation. This improvement will help support larger-scale hyperbolic experiments. we also worked on a Gerrit (code review system that we use) bot that can pull NLSR changes, run integration tests in Mini-NDN and post the review of failure or success. This bot has enabled us to quickly detect breakage of NLSR functionality due to an ndn-cxx/NFD API change, as it tests a change with the latest version of ndn-cxx and NFD. We integrated these updates into Mini-NDN version 0.3.0 on March 3, 2017.

Post release v0.3.0, we worked on the Mini-NDN WiFi project in the 4th NDN hackathon and produced a proof of concept simple experiment based on a Mininet fork, Mininet-Wifi. More work needs to be done on it with respect to stability, features, and researching on how Mininet-Wifi works before this change can be incorporated into Mini-NDN.

6.2 ndnSIM

In the last year we continued support, development, and extensions of ndnSIM, the NDN simulation framework based on NS-3 network simulator. ndnSIM is heavily based on the real codebase of the NDN Forwarding Daemon (NFD) and ndn-cxx library, providing multiple helper components and making NFD/ndn-cxx run in the simulated environment (Figure 6.1). We made two releases (ndnSIM versions 2.2 based on NFD/ndn-cxx version 0.4.1 and ndnSIM 2.3 based on version 0.5) that incorporated updates to the at the time latest versions of NFD and ndn-cxx codebases, as well as multiple updates to the ndnSIM-specific adaptations. In addition to all of the new features directly inherited from NFD/ndn-cxx codebases, the latest ndnSIM release included the following updates:

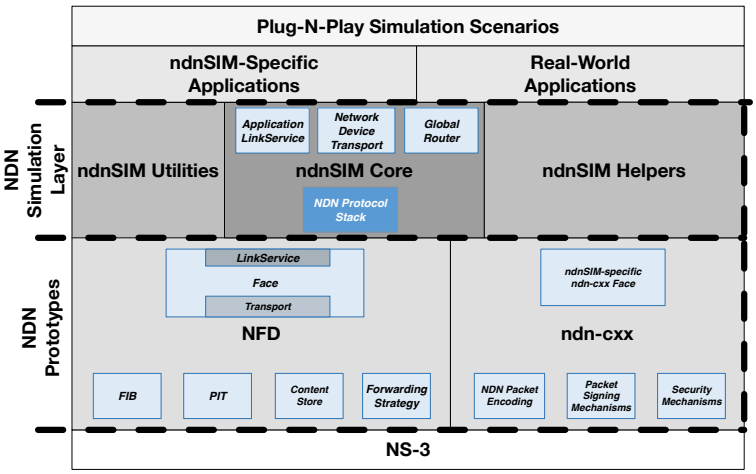


Figure 6.1: ndnSIM framework

- Full support of NDNLPv2, bringing network-level NACK functionality. This feature came at the cost of losing direct support of the NS-3 related packet tags, which now need to be proxied through NDNLPv2.
- NACK-tracing by the network-layer tracers.
- Added a helper to allow configuration of the simulated nodes's NetworkRegionTable
- Internal refactoring to use the ndnSIM-specific transport implementation (`ndn::L3Protocol`, `ndn::StackHelper`, `ndn::LinkControlHelper`, `ndn::GlobalRoutingHelper`, `ndn::Consumer`, `ndn::Producer`).
- The NetDevice address is now represented as a LocalUri instance for NetDevice-based Faces.
- Configurability of NFD's managers, i.e., ability to disable RIB, StrategyChoice, and other managers if they are not used a simulation scenario.
- Updates of the ndnSIM documentation

We released a new versions of the ndnSIM 2 Technical report [2] and submitted a paper to CCR describing evolution of the ndnSIM over the last six years and lessons learned from this process [3]. The ndnSIM mailing list has grown to 475 subscribers with ongoing active discussions.

6.3 NDN Testbed: Deployment, Management, Expansion

The NDN team at Washington University operates and manages the global NDN testbed, which as of April 2017, had 33 nodes in 15 countries. The Washington University team also manages integration testing and deployment activities. They develop and maintain monitoring tools for the NDN Testbed [1], including both a Cacti-based node resource (CPU, memory, etc.) monitoring facility and an NDN-based real time testbed usage mapping tool. A web page at http://ndnmap.arl.wustl.edu displays the usage data. In the past year, major testbed developments, changes, and deployments include: deployment, testing, debugging, and evaluation of hyperbolic routing (Section 5.1), guest prefix registration at NDN gateway nodes, and forwarding strategy evaluation and management.

6.3.1 Lessons Learned from Testbed Use

Having access to an active testbed, one running real code on real systems distributed around the world, allows us to validate our work in practical ways. In the past year, use of the testbed has surfaced and emphasized challenges in areas such as forwarding strategy, guest certificates, and guest name prefixes (in support of dynamic applications) that had not previously been fully understood or appreciated until deployment.

The testbed helped us identify a problem in NLSR v0.3.1 released on Jan. 22, 2017 – we changed the sync and LSA name prefixes used by NLSR but did not include an associated update to NLSR's security rule in the configuration file. Our automatic testing bot built using Mini-NDN was not able to catch the problem, as Mini-NDN was mis-configured to output an exit code of 0, rather than 1, on test failure and the bot accepted the breaking change as working. As soon as NLSR was deployed on the testbed, it failed to converge and issued errors about failed validation. Fortunately, we were able to re-deploy NLSR soon with a minor update of the configuration file. We fixed the problem in NLSR's configuration file in the next minor release of v0.3.2. Mini-NDN was also fixed and new static unit tests were added to NLSR so that this problem can be caught in the future. over yet as a parallel experimental testbed in the same network was leaking LSAs into the main testbed. This is now a reported task to have NLSR not accept LSAs from another networks (Issue 3948).

References

[1] Zeév Lailari, Hila Ben Abraham, Ben Aronberg, Jackie Hudepohl, Haowei Yuan, John D. DeHart, Jyoti Parwatikar, and Patrick Crowley. Experiments with the Emulated NDN Testbed in ONL. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*. ACM, 2015.

[2] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM 2: An updated NDN simulator for NS-3. Technical Report NDN-0028, Revision 2, NDN, November 2016.

[3] Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. On the evolution of ndnSIM: an open-source ecosystem for NDN experimentation. *In submission to CCR*, 2017.

Chapter 7

Impact: Education

Contributors	
PIs	Christos Papadopoulos (CSU), Lan Wang (Memphis), Beichuan Zhang (Arizona), Van Jacobson, Jeff Burke, Lixia Zhang (UCLA)

7.1 University of Arizona

Beichuan Zhang included NDN material in a total of two courses. At the introductory graduate level, a fall 2016 graduate course (CS 525 - Principles Computer Networking) dedicated two lectures on basic NDN concepts and mechanisms. Note this represents a substantial contribution to an already packed course that covers the broad field of computer networks. The most substantial work was in CS 630 - Advanced Topics in Information-Centric Networking. This advanced graduate course focused on paper reading, term projects, and in-class presentations and discussions on NDN-related research.

Following our general approach of both theoretical discussions and hands-on projects, CS 630 included a number of research projects:

- A cross-layer protocol for collecting sensing data in an NDN sensor network
- A Bloom filter based DDoS defense mechanism in NDN
- An analysis of caching opportunities in a Hadoop cluster running various MapReduce benchmarks
- A simulation study comparing the performance of TCP/IP and NDN under host mobility
- A study of current video CDNs (Content Distribution Networks) and the potential of NDN in this area

In addition to developing new course material, the PI also involved one undergrad student in NDN development. The student has been maintaining the Jenkins service for NDN code development, and contributed to the design and implementation of a number of NFD features, including NFD management, the link adaptation layer, the face system, and so on.

7.2 University of Memphis

Lan Wang supervised a visiting student Alejandro Torres from Spain to do an undergraduate thesis on NDN routing. He added routing statistics collection to NLSR and used this capability to evaluate routing performance. Wang also supervised three U. Memphis undergraduate students, Nick Gordon, Ashlesh Gawande and Damian Coomes. They worked on NLSR, hyperbolic routing, and Mini-NDN. Nick and Ashlesh received their B.S. degree in Dec. 2016 and continued to work on the NDN project as software engineers.

7.3 Colorado State University

Christos Papadopoulos continued to use NDN in his graduate networking class (CS557). The class includes at least two lectures on NDN, one describing the architecture and its applications, and one that introduces students to NFD and the programming model in NDN (the equivalent of a socket tutorial). Students have one major project in NDN, where they are asked to re-implement a BitTorrent type application that they had already implemented in IP. The students come up with a naming scheme and build a distributed virtual network to share files. The reason for implementing the same project in NDN and IP is to get a first hand experience of implementing applications in the two architectures. Students also have the option to implement the project on lab machines or DETER.

Students always comment how much easier is to implement the project in NDN (they get about half the time for NDN than IP). I always get requests for extensions for the IP project, but rarely for NDN.

Christos Papadopoulos also offers NDN projects in a research seminar class. One such project that completed this semester is to use NDN to control devices in a room, but only when the user is physically present in the room. The project used the name space "thisRoom" to designate Interests and Data that should be limited to devices in the room. In addition, the project used a novel method to distribute a rotating room key to users so they can prove they are physically present in the room, but lose access when they leave the room. The project resulted in a poster at an internal student exhibition and a paper submission.

7.4 UCLA

Lixia Zhang covers three courses at UCLA, one introduction to networking course for undergraduate students (CS118) and two graduate level courses (CS217A & B). Upon students' enthusiastic requests, Zhang offered a lecture on introduction to NDN in CS118 the first time in fall 2016 which was well received. A number of students followed her to take CS217A ("Internet Architecture & Protocols") in winter 2017 quarter, to learn more about Internet architecture and about NDN. More than half of the CS217A students developed term project proposals on NDN research topics, which they carried over to CS217B in Spring 2017 quarter to finish them.

CS217B, "Advanced Topics in Internet Research", is a graduate seminar course focusing on the NDN architecture design and application development. The course traces back historical literature from a series of research papers spanning the last few decades (e.g. [1–3,5]) to identify the origin of NDN's core architectural ideas. Students also learn about other architectural designs under NSF's FIA program, mainly eXpressive Internet Architecture [6] and Mobility First [7]. Before taking the classes, "Internet architecture" seemed to most students a rather abstract term; through CS217A &B's discussions, they not only grasped the basic concepts but also explored in-depth issues regarding how to evaluate an architecture, how to judge design tradeoffs, and effective ways to roll out new architectures.

In parallel to in-class discussions, the students conducted research projects on NDN design and development. The students finished the following projects during Spring 2016 quarter.

- Evaluation of the design and performance of ChronoSync 2.0 using ndnSIM (see also Section 4).
- Local NDN hub discovery for automatic NDN network attachment.
- Implement nTorrent: A Peer-to-Peer file download system over NDN (now a pending conference submission).
- Mitigating content poisoning attacks in NDN, extending results from [4].
- Implementing NDN Sync Protocol on RIOT OS for Internet of Things.
- Secure poker games that used NDN's security primitives to address a difficult application scenario.
- NDN Open mHealth pilot applications, as part of the NDNfit (Section 2.1.2) development efforts.

The first project turned into a thorough evaluation of the ChronoSync implementation and led to fixing performance issues that affected other NDN projects relying on ChronoSync. The second one turned to a nice tool facilitating NDN autoconfiguration and has been incorporated into the NDN codebase. Both of

them became the student's master projects that fulfilled the M.S. degree requirements.

The research projects being carried out in Spring 2017 quarter include the following:

- Mitigation of Bad Blood in Named Data Networks, a continuing effort in exploring effective mitigations to content poisoning.
- NDN Android Event Scheduler: Meet it up!
- Attribute-based access control in virtual organizations (a collaboration with Aerospace researchers)
- MIDI NDN: Real time musical instrument control using NDN on macOS
- Authenticated denial of existence for NDNS (scalable solution to prove the non-existence of given names)
- ndnMouse: Secure Control Interface for a PC Using a Mobile Device
- NDN over WiFi-direct for Linux
- NDN over Bluetooth

We expect these efforts to contribute to NDN's growing application collections and codebase development. Three other master students also contributed to NDN development while fulfilling their degree requirements. One developed the use of WiFi-direct to connect nearby Android devices in the absence of infrastructure; one utilized such ad hoc connectivity to enable textchat with Android phones; and a third student conducted a comparative evaluation between NDN RIOT and the IETF IoT protocol stack implementation with a focus on energy consumption, which both demonstrated NDN's advantages on constrained devices and helped us identify energy-hungry NDN operations for further improvements.

7.5 NDN Project-Wide Seminars

We continued our NDN seminar series among all participating universities. NDN seminar topics reflect ongoing NDN design and development efforts and promote inter-campus exchanges and collaborations. Speakers are graduate students from different universities. Students from different campuses rotate to take the responsibility for running the seminar series, Spyros Mastorakis of UCLA was in charge over this reporting period.

The following list shows the speakers and topics for this year's NDN seminar series.

- 06/01/2016: Klaus Schneider, "Practical Congestion Control for NDN."
- 10/05/2016: Wentao Shang, UCLA, "The Design and Implementation of the NDN Protocol Stack for RIOT-OS."
- 11/30/2016: Susmit Shannigrahi of Colorado State University, "Applying NDN to large scientific data."
- 1/12/2017: Junxiao Shi, University of Arizona, "Technical Discussion on the Self-Learning Strategy."
- 2/2/2017: Jongdeog Lee, UIUC, "SwiftNDN: NDN support for iOS."
- 2/16/2017: Ashlesh Gawande, University of Memphis, "Scalable Name-based Data Synchronization for Named Data Networking."
- 03/02/2016: Klaus Schneider, University of Arizona, "How to Establish Loop-Free Multipath Routes in Named Data Networking."

References

[1] Paul Baran. On distributed communications networks. *IEEE Transactions on Communications Systems*, 1964.

[2] David D Clark and David L Tennenhouse. Architectural considerations for a new generation of protocols. *ACM SIGCOMM Computer Communication Review*, 20(4):200–208, 1990.

[3] Steve Deering and David Cheriton. Multicast routing in internetworks and extended lans. In *SIGCOMM*, 1988.

[4] Stephanie DiBenedetto and Christos Papadopoulos. Mitigating Poisoned Content with Forwarding Strategy. *Proceedings of the third Workshop on Name-Oriented Mobility: Architecture, Algorithms and Applications (NOM'2016)*, April 2016.

[5] Sally Floyd, Van Jacobson, Steven McCanne, Lixia Zhang, and Ching-Gung Liu. A reliable multicast framework for light-weight sessions and application level framing. In *SIGCOMM*, 1995.

[6] David Naylor, Matthew K. Mukerjee, Patrick Agyapong, Robert Grandl, Ruogu Kang, and Michel Machado. XIA: Architecting a More Trustworthy and Evolvable Internet. *ACM SIGCOMM Computer Communication Review (CCR)*, 44(3):50–57, Jul 2014.

[7] Arun Venkataramani, James Kurose, Dipankar Raychaudhuri, Kiran Nagaraja, Suman Banerjee, and Morley Mao. MobilityFirst: A Mobility-Centric and Trustworthy Internet Architecture. *ACM SIGCOMM Computer Communication Review (CCR)*, 44(3):74–80, Jul 2014.

Chapter 8

Impact: Expanding NDN Community

In the last few years the NDN project has been attracting ever-increasing attention from the global networking community. During this reporting period, in addition to the two internal project retreat meetings (November 3-4, 2016 hosted by Colorado State University, and March 22, 2017 hosted by University of Memphis), the NDN team organized several activities to spread the knowledge about the NDN architecture and to facilitate the community growth.

8.1 NIST Workshop on Named Data Networking (NDN), May 31-June 1, 2016

We worked with the researchers from the National Institute of Standards and Technology (NIST) to organize a two-day workshop on Named Data Networking (NDN) at NIST Headquarters Gaithersberg, MD. This workshop attracted 141 registrations from government branches, industry, and academia around the country to exchange the understanding and latest research results on NDN and its applications to various challenges facing the Internet today, in particular in the IoT area.

NDN team members made the following presentations at the workshop.

1. Van Jacobson delivered opening keynote, "Named Data Networking and the Internet of Things".
2. Alex Halderman, "NDN: A Security Perspective".
3. Lan Wang, "Open IoT/NDN research challenges: Named Data Networking of Things".
4. Christos Papadopoulos, "Named Data Networking In Scientific Applications?".
5. Patrick Crowley, "NDN Platforms & Scalability"
6. Jeff Burke, "Driving Network Architecture through Media Applications," Panel chair.

The workshop homepage at `https://www.nist.gov/news-events/events/2016/05/workshop-named-data-networking` contains pointers to the workshop agenda as well as video recordings.

8.2 The Third NDN Community meeting, March 23-24, 2017

The Third NDN Community Meeting was held at University of Memphis, on March 23-24, 2017. The meeting provided a platform for 73 attendees from 36 institutions across 10 countries to exchange their recent NDN research and development results, to debate existing and proposed functionality in NDN forwarding, routing, and security, and to provide feedback to the NDN architecture design evolution. The meeting agenda is presented below, and meeting homepage at `https://www.caida.org/workshops/ndn/1703/` contains pointers to presentations and the video recordings.

March 23 (Thursday)

- "Past Achievements and Future Plan of the NDN Project", Lixia Zhang (UCLA)
- "Mobile, IoT and Challenged Environments" (Session Chair: Beichuan Zhang)
  - "Opportunities for NDN in Augmented Reality", Jeff Burke (UCLA)
  - "NDN-Opp: NDN in Opportunistic Networks", Seweryn Dynerowicz, Omar Aponte, Paulo Mendes (COPELABS, University Lusofona)
  - "FleetLink: NDN-Powered Low-Cost, Low-Rate, Reliable, Secure Communication for Neighborhood Solar: A Practical Approach to Lowering the Cost of Solar", Alex Afanasyev, Jeff Thompson (UCLA)
  - "Named Data Networking of Things: NDN for Microcontrollers (NDN-RIOT)", Wentao Shang, Alex Afanasyev, Lixia Zhang (UCLA)
  - "Named Data Networking of Things: Trust Management for Autonomous Data-Centric Security", Wentao Shang, Alex Afanasyev, Lixia Zhang (UCLA)
- "Data-Intensive Applications" (Session Chair: Christos Papadopoulos)
  - "Quantifying NDN's improvement to Scientific Data Management", Susmit Shannigrahi, Chengyu Fan, Christos Papadopoulos (CSU)
  - "From Crowd to Cloud: Apply Named Data Networking Principles to Manage Air Sensor Data", Yifang Zhu, Alex Afanasyev and Lixia Zhang (UCLA)
  - "Hadoop over NDN: Initial Experience and Results", Mathias Gibbens, Chris Gniady, Lei Ye, Beichuan Zhang (University of Arizona)
  - "nTorrent: Peer-to-Peer File Sharing in Named Data Networking", Spyridon Mastorakis, Alexander Afanasyev (UCLA); Yingdi Yu (Facebook, Inc); Lixia Zhang (UCLA)
- Panel: "NDN Security - Current Status and Open Issues" Chair by Christos Papadopoulos, with panelists Alex Afanasyev (UCLA), Alex Halderman (U. Michigan), Satyajayant Misra (New Mexico State University), Dante Pacella and Mani Tadayon (Verizon).
  - Dante Pacella and Mani Tadayon (Verizon), "ICN Content Security Using Encrypted Manifest and Encrypted Content Chunks"
- "All Aspects of NDN" (Session Chair: Lotfi Benmohamed)
  - "Adaptive Caching Algorithms with Optimality Guarantees for NDN Networks", Stratis Ioannidis, Edmund Yeh (Northeastern University)
  - "NDN Control Center: NDN Networking Stack and Security Enabler for Desktop Systems", Qi Zhao, Alex Afanasyev, Lixia Zhang (UCLA)
  - "NDN-Android: NDN Networking Stack for Android Platform", Haitao Zhang, Alex Afanasyev, Lixia Zhang (UCLA)
  - "Facilitating ICN Deployment with an Extended OpenFlow Protocol", Piotr Zuraniewski, Ray van Brandenburg (TNO); Borgert van der Kluit (Chess Wise); Niels van Adrichem (TNO)
- "Lightning Talks Introducing Posters and Demos" (Session Chair: Josh Polterock)
  - Demo: "Integrating Named Data Networking with Transportation Simulation", Meng Kuai, Pawan Subedi, Xiaoyan Hong, Bing Zhou (The University of Alabama)
  - Demo: "Implementing an environmental sensor system using ICN", Bengt Ahlgren, Anders Lindgren (Swedish ICT - SICS); Adeel Mohammad Malik (Ericsson Research); Edith Ngai (Uppsala University); Borje Ohlman (Ericsson Research)
  - Demo: "NDN operation in Opportunistic Wireless Networks", Seweryn Dynerowicz, Omar Aponte, Paulo Mendes (COPELABS, University Lusofona)
  - Demo: "Securing Smart Homes using NDN", Lei Pi, Lan Wang (University of Memphis)
  - Demo: "Scalable Real-time Collaborative Communication over NDN using Edge Service Routers", Syed Obaid Amin (Huawei Research Center); Haitao Zhang (UCLA); Asit Chakraborti, Aytac

Azgin, Ravishankar Ravindran, GQ Wang (Huawei Research Center)
  - Demo: "Mini-NDN": a Lightweight and Scalable Emulation Environment for NDN, Ashlesh Gawande, Lan Wang (University of Memphis)
  - Demo: "Open mHealth: What is NDNFit", Haitao Zhang, Zhehao Wang (UCLA)
  - Poster: "Robus and Anonymous Information Sharing among Autonomous Vehicles", Muktadir Chowdhury, Ashlesh Gawande, Lan Wang (University of Memphis)
  - Poster: "Controlling Strategy Retransmissions in Named Data Networking", Hila Ben Abraham, Patrick Crowley (Washington University in St. Louis)
  - Poster: "Network Measurement for NDN", Davide Pesavento, Omar Ilias El Mimouni, Lotfi Benmohamed (NIST)
  - Poster: "Path Tracing in Named Data Networking", Siham Khoussi, Lotfi Benmohamed (NIST)
  - Poster: "NDN Distributed File System(NDFS)", Junior Dongo, Charif Mahmoudi, Fabrice Mourlin (Paris-Est University)
  - Poster: "NDN for Cog Network to Support Next Generation Social Systems", Arata Koike, Yoshiko Sueda (Nippon Telegraph and Telephone Corp.)
  - Poster: "NDN-IoT Framework and Example Application "Flow"", Zhehao Wang, Eitan Mendelowitz, Zoe Sandoval, Jeff Burke (UCLA)
  - Demo: "NDN-Android: NDN Networking Stack for Android Platform", Haitao Zhang, Alex Afanasyev, Lixia Zhang (UCLA)
  - Demo: "NDN Control Center: NDN Networking Stack and Security Enabler for Desktop Systems", Qi Zhao, Alex Afanasyev, Lixia Zhang (UCLA)
  - Demo: "Facilitating ICN Deployment with an Extended OpenFlow Protocol", Piotr Zuraniewski, Ray van Brandenburg (TNO); Borgert van der Kluit (Chess Wise); Niels van Adrichem (TNO)
  - Demo: "ndnSIM v2.3", Spyridon Mastorakis, Alexander Afanasyev, Lixia Zhang (UCLA)
- Andrew Meyers, Vice President for Research and Executive Director, University of Memphis Research Foundation
- Posters and Demos

March 24 (Friday)

- "NDNcomm: Community Feedback"
- "NDN Applications, Libraries and Tools" (Session Chair: Jeff Burke)
  - "ChronoShare: Decentralized File Sharing Application over NDN", Yukai Tu, Alex Afanasyev, Lixia Zhang (UCLA)
  - "NDNS: DNS is NDN", Yumin Xia, Alex Afanasyev, Lixia Zhang (UCLA)
  - "NDN-RTC and Experimental Library Functionality", Peter Gusev, Jeff Thompson, Alex Afanasyev (UCLA)
  - "Common Client Libraries - Update", Jeff Thompson, Jeff Burke (UCLA)
  - "On the Evolution of ndnSIM: an Open-Source Ecosystem for NDN Experimentation", Spyridon Mastorakis, Alexander Afanasyev, Lixia Zhang (UCLA)
  - "NDN Testbed: Status Update", John DeHart (Washington University)
- "Routing and Forwarding" (Session Chair: GQ Wang)
  - "Geohyperbolic Routing and Addressing Schemes For Overlay Networks", Ivan Voitalov, Rodrigo Aldecoa (Northeastern University); Lan Wang (University of Memphis); Dmitri Krioukov (Northeastern University)
  - "How to Establish Loop-Free Multipath Routes in Named Data Networking?", Klaus Schneider, Beichuan Zhang (The University of Arizona)

Figure 8.1: "Ad-hoc NDN Relay with Micro-Forwarder" project team, winner of 3rd NDN Hackathon.

Figure 8.2: Students at the 4th NDN Hackathon.

  - "A Native Content Discovery Mechanism for NDN", Onur Ascigil, Vasilis Sourlas, Ioannis Psaras, George Pavlou (University College London)
  - "Geolocation Compass: Geolocalized Data Registry and Forwarding for ICN Networks", Dante Pacella, Ashish Sardesai, Mani Tadayon (Verizon)
- Panel: "NDN at the Edge and on the Horizon" Chair by Jeff Burke, with panelists Lixia Zhang (UCLA), Sokwoo Rhee (NIST), Luca Muscariello (Cisco)

8.3 NDN Hackathons

Over the last year we organized two NDN hackathons. The third NDN Hackathon was held November 4-5, 2016 at Colorado State University, with sponsorship from CableLabs and Kyrio, in addition to the NDN Consortium (`http://3rd-ndn-hackathon.named-data.net/`). The participants included both NDN team members as well as students from other universities. They worked on eight projects that directly contributed to the NDN codebase development.

The Fourth NDN Hackathon was held in March 25-26, 2017 at University of Memphis (`http://4th-ndn-hackathon.named-data.net/`), attended by over 30 students coming from both the NDN project collaboration sites as well as seven other universities. They worked on the projects ranging from map applications over NDN to wireless mobility support for ndnSIM.

8.4 Other Efforts in Engaging the Broader Community

At the Third ACM Information Centric Networking Conference in September 2016, the NDN team gave a full-day NDN tutorial. The team has been doing NDN tutorials at every ACM ICN conference since the first one in 2014.

Two NDN team members, Jeff Burke and Alex Afanasyev, co-chaired the IEEE GLOBECOM 2016 Workshop on Information Centric Networking Solutions for Real World Applications (ICNSRA), held at Washington DC on December 8, 2016. NDN team members presented papers at the workshop and participated in an exciting panel titled "Application of ICN in Infrastructure-Free Environments: Rural Areas, Disaster Recovery, and Military Tactical Environments".

As we finish up this NDN report, we are starting the preparation for the first NDN tutorial at ACM SIGCOMM, to be held in August 2017 at UCLA.

8.5 Press & Media Coverage

8.5.1 Videos

1. Microsoft Research. "Enabling Connected Cars through Named Data." https://www.youtube.com/watch?v=vIbEqkVOHWU (posted June 21, 2016)

2. Ashlesh Gawande. "NDN Smart Home Demo." https://www.youtube.com/watch?v=xqs2m_S8Hm4 (posted June 10, 2016)

3. Jeff Voas at NIST Headquarters. "Networks of Things (Demystifying IoT)." http://iot.ieee.org/education/networks-of-things.html (presentation on May 31, 2016)

4. Muktadir Rahman Chowdhury. VANET (Vehicular Ad-hoc Network) over NDN. https://www.youtube.com/watch?v=377SfxtoJdM (posted on May 12, 2016)

5. Stephen Lee. EE533 "Named Data Networking Protocol on NetFPGA." https://www.youtube.com/watch?v=sLDULIS0hUc (posted May 9, 2016)

8.5.2 Selected Writing

1. Michael Byrne. "How a Reengineered Internet Could Protect Free Speech," *Motherboard*, *Vice*, December 2016. https://motherboard.vice.com/en_us/article/how-a-reengineered-internet-could-protect-free-speech (posted December 3, 2016)

2. Wilhelm Ner. "NDN soll das Internet Revolutionieren" (ENG: "NDN is to Revolutionize the Internet"). *Golem*, November 2016. https://www.golem.de/news/named-data-networking-ndn-soll-das-internet-revolutionieren-1611-123915.html (posted November 29, 2016)

3. Bob Brown. "Named Data Networking Team Issues Call for Hacks," *Network World*, October 2016. http://www.networkworld.com/article/3128261/internet/named-data-networking-team-issues-call-for-hacks.html (posted October 6, 2016)

4. Michael Recchione and 5G Americas. "Understanding Information Centric Networking & Mobile Edge Computing." December 2016. http://www.5gamericas.org/files/3414/8173/2353/Understanding_Information_Centric_Networking_and_Mobile_Edge_Computing.pdf

8.5.3 Misc. Coverage

1. 5G Americas. "Demystifying MEC and ICN." RSS Newsfeeds, *Marketwired*, December 2016. http://www.marketwired.com/press-release/demystifying-mec-and-icn-2183263.htm (posted December 14, 2016)

2. Chris Uwaje. "LTE, IoT as Next Battle Field for Innovation." *The Guardian*, October 2016. https://guardian.ng/technology/lte-iot-as-next-battle-field-for-innovation/ (posted October 19, 2016)

3. Jeremy Gillula and Noah Swartz. "Values, Governance, and What Comes Next: Afternoon Sessions at the Decentralized Web Summit." *Deeplinks*, Electronic Frontier Foundation, June 2016. https://www.eff.org/deeplinks/2016/06/values-governance-and-what-comes-next-afternoon-sessions-decentralized-web-summit (posted June 9, 2016)

Chapter 9

NDN Publications and Presentations

The NDN PIs participated in many conferences and speaking engagements, as listed below. These formal and informal efforts have helped disseminate research results and project ideas.

9.1 Publications

Listed below are 15 publications by NDN-NP team members during this reporting period (1 May 2016 – 30 April 2017).

[1] Peter Gusev, Zhehao Wang, Jeff Burke, Lixia Zhang, Eiichi Muramoto, Ryota Ohnishi, and Takahiro Yoneda. Real-time streaming data delivery over Named Data Networking (invited paper). *IEICE Transactions*, 2016.

[2] Jianxun Cao, Dan Pei, Zhelun Wu, Xiaoping Zhang, Beichuan Zhang, Lan Wang, and Youjian Zhao. Improving the Freshness of NDN Forwarding States. In *Proceedings of IFIP Networking Conference (IFIP Networking)*, May 2016.

[3] Vince Lehman, Ashlesh Gawande, Rodrigo Aldecoa, Dmitri Krioukov, Lan Wang, Beichuan Zhang, and Lixia Zhang. An Experimental Investigation of Hyperbolic Routing with a Smart Forwarding Plane in NDN. In *Proceedings of the IEEE IWQoS Symposium*, June 2016.

[4] Haitao Zhang, Zhehao Wang, Christopher Scherb, Claudio Marxer, Jeff Burke, and Lixia Zhang. Sharing mhealth data via named data networking. In *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*, pages 142–147. ACM, 2016.

[5] Klaus Schneider, Cheng Yi, Beichuan Zhang, and Lixia Zhang. A practical congestion control scheme for named data networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking (ICN)*, 2016.

[6] Junxiao Shi, Teng Liang, Hao Wu, Bin Liu, and Beichuan Zhang. NDN-NIC: name-based filtering on network interface card. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking (ICN)*, 2016.

[7] Hila Ben Abraham and Patrick Crowley. In-network retransmissions in named data networking (poster). In *Proceedings of ACM Conference on Information-Centric Networking*, September 2016.

[8] Alexander Afanasyev, Alex J. Halderman, Scott Ruoti, Kent Seamons, Yingdi Yu, Daniel Zappala, and Lixia Zhang. Content-based security for the web. In *Proceedings of New Security Paradigms Workshop (NSPW)*, September 2016.

[9] Wentao Shang, Alexander Afanasyev, and Lixia Zhang. The design and implementation of the NDN protocol stack for RIOT-OS. In *Proceedings of GLOBECOM Workshop on Information Centric Networking Solutions for Real World Applications (ICNSRA)*, December 2016.

[10] Katie Shilton, Jeffrey A. Burke, KC Claffy, and Lixia Zhang. Anticipating policy and social implications of named data networking. *Communications of ACM*, 2016.

[11] Jianxun Cao, Dan Pei, Xiaoping Zhang, Beichuan Zhang, and Youjian Zhao. Fetching popular data from the nearest replica in NDN. In *In proceedings of the 25th International Conference on Computer Communication and Networks (ICCCN)*, 2016.

[12] Yongmao Ren, Jun Li, Shanshan Shi, Lingling Li, Guodong Wang, and Beichuan Zhang. Congestion control in named data networking - a survey. *Computer Communications*, 86:1–11, 2016.

[13] Wentao Shang, Zhehao Wang, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. Breaking out of the cloud: Local trust management and rendezvous in Named Data Networking of Things. In *Proceedings of the 2nd ACM/IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI)*, April 2017.

[14] Muktadir Chowdhury, Ashlesh Gawande, and Lan Wang. Secure Information Sharing among Autonomous Vehicles . In *Proceedings of the 2nd ACM/IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI)*, April 2017.

[15] Jongdeog Lee, Akash Kapoor, Md Tanvir Al Amin, Zeyuan Zhang, Radhika Goyal, Zhehao Wang, Ilya Moiseenko, and Tarek Abdelzaher. InfoMax: A Transport Layer Paradigm for the Age of Data Overload. Book chapter in "Advances in Computer Communications and Networks - from Green, Mobile, Pervasive Networking to Big Data Computing", December 2016.

In addition, the following 10 papers will be presented at various technical conferences in next few months.

[1] Huichen Dai, Bin Liu, Haowei Yuan, Patrick Crowley, and Jianyuan Lu. Analysis of tandem pit and cs with non-zero download delay. In *To appear In proceedings of INFOCOM*, May 2017.

[2] Hila Ben Abraham and Patrick Crowley. Controlling strategy retransmissions in named data networking. In *To appear in proceedings of 2017 Symp. on Arch. for Networking and Communications Systems (ANCS 2017)*, May 2017.

[3] Hao Wu, Junxiao Shi, Yaxuan Wang, Yilun Wang, Gong Zhang, Yi Wang, Bin Liu, and Beichuan Zhang. On incremental deployment of named data networking in local area networks. In *To appear in ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, May 2017.

[4] M. Zhang, V. Lehman, and L. Wang. Scalable name-based data synchronization for named data networking. In *To appear in IEEE INFOCOM*, May 2017.

[5] J. Burke, P. Gusev, Z. Sandoval, J. J. Stein, and Z. Wang. The storytelling systems of los atlantis. In *To appear in ACM SIGCHI Case Studies*, May 2017.

[6] Haowei Yuan, Patrick Crowley, and Tian Song. Enhancing scalable name-based forwarding. In *To appear In proc. of 2017 Symp. on Arch. for Networking and Communications Systems (ANCS 2017)*, May 2017.

[7] Mathias Gibbens, Chris Gniady, Lei Ye, and Beichuan Zhang. Hadoop on named data networking: Experience and results. In *To appear in ACM SIGMETRIC*, June 2017.

[8] Junxiao Shi, Eric Newberry, and Beichuan Zhang. On broadcast-based self-learning in named data networking. In *To appear in IFIP Networking*, June 2017.

[9] Spyridon Mastorakis, Alexander Afanasyev, Yingdi Yu, and Lixia Zhang. nTorrent: Peer-to-peer file sharing in Named Data Networking. In *To appear in proceedings of the 26th International Conference on Computer Communications and Networks (ICCCN)*, July 2017.

[10] Xiaoke Jiang, Alexander Afanasyev, Yingdi Yu, Jiewen Tan, Yumin Xia, Allison Mankin, and Lixia Zhang. NDNS: DNS-like name service for NDN. In *To appear in proceedings of the 26th International Conference on Computer Communications and Networks (ICCCN)*, July 2017.

9.2 NDN Technical Reports

Listed below are 12 NDN technical reports produced during this reporting period. All the NDN technical reports are available online at `http://named-data.net/publications/techreports/`

[1] Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. NDN DeLorean: An Authentication System for Data Archives in Named Data Networking. Technical Report NDN-0040, NDN, May 2016.

[2] Vince Lehman, Ashlesh Gawande, Rodrigo Aldecoa, Dmitri Krioukov, Beichuan Zhang, Lixia Zhang, and Lan Wang. An experimental investigation of hyperbolic routing with a smart forwarding plane in ndn. Technical Report NDN-0041, Revision 1, NDN, July 2016.

[3] Satyanarayana Vusirikala, Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. A best effort link layer reliability protocol. Technical Report NDN-0041, NDN, August 2016.

[4] NFD Team. NFD developer's guide. Technical Report NDN-0021, Rev. 7, NDN, October 2016.

[5] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM 2: An updated NDN simulator for NS-3. Technical Report NDN-0028, Revision 2, NDN, November 2016.

[6] Manika Mittal, Alexander Afanasyev, and Lixia Zhang. NDN certificate bundle (version 0.1). Technical Report NDN-0054, NDN, March 2017.

[7] Susmit Shannigrahi, Dan Massey, and Christos Papadopoulos. Traceroute for Named Data Networking. Technical Report NDN-0055, Revision 2, NDN, 2017.

[8] Wentao Shang, Alexander Afanasyev, , and Lixia Zhang. The design and implementation of the NDN protocol stack for RIOT-OS. Technical Report NDN-0043, Revision 2, NDN, March 2017.

[9] Pedro de las Heras Quirós et al. The Design of RoundSync Protocol. Technical Report NDN-0048, NDN Project, March 2017.

[10] Tyler Vernon Smith, Alexander Afanasyev, and Lixia Zhang. ChronoChat on Android. Technical Report NDN-0059, Revision 1, NDN, April 2017.

[11] Klaus Schneider and Beichuan Zhang. How to establish loop-free multipath routes in Named Data Networking. Technical Report NDN-0044, Revision 1, NDN, April 2017.

[12] Zhiyi Zhang, Yingdi Yu, Alex Afanasyev, and Lixia Zhang. NDN certificate management protocol (NDNCERT). Technical Report NDN-0054, NDN, April 2017.

9.3 Technical Presentations

Listed below are presentations given by NDN-NP team members during this reporting period.

1. Alex Afanasyev, "Named Data Networking of Things: NDN-RIOT Progress Update," ICNRG Interim Meeting, Chicago, IL, US, March 2017.

2. Alex Afanasyev, "NDN/CCN Harmonization: Identifying NDN/CCNx 1.x Commonalties and Differences. A High-Level Discussion Summary," ICNRG Meeting, Kyoto, Japan, September 2016.

3. Alex Afanasyev, "Developing Simple NDN Applications," NDN Tutorial at ACM ICN 2016, Kyoto, Japan, September 2016.

4. Alex Afanasyev, "Developing Simple Simulations with ndnSIM," NDN Tutorial at ACM ICN 2016, Kyoto, Japan, September 2016.

5. Alex Afanasyev, "Internet of (Named) Things: NDN Protocol Stack for RIOT-OS," ICNRG Meeting, Berlin, Germany, July 2016.

6. Tarek Abdelzaher, "Social Network Signal Processing for Cyber-physical Systems," Invited Talk, International Workshop on Cyber Physical Systems, Daegu, South Korea, August 2016.

7. Tarek Abdelzaher, "Social Network Signal Processing," Invited Talk, Norfolk State University, Norfolk, VA, November 2016.

8. Tarek Abdelzaher, "Social Media Signal Processing," Invited Talk, Army Science Planning and Strategy Meeting (ASPSM), Adelphi, MD, December 2016.

9. Jeff Burke, "How Emerging Technologies Can Serve Expressive and Social Goals," UCLA Chancellor's Society, St. Francis Yacht Club, San Francisco, February 26, 2017.

10. Patrick Crowley, "Named Data Networking," Universidad de Concepcion. Concepcion, Chile. December 22, 2016.

11. Patrick Crowley, "Named Data Networking," Large Scale Networking Interagency Working Group, United States Networking and Information Technology Research and Development (NITRD) Program. April 11, 2017.

12. Lan Wang, "Improving the Freshness of NDN Forwarding States," IFIP Networking, May 18, 2016.

13. Lan Wang, "Physical and Cyber Security: The View from a Smart City Lens," Panel presentation, MIT Connected Things, March 13, 2017.

14. Beichuan Zhang, "Congestion Control in Named Data Networking," HotICN workshop, June 16, 2016.

15. Beichuan Zhang, "Applications and the Network: An NDN Perspective," Application-Driven Network Forum, Hong Kong, June 21, 2016.

16. Beichuan Zhang, "Named Data Networking," invited talk, Hunan University, China. June 24, 2016.

17. Lixia Zhang, "Looking back, looking forward: Why We Need A New Internet Architecture," keynote, the 11th International Conference on Future Internet Technologies, Nanjing, China. June 15, 2016.

18. Lixia Zhang, "Developing A New Internet Architecture: Progresses and Challenges," invited talk, Tsinghua University, China. June 20, 2016.

19. Lixia Zhang, "Five Years After the First ACM ICN Workshop: What we've learned, What remain to be done," Panel presentation, ACM Information Centric Conference 2016, September 27, 2016.

20. Lixia Zhang, "Untangling Communications from Infrastructures," Panel on Application of ICN in Infrastructure-Free Environments, at *Globecom Workshop on Information Centric Networking Solutions for Real World Applications*, Washington DC, December 8, 2016.

21. Lixia Zhang, "The Pleasure of Finding Things Out," keynote, CoNext 2016 Student Workshop, Irvine California, December 12, 2016.

22. Lixia Zhang, "Report on NDN Community Meeting 2017, " presentation at ICN Research Group meeting, Chicago, March 26, 2017.