# Named Data Networking (NDN) Project
# 2011 - 2012 Annual Report

*Principal Investigators*


Lixia Zhang, Deborah Estrin, and Jeffrey Burke
*University of California, Los Angeles*

Van Jacobson, James D. Thornton, and Ersin Uzun
*Palo Alto Research Center (PARC)*

Beichuan Zhang
*University of Arizona*

Gene Tsudik
*University of California, Irvine*

kc claffy and Dmitri Krioukov
*University of California, San Diego*

Dan Massey and Christos Papadopoulos
*Colorado State University*

Paul Ohm
*University of Colorado*

Tarek Abdelzaher
*University of Illinois at Urbana-Champaign*

Katie Shilton
*University of Maryland*

Lan Wang
*University of Memphis*

Edmund Yeh
*Northeastern University*

Patrick Crowley
*Washington University*

# Contents

**FIA: Collaborative Research: Named Data Networking (NDN) 2012 Annual Report**

# Executive Summary

This annual report of the Named Data Networking (NDN) project summarizes our second year research achievements and future plans. Chapter 1 briefly reviews the NDN architectural model. Chapter 2 gives an overview of our research plan, describes the milestones reached through our second year's effort. Chapter 3 gives a detailed description of the activities and findings in each of the project's research areas.

The heart of the 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 one of its design choices is the root cause of today's Internet problems. The Internet was designed as a *communication network* so the only entities that could be named in its packets were communication endpoints. Recent growth in e-commerce, digital media, social networking, and smartphone applications has resulted in the Internet primarily being used as a *distribution network*. Distribution networks are fundamentally more general than communication networks and solving distribution problems with a communications network is complex and error prone.

NDN retains the Internet's hourglass architecture but evolves the thin waist to allow 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 be anything — an endpoint, a chunk of movie or book, a command to turn on some lights, *etc.* This 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.

PARC's CCN project demonstrated this architectural evolution was feasible. Using CCN as a starting point, the NDN project's research challenge is to evolve it into architectural framework capable of solving real problems, particularly in application areas poorly served by today's Internet. Solving real problems forces architectural details to be filled in and, most importantly, verifies and shapes the architectural direction. We believe that an architectural research effort should be fundamentally experimental in nature. Design specifics cannot be derived from an intellectual exercise nor can validation be done through greenhouse testbed demonstrations. The Internet design was matured via actual usage through early deployment and our research plan follows the Internet's successful footsteps.

We successfully achieved all our major milestones for the second year of the project. Highlights include: (1) developing a name-based dynamic routing protocol, OSPFN, and fully deploying it on the NDN testbed connecting all the participating institutions; (2) completing and publicly releasing an open source NDN simulator, ndnSIM; (3) developing an architecture to secure instrumented environments such as industrial lighting and control; (4) analyzing potential DDoS attacks against NDN networks and the design space for countermeasures; (5) continued exploration of new application patterns enabled by NDN, including field testing of a scalable, multi-camera, multi-consumer, video streaming solution that doesn't require session semantics, and a Sync-based, *serverless*, multiuser chat application (MUC); and (6) using the NDN testbed, augmented with hundreds of Amazon EC2 hosts, to present a live, large-scale demonstration of NDN's unique capabilities at the GENI engineering conference.

One of the more exciting results of the past year was the discovery of a fundamentally new building block for distributed systems that we are calling *Sync*. Built on top of NDN's basic Interest-Data communication model, Sync enables simple, fully distributed, scalable, efficient and robust sharing of collections of data among multiple parties. The MUC application mentioned above is one of our first experiments in using Sync as a communication primitive. Preliminary experimentation has demonstrated its superior performance and robustness compared to centralized server designs. Over the next year, we expect that Sync's role in the NDN architecture will evolve to one similar to TCP's in the IP architecture.

The first two years of the NDN project have produced a wide range of new applications, a rich set of libraries, a functioning testbed, and a deeper understanding of the NDN architecture. These successes give us confidence that the project's third year will yield new discoveries about naming design, distributed data synchronization, trust management, routing and forwarding—core issues in the named-data networking architecture.

# Chapter 1

# Architecture Overview

NDN is an entirely new architecture, but one whose design principles are derived from the successes of today's Internet, reflecting our understanding of the strengths and limitations of the current Internet architecture, and one that can be rolled out through incremental deployment over the current operational Internet.

Today's Internet's *hourglass* architecture centers on a *universal* network layer (i.e., IP) which implements the minimal functionality necessary for global interconnectivity. This thin waist enabled the Internet's explosive growth by allowing both lower and upper layer technologies to innovate independently without unnecessary constraints. The NDN architecture retains the same hourglass shape, but transforms the thin waist to focus on data directly rather than its location. More specifically, NDN changes the semantic of network communication from *delivering a packet to a given destination address* to *retrieving data identified by a given name* (Figure 1.1). The design is also guided by the following principles.

- *Security must be built into the architecture.* Security measures in the current Internet architecture are an afterthought which do not meet the demands of today's diverse environment. NDN provides a fundamental security building block *right at the thin waist* by signing all named data.
- The *end-to-end principle* [4] underlying the TCP/IP architecture enabled development of robust applications in face of unexpected failures. NDN retains and expands this principle by securing data end-to-end.
- *Network traffic must be self-regulating.* Flow-balanced data delivery is essential to the stability of large systems. Unlike IP's open-loop packet delivery, NDN designs a flow-balance feedback loop into the thin waist.
- *The architecture should facilitate user choice and competition wherever possible.* Although not considered in the original Internet design, global deployment has taught us that "architecture is not neutral." [1] NDN makes a conscious effort to empower end users and facilitate competition.

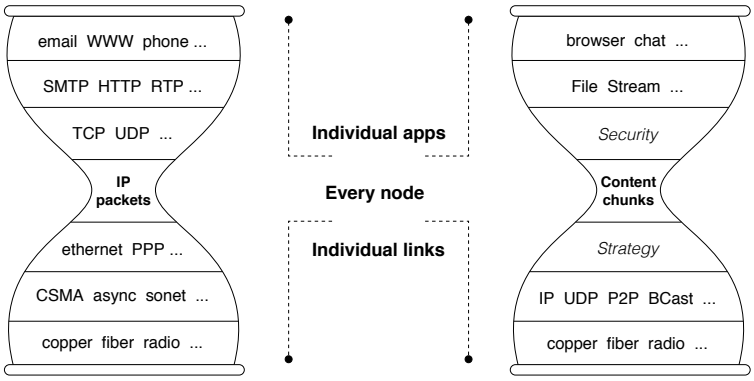


Figure 1.1: Internet and NDN Hourglass Architectures

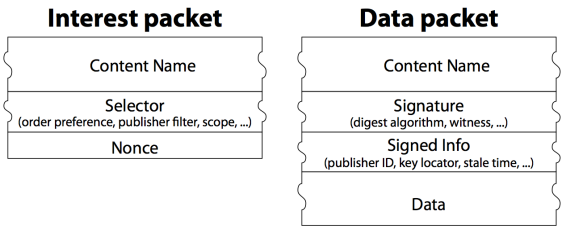


Figure 1.2: Packets In the NDN Architecture.

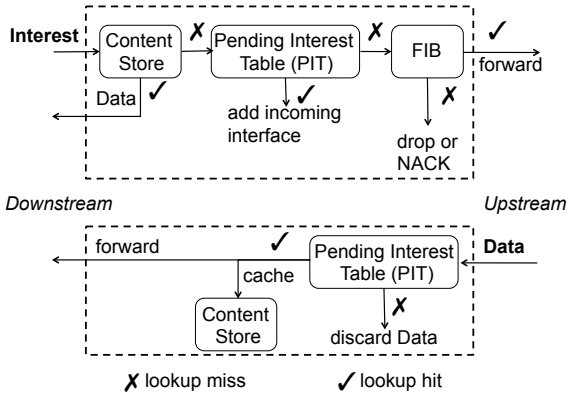


Figure 1.3: Forwarding Process at an NDN Node.

Communication in NDN is driven by the receiving ends, *i.e.*, the data consumers, through the exchange of two types of packets: *Interest* and *Data* (see Figure 1.2). Both types of packets carry a name that identifies a piece of data that can be transmitted in one Data packet. To receive data, a consumer puts the name of desired data into an *Interest* packet and send to the network. Routers use this name to forward the Interest toward the data producer(s). Once the Interest reaches a node $N$ that has the requested data, node $N$ will return a *Data* packet that contains both the name and the content, together with a signature by the producer's key which binds the two (Figure 1.2). This Data packet follows in reverse the path taken by the Interest to get back to the requesting consumer.

To carry out the above Interest and Data packet forwarding functions, each NDN router maintains three major data structures: a *Forwarding Information Base (FIB)*, a *Pending Interest Table (PIT)*, and a *Content Store* (Fig. 1.3). The FIB is populated by a name-prefix based routing protocol and is used to guide Interests toward data producers. The PIT stores all Interests that have been forwarded but not satisfied yet. It records the Interests name, incoming interface(s) and outgoing interface(s). When a router receives multiple Interests with the same name from downstream consumers, it forwards only the first one upstream toward the data producer. The Content Store is a temporary cache of Data packets that the router has received. Because an NDN Data packet is meaningful independent of where it comes from or where it is forwarded to, it can be cached to satisfy future Interests.

When an Interest packet arrives, an NDN router first checks the Content Store for matching data; if it exists the router returns the Data packet on the interface from which the Interest came. Otherwise the router looks up its PIT, and if a matching entry exists, it simply records the Interest's incoming interface in the PIT entry. In the absence of a matching PIT entry, the router will look up its FIB and forward the Interest toward the data producer(s).

When a Data packet arrives, an NDN router finds the matching PIT entry and forwards the data to all downstream interfaces listed in the PIT entry. It then removes that PIT entry, and caches the Data in the Content Store. Data packets always take the reverse path of Interests, and one Interest packet results in one Data packet on each link, providing *flow balance*. Neither Interest nor Data packets carry any host or interface addresses (such as IP addresses); Interest packets are forwarded toward data producers based on the names carried in them, and Data packets are forwarded to consumers based on the PIT state information set up by the Interests at each hop.

## 1.1 Names

NDN names are *opaque* to the network, *i.e.*, routers do not know the meaning of a name, although they know the boundaries between components in a name. This decoupling allows each application to choose the naming scheme that fits its needs and allows the naming schemes to evolve independently from the network. However, the NDN design does assume hierarchically *structured* names, e.g., a video produced by UCLA may have the name `/ucla/videos/demo.mpg`, where '/' delineates name components in text representations, similar

to URLs. This hierarchical structure allows applications to represent relationships between data elements. For example, segment 3 of version 1 of the video might be named `/ucla/videos/demo.mpg/1/3`. It also allows name aggregation: In this example `UCLA` could correspond to an autonomous system from which the video originates. Flat names can be accommodated as a special case and useful in local environments, however hierarchical name spaces are essential in scaling the routing system. Naming conventions among data producers and consumers, *e.g.*, to indicate versioning and segmentation, are specific to applications.

To retrieve dynamically generated data, consumers must be able to *deterministically* construct the name for a desired piece of data without having previously seen the name or the data. Either: (1) a deterministic algorithm allows the producer and consumer to arrive at the same name based on information available to both, or (2) *Interest selectors* in conjunction with *longest prefix matching* retrieve the desired data through one or more iterations. Our experience so far suggests that a simple set of selectors can support retrieving data with partially known names. For example, a consumer wanting the first version of the `demo.mpg` video may request `/ucla/videos/demo.mpg/1` with the Interest selector "leftmost child" and receive a data packet named `/ucla/videos/demo.mpg/1/1` corresponding to the first segment. The consumer can then request later segments using a combination of information revealed by the first data packet and the naming convention of the publishing application to request subsequent packets.

Only names used to retrieve data globally requires *globally uniqueness*. In fact, individual data names can be meaningful in various specific scopes and contexts, ranging from "the light switch in this room" to "all country names in the world". Efficient strategies to fetch data within the intended scope is a new research area.

Name space management is not part of the NDN architecture, just as address space management is not part of the IP architecture. However, naming is the most important part of the NDN design. Naming data enables natural support for functionality such as content distribution, multicast, mobility, and delay-tolerant networking. We are learning through experimentation how applications should choose names that can facilitate *both* application development and network delivery. As we develop and refine our principles for naming, we convert these principles and guidelines into naming conventions and implement them in system libraries to simplify future application development. Fortunately, the opaqueness of names to the network allows architecture development to proceed in parallel with research into namespace structure and navigation in the context of application development.

## 1.2 Data-Centric Security

In contrast to TCP/IP's approach where security (or lack thereof) is a function of where or how the data is obtained, NDN builds security into data itself by requiring data producers to cryptographically sign every piece of data with its name [3]. The publisher's signature enables determination of data *provenance*, allowing the consumer's trust in data to be decoupled from how (and from where) the data is obtained. It also supports fine-grained trust, allowing consumers to reason about whether a public key owner is an acceptable publisher for a particular piece of data in a specific context.

Historically, security based on public key cryptography has been considered inefficient, unusable and difficult to deploy. Besides efficient digital signatures, NDN needs flexible and usable mechanisms to manage user trust. Since keys can be communicated as NDN data, key distribution is simplified. Secure binding of names to data supports a wide range of trust models. If a piece of data is a public key, a binding is effectively a public key certificate. Finally, NDN's end-to-end approach to security facilitates trust between publishers and consumers, and gives applications flexibility in customizing their trust model.

NDN's data-centric security has natural applications to content access control and infrastructure security. Applications can control access to data via encryption and distribute (data encryption) keys as encrypted NDN data, limiting the data security perimeter to the context of a single application. Requiring signatures on network routing and control messages (like any other NDN data) provides a solid foundation for routing protocol security, protecting against spoofing and tampering. NDN's inherent multipath routing, together with the adaptive forwarding plane we describe next, mitigates prefix hijacking because routers can detect the anomaly caused by a hijack and retrieve data through alternate paths.

## 1.3 Adaptive Routing and Forwarding

NDN routes and forwards packets based on names, which eliminates four problems caused by addresses in the IP architecture: address space exhaustion, NAT traversal, mobility, and address management. There is no address exhaustion problem since the namespace is unbounded. There is no NAT traversal problem since a host does not need to expose its address in order to offer content. Mobility, which requires changing addresses in IP, no longer breaks communication since data names remain the same. Finally, address assignment and management is no longer required in local networks, which is especially empowering for embedded sensor networks.

NDN can make use of conventional routing algorithms such as link state and distance vector. Instead of announcing IP prefixes, an NDN router announces *name prefixes* that cover the data that the router is willing to serve. Routers simply treat names as a sequence of opaque components and do component-wise longest prefix match of the name in a packet against the FIB. We describe our approaches to scalable routing in Section 3.2, and our research on various hardware and software techniques for scalable and fast name lookups in Section 3.3.

The PIT state at each router supports forwarding across NDN's data plane, recording each pending Interest and the incoming interface(s), and removing the Interest after the matching Data is received or a timeout occurs. This per-hop, per-packet state is a fundamental change from IP's stateless data plane. The state information enables NDN nodes to monitor packet delivery performance across different interfaces, and adapt to network failures, all at the time scale of a round-trip time. Via a random nonce in the Interest packet, NDN nodes can identify and discard packets that have returned to the same node, preventing forwarding loops. This allows NDN nodes to freely use multiple paths toward the same data producer, efficiently utilizing network resources.

The PIT state also serves several other important purposes. Since it records the set of interfaces over which the Interests for the same data name have arrived, it naturally supports multicast Data delivery. Since each Interest retrieves at most one Data packet, a router can control the traffic load by controlling the number of pending Interests to achieve flow balance. The PIT state can also be used to effectively mitigate DDoS attacks. Because the number of PIT entries is an explicit indicator on the router load, an upper bound on this number sets the ceiling on the effect of a DDoS attack. PIT entry timeouts offer relatively cheap attack detection [5]; and the arrival interface information in each PIT entry gives information to implement a push-back scheme [2].

Each NDN node also implements a *forwarding strategy* module, which does not exist in today's IP nodes. The role of this forwarding strategy module is to make informed decisions about which Interests to forward to which interfaces, how many unsatisfied Interests to allow, what are the relative priority of different Interests, how to load balance Interest forwarding among multiple interfaces, as well as how to choose alternative paths to avoid detected failures.

## 1.4 In-Network Storage

Because each NDN Data packet carries a name and a signature, it is meaningful independent of its source or destination. Thus a router can cache the data in its Content Store to satisfy future requests. The Content Store is analogous to buffer memory in IP routers, but IP routers cannot reuse data after forwarding it, while NDN routers can. NDN treats storage and network channels identically in terms of data retrieval. For static files, NDN achieves almost optimal data delivery. Even dynamic content can benefit from caching in the case of multicast (*e.g.*, realtime teleconferencing) or retransmission after a packet loss. In addition to the Content Store, we are also adding to the NDN architecture a more persistent and larger-volume in-network storage, called a Repository (Repo in short).

Caching named data raises different privacy concerns from those of IP. In IP, one can examine packet headers, and possibly payload, to learn who is consuming what data. Naming and caching of data in NDN networks may facilitate observation of what data is requested, but without destination addresses it is harder to identify who is requesting it (unless one is directly connected to the requesting host). This aspect of the architecture offers privacy protection at a fundamentally different level than current IP networks.

## 1.5 From Transport to Distributed Synchronization

The NDN architecture does not require a separate transport layer. It moves the functions of today's transport protocols into applications, their supporting libraries, and the strategy module of the forwarding plane. NDN does not use port numbers; a host knows to which application to deliver packets based on data names, and applications handle data integrity checking, signing, and trust decisions related to their data. To provide reliable delivery across highly dynamic and possibly intermittent connectivity, such as in mobile environments, nodes will discard Interest packets that remain unsatisfied after some threshold of time. The application that originated the initial Interest must retransmit it if it still wants the data.

NDN's flow balance requirement, together with the ability of nodes to control their own traffic load by limiting the number of pending Interests at each hop means that there is no need for separate end-to-end congestion control, a typical transport layer function in today's networks. If congestion losses occur, caching will mitigate the impact since retransmitted Interests can be satisfied by cached Data packets right before the point of packet losses. Thus NDN avoids the kind of congestion collapse that can occur in today's Internet when a packet is lost near its destination and repeated retransmissions from the original source host(s) consume most of the bandwidth.

Traditional transport services provide point-to-point data delivery and most of today's distributed applications, including peer-to-peer applications, heavily rely on centralized servers. To aid development of robust and efficient distributed applications, we are adding a fundamentally new architectural building block that we call *Sync*. Using NDN's basic Interest-Data exchange communication model, Sync uses naming conventions to enable multiple parties to synchronize their dataset. By exchanging individually computed data digests, each party learns about new or missing data quickly and reliably, and retrieves data efficiently via NDN's built-in multicast delivery.

# References


[1] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow's internet. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 347–356, New York, NY, USA, 2002. ACM.

[2] John Ioannidis and Steven M. Bellovin. Router-based defense against ddos attacks. 2002.

[3] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th ACM International Conference on Emerging Networking Experiments and Technologies*, pages 1–12, 2009.

[4] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design.

[5] Vyas Sekar, Nick Duffield, and Oliver Spatscheck. Lads: Large-scale automated ddos detection system.

# Chapter 2

# Research Plan and Progress

In this chapter we present the major outcomes we are working to deliver and the plan we are executing to move the NDN architecture from ideas to reality and validation. We summarize the execution results during the second year of the project and the milestones for the remaining duration of the report.

The NDN architecture retains the same hourglass shape as the IP architecture, with the narrow waist as its centerpiece. However the minimal functionality of NDN's narrow waist, as we described in Chapter 1, is fundamentally different from IP's. NDN's minimal functionality includes support for consumer-driven data delivery, built-in data security, and use of in-network memory. Together with in-network storage, NDN's stateful forwarding plane scalable data dissemination, flow balancing, multiple path data retrieval, as well as facilitates mobile and delay-tolerant communications.

We aim at the following major deliverables.

1. A specification of the standard formats for the two packet types, Interest and Data, in NDN data delivery. We expect this specification to play a role equivalent to that of RFC791 (Internet Protocol Specification) for NDN networks. The challenge in nailing down this specification is not the format, but the verification and validation of the exact functions that must be supported by the narrow waist.
2. A functional version of each of the necessary supporting modules in an operational NDN network, including libraries to support naming conventions, reliable data delivery, routing protocols and forwarding strategy, trust management, and usable, efficient cryptography for data security[1].
3. A set of applications that operate over an NDN network, including both new ones specifically designed to run on top of NDN, as well as legacy ones in today's Internet.
4. An online documentary of the NDN project process, and a technical report series to capture our thinking along the path of architectural development.

Teams at various campuses are advancing research in various areas, including forwarding performance, routing, security and privacy, and the fundamental theory for this new type of networking. Due to practical limitations of scaling the testbed at this stage, We have developed an ns-3 based NDN simulator, ndnSIM, during the second year, so that we and others can use a common simulation platform to investigate large-scale system properties of NDN.

## 2.1 Progress: Year 2

- Applications (Section 3.1)
  - Developed authenticated actuation approach for instrumented environments.

[1] We see an analogy between the above list and IP's supporting components. Although the IP address allocation system, routing protocols, and DNS are not part of the IP narrow waist, they are nonetheless necessary supporting components to make an operational IP network. The fact that DNS was added *after* the initial IP deployment further underscores the importance of identifying missing components from real deployment.

  - Developed and field-tested NDN-based video streaming solution.
  - Redesigned Personal Data Cloud application to reduce legacy semantics, and tested on Android.
  - Prototyped NDN support in Firefox and a native javascript implementation of the NDN protocol.
  - Implemented multi-user chat over NDN, and used it to explore the design space of a general synchronization protocol to support *serverless* distributed applications.
  - Created Python bindings for CCNx libraries and tested in several application.
- Routing (Section 3.2)
  - Designed and implemented a name-based *dynamic routing protocol*, *OSPFN*, deployed it on the NDN testbed, and developed monitoring tools.
  - Sketched a solution called *forwarding hint* to scale core routing tables in a global NDN network, as well as support mobility and overlay construction.
  - Applied *hyperbolic greedy routing* to the current NDN testbed topology, and ran simulations that demonstrated high success ratio of packet delivery using this method.
  - Simulated NDN, IP, and Path Splicing in the face of prefix hijack, link failure, and congestion, showing that NDN's forwarding strategy can detect and recover from these problems.
  - Designed and implemented a link layer protocol, *NDNLP*, which provides two optional features at each hop: fragmentation/reassembly of NDN packets, and detection/recovery of packet losses over the link.
- Scalable Forwarding (Section 3.3)
  - Presented the concepts, issues and principles of NDN forwarding that can support scalable 1 and 10 Gbps implementations of the forwarding plane.
  - Conducted a large-scale NDN demonstration at the GENI engineering conference, using the NDN testbed and hundreds of hosts to showcase NDN services operating efficiently and robustly in a wide-area context.
  - Demonstrated architectural advantages of the NDN design by comparing content distribution performance using NDN and HTTP in gigabit networks.
- Security and Privacy (Section 3.4)
  - Evaluated and implemented NDN privacy protection and traffic anonymization technique – AN-DaNA (Anonymous Named Data Networking Application).
  - Developed a security architecture for instrumented environments, including support for signed interests as authenticated commands to end devices, e.g., actuators, controllers, and sensors.
  - Developed and implemented security architecture for NDN audio conferencing.
  - Developed and implemented an initial trust model for selected set of applications (in collaboration with NDN application group).
  - Developed "Namecrypto", a generic library for encrypting and signing name components.
  - Conducted initial analysis of (Distributed) Denial-of-Service vulnerabilities and possible attacks against NDN networks, and identified basic attack classes and their impacts.
  - Surveyed and evaluated efficient signature and (symmetric-key) MAC schemes for NDN.
- Network Management (Section 3.5)
  - Introduced the concept of *Auditor Nodes* into the network, specialized nodes that act as safe harbors to detect and overcome content poisoning attacks, and possibly enable value added services by ISPs.
  - Developed NDN traceroute tool and in the process identified challenges in managing an NDN network.
  - Developed NDN Auto Configuration (NAC) server to enable hosts to automatically obtain the necessary configuration information to join an NDN network.
  - Finished the development of an open source NDN simulator, ndnSIM.
- Theory (Section 3.6)
  - Established a framework to systematically develop dynamic forwarding and caching for NDN, allowing for simultaneous optimization of the strategy layer and the caching policy, in order to maximize the total consumed bit rate in the NDN network.
  - Developed a distributed dynamic forwarding and caching algorithm for maximizing the region of Interest packet arrival rates which the NDN network can satisfy.

  - Numerically tested the proposed dynamic forwarding and caching algorithm for a number of interesting network and traffic scenarios, including (1) the Abilene network, (2) the NDN testbed network, (3) access networks, and (4) random spatial networks.
  - Undertook theoretical investigations into the optimality of the proposed dynamic forwarding and caching algorithm, using the theory of nonlinear stochastic control and optimization.
- Values in Design (Chapter 4)
  - Mapped relationships between the campuses, PIs, and work practices of the NDN team.
  - Developed a taxonomy of values concerns in the NDN project.
  - Developed a set of questions to help shape interviews and publications with the NDN team.
  - Worked with the applications team to bootstrap a trust model using existing relationships among researchers harvested from LinkedIn.
  - Began developing a conceptual map of the NDN design, focusing on aspects that will be important or confusing to external communities.
- Education (Chapter 5)
  - Incorporated NDN architecture into teaching curricula at several NDN project campuses.
  - Developed a set of NDN projects and guided graduate students through their research efforts in mastering the architecture concepts and design tradeoffs through concrete design and implementation exercises.
  - Organized two NDN hands-on boot camps to train students in the basic concepts of NDN architecture and hands-on skills for application development.

# Chapter 3

# Architecture Research

This chapter describes activities and findings in each research area for the second year.

## 3.1 Applications

| Contributors | |
|---|---|
| **PIs** | Jeffrey Burke, Deborah Estrin & Lixia Zhang (UCLA), Tarek Abdelzaher (UIUC), Van Jacobson & Jim Thornton (PARC) |
| **Grad Students** | Jiwen Cai, Meki Cherkaoui, Derek Kulinski, Gauresh Rane, Zhenkai Zhu (UCLA); Shen Li, Yusuf Sarwar, Hengchang Liu, Tanvir Amin (UIUC) |
| **Undergrads** | Zening Qu (UCLA) |
| **Staff** | Alex Horn, Alessandro Marianantoni (UCLA); Hongyan Wang (UIUC) |

NDN application research: (1) drives architecture development based on a broad vision for future applications; (2) drives and tests prototype implementations of the architecture using applications in participatory sensing, instrumented environments, and media distribution; (3) verifies and validates performance and functional advantages of NDN in key areas; and (4) shows how NDN's embedding of application names in the routing system promotes efficient *authoring of sophisticated distributed applications*, reducing complexity and thus opportunities for error, as well as time and expense of design and deployment.

In parallel with the continued development effort of instrumented environment and participatory sensing applications, we began to work extensively with streaming multimedia content over NDN and exploring avenues for NDN support in contemporary web browsers, two applications that stand to benefit from content caching, packet verification, and name-based routing/forwarding. We developed our prototype applications using the CCNx software codebase from PARC, including the C API, which provides low-level (socket-like) functionality, and the Java API, which provides higher-level abstractions. Based on our experience from the first year, we also built tools to facilitate application development, completing Python bindings and starting a Javascript implementation. We also began to develop frameworks to test scaling of the network and applications; e.g., using Amazon EC2 instances to add content consumers, and distributing standardized hardware to run applications across the testbed.

### 3.1.1 Instrumented environments

We are creating NDN interfaces for representative, IP-enabled building automation systems, sensing, and multimedia subsystems, and connecting them to the NDN testbed network to explore security, performance, and addressing features enabled by the architecture. In Year 2, we formalized our initial approach to control authentication, and began to incorporate NDN into several practical deployments planned for Year 3.

**Authenticated Actuation - Lighting Control**

We expanded our lighting control application, which explores actuation over NDN as a contrast to the usual focus on content distribution. Our lighting control testbed is in a television studio on the UCLA campus, where we have eleven TCP/IP-enabled color mixing lighting fixtures controlled by the embedded processor that acts as an NDN gateway. To ease development and aid integration with other projects, we reimplemented the lighting control application in Python instead of C, which gave us the opportunity to test the new Python bindings for CCNx on an ARM-based embedded platform. The new version of the control system incorporates multiple keys and control namespaces for the lights, as well as, notably, "authenticated Interests." In collaboration with the NDN security group at UCI, we incorporated a signature into Interest packets that enables them to be interpreted as authenticated RPC calls. (The semantics are consistent with NDN: the controller is interested in the status data resulting from changing an actuator's property to a value expressed in the Interest name, with a signature calculated using the controller's key over the actuator, property, and value.) We have explored both asymmetric and symmetric signatures for the verification mechanism (See also Section 3.4.3).

For example, a lighting fixture might be named using the following pattern:

```
/<path_to_fixture>/<building>/<room>/<lights>/<fixture-path>
/ucla.edu/melnitz/TV1/lights/living-room-left
```

A verifiable control Interest would be:

```
/<path_to_fixture>/<capability>/<param_pattern>/<parameters>/<signature>
/ucla.edu/melnitz/TV1/lights/living-room-left/control/rgb-8bit-hex/F0FF39/[sigbits]
```

The trust model developed for fixture control is SDSI-like [5]. Every entity in the system has a public and private key pair, and a trusted configuration manager assigns keys to fixtures and controllers, expressing the controllers' permissions through the namespace in which its key is published (and signed by the controller). For example, a controller's public key would be available as:

```
/<path_to_fixture>/lighting/<capability>/<app-name>/key
/ucla.edu/apps/lighting/control/light_board/key
```

The details of authenticated Interests are described in [2] of the publication list.

**Sensing in Building Automation / Environmental Sensing**

In Year 2 we began to explore using NDN to publish data from infrastructure-based sensing, analogous to the building automation and "smart grid" visions. UCLA implemented an NDN-based data publisher and consumer for a personnel-tracking solution installed in a studio space, and UIUC's NDN-based platform for data center energy control was opened to researchers outside of the NDN project. These platforms will enable us to simulate building automation feedback loops, to explore how NDN can handle challenges with existing building management systems that use TCP/IP - for example, configuration management and security [1] [2].

We are also incorporating NDN into two existing UCLA sensing projects supported from other sources, in an area called "Ambient Informatics", in which networked objects change properties (e.g., color) to reflect real-time data. The Canary is a wall-hanging sculptural object that changes color to reflect the electrical demand and chilled water usage of UCLA's Strathmore Building, where transportation services are housed. Over summer 2012 it will be transitioned from using IP to NDN, with its commercial-grade building sensor data published on the NDN testbed and read by the Canary's embedded device. In a similar project, an outdoor sculptural indicator of UCLA's BruinBus arrival times will read that data from the NDN testbed. Both cases provide live deployments for which we must develop: (1) namespaces for sensing infrastructure; (2) security approaches for verifying data before actuation; (3) storage for historical data; and (4) reliable performance of the network stack.

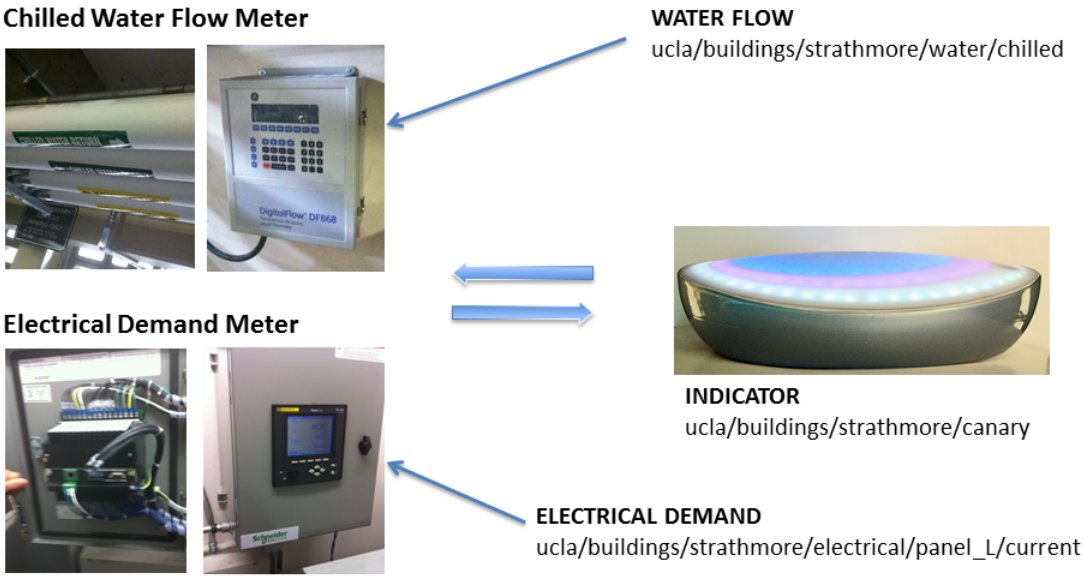


Figure 3.1: "Canary" ambient informatics project and representative NDN namespace.

### 3.1.2 Participatory Sensing

The basic operation of NDN, *i.e.* a consumer expresses interest for data and data flows back following the state set up by the Interest packet, can directly support mobile data consumers and publishers. We have applied NDN technology to two participatory sensing projects: the *personal data cloud* (UCLA), and *GreenGPS* (UIUC).

UCLA has been extending the concept of a host-centric "personal data vault" developed at UCLA and USC [3] to create a geographically distributed *personal data cloud (PDC)*. The PDC work has gone through three iterations that reflect increasing understanding of how to develop applications using NDN, i.e., without session semantics or client/server interaction. The first implementation, completed largely in the first year, implemented data collections, key management, storage, data transfer and authentication/setup phases. The second revision integrated the PDC architecture into a deployed participatory sensing application called Ohmage [4]. It mapped the relational data store used for Ohmage and the PDC data collection model. The most recent revision uses the new Sync protocol in CCNx for transferring content between entities, and removes much of the session-like semantics initially present.

*Social sensing* is motivated by the proliferation of cheap processing technology and personally owned sensors.

*GreenGPS* is a smart-phone-based navigation service for drivers to route their vehicles on the most fuel-efficient route, as opposed to the fastest or shortest. The PIs are transitioning this application to the car fleet of UIUC Facilities and Services to save energy on campus. In year 2, UIUC completed a 25-car human subjects study to test the service, introducing opportunistic car-to-car communication and data prioritization (to support short car-to-car encounters). In Year 2, the PIs transitioned a first prototype of an in-network storage service that manages a content store, for possible use by mobile ad hoc and disruption tolerant networks.

Each application required careful consideration of namespace design for distributed sensing services. In most cases, location and time of data generation matters; the general "formula" for namespaces is:

```
<root>/<service_name>/<content_type>
                              /<originator_ID>/<location>/<time>/<features>
```

The service owns the content, so it is responsible for everything below *service_name* and *content_type*. Object metadata then follows, concluding with the features of the content object, which allows the definition

of "distance functions" between objects based on their names. These distance functions are service- and content-type specific, A distance function and estimate how much information overlap is likely to exist between objects of a given type in a given service. For example, consider the following two object names:

```
/photoNet/pictures/camera1/GPS-addressA/timestampA/RGB-featuresA/pictureA
/photoNet/pictures/camera2/GPS-addressB/timestampB/RGB-featuresB/pictureB
```

The general distance function for pairs of "/photoNet/pictures" objects returns:

$distanceAB = photoNetPicsDistance(name1, name2)$

The returned distance value may enable us to offer a prioritized Sync scheme that sends content in an order that minimizes overlap between objects sent within any given time window. Additionally, a replacement scheme for data repositories may choose to erase content that has the most overlap with surviving objects, which should improve user-perceived throughput or performance in the presence of resource constraints.

### 3.1.3 Media distribution

NDN's caching and name-based protocol has inherent advantages for media streaming, especially in improving the bandwidth efficiency of content distribution. This year we completed a live video streaming application and demonstrated it by streaming an acoustic concert shot in a UCLA television studio to the GENI Engineering Conference over the NDN testbed. We also continued to develop multi-user communication tools, and begun to explore both two of the most important environments for media: the browser, and the networked 3D game engine.

#### NDNVideo - Live and Pre-recorded Video

NDNVideo is an audio and video streaming application that supports streaming of live and pre-recorded audio and video to multiple clients[1]. NDNVideo supports simple, rapid random access into streams using a timecode-based namespace, synchronized playback of multiple clients and devices, long term storage and retrieval of content, and enables passive consumers of content requiring no session semantics or communication with the video source. NDNVideo's architecture demonstrates distributed publishing of named data, as well as versioning and segmenting of live data for caching, content verification, lack of end-to-end negotiation, and pipelining for live streaming and broadcast. NDNVideo is written in Python, using the GStreamer multimedia framework [2] and our Python bindings for CCNx.

NDN's pull-based approach enables the NDNVideo publisher to be simpler than it would need to be in IP to achieve random access. Data are named in a hierarchy that defines the encoding type and provides random access via time (for seeking) and segment (for playing), as shown in Fig. 3.2. The publisher can stream data with minimum to no interaction with consumers. It simply prepares Data packets, signs them, and stores them for retrieval. The consumer no longer needs to inform the publisher about its QoS, because it is in full control of how much and at what rate it receives data. If for any reason the consumer needs to upgrade/downgrade bit rate, it can do so seamlessly by requesting differently named data and seeking to the same point. Challenges raised by the NDNVideo implementation include repository performance for large amounts of data; developing an appropriate Interest pipelining technique to keep both latency and dropouts low; and the challenge of retrieving the newest data in a live stream over a network full of caches without communicating with the source.

To demonstrate the system, we published a live, standard definition H.264-encoded stream (@ 1Mbit/sec) from a musical performance in a UCLA TV Studio over the NDN testbed to the Washington University team's demonstration for the GENI Engineering Conference in Los Angeles in March 2012. We mixed and streamed broadcast quality audio and video feeds from three cameras, publishing the data to a CCN repo at UCLA. The WUSTL team retrieved the video using the player. In these and other tests with standard definition, H.264 video, the streaming works well to end-users, although we have encountered repeated (periodic) packet

[1] https://github.com/remap/ndnvideo
[2] http://gstreamer.freedesktop.org

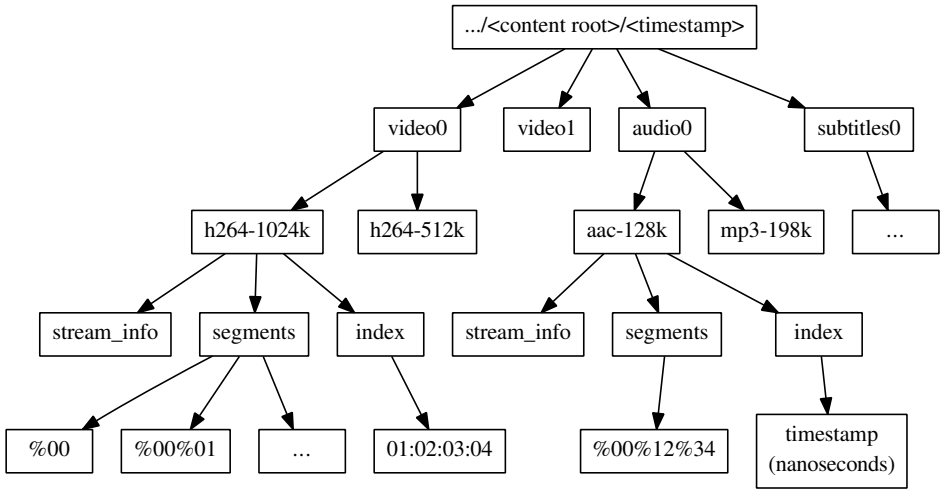


Figure 3.2: NDNVideo Namespace.

timeouts that are not eliminated even with zero network communication and interest delays. We are working to determine the cause (possibly interest reordering in CCNx or other implementation details).

Finally, we began to test and visualize the scaling properties of content distribution over NDN, using Amazon EC2 instances running a (headless) video client. We have scaled up to nine independent nodes in Amazon's cloud, each running a video client, with a content router between the clients and the video source. We measured Content Object propagation by monitoring the status of each daemon, visualizing it in a web browser.

### NDNFox - Enabling the Web Browser

Browsers are now a primary screen through which media are navigated, and standalone applications like NDNVideo are increasingly being replaced by browser-based version. In Year 2, we began two exploratory efforts to accommodate this trend: native code integration of CCNx support into the Mozilla open-source browser and the lwNDN Javascript implementation of the CCNx protocol, discussed in the next section. As a prototype, we aimed to first provide static file retrieval from CCNx repositories and then video playback in the browser via a simple *ccnx:/* URL prefix. The Mozilla (Firefox) web browser handles networking and content rendering separately enabling our initial work to be limited to adding NDN content retrieval mechanism into Firefox's existing network module. The implementation had three steps: a new protocol handler to support an NDN-style URL; use of this library to fetch static text data from the repository and pass it to Firefox's rendering module; and HTML5-based multimedia playback support. The resulting implementation provided NDN-backed video playback in the web browser and supported two types of video transmission: (1) playing a static video file from a repository in the browser, and (2) playing a live stream served by NDNVideo. We used C++ to implement the protocol patch to Mozilla and C to implement a new plugin of the GStreamer library to depacketize NDN content from NDNVideo server.

The current version of NDNFox can fetch and render HTML webpages, high resolution images, and high definition video tracks using NDN Repository without no significant performance troubles. This work has inspired many new questions including, for example, the discussion of content metadata. In traditional HTTP-based web browsing, the MIME type of incoming data is indicated by the "Content-Type" field of the HTTP header. In our initial implementation of a simple protocol, we used the file's extension to infer the type of incoming data. We will explore other solutions, including a higher level protocol or a naming standard for file metadata.

#### Egal - Synchronizing 3D Graphics over NDN

To explore the value of NDN to create distributed experiences with open source graphics engines and game platforms, and to better understand the new Sync primitive available in PARC's CCNx implementation, we developed "Egal", a library for distributed state and asset synchronization in Unity using NDN, and prototyped an implementation in Year 2. As an initial step towards supporting massively multiplayer online games (MMOG) over NDN, we converted a single-player car racing game provided by Unity 3D into a peer-to-peer multiplayer game that communicates asset and state changes via NDN. Asset changes were written to a local CCNx repository and distributed through CCNx sync. State changes were communicated using normal CCNx interest and data exchange in a known namespace, using optimistic prediction and error correction algorithms, and without going through the local CCNx repository. At this time, primary limitations of the CCNx Sync protocol appear to be: (1) its (unwanted) integration with content repository; and (2) its limitation of supporting only directly connected peers. We plan to work with the architecture team on alternatives, including Perhaps Chronos, described below.

#### Multi-User Chat over NDN

The Audio Conferencing Tool created last year [6] has been supplemented by a multi-user text chat tool (MUC)[3]. XMPP-based communication is supported by a wide range of IM clients, including Adium [4], iChat [5], Empathy [6], etc. Similar to the audio conference tool (ACT), the implementation of MUC also uses virtual servers run on the machines of individual participants, which discover each other via broadcast Interests. Whoever wants to join the chat group can simply reply to Interests. The virtual servers also send Interests to each participant to request specific pieces of chat data, which is then automatically multicasted to all other participants once it is published. The local XMPP clients talk to the virtual servers using normal XMPP protocol, and virtual servers translate NDN Data packets from the network into IP packets for the local client, and vice versa. Aside from text chat, the system also supports group whiteboard function, as long as the local clients support it (e.g. Coccinella [7]).

### 3.1.4 Libraries and APIs

We made significant progress in lowering the barrier to NDN experimentation and application development by releasing Python bindings for the CCNx C API, which we used to build the complete video streaming application described above. We have also begun development on a "pure" Javascript implementation of the CCNx client protocol, to promote experimentation with named data networking in web browsers without requiring any native NDN code.

#### PyCCN Python Bindings

PyCCN is a set of Python bindings for CCNx, written in both C and Python[8]. While the CCNx protocol is well-defined, its use in applications is an ongoing, iterative exploration that often requires rewriting code and trying different solutions to problems. Python is popular among students and application programmers alike, an interactive shell, and many supporting libraries and packages that make it easy to quickly write and modify existing programs. In PyCCN, the C code bridges Python to the CCNx API, with higher level features in Python. The Python API is composed of modules that map closely to the basic elements of the protocol, including Name, Key, Interest, and ContentObject. These objects handle marshaling data in and out of the wire format, simplifying development from the C API. In contrast with much of the Java API, they map directly onto the protocol rather than focusing on higher level abstractions.

[3] `https://github.com/bcy/xmpp`
[4] http://adium.im/
[5] `http://www.apple.com/macosx/apps/all.html#ichat`
[6] `https://help.ubuntu.com/community/Empathy`
[7] http://thecoccinella.org/
[8] https://github.com/remap/PyCCN

**lwNDN - Javascript NDN Library**

Based on the experience developing the Python bindings, we considered other opportunities to promote experimentation with named data networking in classes and among the growing research community. In particular, we want to target more browser-based applications in the third year of research. However, building the NDN protocol into a browser (as in NDNFox) currently requires a user to run the full CCNx software stack locally. lwNDN (light-weight NDN) is an exploratory effort to build a "pure Javascript" implementation of the CCNx protocol that will run in modern web browsers without requiring any native code[9]. The current approach will connect a browser via Websockets to a remote CCNx daemon for routing and forwarding. In addition to enabling browser-based experimentation, we believe lwNDN will also prompt and support research on trust management in browsers and application performance evaluation.

The initial lwNDN implementation includes a subset of features from CCNx: (1) Encoding content objects and interest objects into the ccnb wire format; (2) Decoding ccnb packets into content objects and interest objects; (3) Creating and verifying content object signatures; and (4) Sending interest objects and receiving content objects. Work in progress includes: (1) Exclusion filters for Interests; (2) Face registration; (3) Key generation; and (4) Support for a standalone version outside the browser using Node.js. Current communication between lwNDN and CCNx nodes uses UDP through a Java Applet; using WebSockets instead will enable an entirely Javascript solution.

**Chronos - Serverless Distributed Applications**

Chronos is a new design for multi-user realtime applications which synchronizes data in a distributed, serverless fashion over NDN. As one of our first experiments in using Sync as a communication primitive, Chronos is designed to support the Multi-User Chat application described above. The design has two main components: data set state and data storage. The data set state maintains current knowledge of the chat data set in form of a digest tree, as well as maintains history of the data set changes in form of digest logs. Taking advantage of the naming rules, we name the chat data sequentially and the digest tree only needs to keep track of the status of each participant (e.g. the latest sequence number of a participant).

Chronos participants interact with each other using two types of Interest/Data message exchanges: synchronization (sync) and chat data. A sync Interest represents the senders knowledge of the current chat data set in form of cryptographic digest, obtained using digest tree, and is delivered to every other participant. Any recipient of the sync Interest who notices that it has more information, satisfies this Interest with a Data packet that includes the missing piece of the data set. Identification of the missing piece occurs through the use of the digest carried in the sync Interest, in combination with the digest logs. So as soon as a participant discovers changes to the chat room data set, it sends out chat data Interests to pull messages from their originators. After recovers from a network partition, participants may will request recovery information from each other to bring everyone to a common state.

We implemented the Chronos design in our newly developed NDN simulator, ndnSIM, which contains an implementation of the NDN protocol module that approximates behavior of the existing code base of the CCNx implementation. We then conducted extensive evaluation experimentations. The results demonstrate that Chronos, by leveraging benefits of the serverless design and NDN primitives (application naming convention, loop-free Interests broadcasting, Data caching), can provide a multi-user synchronization function that has lower overhead and lower delay in new data delivery than traditional centralized server based implementations, and is robust against link failures and packet losses. The design and evaluation results are included in a paper under submission. We also implemented the design as a real time synchronization library (in C++) and rewrote the multi-user chat application as an example of library usage.

### 3.1.5 Standardized Hardware and Software for the Testbed

To scale up NDN applications and encourage their use by other team members, we have deployed a uniform "application box" (an inexpensive Linux-based workstation) to all interested NDN campuses. These boxes are

[9] https://github.com/remap/lwndn

configured with a standard CCNx distribution and all available NDN applications, and enable remote updates so that we can push new applications and library builds easily. They are connected to their geographically closest NDN hub in the NDN testbed. Through this process we are identifying challenges in deployment and, we hope, in the next year will increase application usage. Currently seven campuses have application boxes that support audio & video conferencing, chat, and video publishing and playback. Additionally, the "app boxes" have helped us to better understand deployment issues such as the impact of MTU limitations in the underlying IP transport layer.

### 3.1.6 New Architectural Findings from Application Development

Application development continues to generate questions and design patterns that inform further development of the NDN architecture. Our findings during year 2 are summarized below.

1. **Repositories:** Our experience shows that repository performance, configurability, and architecture are key topics to explore. Video streaming has generated several challenges for the repository implementation, including the need for rapid packet writes, signing and verification as well as large storage capacities.
2. **Prioritized SYNC:** We identified a common need for synchronizing data with varying priorities. While the simplest synchronization primitive between two nodes is of the form "give me all content you have that I do not have", some applications desire prioritized content exchange and preferential expiration/deletion. For example, when two mobile devices meet (e.g., phones in two moving vehicles as in UIUC's work), the encounter time might be short. The order in which content is to be exchanged can have impact on maximizing information flow. While some factors affecting optimal prioritization are application-specific, general policies may be applicable, such as most recent first, most diverse first, or most corroborated first. A flexible library of common prioritization techniques could significantly facilitate application development.
3. **SYNC beyond Peer Scope:** The CCNx Sync implementation offers promise for efficiently synchronizing collections of data, but currently limits synchronization to directly connected nodes. We will explore solutions for Sync across nodes separated by arbitrary distances, which may involve both new application design styles (with judicious use of broadcast namespaces) and/or changes to the protocol itself.
4. **Preferential deletion:** Current cache replacement policies treat content objects independently, but if their popularity is not independent, UIUC has shown that more sophisticated content replacement/expiration/deletion policies that are cognizant of relations between objects (e.g., information overlap) can significantly improve application performance. NDN allows one to encode such relations in content names, enabling new types of distributed content storage services that optimize application objectives.
5. **Content metadata standards:** For applications such as the NDNFox Mozilla Browser that uses NDN to fetch data, application naming standards that provide content type metadata (analogous to HTTP Content-Type data) will be needed to enable content receivers to render the data they obtain.
6. **Key distribution standards:** Application-specific and highly applicable key distribution services will enable applications to take advantage of NDN's inherent use of signatures and verification.
7. **Long-lived Interests:** With NDN's pull-based architecture, we need efficient solutions to support event-triggered, low-latency data dissemination.

### 3.1.7 Reference Implementation Feedback from Applications

We provided the following feedback to the CCNx reference implementation based on our initial experience in application deployment.

1. In high-throughput applications such as video, Interest timeouts are occurring unexpectedly, even in local communications. Interest re-ordering or other internal implementation details may be the cause.

2. As observed last year, the CCNx daemon may benefit from a plug-in model, such as that of wireshark, apache, and many other applications, which would allow shared libraries between developer community that extend CCND, instead of living on top of or otherwise modifying the ccnx core in a non-standard way. Modularity would also allow an easy way to gauge what to include in ccnx core by essentially allowing evolution of a runtime services layer between ccnx and application. Plugins could support device discovery, enhanced cryptographic functions, key-stores, etc.

## References


[1] D. Dietrich and G. Zucker. Communication and computation in buildings: A short introduction and overview. *IEEE Transactions on Industrial Eleectronics*, 57(11):3577–3584, 2010.

[2] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus. Security in networked building automation systems. In *2006 IEEE International Workshop on Factory Communication Systems*, pages 283–292, 2006.

[3] M. Mun and et al. Personal data vaults: A locus of control for personal data streams. In *Proceedings of The 6rd International Conference on emerging Networking EXperiments and Technologies (CoNEXT) 2010*, 2010.

[4] N. Ramanathan, F. Alquaddoomi, H. Falaki, D. George, C.-K. Hsieh, J. Jenkins, C. Ketcham, B. Longstaff, J. Oomsand J. Selsky, H. Tangmunarunkit, and D. Estrin. ohmage: An open mobile system for activity and experience sampling. (demo). 2012.

[5] Ronald L. Rivest and Butler Lampson. SDSI - A Simple Distributed Security Infrastructure. Technical report, MIT, 1996.

[6] Zhenkai Zhu and et al. Act: An audio conference tool over named data networking. In *ACM SIGCOMM ICN Workshop*, 2011.


## Publications


1. A. Afanasyev et al., "Developing NS-3 based NDN simulator", CCNx Community Meeting held at PARC, 2011.
2. Jeff Burke, Paolo Gasti Naveen Nathan and Gene Tsudik, Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control via Named-Data Networking, in submission.
3. Mani Srivastava, Tarek Abdelzaher, Boleslaw K. Szymanski, "Human-centric Sensing," Philosophical Transactions of the RoyalSociety, special issue on Wireless Sensor Networks, Jan 2012
4. Lu Su, Yong Yang, Bolin Ding, Jing Gao, Tarek F. Abdelzaher, andJiawei Han, "Hierarchical Aggregate Classification with LimitedSupervision for Data Reduction in Wireless Sensor Networks," ACMSensys, Seattle, WA, Nov 2011
5. Tarek Abdelzaher, "Green GPS-assisted Vehicular Navigation," Handbook of Energy-Aware and Green Computing, Chapman & Hall/CRC, 2012.
6. Md Yusuf Sarwar Uddin, Hongyan Wang, Fatemeh Saremi, Guo-Jun Qi,Tarek Abdelzaher and Thomas Huang, "PhotoNet: A Similarity-awarePicture Delivery Service for Situation Awareness," IEEE Real-timeSystems Symposium (RTSS), Vienna, Austria, Nov 2011
7. Shen Li, Hieu Le, Nam Pham, Jin Heo and Tarek Abdelzaher, "JointOptimization of Computing and Cooling Energy: Analytic Model and AMachine Room Case Study," IEEE International Conference on DistributedComputing Systems (ICDCS), Macau, China, June 2012.
8. Shen Li, Shiguang Wang, Tarek Abdelzaher, Maria Kihl, and AndersRobertsson, "Temperature Aware Power Allocation: An OptimizationFramework and Case Studies," Journal of Sustainable Comput-ing:Informatics, and Systems (accepted, 2012).

## 3.2 Routing

| Contributors | |
|---|---|
| **PIs** | Beichuan Zhang (Arizona), Lan Wang (Memphis), Dmitri Krioukov and kc Claffy(CAIDA), Lixia Zhang (UCLA) |
| **Grad Students** | Cheng Yi, Junxiao Shi, Varun Khare and Greg Lutostanski (Arizona); A. K. M. Mahmudul Hoque and Yaoqing Liu (Memphis); Alexander Afanasyev and Ilya Moiseenko (UCLA) |
| **Undergrads** | Yifeng Li (Arizona); Gus Sanders, Adam Alyyan, Oksana Koziaryk, and Andrew Hood (Memphis) |
| **Staff** | Ken Keys, Marina Fomenkov and Alex Ma (CAIDA); Syed Obaid Amin (Memphis) |

The goal of NDN's network layer is to provide a name-based packet delivery service for applications to build upon. To meet the requirements of the future Internet, the network layer must have:

1. *Scalability*: support a large Internet topology and large number of name prefixes.
2. *Security*: provide integrity, provenance, and pertinence of routing messages.
3. *Resiliency*: detect and recover from packet delivery problems quickly.
4. *Efficiency*: exploit multi-path forwarding and data caching for efficient use of network resources.

Our technical approach begins with extending existing routing protocols, such as OSPF (intra-domain) and BGP (inter-domain), to support name-based routing. These extensions include supporting name prefixes in routing updates and route computation, providing multipath capability, and using NDN instead of IP as underlying transport mechanism. We use these extensions to explore longer-term solutions and challenges. One major challenge is scaling routing table size, for which we are pursuing two different technical approaches, described below. We also plan to develop and evaluate resilient and efficient packet forwarding strategies by leveraging per-packet state at each node to observe and respond to data-plane performance problems. Finally, we are exploring how NDN's multipath capability can make more efficient use of available network resources. This section describes the following activities and findings in the routing area during the second year of the NDN project:

- **Dynamic routing protocol**: We have designed and implemented a named-based dynamic routing protocol, *OSPFN* [8], deployed it on the NDN testbed, and developed monitoring tools. With this, dynamic routing is now fully functioning on our testbed.
- **Routing scalability**: We sketched the use of a *forwarding hint* [2] to scale NDN's core routing table size while retaining the benefits of named-based routing.
- **Hyperbolic routing**: Applying hyperbolic greedy routing on the current NDN testbed topology, our calculation shows that the success ratio of packet delivery is high.
- **Forwarding strategy**: Using the newly developed ns-3 NDN simulator, we compared NDN, IP, and Path Splicing in face of prefix hijacking, link failures, and congestion. The results show that NDN's forwarding strategy can effectively detect problems and recover from them [9].
- **Link protocol**: We designed and implemented a link protocol, *NDNLP*[7], which provides two optional features at each hop: fragmentation/reassembly of NDN messages, and detection/recovery of packet loss over the link. NDNLP enables the delivery of NDN packet with arbitrary sizes over Ethernet or other layer-2 protocols directly, and with efficient loss recovery.

### 3.2.1 OSPFN

To provide dynamic routing capability in NDN, we have extended OSPF to distribute name prefixes and calculate routes to name prefixes, and deployed our implementation on the NDN testbed. Here we describe OSPFN's design, implementation, deployment, and status monitoring. We use OSPF's Opaque LSAs or OLSA [3] to announce name prefixes. OLSA allows application-specific information to be advertised in the

network. Although legacy nodes do not use these LSAs to build their topology, they will still forward them, ensuring backward compatibility. Open source routing suites, such as Quagga OSPF, provide an API that allows easy injection and retrieval of OLSAs through OSPF, which makes OLSA a perfect candidate for advertising name prefixes.

Each NDN node runs CCND, OSPFN and OSPFD in parallel. OSPFN builds Name OLSAs and injects them into the local OSPFD, which floods OLSAs to the entire network. When OSPFD at a node receives an OSLA, it delivers the OLSA to its local OSPFN. Since each LSA carries the ID of the router that originated the LSA, OSPFN can obtain the router ID of a Name OLSA and query OSPFD to retrieve the nexthop to the router (note that OSPFD still floods its regular LSAs and computes the shortest path tree based on the topology). OSPFN then installs the name prefix and its associated nexthop in the CCND FIB. Figure 3.3 illustrates the relationship between the three protocols.


Figure 3.3: Relationship between CCND, OSPFN, and OSPFD

| 0 | 8 | 16 | 24 | 31 |
|---|---|---|---|---|
| LS Age | | Options | LS Type | |
| Opaque Type | Opaque ID | | | |
| Advertising Router | | | | |
| LS Sequence Number | | | | |
| LS Checksum | | Length | | |
| Opaque Information (variable size) | | | | |

| | |
|---|---|
| **LS Age** | The age of the LSA in seconds |
| **Options** | Optional capabilities associated with the LSA |
| **LS Type** | The link-state type of the Opaque LSA that identifies the LSAs range of topological distribution. The three types used are as follows:<br>• 9 – link-local scope; Opaque LSAs are not flooded beyond the local (sub)network<br>• 10 – area-local scope; Opaque LSAs are not flooded beyond the local area<br>• 11 – AS-wide scope; Opaque LSAs are flooded throughout the AS |
| **Opaque Type** | Specifies the application specific type of the Opaque LSA |
| **Opaque ID** | Type-specific ID |
| **Advertising Router** | Router ID of the LSAs originator |
| **LS Sequence Number** | Used to detect old and duplicate LSAs |
| **LS Checksum** | Checksum of the complete header excepting the LS Age field |
| **Length** | Total header length |
| **Opaque Information** | Application-specific data |

| **Name LSA Opaque Information** |
|---|
| Size in Bytes of Name Prefix (32 bits) |
| Name Type (8 bits) |
| Name Prefix (variable size) |

Figure 3.4: Name OLSA Message Format

Because OSPF provides only a single nexthop for each destination, OSPFN by default generates a single CCNx route for each name prefix. However, one of the unique features of NDN is its forwarding strategy, which can explore multiple paths to retrieve a piece of named data. As an initial step to enable multipath routing on the testbed, OSPFN allows operators to specify a ranked list of interfaces and inserts the corresponding routes in CCND's FIB, so that CCNx will try them when the best path fails to bring back data. Each interface is associated with a preference; the more preferred interfaces is tried first. We call this feature "configured multipath routing". To ease the burden on operators, the multipath configuration is specified for each node, not for each name prefix. This preference order serves only as a starting point for CCND to explore. Once Interest and Data packets start flowing, CCND will observe data retrieval delay of each interface and take that into account for future forwarding decisions.

The Name OLSA message and field description are shown in Figure 3.4; most fields are assigned by

OSPFD in the advertising router. OLSA has three scopes for flooding the updates and we use LS Type "10" for area scope flooding. The Opaque Type field can contain the value range 127-255 for application specific use and we use 236 for Name OLSA. The Opaque Information field carries a 32-bit field for the name prefix size, an 8-bit field for the name prefix format, as well as the actual name prefix.

NDN teams at the University of Memphis and the University of Arizona developed OSPFN based on Quagga 0.99.17 [6] and tested it Ubuntu 10.04, 10.10, 11.04, 12.04, Fedora 9, and FreeBSD 9.0. We released the first version on Oct. 18, 2011 and the second version on May 15, 2012, via `https://github.com/NDN-Routing/OSPFN2.0`. We have deployed OSPFN on the NDN testbed, and have developed web-based tools to monitor which prefixes are advertised by OSPFN and which neighbors are advertised by OSPFD at each NDN hub, as well as the FIB entries installed in CCND on each NDN hub. These real-time status pages have been instrumental in deploying OSPFN and debugging testbed problems.

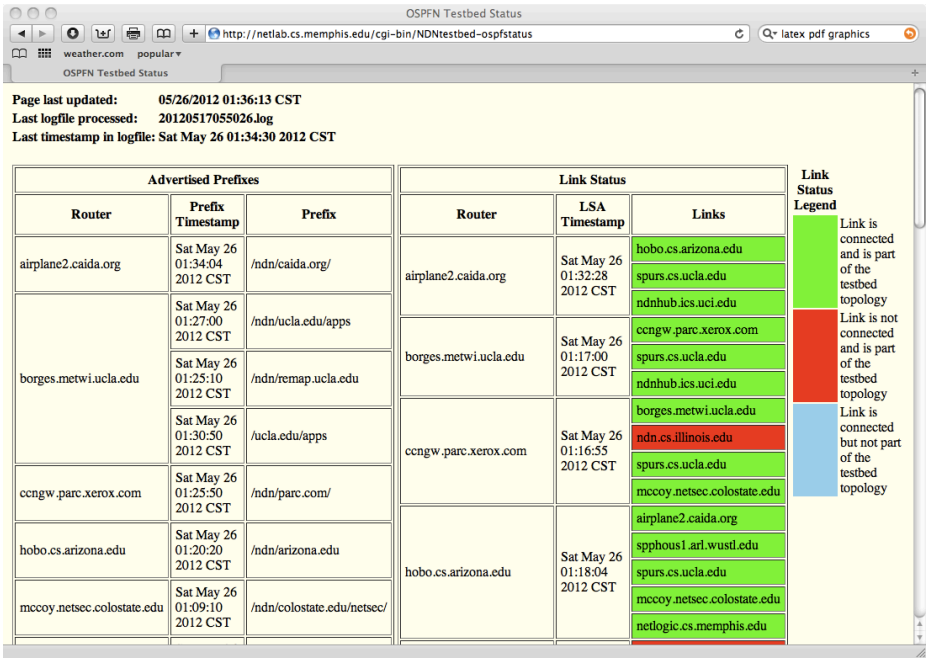


Figure 3.5: OSPFN Status on NDN Testbed

### 3.2.2 Scaling Routing Tables

Map-n-Encap was proposed long ago to scale IP routing in face of a large number of provider-independent (i.e., non-aggregatable) addresses. The idea is to use a mapping system to map a PI destination address to a PA (provider-aggregatable) address, and then tunnel packets to the destination via the PA address. This technique allows the core to maintain only PA address prefixes, by moving the scalability issue from the routing to the mapping system. We propose to apply the same Map-n-Encap idea to scale NDN routing. For each Interest packet that carries a PI application name, we first look up the mapping system to find corresponding ISP information and then use this information to help forward the packet. Today's DNS can easily handle the scale of the mapping system. Since the mapping happens at the edge of the network, it will have no impact on the network core.

The most straightforward way to implement the "encapsulation" step is by *name concatenation*. An ISP name prefix is prepended to the application data name, forming a provider-dependent name to be used in both Interest and Data packets. The Interest packet does not change, but the Data packet is encapsulated: the inner packet contains the original application name, content and signature that binds the two, and the outer packet contains the provider-dependent name, the inner packet as its content payload, and its own signature. Although name concatenation achieves the goal of forwarding packets by ISP name prefixes, it

changes the names used for fetching data, which creates two problems: signing overhead (for inner and outer packet) and topology-dependent names, which reduces caching efficiency and content availability. As an alternative, we added a new field into the Interest packet to carry the ISP name prefix. Application names in Interest and Data packets are intact and visible to routers, allowing Interests to be satisfied by cached Data regardless of where the Data is originated. The forwarding hint is used in routing table lookup when Interests are processed, and suggests to routers where to forward them. Use of the forwarding hint requires no changes to Data packets or their processing, and the only change required for data producers is to tell the mapping system which ISPs provide it transit. Carrying a forwarding hit in an Interest packet does not imply the Interest will necessarily reach the producer or its ISP; Interests will bring back Data from the first router along the path where the Data is cached. To map application names to ISP names, we plant to build a distributed mapping system similar to DNS.

### 3.2.3 Routing on Hyperbolic Metric Space

In a previous work [4] we described how the hyperbolic metric space underlying complex networks enables efficient greedy forwarding (GF) without any global knowledge of the network topology. Since each node in the network has coordinates in this hidden metric space, a node can compute the distance between each of its neighbors in the network and destinations whose coordinates are carried in packets. GF then amounts to forwarding the packet to the neighbor node closest to the destination in the hyperbolic space. In the experiments using the NDN testbed, we tested the modified greedy forwarding (MGF) algorithm that excludes the current node from any distance comparisons and finds the neighbor closest to the destination. The packet is dropped if this neighbor is the same as the packet's previously visited node.

We used two types of metrics to evaluate the efficiency of MGF: (1) the success ratio, which is the percentage of successful paths that reach their destinations; and (2) three types of (average) stretch. The standard hop stretch measured on the actual topology is the ratio between the hop lengths of greedy paths and the corresponding shortest paths in the graph. (Optimal paths have stretch equal to 1.) The second type of stretch is measured in the underlying hyperbolic space, the ratio of the length of a *successful* greedy path to the actual hyperbolic distance between the source and the destination. Finally, the third type of strength is also measured in the underlying hyperbolic space, but is the ratio of the length of the *shortest* path to the actual hyperbolic distance between the source and the destination. The lower these two hyperbolic stretches, the closer the greedy and shortest paths stay to the hyperbolic geodesics, and the more congruent the network topology is with the underlying geometry.

We also modeled network growth on the NDN testbed. Specifically, we assigned to the testbed routers the hyperbolic coordinates of their corresponding ASes obtained in hyperbolic mapping, and then simulated network growth by connecting each node to its m=1,2,3 hyperbolically closest nodes. For each value of m, we measured the success ratio and stretches of three types in the resulting networks, as well as in the networks obtained by all possible removals of one link and one node. All paths in all cases were successful even for small values of m=2 and 3, except for a few unsuccessful paths upon the removal of node 14048 (Memphis), which appears to be the core (highest-degree) node in the resulting network. However, at m=3 the percentage of successful paths was still 94% even with the failure of the Memphis node.

### 3.2.4 Adaptive Forwarding Strategies

Unlike in an IP network where the forwarding process simply looks up the routing table to direct traffic, NDN's forwarding process can adapt its forwarding decisions based on the inputs from both routing tables and data-plane observations. We continued our research on forwarding strategy from last year by fine-tuning the design and comparing performance with Path Splicing, an IP-based multipath solution.

By default, NDN routers discover failures only when timeouts occur, which can be relatively slow. Also, in the case of failures, unsatisfied (but not yet expired) Interests ("*dangling state*") can block other consumers from getting data, since routers have already forwarded the Interest and are waiting for Data to arrive across now failed links. We introduced an *Interest NACK* to address these issues. When an NDN node can neither satisfy nor forward an Interest (e.g., there is no interface available for the requested name), it sends an Interest NACK back to the downstream node. If the downstream node has exhausted all its own forwarding

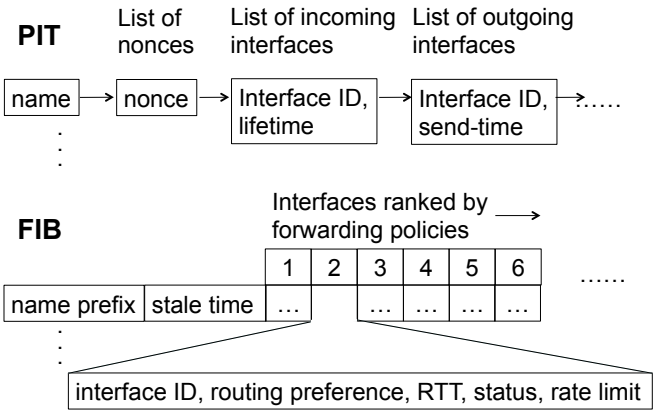


Figure 3.6: Forwarding State in PIT and FIB

options, it will send a NACK further downstream. In the absence of packet losses, every pending Interest is consumed by either a returned Data packet or an Interest NACK, which speeds up problem detection and clears dangling state in the network. An Interest NACK carries the same name and nonce as the original Interest, plus an error code explaining the reason why the Interest cannot be satisfied or forwarded. So far we have implemented codes for duplicate interest, no data, no prefix, and congestion.

Figure 3.6 shows the basic information maintained in a router's PIT and FIB. A PIT entry is created for each requested name. It contains a list of nonces that have been seen for that name, a list of incoming interfaces from which Interests for that name have been received, as well as a list of outgoing interfaces to which Interests have been forwarded. An NDN FIB differs from an IP FIB in two fundamental ways: (1) an IP FIB entry usually contains a single best next-hop, while an NDN FIB entry contains *a ranked list of multiple interfaces*; (2) an IP FIB entry contains nothing but the next-hop information, while an NDN FIB entry records information from both routing and data planes to support adaptive forwarding decisions.

In our first prototype design, a FIB entry records the status of each interface as working (green), possibly working (yellow), or red (not working). A new or newly learned interface's status is Yellow, and turns Green if Data flows back from that interface. A Green interface turns Yellow when a pending Interest times out (i.e., no Data returns within the expected time), after Data ceases flowing for a certain time, or upon a receipt of a "No Data" NACK. An interface turns Red if it goes down, or upon receipt of a "No Prefix" NACK. Other NACK types (Duplicate, No Path, or Congestion) do not affect the interface color, although the Congestion NACK reduces the rate of Interests sent to the corresponding interface. Green interfaces are always preferred over Yellow ones. Using the information stored in the PIT and FIB, the strategy module then determines when and which interface to use to forward an Interest, adapting to network conditions.

Our initial design includes the handling of new Interests, retransmitted Interests, Interest NACKs, and also proactive probing of interfaces in the following ways:

- **New Interest:** If a router receives a new Interest without a match in the PIT, it will create a PIT entry and forward the Interest to the highest-ranked available Green (or if no Green, Yellow) interface. If there is no available interface, it returns a NACK with code "Congestion." When forwarding an Interest, the router starts an *exploration timer* to limit how long to explore available interfaces (if a NACK is returned) to retrieve the Data.
- **Retransmitted Interest:** If an Interest matches an existing PIT entry and is received from an interface listed in this PIT entry, but Interest's nonce does not exist in the nonce list, the router regards this Interest as a retransmission. If the exploration timer has expired, the router will forward the Interest and restart the exploration timer.
- **Interest NACK:** Upon receiving an Interest NACK, a router will send an Interest with the same name and nonce to the next highest-ranked available interface only if the exploration timer of the corresponding PIT entry has not expired. After the timer goes off, exploration can only be triggered by another Interest with the same name.

- **Interface Probing:** A router prefers currently working (Green) interfaces. It also periodically probes Yellow faces to discover recovered or better performance paths.

We have implemented the basic NDN data plane and forwarding scheme in ndnSIM [1], and used simulations to compare NDN and Path Splicing [5], an IP-based adaptive multipath forwarding scheme, under network faults including traffic congestion, prefix hijacking, and link failures. In Path Splicing, each router has multiple routing tables, called slices, each computed using the same topology and the same shortest-path algorithm but a different set of link weights. When an end-host sends a packet, it adds an ordered list of tags to the packet header; each tag is an index to the slice to use at each hop, and routers forward the packet according to the tags. Thus end hosts can choose a different path for a packet by tagging it differently, without knowing the exact path it may take. In path splicing, end hosts can observe two-way traffic and explore different paths via extra state installed in Path Splicing routers; in NDN this capability extends to all nodes in the network.

We simulated "blackhole" hijacking over NDN, IP, and Path Splicing, exhausting all combinations of (consumer, producer, attacker) tuples in the topology. In NDN, over 94% cases of all consumer-producer communications were unaffected by hijacks, and in the affected cases NDN could recover all except the unreachable cases where the consumer or producer was single-homed to the attacker. The same hijack in an IP network affected 50% of cases, because traffic affected under one producer/attacker pair is unaffected when the producer and attacker switch roles. Since IP routers strictly follow paths given by routing protocols, it is no surprise that none of the affected traffic is recoverable. In simulating Path Splicing under the same hijack attacks, the results showed the same number of affected traffic as in the IP case, but 6% of affected cases recovered. In these cases, when a consumer's request times out, the consumer retransmits the request with different tags, causing the packet to follow a different path. When content returns, the consumer keeps using the same tag. Thus, Path Splicing gives end hosts some influence over path selection, but hosts have no knowledge of network internals so can only randomly select tags, and the number of route options is bounded by the number of precomputed slices.

NDN's resiliency and performance is due to its fast local recovery. When a link fails, the NDN router will mark this interface RED and send Interests to other interfaces per its forwarding strategy. When a router detects looping Interests, it tries the next interface. If the router has tried all possible interfaces, it returns a NACK to the downstream node, which will explore its own alternative interfaces. Routers label interfaces they have received data from as GREEN, and use them to forward future Interests. Therefore, the exploration starts when and where the failure happens and gradually pushes back toward the consumer until a working path is found. Since a FIB entry's outgoing interfaces are ordered based on routing preference and previous performance, a router tries better interfaces before moving down the list, and finds working paths quickly. IP forwarding does not have any such adaptability, and the adaptability of Path Splicing is limited by the requirement to initiate exploration at the end hosts rather than local to the failure incident, and without any hints from the network. As a result, Path Splicing takes much longer time to retrieve data than NDN and in general the found paths are also longer.

### 3.2.5 Link Protocol

We identified two issues when running current CCNx on non-TCP tunnels such as Ethernet or UDP tunnels: (1) CCNx does not fragment/reassemble packets, thus packets larger than the link MTU cannot be sent; and (2) packet losses over unreliable links can impact the upper layer's performance. To solve these issues, we developed the NDN Link Protocol (NDNLP) to provide optional fragmentation/reassembly and loss detection/recovery. NDNLP operates between CCND and layer-2 protocol (e.g., Ethernet) or underlying tunnels (e.g., UDP tunnels). Each feature can be used independently; for example, on top of Ethernet, fragmentation/reassembly should be turned on, but loss detection/recovery is optional.

To extend CCNx with this NDNLP protocol, we implemented a daemon program *ndnld*. Acting as a CCND client, ndnld creates a face for each incoming connection on ccnd, and delivers messages coming from this face to a remote host also running ndnld. The ccnd process handles routing and forwarding, while ndnld handles communication with local link neighbors. Another utility program, ndnldc, sends control commands (Interests) to a separate *control face*, conveying information about local link neighbors, such as

MAC addresses, MTUs, etc. We implemented ndnld and ndnldc as CCNx applications, and tested them over Ethernet, UDP tunnels, and TCP tunnels. Both features worked well. This software will be released soon; details are in [7].

## References


[1] Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM: NDN simulator for NS-3. Technical Report NDN-0005, NDN Project, July 2012.

[2] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. Scaling NDN Routing: Old Tale, New Design. Technical Report NDN-0004, NDN Project, July 2012.

[3] L. Berger, I. Brystkin, A. Zinin, and R. Coltun. The OSPF opaque LSA option. *RFC 5250*, July 2008.

[4] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, Sep 2010.

[5] Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. Path splicing. In *Proceedings of ACM SIGCOMM*, 2008.

[6] Quagga routing software suite. `http://www.quagga.net`.

[7] Junxiao Shi and Beichuan Zhang. A Link Protocol for NDN. Technical Report NDN-0006, NDN Project, July 2012.

[8] Lan Wang, A K M Mahmudul Hoque, Cheng Yi, Adam Alyyan, and Beichuan Zhang. OSPFN: An OSPF Based Routing Protocol for Named Data Networking. Technical Report NDN-0003, NDN Project, July 2012.

[9] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. A Case for Stateful Forwarding Plane. Technical Report NDN-0002, NDN Project, July 2012. (submitted to Computer Communications, 2012).


## Publications


1. Cheng Yi, Alexander Afanasyev, Lan Wang, Beichuan Zhang, Lixia Zhang, "Adaptive Forwarding in Named Data Networking," *ACM SIGCOMM CCR*, 2012.

2. Cheng Yi, Alexander Afanasyev, Lan Wang, Beichuan Zhang, Lixia Zhang, "Smart Forwarding: A Case for Stateful Data Plane," *submitted to* Computer Communications, 2012.

3. Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, Lixia Zhang, "Scaling NDN Routing: Old Tale, New Design," NDN Technical Report, 2012.

4. Junxiao Shi, Beichuan Zhang, "A Link Protocol for NDN," NDN Technical Report, 2012.

5. Lan Wang, A K M Mahmudul Hoque, Cheng Yi, Adam Alyyan, Beichuan Zhang, "OSPFN: An OSPF Based Routing Protocol for Named Data Networking," NDN Technical Report, 2012.

6. F. Papadopoulos, M. Bogu, and D. Krioukov, "Popularity versus Similarity in Growing Networks", Technical Report, http://arxiv.org/abs/1106.0286, April 2012.

## 3.3 Scalable Forwarding

| Contributors | |
|---|---|
| **PIs** | Patrick Crowley (WashU) |
| **Grad Students** | Haowei Yuan, Shakir James (WashU) |
| **Undergrads** | Jyoti Parwatikar, John Dehart (WashU) |

With respect to forwarding research, our goal is to develop fast, scalable NDN node prototypes. By fast, we mean that we intend to support 1 Gbps links at line-rates in software implementations; we expect our hardware-based, and hardware-accelerated implementations to exceed this rate by at least an order of magnitude. By scalable, we mean that we intend to develop an NDN forwarding plane that can support hundreds of thousands of names, each of arbitrary length. (Our scalability goals are not as explicit as our forwarding rate goals because we aren't yet confident that we understand the requirements or relevant range of name counts and lengths.) In addition, we provide feedback to the areas of architecture and routing whenever their design choices might have substantial performance consequences in the data plane. We also need to keep NDN forwarding comparable, with respect to features and performance, to other name-based forwarding schemes.

Our technical approach relies on the Open Network Lab, an NSF-funded testbed where experimenters can design a network topology, configure and program gigabit routers, and control Linux end-hosts. ONL currently contains 14 programmable routers, over 100 Linux hosts, and 1 Gb and 10 Gb links and switches. To support NDN experimentation, we added a new end-host type to ONL that boots with CCNx, the NDN reference implementation, installed and properly configured. CCNx routes and statistics can be changed and monitored via the RLI. The RLI supports real-time charts, which can be used to report end-hosts characteristics in real-time, including link bandwidth, packets/s, queue lengths, CPU utilization, and memory utilization. ONL accounts and usage are free. Registration and step-by-step CCNx instructions can be found at http://wiki.arl.wustl.edu/onl/index.php/CCNx_in_ONL.

The rest of this section provides more details about the following activities and findings in the forwarding area during the second year of the NDN project:

- We have clarified the content, issues and principles in scalable NDN forwarding design, and proposed a simplified operational flow based on the NDN reference implementation, CCNx.
- We presented a live NDN demo at the GENI engineering conference.
- We demonstrated architectural advantages of the NDN design by comparing content distribution performance using NDN and HTTP, in both wired and emulated wireless networking environments.

### 3.3.1 Concepts, Issues, and Principles

The challenges of designing a scalable forwarding plane derive primarily from variable-length names and the read/write nature of packet forwarding. As a result, efficient algorithms and data structures, as well as advanced hardware devices, are needed to accelerate NDN forwarding. Our measurements show that the peak throughput of the CCNx implementation is much lower than the 1 Gbps link rate that we consider a minimum requirement. It is important to understand the status and practical limits of the current NDN prototype implementation in order to guide future research in this field. We note that recent papers have considered closely related topics, such as general content-centric router design, cost estimates for building content centric networks and theoretical analyses of the performance of content centric networking in general. To date, however, there has been no discussion or investigation of how data structures and algorithms should be applied to the specific topic of name-based forwarding, as embodied in the NDN forwarding plane. In particular, there is no literature on how the current CCNx software is implemented and how it performs in detail. We feel the essential problems in NDN forwarding need to be further clarified and that a formal description of these challenges will be beneficial for the broader research community.

For context, we summarize some major concepts relevant to NDN forwarding,

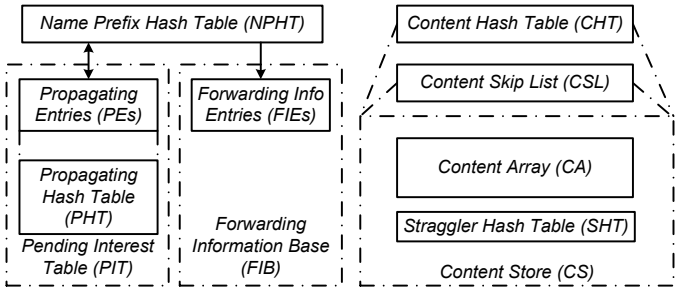


Figure 3.7: CCNx Data Structures

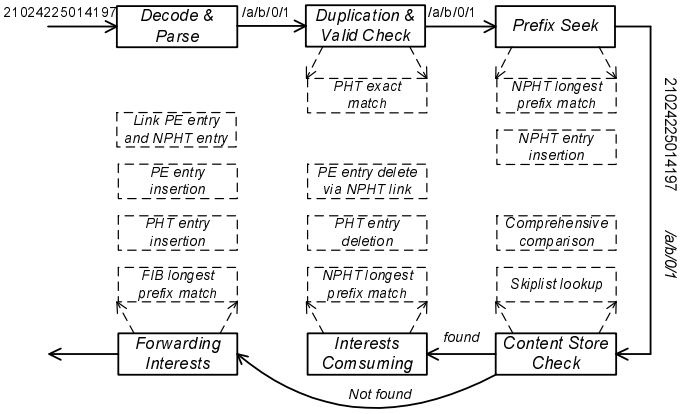


Figure 3.8: Operational Flow of Processing Interest Packets in CCNx

1. **CCNx Data Structures** Each NDN forwarding node has three logical components: Content Store, Pending Interest Table and Forwarding Information Base. Figure 3.7 shows the data structures that implement these components in CCNx. The FIB and PIT share a hash table named *Name Prefix Hash Table (NPHT)*, which indexes the *Propagating Entries(PEs)* and *Forwarding Info Entries (FIEs)*. Each bucket in the NPHT has pointers pointing to PEs and FIEs, structures storing detailed pending Interest information and forwarding information, respectively. The *Propagating Hash Table (PHT)* is keyed by the *nonce* field, which is unique for each Interest packet. The PHT stores all the nonce field of the Interest packets presented in PIT (in the form of PEs). PHT prevents loops in the network.

   For the Content Store, the forwarding daemon assigns each Data packet a unique *accession number* which increments with each incoming Data packet. The cached Data packets are stored in an array named *Content Array (CA)* indexed by the accession numbers. Old cached content expires from the CS, and the starting and end accession numbers of the CA increase, similar to a sliding window. Old but popular Data packets whose accession number are out of range are stored in the *Straggler Hash Table (SHT)* to save space. Two data structures summarize the Content Store: *Content Hash Table (CHT)* and *Content Skip List (CSL)*. The CHT is keyed by the Data packet full name; CSL is a standard skip list data structure, chosen because it supports content-order lookup.

2. **Interest Packet Handling**

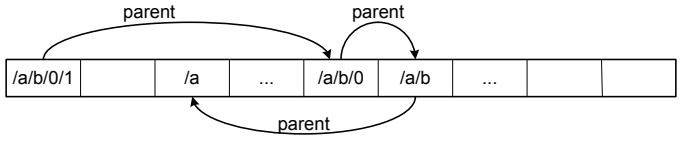


Figure 3.9: NPHT After Processing /a/b/0/1

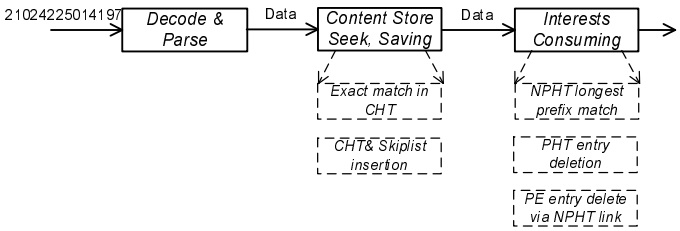


Figure 3.10: Operational Flow for Handling Data Packets in CCNx

the CCNx implementation. Once received, Interest packets are decoded, parsed, and values in their fields stored (Figure 3.8). Then an exact string matching of the nonce key is performed on the PHT to detect loops and discard looping Interests. If the Interest packet is new, CCNx checks the NPHT to make sure all prefixes of this Interest packet exist there (*Prefix Seek*), inserting new entries. Prefixes are linked by *parent* pointers for fast name lookup. After the Prefix Seek stage, the packet name is sent to the Content Store to check whether cached content can satisfy this Interest or not. In the Content Store, the Content Skip List is queried to find potential matching content following standard skip-list lookup procedure. If a potential match is found, the content is pulled from the CS, and a comprehensive match is performed to check whether the content indeed satisfies the Interest. This checking is needed because there are other restrictions in the Interest packet that do not get represented by packet names, such as the exclusion filters to exclude certain content.

If this Data packet is a true match, the Interest packet will be consumed. Currently, CCNx checks all the prefixes (not just longest) of the content name to consume as many Interests as possible. We believe performing all prefix checks in the NDN forwarding plane is an open question; it may not always be required, in which case we propose a simplified operational flow design. After sending this Data packet to all matching Interest senders, CCNx deletes the consumed Interest names from the NPHT and PHT.

If no Data packet in the CS can satisfy this Interest, CCNx consults the FIB for next-hop information, first looking for a valid FIE for the longest prefix, i.e., the full packet name, then checking prefixes in length-decreasing order by following the parent pointers. The process stops when a valid PE is found or the root prefix is reached. When this packet is forwarded, a Propagating Entry containing the entire Interest message is constructed and then linked to the NPHT bucket that stores the full name of this packet. Then the nonce field of this Interest packet is inserted into the PHT.

3. **Data Packet Handling**
   Figure 3.10 shows the operational flow of processing Data packets in the CCNx implementation. As with Interest packets, each Data packet is decoded, parsed and string-matched on the CHT to make sure this packet is not already in CS and that the lookup key is the full name of the Data packet. For non-duplicate data packets, CCNx inserts the content name into the the CHT and CSL, and the content itself into the CS. Then this Data packet will try to consume as many Interests as possible, as we just described.

NDN forwarding is complicated and more challenging than IP forwarding; we recognize three critical challenges in achieving 1 Gbps forwarding rates in software and 10 Gbps with hardware acceleration: exact string matching with fast updates; longest prefix matching for variable-length and unbounded names; and large-scale flow table maintenance. Although all three areas have been heavily studied for the currently Internet, none of the solutions apply to NDN networks since the NDN architecture requires fast per-packet updates of the PIT and CS, variable length names in packets, and much larger flow tables. To meet these fast forwarding goals, we offer five design principles:

1. Aim for constant-time operations, which might require mapping longer names to shorter ones at each NDN node

2. Exploit format of URLs, e.g., number of name components

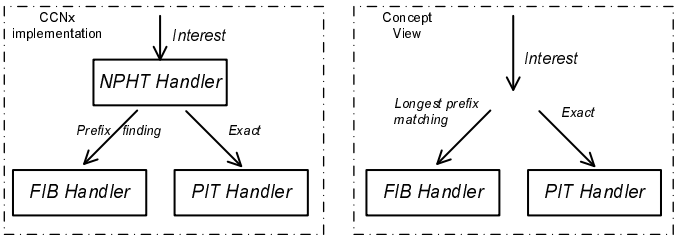


Figure 3.11: CCNX Implementation vs. Concept View of NDN Forwarding Plane

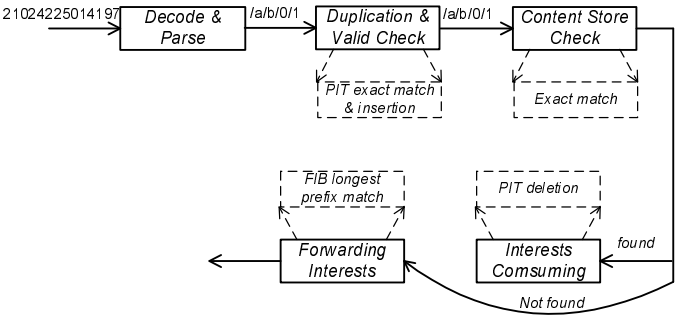


Figure 3.12: Simplified Operational Flow for Handling Interest Packets

3. Simple data structures to support fast updates, such as hash tables, and Bloom filters
4. Efficient packet encoding/decoding, either via CCNx's existing XML or a new format
5. Different content store policies in different parts of the network, so the core can be optimized

CCNx is a prototype NDN forwarding node, and some of its implementation decisions are not required by NDN. Figure 3.11 shows the CCNx implementation model and a conceptual view of the NDN forwarding plane. In CCNx, NPHT is an intermediate data structure that stores prefixes for the FIB and PIT. As a result, the longest prefix match on the FIB is converted to an exact string match in the NPHT. Sharing the NPHT between the FIB and PIT reduces memory utilization. Longest prefix matching in the FIB and exact string matching in the PIT are core NDN forwarding functions. Figures 3.12 and 3.13 show simplified operational flows for Interest and Data packets, respectively. The conceptual flow is the same as in CCNx (Figure 3.8) except that we remove the NPHT data structure, which allows efficient data structures and algorithms to optimize forwarding performance, e.g, pipelined-based architectures designed into hardware.

### 3.3.2 Live NDN Demo at GENI Engineering Conference

We coordinated a live NDN demo at the 13th GENI engineering conference in March 2012 in Los Angeles, using the NDN testbed, augmented with several hundred Amazon EC2 instances. We also developed a map-based web app that visualized real-time link utilization on the NDN testbed (Figure 3.14). The demo had three parts: (1) a live text chat; (2) video streaming, illustrating that scaling the number of clients does not increase server load; (3) robo-chat, an active chat room with approximately 250 hosts participating; (4) file transfer, illustrating how the NDN strategy layer can quickly react to a sequence of dropped links without

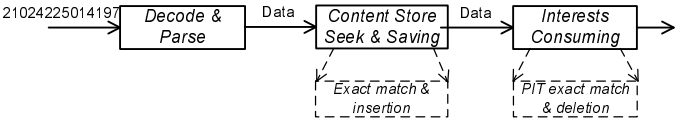


Figure 3.13: Simplified Operational Flow for Handling Data Packets

| Demo Feature | Realization |
|---|---|
| Large-scale, wide-area operation | All 4 US time zones, 300 machines |
| Mix of content distribution and interactive apps | 4 distinct services |
| Visualization of both app-level and net-level activity | Visualization Web App |
| Demonstrate both steady-state and react-to-change modes | Drop links during app sessions |
| Something IP+HTTP cannot do | Video streaming, multi-path routing |

Table 3.1: NDN Live Demonstration Goals

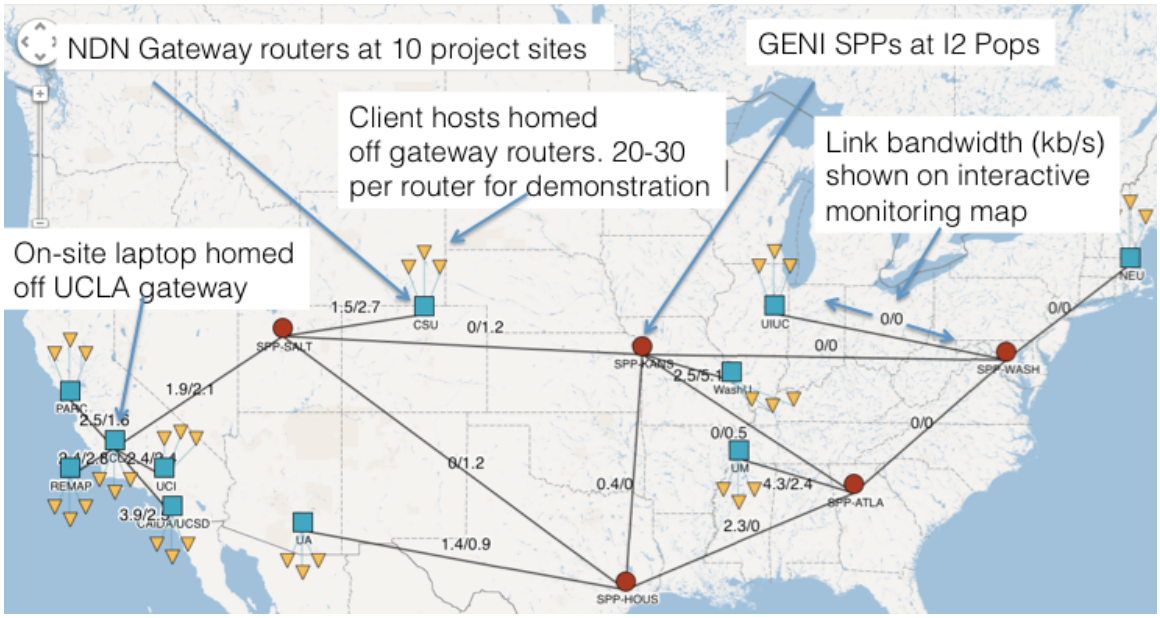


Figure 3.14: NDN Testbed and Live Visualization

interrupting the file transfer; and (5) video and audio from a live musical performance from Jeff Burke's REMAP studio space at UCLA. The reaction at the conference was very positive.

### 3.3.3 Experimental Evaluation of Content Distribution with NDN and HTTP

**Experimental Setup** We used a set of CCNx software tools, mostly provided by the CCNx software distribution, and some written on our own, to perform file distribution. The CCNx implementation we evaluated is ccnx-0.4.0, released on September 15, 2011, and with *ccnd* in default configuration. The CCNx built-in *ccncatchunks2* program takes a file name as an input argument and generates a sequence of Interest packets to fetch that file. The generated Interest starts with name ccnx:/filename/0, where the last portion is the chunk index, which increases by 1 for each generated Interest. As a result, the Interest packets generated by *ccncatchunks2* share the file name as a common prefix. There is no corresponding server side application in the CCNx distribution. We built a server side programs, *ccnfileserver* that generates Data packets with content from actual files. On the HTTP side, we chose open-source HTTP server *lighttpd*-1.4.28 and web-caching software *Squid*-3.1.11 for evaluating HTTP and Web-Caching system performance, in their default configurations, and *wget* to download files.

We compared the performance of CCNx vs lighttpd (measured by download time) in ideal wired network conditions, and then analyzed factors including the number of clients, the level of caches used by CCNx and Squid, link loss rate and link delay. When clients fetched files from a lighttpd server directly, the average file download time increased linearly as the number of clients increases. The lighttpd server's physical link (1Gbps) was saturated and thus became the bottleneck. In the case of CCNx, when there were fewer than

20 clients, the file download time was stable regardless of how many clients were fetching the file. When there were more than 20 hosts, the download time increased. because the *ccnd* daemon host itself saturated and became the bottleneck. We also compared the file distribution performance of CCNx against Squid, with similar factors as with lighttpd and got similar results. For the two-level cache case, both Squid and CCNx perform much better, as the physical link bottleneck and the *ccnd* daemon processing bottleneck disappear. Overall, the current CCNx implementation is about 10 times slower than Squid, which can partly be explained by CCNx still being in an early development stage, while Squid is a mature system.

## References


1. Somaya Arianfar, Pekka Nikander, and Jorg Ott, "On Content-Centric Router Design and Implications", in Proc. ACM ReArch 2010.

2. Diego Perino and Matteo Varvello, "A Reality Check for Content Centric Networking." in Proc. of the ACM ICN 2011.

3. Giovanna Carofiglio, Massimo Gallo, and Luca Muscariello, "Bandwidth and Storage Sharing Performance in Information Centric Networking." in Proc. of the ACM ICN 2011.

4. Jiachen Chen, et al., "COPSS: An Efficient Content Oriented Pub- lish/Subscribe System." in Proc. of ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2008.


## Publications


1. "Performance Measurement of Name-Centric Content Distribution Methods." Haowei Yuan, and Patrick Crowley. In Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '11), Brooklyn, NY, October 2011.

2. "Named Data Networking." Patrick Crowley et al. Poster presented at the 13th GENI Engineering Conference (GEC 13). Los Angeles, CA. March, 2012.

3. "Scalable NDN Forwarding: Concepts, Issues, and Principles." Haowei Yuan, Tian Song, and Patrick Crowley. To appear in International Conference on Computer Communication Netwroks(ICCCN). Munich, Germany. August 2012.

## 3.4 Security and Privacy

| Contributors | |
|---|---|
| **PIs** | Gene Tsudik (UCI) and Ersin Uzun (PARC) |
| **Grad Students** | Naveen Nathan (UCI), Yanbin Lu (UCI), Cesar Ghali (UCI) |
| **Staff** | Paolo Gasti (UCI) |

Rather than securing the communication channel at the application/session layers, as in today's Internet (through SSL/TLS), NDN secures authenticity and integrity of content packets via digital signatures. Named, signed and potentially encrypted content forms a solid foundation for routing security, but poses three major research challenges: cost-effective fine-grained security operations, functional and usable trust management, and protection of privacy.

### 3.4.1 Efficient Signing Primitives

During the second year we incorporated support for DSA [6] and EC-DSA [5] signature schemes in NDN. (In the past, only RSA signatures were supported.) DSA allows efficient signing, at the cost of slightly less efficient verification (compared to RSA [7] signing and verification). EC-DSA is a variant of DSA over elliptic curves. It reduces both signing and verification costs while balancing performance between the two operations. We also implemented content packet authentication using symmetric message authentication codes (MACs [1]). With MACs, packet authenticity and integrity are not publicly verifiable, since only a party that knows a symmetric key can verify a MAC. However, a MAC can be implemented using an efficient hash function (e.g., HMAC [1]) or a block cipher (e.g., CBC-MAC [2]), which makes them appealing in computation- and/or bandwidth-limited settings, such as wireless/mobile networks. Lack of public verifiability makes MAC-authenticated content objects unsuitable for generic content distribution over wide-area networks. However, our results show that MACs provide significant performance advantages for applications scoped in local-area networks, where public verifiability may not be relevant, (i.e., there is no NDN routers between communication end-points), in closed environments (i.e., where all parties can share authentication keys) or in low-latency applications such as SSH and VoIP, where there is little or no benefit to caching.

### 3.4.2 (Distributed) Denial of Service Exploration

During the second year we analyzed the impact of current DoS attacks on NDN, identifying new attacks that rely on NDN features, and proposing countermeasures. First, we showed that DoS attacks effective in the current IP-based Internet are largely ineffective against NDN. Second, we identified and described two new types of NDN-specific DoS attacks, based on: (1) **interest flooding**, and (2) **content/cache poisoning**. Third, we analyzed the impact of several flavors of both attack types and proposed a set of potential counter-measures. Interest Flooding and Content/Cache Poisoning capitalize on two features that distinguish NDN routers from IP routers: PITs and caches. An adversary can take advantage of PIT state to mount a DoS attack, which we termed "interest flooding", aiming to (a) overwhelm (PIT-s) in routers so they cannot process legitimate interests, and/or (b) swamp the targeted content producer(s). Since NDN interests lack source address and are not secured (e.g., not signed) by design, it is difficult to determine the attack originator(s) and target countermeasures.

We considered several countermeasures. NDN routers can easily track unsatisfied (expired) Interests and use this information to limit the numbers of pending interests per outgoing interface, per incoming interface, and per namespace. NDN creates flow balance between Interests and content, i.e., for each Interest sent upstream, at most one Data packet satisfying that Interest can flow downstream. So routers can calculate and never send more Interests than an interface can satisfy based on average content size, Interest timeouts, and bandwidth-delay product of the attached link. Flow balance also implies that a router can detect when a downstream router is sending too many interests that cannot be all satisfied due to the physical limitations of the downstream link. Even better, when a certain prefix under DoS attack, routers on the way (especially

those closer to the data producer) can detect and throttle a spike in the number of unsatisfied Interests in their PITs for that prefix.

We also considered a multi-router-based countermeasure to interest flooding – a push-back mechanism that allows routers to isolate attack source(s). When a router suspects an on-going attack for a particular namespace (e.g., when it reaches its PIT *quota* for that namespace on a given interface), it throttles any new Interests for that namespace and reports them to routers connected on that interface. These routers can propagate this information upstream towards offending interfaces, while limiting the rate of forwarded interests for the namespace under attack. The goal is to push an attack back to its source(s), or at least to where it is detectable. This countermeasure requires no modifications to the current NDN architecture.

In parallel, we considered DoS attacks that target content, i.e., the adversary's goal is to cause routers to forward and cache corrupted or fake data packets, preventing consumers from retrieving legitimate content. We consider a data packet corrupted if its signature is invalid, and *fake* if it has a valid signature but generated with a wrong (private) key.

### 3.4.3 Securing Instrumented Environments

In designing an NDN lighting control framework (see Section 3.1), our goals were (1) low latency between controllers and fixtures; (2) Use NDN naming to address all components of the system, with names related to their identity or function rather than a combination of addressing that spans layers and systems (e.g., VLAN tag, IP address of gateway, port of protocol, address of fixture) as in current implementations; (3) Control access to fixtures via authorization policies, coupled with strong authentication; (4) Use NDN naming itself to reflect access restrictions; and (5) Develop security mechanisms suitable for low-power systems.

### 3.4.4 Efficient Authenticated Publicly Verifiable Acknowledgments in NDN

A natural and efficient alternative to public-key signatures are symmetric MACs, but they do not work if devices communicate through a public network. Since MACs are not publicly verifiable, an intervening NDN router has no means of authenticating MAC-d content. While this limitation is may be reasonable for some applications such as VoIP or SSH, where both parties exchange data, in many circumstances where only one party is transmitting information (e.g., an application sending commands to a fixture) there exist efficient and publicly verifiable alternatives. We designed two techniques that allow public verifiability of acks without requiring public-key operations by communicating parties or NDN routers. Both techniques require the two parties to share a secret key.

The first technique is uses encrypted challenges. Each Interest contains, encoded as one component, an encrypted value $v$ (under the shared key) and its hash. In order to acknowledge such an Interest, the receiving party constructs a content object – without signature – containing $v$. Routers can verify efficiently whether the hash of $v$ corresponds to the value in the acknowledged interest.

The second technique is based on hash chains. The receiving party creates a hash chain of length $n$ and sends the last element $x_n$ to the sending party. The sending party includes the current element $x_i$ as a component of an interest; the receiver acknowledges the interest constructing an unsigned content object containing the hash pre-image of $x_i$.

### 3.4.5 Securing Audio Conference Tools over NDN

During our second year we worked with the application group to develop and implement a security framework for the NDN Audio Conference Tool (ACT). An ACT audio conference is created by an "organizer", who can add or remove participants from the conference, create and enforce control policies and alter the conference metadata (e.g., conference description). Access control is enforced through encryption, while data authentication is obtained through signed content objects, standard in NDN.

Our design uses only simple cryptographic tools, but represents a fundamental departure from conventional approaches that rely on centralized controllers and session-based security. Through directly securing data rather than its containers as well as a strong separation between encryption/authentication and trust

management, our design secures communication in a distributed way and enables each conference group to devise and enforce their own security policies. Moreover, conference organizers can create secret participant lists, where participants do not learn each other's identity. The developed security framework also allows efficient user revocation.

### 3.4.6 Trust Model and Key Management

NDN does not mandate the use of any particular trust model: each application is free to adopt its own. One trust model we explored this year was based on creating rigid relations between keys and the namespaces for which they are valid for publishing content, implemented via public-key cryptography. Zero or more public keys are associated with each namespace. A content object published under namespace *name* must be signed using the key associated to *name* or any of its ancestors. The root of trust in this model is multiple trusted third parties (TTP) – e.g., a CA – where the corresponding public keys for those TTPs are securely distributed to clients via out-of-band channels. For this trust model, hierarchical identity-based signature (HIBS) [4] schemes represent a viable alternative to the currently used RSA signatures in NDN. A HIBS scheme is a signature scheme where any string can be a public (i.e., verification) key. Given a signing key $sk$ corresponding to a string $s$, $sk$ is also a valid signing key for any string $s||t$ where "||" denotes string concatenation. Moreover, $sk$ can be used to compute a new signing key $sk'$ corresponding to $s||u$ for any string $u$. With HIBS, the public key corresponding to a namespace *name* can be the string representing *name* itself, while TTPs act as key generation centers in issuing signing keys to producers. Currently, the NDN codebase does not support HIBS signatures and the prototype we implemented for lighting control relies on RSA signatures. However, we are currently investigating adding HIBS support especially if above described trust model is well received by the NDN community.

Another model we began exploring this year is publishing keys in a common namespace that is efficiently distributed and cached for fast and easy access to verification keys. We have utilized the repo and Sync tools provided in the CCNx/NDN distribution, where the system publishes NDN testbed keys under a strict trust chain starting from a single root. Each NDN testbed site has a site key, signed by the root key, and user keys are signed by corresponding site keys. All keys, which are published under a common namespace $NDN/keys$ are stored on the repo of each site and continuously synced using the sync tool. The verification of keys follow the trust chains until a trusted anchor (either the root key or any cached key) is reached. We built a simple key publication and synchronization application for operators to publish user's keys in NDN testbed, and also a key verification library (in C/C++) for application developers to take advantage of NDN's built-in security. This model is currently in the process of being deployed on the testbed and we will initially use our audio conference tool application to test it.

## References


[1] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *CRYPTO 96*, pages 1–15, 1996.

[2] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.

[3] Steven Dibenedetto, Paolo Gasti, Gene Tsudik, and Ersin Uzun. ANDaNA - Anonymous Named Data Networking Application. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, 2012.

[4] C. Gentry and A. Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.

[5] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *Int. J. Inf. Sec.*, 1(1):36–63, 2001.

[6] NIST. FIPS 186-3, June 2009.

[7] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[8] Zhenkai Zhu, Jeffery Burke, Lixia Zhang, Paolo Gasti, Yanbin Lu, and Van Jacobson. A new approach to securing audio conference tools. In *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC '11, 2011.

## Publications

1. Steven Di Benedetto, Paolo Gasti, Gene Tsudik and Ersin Uzun; "ANDaNA - Anonymous Named Data Networking Application", 2012 Symposium on Network and Distributed Systems Security (NDSS'12) [3].

2. Zhenkai Zhu, Paolo Gasti Yanbin Lu, Jeff Burke, Van Jacobson and Lixia Zhang, "A New Approach to Securing Audio Conference Tools", 2011 Proceedings of the 7th Asian Internet Engineering Conference (AINTEC '11) 0[8].

3. Paolo Gasti, Gene Tsudik, Ersin Uzun and Lixia Zhang, "DoS & DDoS in Named-Data Networking", in submission.

4. Jeff Burke, Paolo Gasti Naveen Nathan and Gene Tsudik, "Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control via Named-Data Networking", in submission.

5. C. Bian, Zhenkai Zhu, Ersin Uzun and Lixia Zhang, "Deploying Key Management System in the NDN Test Bed", Technical Report – http://irl.cs.ucla.edu/ zhenkai/key.pdf

## Presentations

1. Paolo Gasti, "Security and Privacy Challenges in NDN", invited talk, INRIA Sophia Antipolis, France, 2011.

2. Paolo Gasti, "Security and Privacy Challenges in Future Internet Architectures – The Case of Named Data Networking", invited talk, Northeastern University, Boston, MA, 2012.

3. Paolo Gasti, "Security and Privacy in Emerging Applications and Network Architectures", invited talk, RSA Labs, Boston, MA, 2012.

4. Gene Tsudik, "Security and Privacy in Named-Data Networking", keynote talk, IEEE International Conference on Computer Communication Networks (ICCCN'12), Munich, Germany, 2012.

## 3.5 Network Management, Monitoring, and Simulation

| **Contributors** | |
|---|---|
| **PIs** . . . . . | Dan Massey & Christos Papadopoulos (CSU), kc claffy (UCSD), Lixia Zhang (UCLA) |
| **Grad Students** . . | Steve DiBenedetto & Kaustubh Gadkari & Susmit Shannigrahi & Chengyu Fan (CSU), Alexander Afanasyev & Ilya Moiseenko (UCLA) |

The network management, measurement, and evaluation work includes fundamental research related to network policy management, as well as near term tool building and deployment, including tools for troubleshooting, testbed configuration, traffic generation, and a simulator. Table 3.2 summarizes the policy knobs available to NDN network operators. In addition to routing plane knobs, NDN offers data plane knobs that operate on new or modified NDN components (PIT, FIB, CS). We are experimenting with applying these knobs to solve both policy and security issues. We introduced the concept of specialized *Auditor Nodes*, which act as safe harbors for content and can be exploited to detect and overcome content poisoning attacks.

### 3.5.1 Network Monitoring Tools

Users need to be able to trace the location of problems when an application fails (e.g. they need traceroute for NDN). Traceroute in the context of CCN is not so well-defined as in IP, where traceroute traces the path between two locations (IP addresses). However, there is no explicit location for the data. The desired data may come from a node caching a copy of the content or one of the data's multiple publishers. Therefore, doing traceroute in a CCN network can take different meanings depending upon the context: finding a path (or all paths) to the nearest publisher of the content, or finding all paths to the nearest copy (or all copies) copies of the content. This year we developed a prototype NDN traceroute tool; testing is ongoing. Our current implementation enumerates all paths to all reachable publishers of the specified content, essentially a a topology map for a certain piece of content.

| Component | Name | Controls | Setting | Description |
|---|---|---|---|---|
| Routing Plane | Routing/RIB | Routes | Not Selected | Route cannot be used to forward Interest packets. |
| | | | Selected | Route may be used for forwarding Interest packets. |
| | | | Select and Announce | Route may be used for forwarding Interest packets and may be announced to peer groups. |
| Content Store | Cache Access | Interests | Allow | Check Interest against cache and proceed to PIT if no matches are found. |
| | | | Cache Only | Check Interest against cache and drop if no matches are found. |
| | | | Deny | Drop Interest without checking cache. |
| FIB | FIB Usage | | Full Usage | New matching faces may be found and added to the existing face list. |
| | | | Limited Usage | No new faces may be added to the existing face list. |

Table 3.2: Examples of NDN policy knobs (not exhaustive).

Our traceroute application operates at the application layer, and does not have direct access to the underlying router data structures and values, thus does not require any modifications to NDN routers. NDN traceroute issues Interest packets specifying the desired content, similar to normal Interests but with a /trace at the beginning of the name, and a random number at the end of the name to allow identification of duplicate Interests. For example, an Interest to trace the route to "/csu/index.html" might look like "/trace/csu/index.html/1234556" To report the route back, we use a unique identifier (IP address or name)

for each router. Upon start up, a traceroute server application registers an Interest for namespace "/trace", NDN routers forward all "trace" packets to their trace server application, which does the normal FIB check for entries matching the content name being traced, and responds with a message saying "local content" if the content is local. If there a remote entry in the FIB, the server gets the list of outgoing faces, adds new entries for the trace Interest names, and forwards the trace Interest. It then parses and returns any corresponding replies, adding its own (router) identifier to the path, and indicating any timeouts that occurred. Finally, the server deletes entries for the trace Interest.

### 3.5.2 Auto-Configuration

We built NDN Auto-configuration (NAC) to help edge nodes (e.g., laptops or other hosts) automatically self-configure to join the NDN testbed network automatically (e.g., DHCP for NDN). We utilized the DHCP protocol to distribute NDN gateway addresses and user namespace information (so users can publish data). Users store this information in a file nac.conf, and then can easily connect to NDN network. Designing this protocol over the popular DHCP architecture simplifies the process of building NDN overlay networks, since entering nodes must get an IP address first.

### 3.5.3 Simulation Tool Development

We developed an open-source scalable NDN simulator, ndnSIM, based on NS-3 network simulator framework, with the following goals: faithfully simulate all basic NDN protocol operations with at least 100 nodes; maintain packet-level interoperability with CCNx implementations, to allow sharing of data and tools between CCNx and ndnSIM; and facilitate experimentations with data caching, packet forwarding, routing, and congestion management. Although NS-3 is relatively new and still does not have everything that the commercial Qualnet or the legacy ns-2 simulator have (e.g., support for simulating IP routing protocols), it offers a consistent design, documentation, and implementation flexibility. ndnSIM is implemented as a new network-layer protocol model, which can run on top of any available link-layer protocol model (point-to-point, CSMA, wireless, etc.), as well as on top of other network-layer (IPv4, IPv6) and transport-layer (TCP, UDP) protocols. This flexibility allows ndnSIM to simulate homogeneous and heterogeneous networks (e.g., NDN-only, NDN-over-IP, etc.). ndnSIM also includes basic traffic generation and modules to simplify creation of simulation scenarios and tools to gather simulation statistics for measurement purposes.

The wire format of Interest and Data packets follows the format of the existing CCNx Project's NDN implementation, allowing reuse of existing tools and traces. The design of ndnSIM includes a number of optional modules, including (1) a placeholder for data security (the current code attaches a signature to data packet but does not compute it, and does not perform signature verification at routers), (2) experimental support of negative acknowledgments (NACK Interests) to provide fast feedback about data plane problem, and (3) a pluggable Interest rate limit and interface availability component. We implemented the following component-level abstractions of ndnSIM:

- **Core NDN protocol implementation:** L3Protocol in ndnSIM is a central architectural entity and stands in the same line of class hierarchy as the corresponding Ipv4L3Protocol and Ipv6L3Protocol classes of the NS-3 framework that implement IPv4 and IPv6. This component performs all incoming Interests and Data packets processing, including adding and removing faces, performing lookups to ContentStore, PIT, FIB, forwarding Data packets according PIT state, and applying forwarding strategy decisions to handle Interests. The L3Protocol class also provides access to network-layer statistics ("tracing") about Interest packets (InInterests, DropInterests), Data packets (OutData, InData, DropData), and optional Interest NACKs (OutNacks, InNacks, DropNacks), and access to configure optional components of the implementation (NACKs and caching unrequested data).
- **Face abstraction**, which enables uniform and flexible communication with applications (AppFace) and other simulated nodes (NetDeviceFace) with pluggable (and optional) support of link-level congestion mitigation modules. We also implemented a module to prevent local-link congestion by limiting the Interest rate based on a leaky bucket method, assuming that the Data packet size is known a priori or estimated using a moving average of previously seen Data sizes.

- **ContentStore**: an in-network storage for Data packets with modular replacement policy, providing efficient error recovery and delayed multicast support. The current version of ndnSIM contains a Least-Recently-Used (LRU) replacement policy module.
- **Pending Interests Table (PIT)**, which maintains state for each forwarded Interest packet in order to provide directions for Data packet forwarding. Each PIT entry contains the content name, a list of incoming Faces (from which Interest packets for that name have come); a list of outgoing Faces (to which Interests have been forwarded); and the time when the entry should expire (the maximum lifetime among all the received Interests for the same name). The PIT is structured as an unbounded dynamic container with multiple indices. The first index (by name hash) provides support for quick PIT entry lookup using hashes of names from Interest or Data packets. The second index (by expiration time) allows efficient removal of timed out Interests. When a non-unique Interest is received, the list of incoming Faces for the existing PIT entry is updated accordingly, effectively aggregating (suppressing) similar Interests.
- **Forwarding Information Base (FIB)**, which holds information required for Interest forwarding decisions. The FIB is structured as a hash-indexed container, where each entry contains a prefix and an ordered list of Faces through which the prefix is reachable. In addition to the longest-prefix match, ndnSIM implements several Interest selectors, including two types of exclude filters and min/max name components filter. One can manually configure every node's FIB, use use a central global NDN routing controller to automatically populate all routers' FIBs. The global routing controller has information about all existing NDN nodes and all exported prefixes, so can calculate optimal paths between every node pair and updates all the FIBs. Currently, this controller uses Dijkstra's shortest path algorithm (using the Face metric), installing only a single outgoing interface for each name prefix.
- **pluggable Internet forwarding strategies**: Our design supports experimentation with forwarding strategies without modifying the core components. We implemented two forwarding strategies: a flooding strategy, which forwards Interest packets to all GREEN and YELLOW Faces in the corresponding FIB entry (except the incoming face), and the best-route strategy, which forwards Interest packets to the highest-ranked GREEN (if available) or YELLOW face (except the incoming face). An optional Interest limiting feature can limit the number of Interests transmitted on a given face, to avoid congestion or maximize utilization.
- **reference NDN applications**: ndnSIM provides a base App class that creates AppFace and registers it inside the NDN protocol stack, as well as default processing for incoming Interest and Data packets. Reference traffic-generating applications currently available in ndnSIM can generate Interests with predefined interarrival distributions, generate a specified number of Interests (Batches) at specified points of simulation, generate variable rate Interest traffic using the NACK-Internet feature and a sliding window; and a "Producer", or Interest-sink application, which replies every incoming Interest with Data packet with a specified size and name same as in Interest.

## Publications

1. Steve Dibenedetto, Christos Papadopoulos, Daniel Massey, "Routing Policies in Named Data Networking". SIGCOMM 2011 Workshop on Information Centric Networking, August 2011

## 3.6 Fundamental theory for NDN

| Contributors | |
|---|---|
| **PIs** | PIs: Edmund Yeh (Northeastern) |
| **Grad Students** | Grad Students: Ying Cui (Hong Kong University of Science and Technology) |
| **Undergrads** | Undergrad Students: Kyle Dumont, Patrick Cunniff (Northeastern) |

A fundamental theory for NDN networks can potentially demonstrate that the NDN architecture can quantitatively outperform the IP architecture with respect to appropriate performance metrics (discussed below), when given the same set of network resources. But classical information theory is connection-based, limiting its utility even for IP, much less for a name-based architecture such as NDN. The multi-dimensional generalization of classical point-to-point Shannon capacity is the *network capacity region*, consisting of all the source-destination communication rates achievable using any feasible coding scheme [1]. The corresponding *coding* problem is to find low-complexity error correcting codes which allow for reliable communication at the rates given by the capacity region. To realize the true potential of the NDN paradigm, we need a new fundamental theory of networked communication. This poses some key challenges. First, the classical information theory assumption that data streams sent over different source-destination pairs are independent will not work in NDN. In NDN, data traffic responding to Interests is likely to be statistically dependent, requiring new optimal coding theory and practice for multiple-source multicast. The critical role of caching data in NDN also raises the importance of optimal asynchronous or delay-tolerant multicast.

A theory for NDN must capture the essential tradeoff between wires and storage in optimizing communication performance, so that for a given set of link capacities and buffer sizes, we can analyze combinations of transmission and caching strategy to optimize network performance. Two additional issues not prominent in classical theory are the *time value of information* (information cached too long may be useless once reaching receivers) and the central role of mobility in assessing the tradeoff between storage and transmission [2].

We first developed appropriate network performance measures within this new framework. Instead of considering source-destination transmission rates, we focused instead on measuring the total consumed entropy rate in bits per second, *i.e.*, the total rate of statistically independent information arriving at nodes requesting content. This definition accounts for transmission of information over time and appropriately corrects for redundant content, but does not capture the impact of storage and mobility on transmission of information across space. To further account for the importance of spatial transmission, we will measure NDN capacity in bit-meters (one bit of information transported 1 meter toward a node requesting content) per second [3]. We can then pose questions similar to those in classical information theory: for a given set of link capacities, buffer sizes, interest and data generation rates, what are the achievable rate regions and complexity costs for feasible joint routing, forwarding, scheduling, coding, and buffer management schemes? Then we will incorporate issues beyond network capacity, such as latency and fairness.

### 3.6.1 Optimal Forwarding and Caching Algorithms

We developed a distributed dynamic forwarding and caching algorithm for maximizing the region of Interest packet arrival rates which the network can satisfy (provide a data packet for). This algorithm makes decisions regarding forwarding, caching, and cache replacement by examining the number of Interest packets for each content object at each node. The algorithm allocates the link capacities to Interest packets and cache space to Data packets in such a way that the load of Interest packets for the content objects is maximally balanced in the network. That is, the algorithm does not typically cache retrieved data packets everywhere along the return path, but only when it is conducive to maximizing the network performance metric of total consumed bit rate. This approach minimizes the probability that any part of the network becomes unstable (i.e. the number of interest packets becomes excessively large), thereby maximizing the region of interest packet arrival rates which the network can satisfy. The implementation of the algorithm is distributed in the sense that a node needs to exchange its state only with its neighbors.

Our dynamic forwarding and caching algorithm provides a systematic way to simultaneously design the

NDN strategy layer, based on state provided by the number of Interest packets for different content objects at each node. We simulated the algorithm on the following network topologies: (1) the Abilene network, (2) the NDN testbed network, (3) access networks, and (4) random spatial networks. Initial results have been encouraging. In particular, the forwarding and caching decisions produced by the algorithm accord with engineering heuristics regarding optimal forwarding and caching decisions.

## References


[1] Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley-Interscience, New York, NY, USA, 1991.

[2] Matthias Grossglauser and David N. C. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Trans. Netw.*, 10(4):477–486, 2002.

[3] Piyush Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.


## Publications

1. Edmund M. Yeh and Tracey Ho, "Distributed Dynamic Forwarding and Caching in NDN Networks." In preparation, 2012.

# Chapter 4

# Values in Design

| Contributors | |
|---|---|
| **PIs** ..... | Katie Shilton (UMD),Paul Ohm (U of Colorado) |
| **Undergrads**..... | James Neal (UMD),TBD (U of Colorado) |

The NDN project was extended this past year to include two new activities: Values in the Design of the Named Data Networking Architecture (VD-NDN) and Values from Law and Policy in the Named Data Working Architecture (VLP-NDN). These efforts are aimed at understanding the social values, policy and law implications of the design of NDN architecture as well as the design process. The research objectives for VD-NDN are: 1) understanding social challenges and values engendered by the proposed architecture, 2) investigating design activities that encourage discussion of values during design, 3) operationalizing values such as privacy, anonymity, security and information equity as design principles and features, and 4) understanding NDN's impacts on information policy. VLP-NDN focuses on values derived from law and policy. The objectives are to: identify how today's laws would respond to the NDN design; investigate how law expresses important values that designers should consider; prepare researchers for the inevitable conflicts likely to arise in a future NDN Internet; and understand the political economy of how engineering designs can anticipate, and surmount, political roadblocks on the path to implementation. The approach is to first develop a conceptual non-technical map of the NDN architecture, and undertake a legal issue-spotting exercise, listing laws or public policy considerations that may be consistent or at odds with the NDN architecture. We will then conduct what-if projections, anticipating the reaction of law, policy, and politics to a hypothetical future NDN-based Internet. We will also try to bridge the values of network design and values of external legal and policy-focused communities.

## 4.1 Social relationships, challenges and values (VD-NDN

We mapped social relationships between the campuses, PIs, and work practices of the NDN team, as preparation for our interviews. PI Shilton attended one all-hands meeting with the NDN PIs, a second all-PI meeting for the FIA program, and participated in numerous conference calls to take notes and observe. As a result of this activity, Shilton and graduate assistant Neal developed a taxonomy of values concerns in the NDN project. Based on qualitative coding of NDN publications as well as field notes, the taxonomy highlights project concern for dynamism, efficiency, privacy, security and trust. We are currently building on this taxonomy to understand social challenges and values in the NDN project.

The PI and graduate assistant have also developed a set of questions that will shape interviews and publications with the NDN team. The questions, focused on the domains of infrastructure development, security, applications, law enforcement and governance, and political economy, interrogate social consequences of NDN architectural decisions. The application team is developing a file-sharing program based on NDN, and we are experimenting with bootstrapping trust relationships for that application using existing relation-

ships between researchers. We are modeling these relationships with data about relationships and groups harvested from professional social networking site LinkedIn.

## 4.2 Legal and policy issue mapping (VLP-NDN

During our first (partial) year, we began developing the conceptual map of the NDN architecture, and a preliminary mapping of legal issues to include in the issue spotting exercise. In addition, we have organized a conference together with the Silicon Flatirons Center for Law, Technology, and Entrepreneurship (which the PI co-directs), currently scheduled for Friday, January 11, 2013, on the topic of the "Technology of Privacy", at the University of Colorado Law School, in Boulder, Colorado. One key topic for this conference will be values in design, and invitations have been delivered to leading figures in this field, including several associated with the FIA project. The PI has struggled to find a law fellow to execute the bulk of the proposed work, partly since most law schools (including the PI's home institution) do not have graduate students. We hope to hire a law fellow by summer 2012.

# Publications

1. Theorizing Future Internet Architectures: Values in the Design of Named Data Networking, at the Theorizing the Web Conference, College Park, MD, April 2012.
2. Organizing panel titled "Technology as Policy" which will focus on VID as a major theme at the Association of American Law Schools' Mid-Year meeting, taking place in mid-June 2012, in Berkeley, California.

In addition, we have organized one panel, entitled "Technology as Policy" which will focus on VID as a major theme at the Association of American Law Schools' Mid-Year meeting, taking place in mid-June 2012, in Berkeley, California.

# Chapter 5

# Education

| Contributors | |
|---|---|
| **PIs** | Dan Massey & Christos Papadopoulos (CSU), Van Jacobson, Jim Thornton, Sharon Johnson (PARC), Beichuan Zhang (Arizona), Lixia Zhang (UCLA) |

The NDN architecture development gives us a unique opportunity to teach students how to think architecturally. Today instructors may strive to teach architectural concepts, but unfortunately students often focus on mechanics, e.g., *what* fields are in an IP header, rather than *why* those fields are in the IP header. We use the comparison between the IP architecture and the NDN architecture as a specific case study to show students architectural evolutions and to understand the big picture, as well as how individual components fit in. This theoretical view is complemented by a more tangible projects where students use the actual NDN prototype to assess the systemic impacts of design choices. Students are exposed to NDN architectural concepts in the context of real software development challenges, and use APIs to the core NDN layer to build software that explores these concepts. Implementing projects under both IP and NDN paradigms inform theoretical and practical discussion of networking concepts. Running the NDN variants on our testbeds lead to unexpected network behavior that challenges the testbed student operators. Finally, we record exchanges and discussions from the NDN design and development efforts, capturing the *process* of creating a new architecture to enable such research and teaching in the future.

## 5.1 Integrating NDN research into education

Our achievements include running weekly NDN reading groups (Colorado State, UIUC, UCLA, Yale), adding introductory lectures in undergraduate courses (Arizona, Memphis), graduate seminar courses dedicated to NDN architecture research and development (Colorado State, UCLA), in addition to having both graduate and undergraduate students participating in NDN research efforts. Various PIs have also given keynote talks, panel presentations, GENI demos, and other lectures introducing NDN to a wide variety of audiences. Slides are available online from the NDN website and continue to improve with each presentation. We continue to work on a set of slides on "Thinking Architecturally" for inclusion in graduate and advanced undergrad courses, presenting our approach in 2011 at the SIGCOMM education workshop.

**University of Arizona** Beichuan Zhang included NDN material in 3 courses: CS 296H/496H, a research seminar for undergraduate students; introductory graduate level, a fall 2011 graduate course (CS 525 - Principles of Computer Networking), two lectures; and graduate seminar CS 630 - Advanced Topics: Information-Centric Networking. This last course required course projects where students were asked to: (1) enable the use of legacy applications (SSH, SCP) by running IP on top of NDN; (2) design and implement a link protocol (which handles fragmentation/reassembly and loss detection/recovery) between the NDN and MAC layers; and simulate different NDN forwarding strategies and compare performance in terms of the delay in content retrieval and the number of interests sent.

**Colorado State** Dan Massey included an NDN lecture in his undergraduate computer network course, CS 457. Christos Papadopoulos taught NDN in his first year graduate networking course CS557 (4 of 30 lectures on NDN, 1 of 3 class projects on NDN), and an advanced graduate course CS 657 a seminar devoted primarily to NDN. Class projects included analyzing campus traffic to understand the impact of NDN caching on both network load and users performance experience; developing an NDN traffic generator; and performance evaluation of fetching files over the NDN network (using a testbed overlay on PlanetLab). Students produced posters summarizing their results and presented the results at the January 2012 NDN project retreat hosted by Colorado State.

**UCLA** Lixia Zhang taught graduate seminar course "CS217B: Advanced Topics in Internet Research" which has been devoted to teaching the NDN architecture for the last 3 years. This course also provided an in depth study of the other three FIA project designs, as well as two European Union funded projects SAIL and PURSUIT, to gain a deep understanding of global future internet architecture research efforts. The students compared designs, analyzing assumptions and tradeoffs in each design. Class projects included: developing solutions to support mobile producers; developing solutions to derive trust management from social networking relationships; developing an 'NDN-dropbox' service using NDN's basic features of naming, securing data directly, and distributed synchronization ("SYNC"); supporting joint file (photo) editing over NDN; and a light-weight javascript based NDN node implementation to run inside browsers, making it easier for other applications to utilize NDN transport.

## 5.2 NDN Camps and Tutorials

We organized two multi-day "boot camps": at PARC on September 12-14, 2011, and at by AsiaFI at Seoul National University, Korea, on March 19-21, 2012. These events taught the basic concepts in the NDN architecture and its companion CCNx software prototype through tutorial lectures, discussions, and realtime program development. The first boot camp was adjacent to PARC's CCN Community Meeting and was attended by participants from th e US, Europe, and Asia regions; the second event was largely attended by students and researchers from Asia Pacific region. We developed and presented in-depth tutorial material combined with hands-on activity, where attendees learned the basic principles and concepts, write code, examine packets, and solve problems. Building on this model, the NDN team collaborated with AsiaFI to organize the NDN Hands-On workshop in Korea. Over 100 students and researchers attended the workshop and learned NDN concepts and development skills to utilize the CCNx prototype in their research.

# Publications


1. "Teaching Network Architecture through Case Studies", Dan Massey, Christos Papadopoulos, Lan Wang, Beichuan Zhang, Lixia Zhang. SIGCOMM 2011 Education workshop, August 2011.
2. NDN Project Education Webpage, `http://named-data.net/education.html`
3. CCNx BootCamp Webpage, `http://www.ccnx.org/ccnxbootcamp2011/`
4. AsiaFI NDN Hands-on Workshop Webpage, `http://www.asiafi.net/org/ndn/hands-on2012/`

# Chapter 6

# Global Expansion of NDN Efforts

The NDN project continues to attract attention from the global networking community, and research efforts closely aligned with NDN are underway in a number of institutions around the world.

PARC hosted the first CCNx Community Meeting and Bootcamp in September 2011 (see `http://www.ccnx.org/CCNxCon2011Program`). Approximately 120 people from eight countries attended. It was a very successful meeting, and the attendees enthusiastically endorsed the idea of holding another, expanded meeting this year. The second CCNx Community Meeting will take place September 12-13, 2012 at INRIA's Sophia Antipolis Méditerranée Research Center, France (see `http://www.ccnx.org/ccnxcon2012/`).

NDN continues to draw a strong interest from Asia Future Internet Forum (AsiaFI, `http://www.asiafi.net/`). AsiaFI organized "Asia Workshop on Future Internet Technologies (AWFIT2011)" in Bangkok, Thailand in November 2011 (`http://www.interlab.ait.ac.th/aintec2011/asiafiws.php`), and invited Lixia Zhang to deliver a keynote address on "Evolving Internet to the Future via Named Data Networking". In addition to sending delegations to the CCNx Community Meeting and Bootcamp at PARC in September 2011, AsiaFI also hosted its own NDN Hands-on Workshop on March 19-21, 2012, at Seoul National University, Korea see `http://www.asiafi.net/org/ndn/hands-on2012/`).

A primary goal of this workshop was to promote and facilitate the ongoing research efforts in the NDN direction in Asia Pacific region. Jim Thornton and Van Jacobson gave a 2-day CCNx tutorial to approximately 100 attendees, most of them were students and faculty members the region. The tutorial combined the basic design of the NDN architecture with its prototype implementation of CCNx code base, enabling the attendees to carry out NDN related research with experimental designs and implementations. The program also included a one-day NDN Research Workshop where Jim Thornton presented an NDN project progress report, including the demo given at the GENI Engineering Conference (GEC) in March 2012, and Lixia Zhang presented a talk on exploring new application design directions with NDN. Van, Jim and Lixia also participated in a panel discussion regarding untapped research areas in Information-Centric Networking. This event was a great success and it has also created an initial collection of NDN educational material, including a set of basic introductory homework exercises and a set of simple sample programs.

The CONNECT project (`http://anr-connect.org/`), sponsored by The French National Research Agency, ANR, is a large research initiative with four university and two industry partners. The research of the CONNECT project nicely complements several aspects of the NDN project and the two projects have engaged in collaborative activities during the past year. Lixia Zhang (NDN) and Giovanna Carofiglio (CONNECT) co-chaired the NOMEN Workshop on Emerging Design Choices in Name-Oriented Networking held in conjunction with the IEEE INFOCOM held March 30, 2012.

PARC has launched an Emerging Networks Consortium (ENC) to foster innovation and collaboration around CCNx. The inaugural Summit meeting of the founding ENC members was held April 23-25 at PARC (`http://www.parc.com/event/1718/emerging-networks-consortium-at-parc----spring-summit-2012.html`). Patrick Crowley and Lixia Zhang from the NDN team also participated in this meeting. Patrick gave a talk on the NDN project, including a portion of the demo that was presented at GEC, and Lixia moderated a panel discussion on Industry Perspectives on Standardization.