On the Evolution of ndnSIM: an Open-Source Simulator for NDN Experimentation

Spyridon Mastorakis
UCLA
mastorakis@cs.ucla.edu

Alexander Afanasyev
UCLA
aa@cs.ucla.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

As a proposed Internet architecture, Named Data Networking (NDN) takes a fundamental departure from today's TCP/IP architecture, thus requiring extensive experimentation and evaluation. To facilitate such experimentation, we have developed ndnSIM, an open-source NDN simulator based on the NS-3 simulation framework. Since its first release in 2012, ndnSIM has gone through five years of active development and integration with the NDN prototype implementations, and has become a popular platform used by hundreds of researchers around the world. This paper presents an overview of the ndnSIM design, the ndnSIM development process, the design tradeoffs, and the reasons behind the design decisions. We also share with the community a number of lessons we have learned in the process.

CCS Concepts

•**Networks** → **Network simulations;**

Keywords

Information-Centric Networking, Named Data Networking, ndnSIM, Evaluation, Simulation, NS-3

1. INTRODUCTION

Named Data Networking (NDN) [54] is a proposed Internet architecture. Instead of delivering packets to given IP addresses as TCP/IP does, NDN retrieves desired content items by names. NDN names can name anything: a text message, a data block generated from a video conference call, a command to a light bulb, or a communication endpoint. Such a fundamental change of the communication model requires extensive evaluation and experimentation. To create a common evaluation platform for the research community to experiment with the latest advancements of NDN research at scale, we have developed ndnSIM [10, 17, 36]; an open-source, modular NDN simulation package based on the NS-3 framework [12]. The ndnSIM development effort started in 2011 and its first beta version was released in February 2012. Since then, it has undergone substantial design changes and extensive development, and has been used by an increasing number of researchers from the broader networking community.

Over the years, ndnSIM has served as an enabler for a wide scope of experimentation with NDN architecture. The latest release of ndnSIM integrates the NDN Forwarding Daemon (NFD) [18] and its supporting library (ndn-cxx) [38], providing a level of interoperability between simulation and prototyping, further increasing the value of ndnSIM experimentation in understanding the behavior of NDN forwarding [53, 22, 40] and in-network caching [29, 50]. ndnSIM has also facilitated the development of NDN applications [37, 27, 55, 31], the exploration of applying NDN to different network environments (e.g., vehicular [33, 34], ad hoc wireless [21], mobile [44], and IoT [43, 20]), the designs of congestion control [42], the evaluation of link layer [49] and routing [47] protocols, including the NDN routing protocol NLSR [35].

To help more people gain familiarity with ndnSIM, in this paper we first present a broad overview of the ndnSIM design (Section 3). We then discuss the major design tradeoffs we encountered, the reasons behind our design decisions, and quantitatively evaluate the cost of our decisions (Section 4). In Section 5, we gauge the software development effort and the community adoption of ndnSIM, and in Section 6 we share the lessons that we have learned from developing an open-source simulator for a new networking architecture. Section 7 identifies several limitations in the current version of ndnSIM, lists challenges of the development process, and sketches our future work. Finally, Section 8 discusses related work and Section 9 concludes the paper.

2. BACKGROUND

In this section, we provide a brief background on Named Data Networking and NS-3 to prepare the reader for the ndnSIM design discussions in the rest of this paper.

2.1 Named Data Networking

The NDN project started in 2010 under the sponsorship of the NSF Future Internet Architecture (FIA) program. Since then, it has grown from the initial blueprint to operational prototype implementations running over a multi-continental testbed ns3-structure[7] and supporting a variety of applications [6].

NDN protocol stack follows the same hourglass shape as TCP/IP, but changes the "thin waist" of the network architecture from address-based packet delivery to fetching named and secured data (Figure 1). More specifically, when an application requires a piece of data, it simply creates an Interest packet with the name of the desired content and sends ("express") this Interest to the network. The forwarding daemons residing on each NDN node (Figure 2) then use these hierarchically structured and semantically meaningful names to forward Interest packets towards data producers (upstream direction) using the forwarding strategy engine. The strategy uses inputs from the Forwarding Information Base (FIB) and measurements from earlier fetching to decide whether, when, and where to forward the Interests. An NDN FIB is similar to an IP FIB, except that it contains name prefixes instead of IP address prefixes, and each name prefix may points to multiple next-hop interfaces (called faces in the NDN context), instead of a single next hop per IP destination in TCP/IP. A forwarder can also send back downstream a Negative Acknowledgment (NACK) if it cannot forward

the Interest because of the lack of information in FIB for the prefix, congestion on the path, or it encounters other errors. In order to return the requested data to the requester, NDN forwarders also maintain a state for each of the forwarded Interests in its Pending Interest Table (PIT), recording faces on which the Interests are received and aggregating Interests for the same data.

Each NDN data packet includes the name, data, and a cryptographic signature created by the original data producer that binds together the packet's name and the content. Because of this binding, consumers can ensure integrity, authenticity, and provenance of each data regardless of how or from where it was retrieved: from the original producer, the managed storage, or opportunistic caches (Content Stores, CS) that can be part of each NDN forwarder.

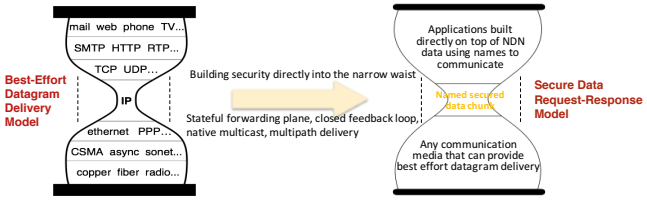

Figure 1: NDN keeps the hourglass-shaped architecture model, but enables secure data retrieval directly at its "thin-waist"

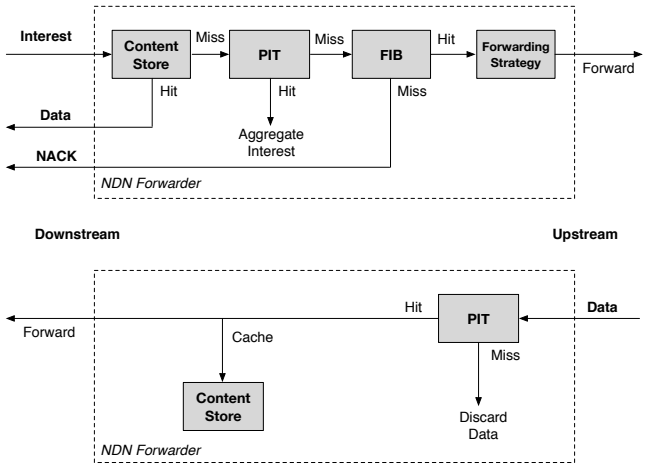

Figure 2: Overview of packet processing by an NDN forwarder

2.1.1 *NDN Forwarder Prototype*

Named Data Networking Forwarder (NFD) [18] is the reference implementation of the NDN forwarder. NFD is developed as a community effort and supports a diverse set of experimentations with the NDN technology by emphasizing modularity and extensibility (Figure 3). It includes realizations of the three basic data structure (FIB, PIT, CS) along with several cache policies and forwarding strategies (basic best-route and multicast strategies, self-learning access router strategy, and two adaptive SRTT-based strategies). The key abstraction for communicating between the forwarder, local applications, and remote forwarders—Face—has modular Face-LinkService-Transport design, separating generic high-level network-level functions (packet encoding/decoding and packet dispatch), link adaptation functions (fragmentation, optional recovery from link errors, etc.), and low-level details of sending and retrieving packets to/from specific links.

Many of the core NDN operations in NFD are implemented using the ndn-cxx library [38], providing routines for packet encoding/decoding, extensive set of security mechanisms, as well as a special application-directed Face realizations (ndn-cxx Face). Note that the latter are semi-equivalent of BSD sockets providing NDN applications a basic API to express Interests and publish data, while Face abstraction inside the forwarder (NFD Face) is similar to OS kernel's routines to send/receive packets through the network interfaces.

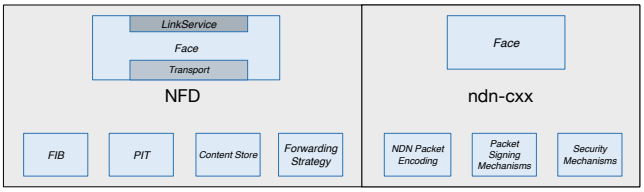

Figure 3: Design overview of NDN prototype implementations

2.2 NS-3 Framework

NS-3 is an open-source network simulation platform based on discrete event scheduling. It is written in C++ and fulfills the needs of modern networking research. With NS-3, users can create their own simulation topologies with custom node and link parameters, simulate the full TCP/IP protocol stack (physical, link, network, transport, application layer protocols), trace and collect simulation data and visualize the simulation execution. To reduce memory requirements in large-scale simulations, NS-3 packets can contain virtual payload.

Network simulations in NS-3 are based on the following key abstractions: 1) *Node*: the basic computing entity, which can be programmed by users; 2) *Application*: a user-defined program defining some functionality to be simulated; 3) *Channel*: a communication channel entity connected to a Node; 4) *Net Device*: representation of both the simulated hardware and software drivers that enable a Node to communicate through Channels with other Nodes; and 5) *Topology Helpers*: software components used to facilitate the creation, coordination and parameter configuration of the previous abstractions.

The software is organized as a number of modules, each module typically consisting of one or more models (representations of network protocols, devices, routers, etc.) and a number helpers classes. Figure 4 summarizes the modules, models, net device implementations and the overall integration and support offered by NS-3.

3. NDNSIM DESIGN OVERVIEW

In this section, we present the overall structure of ndnSIM environment (Figure 5), consisting of NS3, NFD, and ndn-cxx, as well as an NDN simulation layer, ndnSIM-specific and real-world applications ported to ndnSIM, and a number of plug-and-play simulation scenarios. We also identify a set of features that makes it a useful tool to the research community and present the design workflow by discussing the process of exchanging NDN packets between two simulated nodes.

First, we would like to explain the term "open-source simulation package": the term "open-source" refers to the fact that the ndnSIM codebase is available to the public and users are welcome to download and modify it based on their individual needs. Users

Modules	Models	Net Device Implementations	Integration/ Support
Network	Internet	Bridge	BRITE
LTE	Ad Hoc On-Demand Distance Vector (AODV)	CSMA	Click Modular Router
Antenna	Optimized Link State Routing (OLSR)	PointToPoint	OpenFlow switch
Wi-Fi	Low-Rate Wireless Personal Area Network (LR-WPAN)	File Descriptor	
Wi-Fi Mesh	6LoWPAN	Tap	
Spectrum	Propagation	Wimax	
	Mobilty		

Figure 4: Modules, models, net device implementations and integration/support offered by NS-3

are encouraged to participate to the simulator development. The term "simulation package" demonstrates that ndnSIM consists of multiple software components that have been integrated altogether to provide a concrete framework for high-fidelity NDN simulations.

3.1 ndnSIM Structure

In its core, ndnSIM is based on NS3 simulation framework and leverages it in the following ways:

- To create simulation topologies and specify topology parameters (e.g., link bandwidth, node queue size, link delays).
- To simulate available link-layer protocol models (e.g., point-to-point, wireless, CSMA).
- To simulate the exchange of NDN traffic among the simulated nodes.
- To trace simulation events and (optionally) visualize the simulation execution.

Therefore, ndnSIM simulations can use any of the existing modules, models, NetDevice implementations, and integrated components of NS3.

To realize the core NDN forwarding functions, ndnSIM integrates NFD and ndn-cxx codebases, rewiring key logic elements such as the event processing and network operations to the NS3 specific routines. The result of this integration is that the code used for experiments with NDN forwarding in ndnSIM can be directly used by the real NFD implementation and vice versa. Moreover, ndnSIM allows simulating the real-world NDN applications based on ndn-cxx library (with a few constraints described in Section 3.2.2).

On top of the NFD integration, ndnSIM includes an addition NDN simulation layer to streamline creation and execution of simulations and to obtain key metrics. ndnSIM package also offers a collection of tutorial simulation scenarios that provide examples of ndnSIM features.

The following summarizes ndnSIM components and their features:

- **Core (Integration and Models)**: the NDN protocol stack, the realization of NFD's Transport to provide communication on top of NS3 NetDevice and Channel abstractions, the realization of NFD's LinkService to facilitate direct communication between ndnSIM-specific applications and local forwarder instances, and the global routing controller to facilitate static configuration of FIB (based on the Dijkstra's shortest path algorithm).
- **Utilities:** a number of packet tracers to obtain simulation results (link-, network-, and application-level tracing) and topology readers to simplify definition of simulation topologies.
- **Helpers:** a set of helpers to install and configure NDN stack and simulated applications on nodes, to manage (statically or during simulation) FIB, forwarding strategies, and cache replacement policies, to simplify modifying states (up/down) of the links in the simulated topologies.

3.2 Applications

ndnSIM can simulate two distinct types of NDN applications: ndnSIM-specific and real-world applications.

3.2.1 ndnSIM-Specific Applications

The ndnSIM-specific applications are a convenient way to generate basic Interest/Data packet flows for various network-level evaluations, including behavior of forwarding strategies, cache policies, etc. These applications are realized based on NS3's Application abstraction and include several built-in tracing capabilities, including times to retrieve data.

The built-in ndnSIM-specific applications include:

- *ConsumerCbr*: the consumer application that generates Interest traffic with constant-frequency pattern.
- *ConsumerZipfMandelbrot*: the consumer application that generates Interests with name popularities following the Zipf-Mandelbrot distribution [16].
- *ConsumerBatches*: the consumer application that generates a specified number of Interest packets at certain points of the simulation execution. It accepts a pattern for Interest generation specifying a set of points of time in the simulation and a number of Interests to be generated at those points.
- *ConsumerWindow*: the consumer application that generates Interests based on a sliding window mechanism.
- *Producer*: the application that responds to each received Interest with a data packet carrying the same name as the Interest and with a specified size.

A few examples of NDN application designs that have been implemented and evaluated as ndnSIM-specific applications are: an application to achieve peer-to-peer file sharing in NDN (nTorrent) [37] and a framework that features a number of adaptive multimedia streaming (amus-ndnSIM) [27].

3.2.2 Real-World Applications

The real-world applications are generic applications and libraries that fully leverage the high-level NDN and asynchronous input/output APIs provided by the ndn-cxx library. In other words, these applications can express Interests and dispatch the retrieved Data to the supplied callbacks (as opposed to pre-defined callback for ndnSIM-specific applications), detect Interest timeouts, register prefixes with local NFD, use packet signing and verification APIs. Because ndnSIM integrates with the specially adjusted ndn-cxx library for the simulation environment, such applications can be first developed against the real prototypes and then run inside the simulation environment. Alternatively, the existing applications can be ported to run in ndnSIM to evaluate them at scale.

It is important to note that the existing applications may require several modifications to satisfy requirements imposed by the nature of discrete event simulations. Specifically, an application:

- should not use global variables to define its state, since simulations may need to create multiple instances of the same application that share the same memory;
- should not use disk operations, unless application instances access unique parts of the file system, since in the simulated environment, all application instances access the same local file system;
- should only use ndn-cxx APIs for network-level (ndn-cxx Face), event scheduling (ndn-cxx Scheduler), and absolute time operations, as otherwise the simulations may incorrectly combine simulated and real-world functions;
- must not contain any GUI or command-line interactions; and
- the entry point to the application should be configurable (e.g., provided as a C++ class), allowing customized instantiations of the application. This is a generic requirement, which in most cases is satisfied if the application/library is already provisioned for unit-testing.

So far, we and the researchers in the community have adapted and then evaluated with ndnSIM several real-world applications, including:

- *NLSR [35, 11]*: NDN Link State Routing Protocol daemon.
- *ChronoSync [55, 4]*: one of the earliest distributed protocols for dataset synchronization in NDN.
- *RoundSync [31, 15]*: a revised design of the dataset synchronization to achieve fast synchronization in face of simultaneous data productions.

Based on our experience in assisting the community with NDN application development, we have noticed that the majority of researchers prefer the simplified ndnSIM-specific application prototypes. While these applications are a good fit to drive the network-level evaluations, they have a limited approximation of NDN application semantics and require substantial effort to implement even simple application behavior (e.g., they need custom timers to handle Interest timeouts, custom callback dispatch mechanism, etc.). Therefore, we would like to encourage the community to experiment with the development of real-world NDN applications that simplify realization of NDN semantics and can be tested both inside the simulator and on the NDN testbed [7].

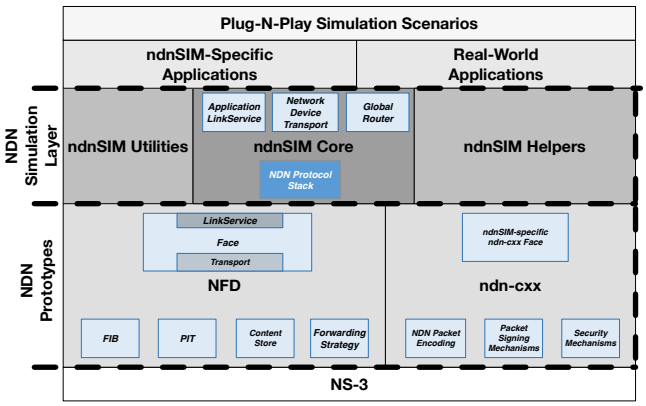

Figure 5: Structure of the ndnSIM simulation package

3.3 NDN Packet Flow in ndnSIM with Integrated ndn-cxx and NFD

The packet flow in ndnSIM involves multiple elements, including NS3's packet, device, and channel abstraction, ndnSIM core, and processing by the integrated NFD with the help of ndn-cxx library. An example of the overall workflow to forward NDN packets between two simulated nodes is illustrated in Figure 6.

The whole process is initiated by an application expressing an Interest or publishing a Data packet (after receiving an Interest for it) and includes generation of Interest and Data packets (and appropriately signing them) using abstractions provided by the integrated ndn-cxx library. Using the specialized ndnSIM Face (using AppLinkService) or a customized version of the ndn-cxx Face (using InternalClientTransport and InternalForwarderTransport underlying abstractions), these packets are injected into the NFD instance installed on the corresponding simulated node. After that, NFD will apply the necessary processing logic and will determine how to process the packet, i.e., it will create a PIT entry and forward to an outgoing Face determined by the strategy for the Interest's namespace, aggregate or drop Interest, satisfy Interest from the internal cache, cache the received Data packet and forward it to the incoming Faces, or drop the Data packet. If a packet is determined to be send out to a Face that corresponds to a simulated link, this packet is getting encoded into NS3's Packet and scheduled for transmission over NS3's NetDevice/Channel using a Face that leverages the specialized NetDeviceTransport abstraction. On the other end, NDN packet is extracted form the NS3 Packet and injected into the NFD instance for further processing.

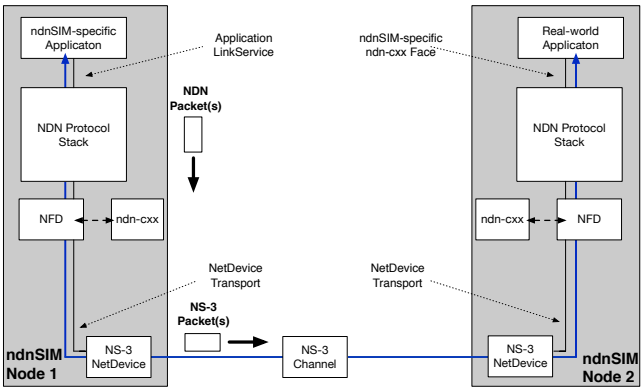

Figure 6: NDN packet exchange process between two simulated nodes

4. PROTOTYPE INTEGRATION PROCESS

In 2014, the NDN team started development of the new reference implementation of NDN forwarder and supporting library to enable multiple new NDN features (per-namespace strategies, formalized forwarding pipelines, data-centric security abstractions with trust schemas, forwarding hints, etc.) and allowing extensions through the modularity. To ensure that these advancements can be properly and with high-fidelity evaluated in the simulation environment we made a decision for ndnSIM 2.x to integrate these codebases inside ndnSIM, instead of reimplementing the same features specifically for NS3 (as was the case for ndnSIM 1.x). In the rest of this section, we discuss the challenges and trade-offs of this integration and then present our experimental evaluation of the integration overheads.

4.1 Integration Challenges

To fit the real prototypes into the NS3 world, we had to: 1) redirect the scheduler and logger of NFD and ndn-cxx to use the scheduler and logger of NS3 respectively, 2) bind the events in NFD

with the tracing facilities of NS3 and extend its data structures and pipelines to add NS-3 tracing points, and 3) enable simulation time in NFD by using an abstraction provided by ndn-cxx to convert NS-3 time to NFD system time.

Additional challenge of instantiating real NFD instances on simulated nodes is additional memory and processing requirements for simulation scenarios. To partially mitigate these overheads, we implemented a custom KeyChain to create "mocked" signatures of Data and Interest packets without incurring cost of cryptographic operations and enabled facilities to opt-out of some of the NFD components if they are not needed in specific simulations.

4.2 Integration Trade-offs

The integration of NFD and ndn-cxx into ndnSIM faced a number of trade-offs. The major change of the packet processing flow broke compatibility with the previous versions of ndnSIM. Because of the tight integration with the real NDN library that assumes properly allocated memory blocks for NDN packets, ndnSIM 2.x can no longer take advantage of the virtual payload abstraction, increasing memory requirements for packet processing and storage. In addition, the memory overhead takes additional increase because the integrated prototype forwarder realized fully featured data structures of PIT, FIB, CS, and various management structures.

At the same time, ndnSIM is now fully up-to-date with the latest advancements of NDN architecture, allowing flexible two-way experimentation and evaluation—i.e., the prototyped code can be simulated in ndnSIM and simulated code can be evaluated in real (or emulated) environments. Moreover, the integration eliminated the overhead of maintaining and synchronizing two independent codebases and united the research community. As anecdotal evidence of the latter, many questions on ndnSIM mailing list are getting answered by developers and users of NFD, some resulting in discovery of issues and requests of new features for the reference implementations.

4.3 Integration Overhead Evaluation

To quantify effects of the major re-design of ndnSIM 2.x compared to ndnSIM 1.x, we evaluated several basic use-cases of the simulator: cache replacement policies, forwarding strategies, and applications. We used a simple topology consisting of two connected nodes and measure the system execution time and average memory requirements for CS and PIT. Unless otherwise stated, each node in our evaluations had installed NDN protocol stack (an instance of NFD and the specialized Faces to communicate with NDN applications and remote simulation nodes), one node with a ConsumerCbr application instance generating 1,000 Interests/sec, and one with a Producer instance responding to the Interests. The experiments were performed on an Intel Core i7 processor (2.4 GHz) with 8 GBytes of memory machine, each experiment simulating 30 seconds of packet exchanges. We repeated each simulation ten times and we report on the minimum, maximum, and average values of all the runs.

4.3.1 Cache Replacement Policy Development Overhead

ndnSIM 2.x uses the CS implementation of NFD,[1] therefore, to create a new cache replacement policy, users need extend NFD's Policy class to implement new callbacks that are invoked when a new data packet is inserted to the CS, an existing data packet is deleted from the CS, and a data packet is about to be returned after a lookup match.

To develop a custom cache replacement policy, the API provided by NFD can be modified more easily than implementing a new C++ template class required in ndnSIM 1.x, especially for users not so experienced with software development in C++. On the other hand, NFD' cache policies framework incurs additional overhead for virtual function dispatch, resulting in a small processin penalty.

Table 1 highlights the memory overhead per CS and PIT entry, and the system execution time with the Least Recently Used (LRU) and First In First Out (FIFO) replacement policies for ndnSIM 1.x and ndnSIM 2.x. The CS has a capacity of 100,000 entries, enough to hold all Data packets during each simulation run, allowing us to measure the memory overhead as the CS size grows. As expected from the loss of virtual payload capability, we observe that the memory overhead per CS entry in ndnSIM 2.x is higher than of ndnSIM 1.x. At the same time, the observed seven-fold average increase is larger than what we expected and we are currently investigating the underlying reasons for this change, including effects of the memory allocation for C++ STL data structures, memory fragmentation, and potential memory leaks in the prototype implementation. The system execution time for ndnSIM 2.x is as expected longer, as it runs the real NFD prototype code that performs more sophisticated processing than the simplified packet forwarding logic in the original ndnSIM 1.x.

4.3.2 Forwarding Strategy Development Overhead

In ndnSIM 1.x, the forwarding plane used a single forwarding strategy chosen for the whole duration of the simulation. Among available strategies were Multicast (a.k.a. Broadcast), BestRoute, and adaptive "Green-Yellow-Red" variants of both.[2] Among the important facilities that can be leveraged by the strategy (not yet available in NFD and ndnSIM 2.x) was per-face and per-FIB entry rate limits on number of Interests. In ndnSIM 2.x, a forwarding strategy is implemented as a part of the forwarding plane of NFD and can be selected to activate in a specific namespace. To add a new forwarding strategy, users need to extend the Strategy class of NFD and implement certain callbacks that will be invoked when an Interest is received, before an Interest is satisfied by a data packet, and when a NACK is received.

The logic of adding a new forwarding strategy in ndnSIM 1.x and 2.x is similar, however, the APIs provided by NFD are more specialized to make determination where/whether to forward the Interest, while ndnSIM 1.x provided more generic APIs for strategies to control all aspects of all type of NDN packet forwarding. In other words, NFD clearly separates the generic logic of Interest (duplicate suppression, aggregation, etc), Data, and Nack packet processing (so called "pipelines" in [18]) and customizable decision (and feedback) to forward Interests, compared to ndnSIM 1.x that combined all of these under a single umbrella of extensible forwarding strategy API. Table 2 shows the memory overhead per CS and PIT entry and the system execution time in the case of the BestRoute and Multicast forwarding strategies for ndnSIM 1.x (with Green-Yellow-Red scheme) and ndnSIM 2.x (non-adaptive variants). The results are similar to those presented in section 4.3.1 with similar conclusions and future ares of improvements.

4.3.3 Application Development Overhead

ndnSIM 2.x enables applications to communicate directly with the local NFD instance, therefore it supports simulations of both ndnSIM-specific and real-world applications. In ndnSIM 1.x, an application communicates with a simulated forwarding plane, there-

[1] Version 2.x of ndnSIM includes support for the ndnSIM 1.x version of cache policies, which will be phase out in future releases

[2] Additional strategies are available as part of car-to-car [**?**], interest flooding [**?**], SAF [40], and several other research efforts.

Table 1: Cache Replacement Policy Development Overhead between ndnSIM 1.x & ndnSIM 2.x

	ndnSIM 1.x (min/max/avg)		ndnSIM 2.x (min/max/avg)	
Cache Replacement Policy	LRU	FIFO	LRU	FIFO
PIT Entry Overhead (Kilobytes)	5.22/5.33/5.29	5.20/5.31/5.24	37.82/40.21/39.23	38.04/40.05/39.12
CS Entry Overhead (Kilobytes)	0.75/0.79/0.77	0.74/0.77/0.75	5.41/6.92/6.24	5.41/7.38/6.91
System Time (s)	12.84/14.13/13.56	12.49/14.71/13.92	23.56/24.93/24.03	27.57/29.11/28.53

Table 2: Forwarding Strategy Development Overhead between ndnSIM 1.x & ndnSIM 2.x

	ndnSIM 1.x (min/max/avg)		ndnSIM 2.x (min/max/avg)	
Forwarding Strategy	Best Route	Multicast	Best Route	Multicast
PIT Entry Overhead (Kilobytes)	5.23/5.39/5.30	5.22/5.32/5.28	37.73/40.12/39.35	38.81/40.21/39.51
CS Entry Overhead (Kilobytes)	0.74/0.77/0.76	0.73/0.77/0.75	6.34/6.45/6.40	6.38/6.41/6.39
System Time (s)	12.87/14.58/13.71	13.04/14.72/14.09	22.67/23.98/23.12	21.72/22.x9/22.01

fore, it can only simulate ndnSIM-specific applications but cannot run real-world applications.

To make a fair comparison, Table 3 presents the memory overhead per CS and PIT entry and the system execution time for ndnSIM 1.x and ndnSIM 2.x when running two ndnSIM-specific applications: 1) the ConsumerCbr application with a constant rate of 2000 Interests/second, and 2) the ConsumerZipfMandelbrot application generating 1000 Interests/second, where the names of the generated Interests are based on an NS3 ZipfRandomVariable instance with α = 1.x and N = 100.

Because of the methodology of our experiment (we are measuring the overall memory use that includes tge overhead of all other auxiliary data structures), the overall overhead of ConsumerZipfMandelbrot application experiment is higher for both ndnSIM 1.x and 2.x that that of ConsumerCbr because of the smaller number of generated of Interest and Data packets. Beyond that, the overhead for memory use and processing time follows the same pattern as reported in previous sections, with higher numbers for ndnSIM 2.x than in ndnSIM 1.x because of the additional requirements of the fully featured NFD implementation.

5. SOFTWARE DEVELOPMENT, GOVERNANCE MODEL & COMMUNITY ADOPTION

In this section, we present statistics from our GitHub repository [8] and our mailing list [9] along with the the number of technical report citations (according to Google Scholar) to measure the ndnSIM software development effort and adoption in terms of community growth respectively. We also present the open-source governance model and stakeholders of ndnSIM.

5.1 Software Development Effort

In Figure 8, we present the number of commits, lines of code added, and deleted and total contributors per year from our GitHub repository [8].

During the first 3 years of development (before the integration with the prototypes), we had to maintain simulation-specific software, which required a lot of effort in terms of coding. In 2014, NFD and ndn-cxx were initially sub-folders of the simulator codebase (their commit history was added to the commit history of ndnSIM, resulting in the corresponding spike in Figure 8). Starting from 2015, to allow easier integration of every new release with the simulator, we decided to maintain them as separate GitHub submodules (a number code lines were deleted), therefore, the coding effort itself became less demanding. However, the cooperative software design effort with the rest of the NDN team significantly increased to make sure that the features developed for NFD and ndn-cxx are compatible with ndnSIM and NS-3.

The NDN Team follows the full cycle of software development (issue tracking, code review, unit-testing, etc.) used in the software development industry. The code review process introduced for ndnSIM in 2015 to ensure that each commit is comprehensive and well-designed before it is pushed to our GitHub repository.

We should note that we receive a limited number of GitHub pull requests and issues, therefore, these metrics are not representative of the ndnSIM software development effort. The majority of our users submit their questions and code related issues on our mailing list, which has created a user community at large, where people help out with each other's questions (as explained in detail in Section 6). Users also typically fork the official ndnSIM GitHub repository and do their development on their personal repositories, while they share code patches and extensions through the mailing list and submit their code to our code review system to get it merged to our official GitHub repository.

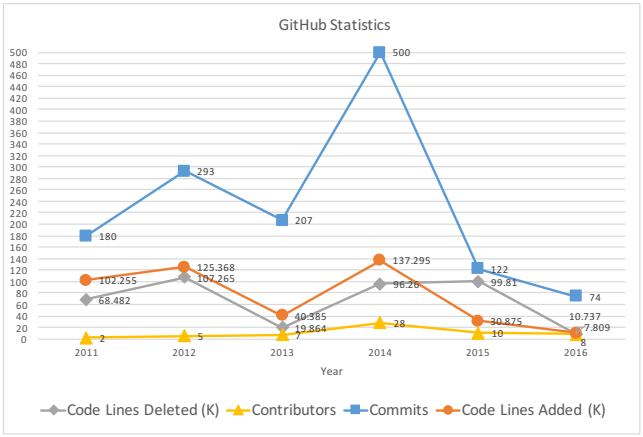

Figure 7: Statistics from the ndnSIM GitHub repository

5.2 Open-Source Governance Model & Stakeholders

ndnSIM is an open-source project, where every researcher can commit their code after going through the code review process (the committers do not need to be NDN Team members). As shown in Figure 8, a number of committers external to the NDN Team have contributed to the development of the simulator over the years.

Table 3: Application Development Overhead between ndnSIM 1.x & ndnSIM 2.x

	ndnSIM 1.x (min/max/avg)		ndnSIM 2.x (min/max/avg)	
Consumer Application	ConsumerCbr	ConsumerZipfMandelbrot	ConsumerCbr	ConsumerZipfMandelbrot
PIT Entry Overhead (Kilobytes)	5.13/5.24/5.18	8.08/9.45/9.03	38.02/40.22/39.54	287.69/299.167/294.21
CS Entry Overhead (Kilobytes)	0.73/0.76/0.75	2.59/3.64/3.06	6.17/6.42/6.31	82.83/84.08/83.54
System Time (s)	25.22/29.69/27.36	3.44/3.72/3.51	45.65/48.13/46.22	2.58/2.87/2.71

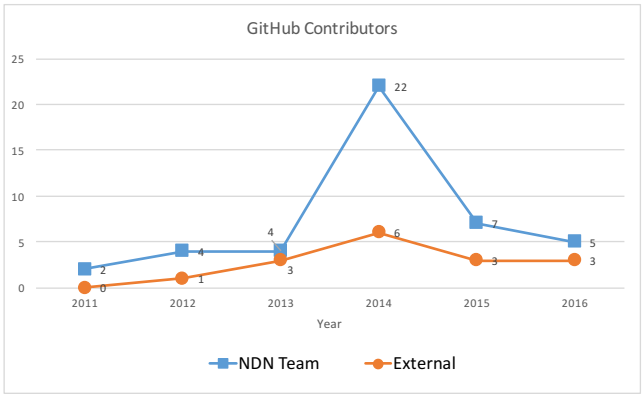

Figure 8: ndnSIM internal (NDN team) and external contributors

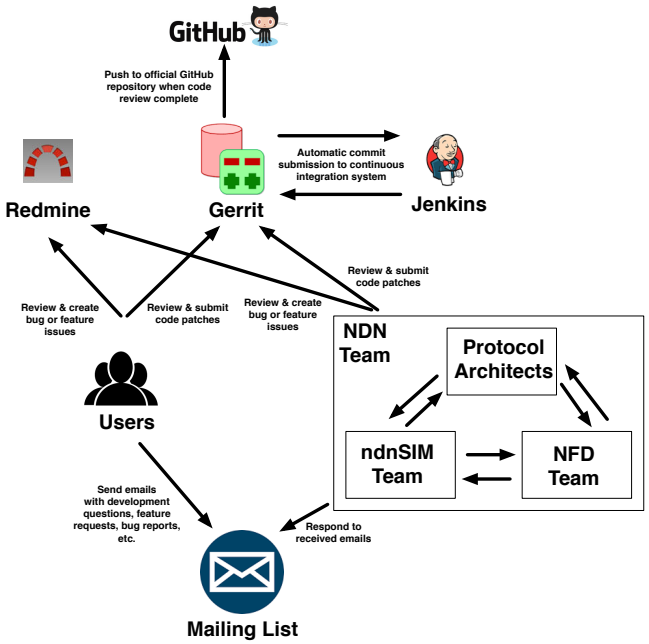

Figure 9: Stakeholders, their relationship and the development workflow facilitated by ndnSIM

In Figure 9, we present the ndnSIM stakeholders, their relationship and the facilitated development workflow. At the bottom left, we have the users that can: 1) directly submit feature and bug reports to our Redmine issue tracking system, 2) submit their code for review to our Gerrit code review system, and 3) send an email to the mailing list with development related questions, feature requests and bug reports.

At the bottom right, we have the ndnSIM team, which is a part of the overall NDN project team. The ndnSIM team works closely with the NDN protocol architects and the NFD team to participate to the protocol design effort and ensure that the software changes done in NFD are compatible with ndnSIM. The ndnSIM team with the participation and help of the entire NDN team responds to the user emails received on the list, reviews the already submitted and creates new issues on Redmine, reviews already submitted (either by users or members of the NDN and ndnSIM team itself), and submits new code patches (commits) to Gerrit.

Once a commit is submitted to Gerrit, it is automatically submitted to continuous integration system powered by Jenkins-CI, where it is compiled and tested on a number of different operating systems including several Ubuntu and macOS distributions. Once the commit is verified by Jenkins-CI and the code review process is complete, it is merged to our official GitHub repository.

We should note that users can also participate in discussion related to the specific Redmine issues (in Redmine and the mailing list) and Gerrit code review process, ensuing critical bugs fixed and important features are implemented in a timely manner.

5.3 Community Growth

The ndnSIM community has grown from a few dozens to some hundreds of members over the last five years. At the time of this writing, the ndnSIM mailing list has approximately 550 subscribers and the technical reports have been cited 425 times in total. As it is shown in Figure 10 and Figure 11, the mailing list becomes more active every year (steady increase in the number of threads and individual emails on the mailing list over the last couple of years), while ndnSIM is used and cited by more researchers.

We would like to thank all the members of the community for their help and feedback and, especially, our much-appreciated contributors; Jiangzhe Wang, Cheng Yi, Saeid Montazeri, Xiaoke Jiang, Saran Tarnoi, Hovaidi Ardestani Mohammad, Michael Sweatt, Wentao Shang, Christian Kreuzberger, Yuanzhi Gao, and Mohammad Sabet, Junxiao Shi, Susmit Shannigrahi, John Baugh, Ashlesh Gawande, and many others who have reported bugs, submitted patches, and helped ndnSIM users on the mailing list.

6. LESSONS LEARNED

Developing an open-source simulation platform used by a growing and active user community is a rewarding process that has helped to us learn a number of lessons. In this section, we would like to share those lessons with the research community.

A well-designed simulation platform facilitates the protocol design effort. It helps researchers understand the architectural trade-offs by enabling large scale experimentation; some design deficiencies come up only after intensive experimentation. A recent example is ChronoSync [3], where a design bug that could lead to large delays in the case of multiple simultaneous data generations was discovered and fixed only after large scale simulations.

Researchers should be able to reproduce each other's experiments. Reproducibility is one of the most important aspect of experiment-based research. Because of the continuous evolution

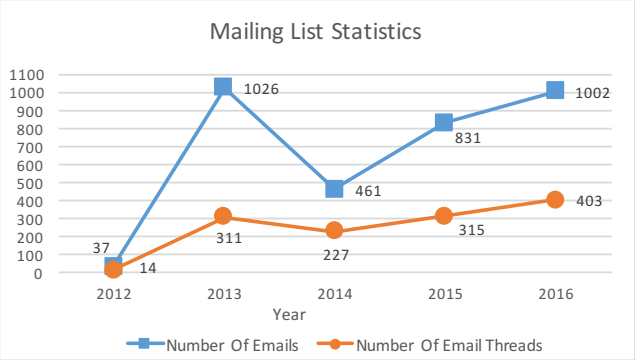

Figure 10: Statistics from the ndnSIM mailing list

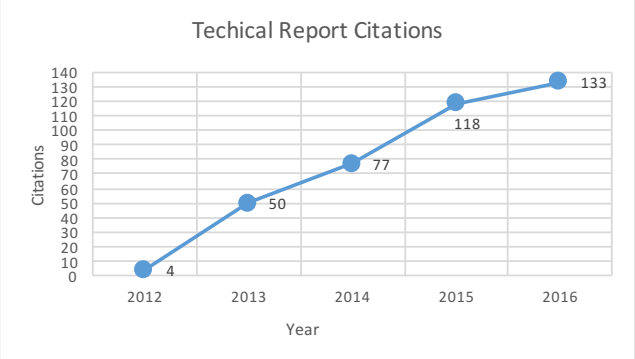

Figure 11: Number of citations of ndnSIM technical reports

of NDN, the reference implementation, and ndnSIM, it is critical to capture the specific version of ndnSIM used for simulations. To promote such recording and ensure that simulation results can be easily reproduced, we created a specialized template [**?**] to simplify managing simulation scenarios and ensuring future-proof ability to re-run experiments. In addition to that, we are planning to setup a database to collect all the simulation scenarios used in scientific papers, so that our users have direct access to each other's experiments.

An open communication channel with the user community is crucial for an open-source project. A number of times, users have helped each other by responding to questions on the mailing list, but also contributed to the actual software development by implementing specific features.

We have established this channel in two ways:

- By extensively documenting ndnSIM and providing a large set of basic simulation scenario examples on ndnSIM website [10], encouraging user participation and helping getting familiar with the codebase.
- By maintaining a mailing list to allow further collaboration and assistance to questions that have not been addressed in the documentation. The mailing list has helped us further improve our documentation by identifying and addressing a number of frequently asked user questions that fall into one of the following categories: 1) understanding of the simulation outcomes related to packet tracing and simulation execution visualization, 2) experimentation with the NDN architectural parameters, and 3) software development questions, as users are required to have a good understanding of C++ and C++11 for NFD and ndn-cxx extensions.

Developing an open-source software project is an iterative process. The developed features may need to be redesigned, refactored, or extended based on the feedback from the users. For instance, right after the NFD integration, we received a number of emails on the mailing list (implicitly) requesting a later introduced API to optionally disable some of the NFD features not needed for basic scenarios. When the support for NACKs was added to NFD, a number of users requested this feature in the simulator, which required the following refactoring of ndnSIM internals with adjustment of several trade-offs (limiting ability to optimize memory overhead).

The simulator and the prototypes facilitate and influence each other's development. When the NDN team started working on ndn-cxx and NFD, parts of the ndnSIM codebase were used to facilitate their development. Eventually, when the prototypes were developed, they were integrated in ndnSIM.

Since the beginning of this integration, we have been working closely with the NFD Team to make the forwarder more modular and compatible with the simulator. An outcome of this collaboration is the higher level of modularity of NFD and ndn-cxx with additional parts that are conditionally compiled. Initially, we had to manually remove parts of the NFD and ndn-cxx codebase not compatible with the simulation environment , including implementations of TCP/UDP channels, support for Unix sockets, logging, etc. The latest version of NFD and ndn-cxx is integrated within ndnSIM with minimal changes.

7. LIMITATIONS AND FUTURE WORK

In this section, we discuss the current simulator limitations, the pain points of its development process, and our future work plan.

7.1 Current ndnSIM Limitations

ndnSIM is currently supported on Linux and Mac OS platforms, but unavailable on Windows (NS3 in general has limited support on Windows platform). It also does not support connecting the simulation network with an NFD, ndn-cxx, or an application instance running on an external host.

As stated in section 3.2, real-world applications need to be modified in certain ways in order to run in ndnSIM. The memory requirements can become a limiting factor if one runs ndnSIM on devices with limited hardware resources (e.g., old or low-end laptops) to simulate large scale scenarios with more than a few hundred nodes. The lack of full backward compatibility of new releases may also limit the portability of user-implemented features.

Overall, development of ndnSIM faces a number of challenges, including:

- Every release of NFD and ndn-cxx need to be manually integrated with ndnSIM by applying a set of customization commits to NFD and ndn-cxx (although the set has been shrinking).
- To enable the visualization of NDN simulation scenarios and data structures (FIB, PIT, CS), we need to patch and extend the provided NS-3 python bindings which requires substantial efforts, especially when NFD/ndn-cxx make significant changes in the supporting data structures.
- Unit-testing of software built on top of NS-3 is not always a straightforward process. The sequence of the events scheduled in the simulation environment may vary for each execution, therefore we need to ensure that existing and newly added unit tests are resistant against such random variations.

7.2 Future Work

To make the research community aware of the lower level details of the simulation development process, we plan to extend our documentation and publish programming "HOW-TOs" on the ndnSIM website. We also plan to enrich our collection of plug-n-play simulation scenarios to demonstrate new interesting use cases and network environments that can take advantage of NDN's communication model.

As mentioned in Section 6, we have been working closely with the NFD Team to enable conditional compilation of NFD components for ndnSIM. This can be extended to the implemented NFD and ndn-cxx optimizations for ndnSIM, so that every new release of the NDN prototypes can be made automatically compatible with ndnSIM without requiring manual integration.

To further improve ndnSIM's scalability, we plan to investigate in detail the memory consumption of each simulated forwarder instance, and come up with a concrete plan to reduce this memory consumption. We also plan to make the backward compatibility a high priority in future releases, to the extend possible we will work with the ndnSIM user community to minimize, if not eliminate, the need for users to modify their scenarios and ndnSIM extensions used in the previous simulations.

In the future, we would also like to incorporate ndnSIM into the standard NS-3 codebase to promote NDN research and encourage all NS-3 users to participate.

8. RELATED WORK

Three most common approaches to experimental evaluation of network architecture designs include testbed deployment, emulation, and simulation.

8.1 Testbed Deployment

The NDN team has been running a testbed since the beginning of the NDN project. The testbed currently consists of 35 nodes spanning four continents. It runs the latest versions of NFD and is open to all interested researchers to use for their own NDN experiments. However, one needs to coordinate with the testbed operators first if one's experiment requires modifications to any parts of the NFD and ndn-cxx instances on any of the testbed nodes.

The NDN project team also offers NDN experimentation on the Open Network Lab (ONL) [14], which contains 14 programmable routers and over 100 client nodes. Compared to the testbed, ONL offers a more tightly controlled experimental environment with a rich set of measurement and monitoring tools.

Years of research efforts have resulted in a number of emulation-based (i.e., based on various virtualization and containerization technologies) testbeds being developed and deployed, such as 1) NITOS [39], a facility for cloud-based wireless experimentation; 2) GENI [23], a federated testbed for network experimentation; 3) Planetlab [28], a general purpose, overlay testbed for broad-coverage network services; 4) Motelab [51], a testbed consisting of wireless sensors; and 5) ORBIT [41], a radio grid facility for wireless protocols.

The testbed approach has the advantage of requiring no changes to the software being tested, which simulation approaches often require porting prototype software to simulation tools. However it only allows for experimentation with the scale to the number of the testbed nodes (in some cases, it may require manual setup of the experimentation software on each node). For experimentation with larger networks, researchers need to resort to simulations.

8.2 Emulation

In addition to the testbed, the NDN team also developed an NDN emulator, called mini-NDN [5], which is based on the Mininet emulator [46]. A number of other networking emulator extensions have been built on top of Mininet as well: 1) Mini-CCNx [24], an emulator for Content Centric Networking (CCN) [19]; 2) Mininet-WiFi [32], an emulator for Software Defined Wireless Networks; 3) SDDC [30], a software defined datacenter experimental framework; and 4) Maxinet [52], a distributed emulator of software-defined networks.

Generally speaking, an emulation framework provides more realistic experimental conditions than a simulation framework. In the case of the NDN frameworks, mini-NDN and ndnSIM provide comparable result fidelity and result reproducibility. NFD, NLSR and real-world applications can run on mini-NDN without any changes, making an emulation experimentation easier than using ndnSIM. However, mini-NDN can scale up to medium-sized networks (up to a couple hundreds of nodes), therefore ndnSIM is again needed for larger scale experimentations.

8.3 Simulation

ccnSim [26] is a chunk-level simulator for CCN networks [19] (a realization of ICN). It is written in C++ under the Omnet++ framework [48]. Its implementation focuses on the analysis of in-network caching performance, without being a fully-featured ICN simulator. CCNPL-SIM [1] is another CCN simulator leveraging a platform-specific implementation of the CCN principles; every time the CCN architecture changes, the simulator codebase needs to be manually updated to include the new features, since it does not support the integration of the CCN prototype software into the simulator.

The approach of integrating prototype software in NS-3 has been taken by a few other simulators as well. OFSwitch13 [25] is a simulation framework that enhances NS-3 with OpenFlow 1.3 support. Both OFSwitch13 and ndnSIM utilize the standard NS-3 Channel and NetDevice abstractions to create communication channels and make use of an external library, ofsoftswitch13 and ndn-cxx, respectively. However, OFSwitch13 models OpenFlow hardware operations and extends the NS-3 Queue class to provide some basic QoS, while ndnSIM does not deal with any hardware operations.

The NS3 DCE CCNx [13] project leverages the Direct Code Execution (DCE) module of NS-3 to simulate the CCNx prototype [2], which is the software implementation of the legacy version of the CCN protocol. NS3 DCE CCNx uses the TapBridge model provided by NS-3 to connect a real-world host with the simulation network, while ndnSIM exclusively uses the NS-3 NetDevice abstraction and does not support the connection with an external Linux process (e.g., an external NFD instance). The DCE module is known to cause a number of scaling issues, since every node in the simulation has to run a full-sized instance of the simulated protocol code plus a DCE software layer on top of that.

DCE-Cradle [45] is a simulation framework that extends NS-3 to enable the simulation of native Linux protocol stacks by reusing their original code. DCE Cradle replaces the NS-3 Socket abstraction to enable NS-3 applications to access the Linux network stack, which is similar to the direction that ndnSIM takes by allowing both ndnSIM-specific and real-world applications to access the original NDN protocol stack, thus reusing the prototype code.

9. CONCLUSIONS

It has been a rather rewarding experience for us to busy work on ndnSIM extensions while watching the ndnSIM community growing over the last few years. In this paper we share the design challenges we encountered, the tradeoffs from the design decisions, and

the lessons we have learned. We hope that these insights are informative not only to the existing ndnSIM user community but also to the network research community at large, and that this paper could serve as an invitation to everyone to use ndnSIM as a handy tool in exploring NDN research.

ACKNOWLEDGMENT

This work is partially supported by the National Science Foundation under award CNS-1345318 and CNS-1629922.

10. REFERENCES

[1] CCNPL-SIM simulation framework. http://systemx.enst.fr/ccnpl-sim.

[2] CCNx Project. http://blogs.parc.com/ccnx/.

[3] ChronoSync Redmine Issue 3928. https://redmine.named-data.net/issues/3928.

[4] ChronoSync Simulation Repository. https://github.com/spirosmastorakis/ChronoSync.

[5] Mini-NDN GitHub. https://github.com/named-data/mini-ndn.

[6] NDN Applications. https://named-data.net/codebase/applications/.

[7] NDN Testbed. http://ndndemo.arl.wustl.edu.

[8] ndnSIM GitHub Repository. https://github.com/named-data-ndnSIM/ndnSIM.

[9] ndnSIM Mailing List. http://www.lists.cs.ucla.edu/mailman/listinfo/ndnsim.

[10] ndnSIM Website. http://ndnsim.net.

[11] NLSR-SIM. https://github.com/3rd-ndn-hackathon/ndnSIM-NLSR.

[12] ns-3. http://www.nsnam.org/.

[13] NS3 DCE CCNx Quick Start. http://www-sop.inria.fr/members/Frederic.Urbani/ns3dceccnx/index.html.

[14] Open Networking Lab. http://onlab.us.

[15] RoundSync Simulation Repository. https://github.com/spirosmastorakis/RoundSync.

[16] Zipf-Mandelbrot Law.

[17] Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM: NDN simulator for NS-3. Tech. Rep. NDN-0005, NDN, 2012.

[18] Alexander Afanasyev, Junxiao Shi, et al. NFD Developer's Guide. Tech. Rep. NDN-0021, NDN, 2015.

[19] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7), 2012.

[20] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. Multi-source data retrieval in iot via named data networking. In *Proceedings of the 1st international conference on Information-centric networking*, pages 67–76. ACM, 2014.

[21] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. Forwarding strategies in named data wireless ad hoc networks: Design and evaluation. *Journal of Network and Computer Applications*, 50:148–158, 2015.

[22] Hila Ben Abraham and Patrick Crowley. Forwarding strategies for applications in named data networking. In *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*, pages 111–112. ACM, 2016.

[23] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[24] Carlos Cabral, Christian Esteve Rothenberg, and MaurÃ■cio Ferreira MagalhÃčes. Reproducing real ndn experiments using mini-ccnx. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 45–46. ACM, 2013.

[25] Luciano Jerez Chaves, Islene Calciolari Garcia, and Edmundo Roberto Mauro Madeira. Ofswitch13: Enhancing ns-3 with openflow 1.3 support. In *Proceedings of the Workshop on ns-3*, pages 33–40. ACM, 2016.

[26] Raffaele Chiocchetti, Dario Rossi, and Giuseppe Rossini. ccnsim: An highly scalable ccn simulator. In *Communications (ICC), 2013 IEEE International Conference on*, pages 2309–2314. IEEE, 2013.

[27] Benjamin Rainer Christian Kreuzberger, Daniel Posch and Hermann Hellwagner. Demo: amus-ndnSIM – adaptive multimedia streaming simulator for ndn.

[28] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[29] Ali Dabirmoghaddam, Maziar Mirzazad Barijough, and JJ Garcia-Luna-Aceves. Understanding optimal caching and opportunistic caching at the edge of information-centric networks. In *Proceedings of the 1st international conference on Information-centric networking*, pages 47–56. ACM, 2014.

[30] Ala Darabseh, Mahmoud Al-Ayyoub, Yaser Jararweh, Elhadj Benkhelifa, Mladen Vouk, and Andy Rindos. Sddc: A software defined datacenter experimental framework. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 189–194. IEEE, 2015.

[31] Pedro de-las Heras-QuirÃşs, Eva M. Castro, Wentao Shang, Yingdi Yu, Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. The design of RoundSync protocol. Technical Report NDN-0048, NDN, April 2017.

[32] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 384–389. IEEE, 2015.

[33] Giulio Grassi, Davide Pesavento, Giovanni Pau, Rama Vuyyuru, Ryuji Wakikawa, and Lixia Zhang. Vanet via named data networking. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 410–415. IEEE, 2014.

[34] Giulio Grassi, Davide Pesavento, Giovanni Pau, Lixia Zhang, and Serge Fdida. Navigo: Interest forwarding by geolocations in vehicular named data networking. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*, pages 1–10. IEEE, 2015.

[35] AKM Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. NLSR: Named-data Link State Routing Protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 15–20. ACM, 2013.

[36] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnsim 2: An updated ndn simulator for

ns-3. Technical report, Technical Report NDN-0028, Revision 2, NDN, 2016.

[37] Spyridon Mastorakis, Alexander Afanasyev, Yingdi Yu, and Lixia Zhang. nTorrent: Peer-to-Peer File Sharing in Named Data Networking. In *26th International Conference on Computer Communications and Networks (ICCCN)*, July 2017.

[38] NDN Project Team. ndn-cxx.

[39] Katerina Pechlivanidou, Kostas Katsalis, Ioannis Igoumenos, Dimitrios Katsaros, Thanasis Korakis, and Leandros Tassiulas. Nitos testbed: A cloud based wireless experimentation facility. In *Teletraffic Congress (ITC), 2014 26th International*, pages 1–6. IEEE, 2014.

[40] Daniel Posch, Benjamin Rainer, and Hermann Hellwagner. Saf: Stochastic adaptive forwarding in named data networking. *IEEE/ACM Transactions on Networking*, 2017.

[41] Dipankar Raychaudhuri, Ivan Seskar, Max Ott, Sachin Ganu, Kishore Ramachandran, Haris Kremo, Robert Siracusa, Hang Liu, and Manpreet Singh. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pages 1664–1669. IEEE, 2005.

[42] Klaus Schneider, Cheng Yi, Beichuan Zhang, and Lixia Zhang. A practical congestion control scheme for named data networking. In *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*, pages 21–30. ACM, 2016.

[43] Wentao Shang, Adeola Bannis, Teng Liang, Zhehao Wang, Yingdi Yu, Alexander Afanasyev, Jeff Thompson, Jeff Burke, Beichuan Zhang, and Lixia Zhang. Named data networking of things. In *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*, pages 117–128. IEEE, 2016.

[44] Hassan Sinky and Bechir Hamdaoui. Cloudlet-aware mobile content delivery in wireless urban communication networks. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–7. IEEE, 2016.

[45] Hajime Tazaki, FrÃľdÃľric Urbani, and Thierry Turletti. Dce cradle: simulate network protocols with real stacks for better realism. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pages 153–158. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013.

[46] Mininet Team. Mininet. http://mininet. org, 2014.

[47] Michele Tortelli, Luigi Alfredo Grieco, Gennaro Boggia, and Kostas Pentikousisy. Cobra: Lean intra-domain routing in ndn. In *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*, pages 839–844. IEEE, 2014.

[48] AndrÃąs Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[49] Satyanarayana Vusirikala, Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. Hop-by-hop best effort link layer reliability in named data networking. *NDN, Technical Report, NDN-0041*, 2016.

[50] Yonggong Wang, Zhenyu Li, Gareth Tyson, Steve Uhlig, and Gaogang Xie. Optimal cache allocation for content-centric networking. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10. IEEE, 2013.

[51] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. Motelab: A wireless sensor network testbed. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 68. IEEE Press, 2005.

[52] Philip Wette, Martin Draxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. Maxinet: Distributed emulation of software-defined networks. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.

[53] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. A case for stateful forwarding plane. *Computer Communications*, 36(7):779–791, 2013.

[54] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, et al. Named data networking. *Comp. Comm. Review*, 2014.

[55] Zhenkai Zhu and Alexander Afanasyev. Let's chronosync: Decentralized dataset state synchronization in named data networking. In *21st IEEE International Conference on Network Protocols (ICNP 2013)*, 2013.