NDN, Technical Report NDN-0028, 2015. http://named-data.net/techreports.html
Revision 1: January 27, 2015

ndnSIM 2.0: A new version of the NDN simulator for NS-3

Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko and Lixia Zhang
University of California, Los Angeles
{mastorakis, afanasev, iliamo, lixia}@cs.ucla.edu

Abstract—The fundamental departure of the Named-Data Networking (NDN) communication paradigm from the IP principles requires extensive evaluation through experimentation, and simulation is a necessary tool to enable the experimentation at scale. We released the first version of ndnSIM, an open source NS-3-based NDN simulator, back in June 2012. Since then, ndnSIM has undergone substantial development resulting in ndnSIM 2.0, which was released in January 2015. This paper reports the design and features of this new simulator version. The goal of the new release is to match the simulation platform to the latest advancements of NDN research. Therefore, it uses the ndn-cxx library (NDN C++ library with eXperimental eXtensions) and the NDN Forwarding Daemon (NFD) to enable experiments with real code in a simulation environment.

1 Introduction

Named Data Networking (NDN) [1, 2, 3, 4] represents a fundamental departure from today's Internet architecture which names the communication endpoints, and aspires to be the successor of the TCP/IP protocol stack. As a result, the various design options in the NDN architecture needs to be evaluated in large-scale experimentation. However, it is infeasible to conduct such experimentation with real-world infrastructure and simulation-based evaluation becomes necessary. The main goal of ndnSIM has always been offering the NDN community a common, user-friendly, and open-source simulation platform based on the NS-3 simulator framework [5].

The first public release of ndnSIM took place in June 2012 [6]. Since then, ndnSIM has become a popular tool used by many researchers around the globe. At the time of this writing, ndnSIM mailing list has over 300 subscribers, and more than 100 papers have been published based on research done using ndnSIM.

Since the first ndnSIM release, though, the NDN team has released an updated version of the protocol, mainly featured by the ndn-cxx library (NDN C++ library with eXperimental eXtensions) [9], and a new modular NDN Forwarding Daemon (NFD) [7, 8]. The ndn-cxx library implements the major NDN primitives that can be used to implement various applications. It is an actively developed project and it is used in practice for real application experiments. NFD is a network forwarder that implements and evolves together with the NDN protocol. The main design goal of NFD is to support diverse experimentation with the NDN architecture, while emphasizing modularity and extensibility to allow easy experiments with new protocol design features, algorithms and applications. The main functionality of NFD is to forward Interest and Data packets. To do this, it abstracts lower-level network transport mechanisms into NDN Faces, maintains basic data structures like Content Store (CS), Pending Interest Table (PIT), and Forwarding Information Base (FIB), and implements the packet processing logic. In addition to basic packet forwarding, it also supports multiple forwarding strategies, and a management interface to configure, control, and monitor NFD

The goal of this new ndnSIM release is to match the simulation platform to the aforementioned latest advancements of NDN research and also consolidate the efforts of using code written for the simulator in real experiments and code from real experiments in the simulated environment. In this way, ndnSIM 2.0 offers a better user experience and more realistic simulation behavior. Namely, ndnSIM 2.0 has the following major enhancements and features compared to the first release:

- All NDN forwarding and management is implemented directly using the source code of NFD.
- ndnSIM directly uses implementation from the ndn-cxx library.
- The used packet format changed to the latest NDN packet format.

This version of the simulator, just like the previous one, is implemented in a modular way using different C++ classes to model the behavior of each NDN entity: Application and Network Device Face, NFD's Forwarding Interest Table (FIB), Pending Interest Table (PIT) and Contest Store (CS), etc. This modular structure allows the easy modification or replacement of any component with no or minimal impact on the other components. In addition, the new release provides a more extensive collection of interfaces and helpers to perform detailed tracing of every component, as well as of the NDN traffic

flow.

In order to improve the user experience even more, we encourage the community to provide us valuable feedback by submitting bug reports. We also welcome requests for new feature development.[1] More information about the simulator, basic examples, and tutorials are available on the ndnSIM website: http://www.ndnsim.net/.

2 Design

The design for ndnSIM 2.0 has been directed by our aim to achieve full integration with Named Data Networking Forwarder (NFD) [7, 8]. In this section, we present the overall design of ndnSIM and demostrate its main structural components and the way that they interact with each other.

2.1 Design summary

The design of ndnSIM 2.0 includes various changes compared to that of the first release, which were mainly prompted by the NFD integration. Despite the fact that the NDN protocol stack (ndn::L3Protocol) that is installed in each simulation node still remains the core component of the ndnSIM, the packet processing is now performed based on the NFD implementation (with minor NS-3 specific changes).

As a result of the NFD integration, the code used for any experiments with NDN forwarding (e.g., custom forwarding strategies) by the real NFD implementation can be directly used by ndnSIM, and vice versa, offering to the researchers the flexibility to simulate scenarios with different strategies assigned to different namespace. In other words, the forwarding plane extensions can be used in both ndnSIM simulations and real NFD deployment, a feature that was not feasible using the previous version of the simulator. Moreover, the per-namespace strategy feature, the full-featured support for Interest selectors and crypto operations and the use of the full-featured NDN packet format were not possible to be simulated before. These new features ensure that the simulations are maximally realistic. Despite these introduced features, ndnSIM 2.0 is only slightly slower than ndnSIM 1.0. However, its memory consumption is higher than ndnSIM 1.0, but it is still within a reasonable limit, so that demanding simulations can run on some general-purpose hardware.

ndnSIM is implemented as a new network-layer protocol model and can run on top of any available link-layer protocol model (point-to-point, CSMA, wireless, etc.). In addition to that, the simulator provides an extensive collection of interfaces (i.e., Face, Network Device Face and Application Face abstractions) and helpers (i.e., Application, FIB, Global Routing, Link Control, NDN Stack and Strategy Choice helpers) to perform detailed tracing behavior of every component, as well as NDN traffic flow.

1. Bug reports and feature recommendations can be submitted on the NDN project issue tracking system website: http://redmine.named-data.net/projects/ndnsim

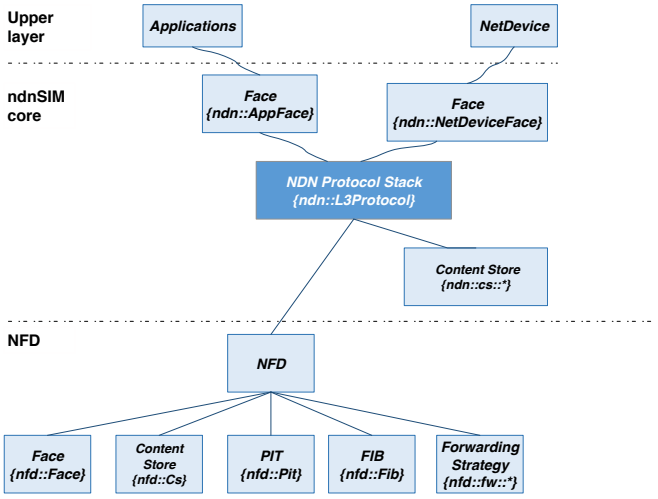

Fig. 1: Structural diagram of the ndnSIM design components

The basic components of this ndnSIM release are shown in Figure 1 and a comparison among their design principles and features for the new and the previous release of the simulator is presented in Table 1. These components are also listed below:

- **ndn::L3Protocol**: NS-3 abstraction of the NDN stack implementation. Its main task is initialization of the NFD instance of each node that participates in the simulation scenario and provides tracing sources to measure NDN performance (sent/received interest and data, satisfied/unsatisfied interests).
- **NFD**: implementation of the Named Data Networking Forwarding Daemon, including:
  - **nfd::Forwarder**: main class of NFD, which owns all faces and tables of the NDN router node and implements NDN forwarding pipelines.
  - **nfd::Face**: base class of NFD Face abstraction that implements the required communication primitives to actually send and receive Interest and Data packets.
  - **nfd::Cs**: the cache of Data packets that is used by NFD. The current release of ndnSIM also includes the old **ndn::ContentStore** abstraction ported from the previous release to enable richer options for simulation of content store operations (nfd::Cs is not yet as flexible when it comes to cache replacement policies).
  - **nfd::Pit**: the Pending Interest Table (PIT) of NFD keeps track of Interest packets that were forwarded upstream toward one (or more) content source(s). In this way, Data can be sent downstream to one (or more) requester(s).
  - **nfd::Fib**: the Forwarding Information Base (FIB) is used to forward Interest packets toward one (or

TABLE 1: Comparison among the components of ndnSIM 2.0 and ndnSIM 1.0

Component of ndnSIM 2.0	Component existed in ndnSIM 1.0?	Features/Design principles inherited from ndnSIM 1.0	Features/Design changes introduced in ndnSIM 2.0
ndn::L3Protocol	Yes	The core component of ndnSIM	NFD integration
nfd::Forwarder	Existed as ndn::ForwardingStrategy	–	As a result of integration with NFD, packet forwarding is split into forwarding pipelines and forwarding strategy decisions
nfd::Face	Existed as ndn::Face	Base class for ndn::AppFace and ndn::NetDevice-Face	Abstraction implemented by NFD
ndn::AppFace	Yes	Enables communication with applications	Realization of nfd::Face abstraction
ndn::NetDeviceFace	Yes	Enables communication with other simulated nodes	Realization of nfd::Face abstraction
ndn::cs	Yes	Same design	NFD integration
nfd::Cs	Existed as ndn::cs	–	1) Interest selectors handling 2) Not yet flexible to cache policies
nfd::Pit	Existed as ndn::pit	–	Abstraction implemented by NFD
nfd::Fib	Existed as ndn::fib	–	Abstraction implemented by NFD
nfd::fw::Strategy	Existed as ndn::ForwardingStrategy	–	1) Per namespace strategy 2) Different built-in strategies
Applications	Yes	Equivalent functionality	Use of the ndn-cxx library
Trace helpers	Yes	Equivalent functionality	Directly trace events from NFD

more) potential source(s).
  - **nfd::fw::Strategy**: the forwarding strategy in NFD makes the decisions regarding whether, when, and where the Interest packets will be forwarded. nfd::fw::Strategy is an abstract class that needs to be implemented by all the built-in or custom forwarding strategies.
- **ndn::AppFace**: realization of the nfd::Face abstraction to enable communication with applications.
- **ndn::NetDeviceFace**: realization of the nfd::Face abstraction to enable communication with other simulated nodes.
- **Basic NDN applications**: implementation of built-in NDN consumer and producer applications that can generate and sink NDN traffic. These applications include parameters that can be configured by the user in the simulation scenario and thus generate NDN traffic according to a user-defined pattern.
- **Trace helpers**: a collection of trace helpers that simplify collection and aggregation of various necessary statistical information about the simulation and write this information in text files.

2.2 Core NDN protocol

The core component of the ndnSIM architecture is ndn::L3Protocol. Like the previous version of the simulator, this component is an implementation of the NDN protocol stack, which can be installed in each node in a way simiar to the IPv4 or IPv6 protocol stacks. In this version of the simulator, however, when it is installed on a NS-3 node, it performs the initialization of the NFD instance and creates the necessary NFD managers (i.e., FibManager, FaceManager, StrategyChoiceManager), tables (i.e., PIT, FIB, Strategy Choice, Measurements), and special faces (i.e., Null Face, Internal Face). In addition to that, the ndn::L3Protocol class defines a refactored API to handle the registration of new nfd::Face instances to NFD using the AddFace method and enables NDN-level packet tracing.

2.3 Named Data Networking Forwarding Daemon

This is an entirely new component of the ndnSIM architecture. Its main functionality is to forward Interest and Data packets. Towards that goal, NFD abstracts lower-level network transport primitives into nfd::Face instances, maintains the well-designed data structures of CS, PIT and FIB, and implements the packet processing logic. ndnSIM integrates NFD codebase to do all the Interest and Data packet processing actions. In the first half of this section, we describe NFD in detail and, in the second half, we mention the major challenges that we had to address towards the aforementioned integration.

2.3.1 NFD Internal Structure

According to the NFD Developer's Guide [7], the basic modules of NFD are the following:

- **ndn-cxx Library, Core, and Tools**: Provides various common services shared between different NFD modules.
- **Faces**: Implements the NDN Face abstraction on top of various lower level transport mechanisms.
- **Tables**: Implements the Content Store (CS), the Pending Interest Table (PIT), the Forwarding Information Base (FIB), StrategyChoice, Measurements, and other data structures to support forwarding of NDN Data and Interest packets.
- **Forwarding**: Implements basic packet processing pipelines, which interact with Faces, Tables, and Strategies.
- **Management**: Implements the NFD Management Protocol, which allows applications to configure NFD and set/query NFD's internal states.
- **RIB Management**: Manages the routing information base (RIB). This component is not yet supported by ndnSIM.

The packet processing in NFD consists of forwarding pipelines. A **forwarding pipeline** (or just pipeline) is a series of steps that operates on a packet or a PIT entry, which is triggered by the specific event: reception of the Interest, detecting that the received Interest was looped,

when an Interest is ready to be forwarded out of the Face, etc. A **forwarding strategy** (or just strategy) is a decision maker about Interest forwarding, which is attached at the end or beginning of the pipelines. In other words, the strategy makes decisions whether, when, and where to forward an Interest, while the pipelines supply the strategy the Interests and supporting information to make these decisions. Because of the fundamental differences in the processing of Interest and Data packets in NDN (i.e., the one serves as a request, while other satisfies pending requests), forwarding pipelines of NFD are separated into Interest processing path and Data processing path. The concepts of forwarding pipeline and forwarding strategy are described in detail below.

Many aspects of NFD are configurable through a configuration file. Currently, NFD defines 6 top level configuration sections:

- **General**: The general section defines various parameters affecting the overall behavior of NFD.
- **Tables**: The tables section is designated for configuration of NFD tables: Content Store, PIT, FIB, Strategy Choice, and Measurements.
- **Log**: The log section defines the logger configuration.
- **Face system**: The face system section fully controls allowed face protocols, channels and channel creation parameters, and enabling multicast faces.
- **Authorizations**: The authorizations section provides a fine-grained control for management operations.
- **Rib**: The rib section controls behavior and security parameters for NFD RIB manager.

2.3.2 Challenges of NFD integration

Towards this integration, we had to address the following challenges:

- We had to enable the use of simulation time in NFD. Therefore, we took advantage of the CustomClock class provided by the ndn-cxx library in order to convert ndnSIM time to system_clock::time_point and steady_clock::time_point.
- The scheduler of NFD was redirected to ns3::Simulator, so that NFD can schedule events that will be executed by the simulator.
- To optimize the signing process used by NFD for the interaction with its managers through its management protocol, we designed a custom keychain that provides high performance (i.e., minor crypto overhead) during the simulation. However, for simulations that need real crypto operations, the use of a full-featured keychain structure can be selected in the simulation scenario.
- The forwarding pipeline of NFD had to be extended with the beforeSatisfyInterest and beforeExpirePendingInterest signals, so that the tracing of the SatisfiedInterests and TimedOutInterests events is enabled to the simulator.
- We enabled the configurability of NFD parameters internally using specially designed configuration files to avoid the overhead of parsing raw external files and thus optimize the simulation process.

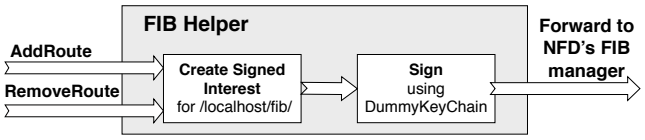

Fig. 2: Operations of the FIB helper

2.3.3 Face abstraction

It is similar to the corresponding abstraction of the previous ndnSIM version. However, in this release, an updated implementation of the Face abstraction is used (i.e., nfd::Face), which contains the required low-level communication primitives to handle Interest and Data packets. As mentioned in [7], these primitives include functions to send an Interest/Data packet and terminate the communication on a Face.

2.3.4 NFD's Content Store

In the NDN communication model, Content Store offers in-network caching for Data packets. Arriving Data packets are placed in the cache as long as possible, so that to satisfy future Interests that would request the same Data. In this way, the protocol performance is enhanced making NDN robust against packet losses and errors and capable of inherent multi-casting.

As with many other forwarding components, this version of ndnSIM uses content store implementation from the NFD codebase. This implementation takes full consideration of Interest selectors, however is not yet flexible when it comes to cache replacement policies. The feature to extend CS flexibility is currently in active development and, for the time being, we have also ported the old ndnSIM 1.0 content store to the new code base, which is discussed in 2.6.

2.3.5 Pending Interest Table (PIT)

In our implementation, the class nfd::Pit of NFD is used as the PIT abstraction. PIT maintains the state for the Interest packets that have been forwarded upstream toward one (or more) potential data source(s) of matching Data. It provides directions for the reverse forwarding of the Data packets toward the data consumer(s). In addition to that, PIT also contains recently satisfied Interest packets for the purposes of loop prevention.

For more information about the PIT structure and the operations performed on it, one can refer to [7].

2.3.6 Forwarding Information Base (FIB)

The class nfd::Fib of NFD is used as the FIB abstraction.This abstraction is used by the forwarding strategies for Interest forwarding toward potential content source(s). For each Interest that needs to be forwarded, a longest prefix match lookup is performed on the FIB.

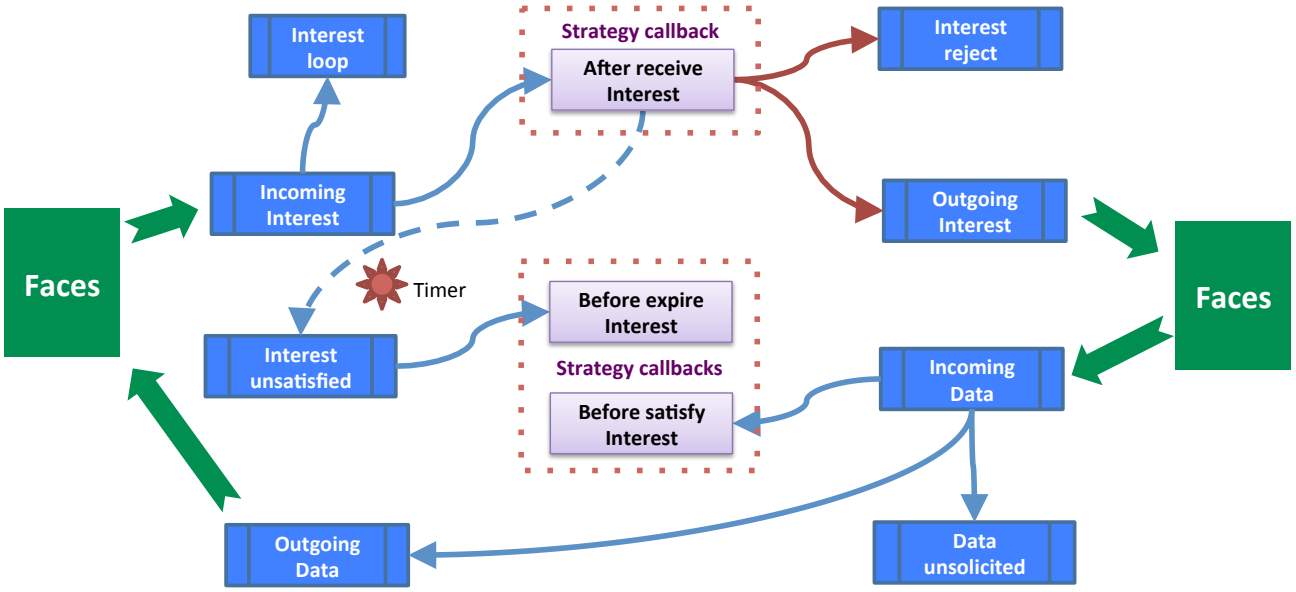

Fig. 3: Overview of ndnSIM/NFD forwarding pipeline

The FIB is updated only through the FIB management protocol, which is operated on the NDN forwarder side by the FIB manager. To simplify common operations, we created a FIB helper that, for the high-level FIB operations, prepares special signed Interest commands and sends them towards the FIB manager. Currently, the FIB helper implements two high-level operations (Figure 2):

- **AddRoute**: Create a new FIB entry, add a route to the FIB entry, or update the cost of the existing record in the FIB entry.
- **RemoveRoute**: Remove a route record from the FIB entry (a FIB entry with empty NextHop records will be automatically deleted).

The Interest commands sent to the FIB manager are signed using the custom key chain mentioned in previous section, which is specially designed to eliminate signing crypto overhead for simulation purposes. If necessary, the full featured crypto support can be re-enabled by switching to the standard KeyChain provided by the ndn-cxx library.

2.3.7 *Forwarding Strategy abstraction*

As mentioned before, the forwarding strategy abstraction of NFD makes the decisions regarding the Interest forwarding. That is to say, whether an Interest would be forwarded or not, the upstream face(s), where it would be forwarded, and when it would be forwarded to the selected upstream face(s). ndnSIM/NFD features an abstract interface (strategy API) that provides the basic implementation of the forwarding strategies without the need of re-implementing the full Interest processing pipeline. An overview of the forwarding pipeline is presented in Figure 3 and is described in detail in the rest of this section.

The implemented forwarding pipeline allow **per-namespace selection of a specific forwarding strategy**. This per-namespace forwarding strategy is registered and maintained at the Strategy Choice table. The Strategy Choice table is updated through the management protocol, operated by the Strategy Choice manager. Similarly to FIB operations, we created a Strategy Choice helper that prepares and sends special signed Interest commands to the manager when strategy selection is requested in the simulation scenario.

The following built-in forwarding strategies are currently available:

- **Broadcast**: Forwards every Interest to all upstream faces.
- **Client Control Strategy**: Allows a local consumer application to choose the outgoing face of each sent Interest packet.
- **Best Route**: Forwards an Interest packet to the upstream face with the lowest routing cost.
- **NCC**: Re-implementation of the CCNx 0.7.2 default strategy.

A new forwarding strategy can implement a completely custom processing or override specific actions in the existing forwarding strategy. The initial step in creating a new strategy is to create a class, say MyStrategy that is derived from the nfd::Strategy class. This subclass must at least override the triggers that are marked as pure virtual and implement them with the desired strategy logic. It may also override any other available triggers that are marked as just virtual.

If the strategy needs to store information, it is needed to decide whether the information is related to a namespace or an Interest. Information related to a namespace but not specific to an Interest should be stored in Measurements entries; information related to an Interest

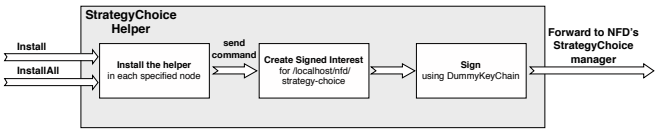

Fig. 4: Operations of the StrategyChoice helper

should be stored in PIT entries, PIT downstream records, or PIT upstream records. After this decision is made, a data structure derived from StrategyInfo class needs to be declared. In the existing implementation, such data structures are declared as nested classes as it provides natural grouping and scope protection of the strategy-specific entity, but it is not required to follow the same model. If timers are needed, EventId fields need to be added to such data structure(s).

The final step is to implement at least the "After Receive Interest" trigger and any (or none) of the three other triggers listed below:

- **After Receive Interest**: When an Interest is received, passes necessary checks, and needs to be forwarded, the Incoming Interest pipeline invokes this trigger with the PIT entry, incoming Interest packet, and FIB entry.
- **Before Satisfy Interest**: When a PIT entry is satisfied, before Data is sent to downstream faces (if any), the Incoming Data pipeline invokes this trigger with the PIT entry, the Data packet, and its incoming face.
- **Before Expire Interest**: When a PIT entry expires because it has not been satisfied before all in-records expire, before it is deleted, Interest Unsatisfied pipeline invokes this trigger with the PIT entry.

Actions are the forwarding decisions made by each forwarding strategy and are implemented as non-virtual protected methods of the nfd::Strategy class. The provided actions are listed below:

- **Send Interest**: It triggers when entering the Outgoing Interest pipeline.
- **Reject Pending Interest**: It triggers when entering the Interest reject pipeline.

To simplify the operations of specifying the desired per-name prefix forwarding strategy for one, more or all the topology nodes, we provide a Strategy Choice helper that interacts with the Strategy Choice manager of NFD by sending special signed Interest commands to the manager. The operations of this helper are illustrated in Figure 4

2.4 Application Face

This class enables the communication of the simulated applications with the NDN network. Specifically, this abstraction provides functions for sending interests and data packets as well as overloads of the sendInterest and sendData methods to receive packets from the NDN stack. We should note that the wording "send" refers to packets that are sent from the NDN stack and thus are received from the application.

2.5 Network Device Face

This component enables the communication between the simulated nodes. Each ndn::NetDeviceFace instance is permanently associated with a NetDevice object and this object cannot be changed for the lifetime of this face. For sending packets between simulated nodes, Interest and Data packets are converted into the NDN packet format, using routines of the ndn-cxx library, and then are encapsulated to a packet instance of NS-3.

2.6 "Old" Content Store

As mentioned above, because of the fact that NFD's Content Store is not yet flexible when it comes to cache replacement policies, we have also ported the old ndnSIM 1.0 content store implementations to the new code base (Table 2). These implementations feature different cache replacement policies, but have limited support for Interest selectors.

TABLE 2: "Old" Content Store Implementations

Simple content stores	
cs::Lru	Least recently used (LRU) (default)
cs::Fifo	First-in-first-Out (FIFO)
cs::Lfu	Least frequently used (LFU)
cs::Random	Random
cs::Nocache	Policy that completely disables caching
Content stores with entry lifetime tracking These policies allow the evaluation of CS enties lifetime (i.e., how long entries stay in CS)	
cs::Stats::Lru	Least recently used (LRU)
cs::Stats::Fifo	Least frequently used (LFU)
cs::Stats::Lfu	Random
cs::Stats::Random	Policy that completely disables caching
Content stores respecting freshness field of Data packets These policies cache Data packets only for the time indicated by FreshnessPeriod.	
cs::Freshness::Lru	Least recently used (LRU)
cs::Freshness::Fifo	Least frequently used (LFU)
cs::Freshness::Lfu	Random
cs::Freshness::Random	Policy that completely disables caching
Content store realization that probabilistically accepts data packet into CS (placement policy) These policies cache Data packets only for the time indicated by FreshnessPeriod.	
cs::Probability::Lru	Least recently used (LRU)
cs::Probability::Fifo	Least frequently used (LFU)
cs::Probability::Lfu	Random
cs::Probability::Random	Policy that completely disables caching

2.7 Basic NDN applications

The basic applications included in the current ndnSIM release are the same applications that have been included in its previous release with minor changes due to the introduction of the ndn-cxx library:

- **ConsumerCbr**: a consumer application that generates Interest traffic according to a user-defined pattern (e.g., predefined frequency, constant rate, constant average rate with inter-Interest gap distributed uniformly at random, exponentially at random, etc.). A user-defined Interest name prefix and sequence number are available. Moreover, this application

provides Interest retransmission according to an RTT-based timeout period similar to the TCP RTO.
- **ConsumerBatches**: a consumer application that generates a specified number of Interest packets at specified time points of the simulation.
- **ConsumerWindow**: a consumer application that generates Interest traffic of variable rate. It implements a simple sliding-window-based Interest generation mechanism.
- **ConsumerZipfMandelbrot**: a consumer application that requests contents (i.e., names in the requests) following the Zipf-Mandelbrot distribution.
- **Producer**: a simple application that sinks Interest traffic and generates Data traffic. Specifically, it responds to each incoming Interest packet with a Data packet that has the same size and name as the corresponding incoming Interest packet.

The interaction of the applications with the core of the simulator is achieved using the ndn::AppFace realization of the nfd::Face abstraction. The base class ndn::App is responsible for the creation/deletion of the ndn::AppFace instances and their registration in the protocol stack.

2.8 Trace helpers

The trace helpers simplify the collection and aggregation of various necessary statistical information about the simulation and write this information in text files. In our implementation, the capability of tracing events directly from NFD has been added to the tracers. There are three sorts of such helpers:

- **Packet-level trace helpers**: This group includes **L3RateTracer** and **L2Tracer**. The former traces the rate in bytes and in number of packets of Interest/Data packets forwarded by an NDN node, while the latter traces only packets that are dropped on layer 2 (e.g., due to a transmission queue overflow).
- **Content store trace helper**: With the use of CsTracer, it is possible to obtain statistics of cache hits/misses on the Content Store of the simulated nodes.
- **Application-level trace helper**: With the use of AppDelayTracer, it is possible to obtain data regarding delays between issuing Interest packets and receiving the corresponding Data packet.

3 Limitations of current version and future plan

Our ultimate goal for the simulator is to provide an API, which will be used by researchers in order to port and simulate any real NDN application in a convenient and handy way. However, this feature is not still available. In order to accomplish this goal the ApiFace has to be replaced with the emulation of ndn-cxx ndn::Face, which is used by real NDN applications. In addition to that, ndnSIM 2.0 does not support the use of NFD's RIB manager. This is also one of the features that we are planning to provide in the near future. The provided FibHelper also does not implement some function for route removal from the FIB of NFD, but only for route creation/update, which is something that we are planning to support in the near future. Last but not least, the TcpFace and UpdFace provided by the previous simulator version have been temporarily disabled, but we are planning to enable them soon.

4 Related Work

Within the recent years, the interest for NDN research has grown. As a consequence, the development of common and handy ways for the evaluation of the proposed NDN research approaches has been absolutely necessary.

One of the first and popular approaches towards that goal is the previous version of ndnSIM [6]. This version, exactly like the current one, is implemented in a modular way and was optimized for simulation purposes. However, the previous version included an independent implementation of NDN packet forwarding and used a deprecated NDN packet format. Moreover, the ndnSIM 1.0 never implemented the full-featured processing of NDN selectors and, as a result, has limitations regarding the accuracy of the simulation results.

Another existing effort is presented by Chioccheti et al. [10, 11]. ccnSim is a scalable chunk-level simulator suitable for the analysis of caching performance of NDN networks. It is developed using the OMNeT++ framework in C++. However, it is mainly optimized for the experimentation on various cache replacement policies for NDN routers and does not provide any flexibility of the forwarding process. As a result, ccnSim cannot be used for the experimentation on a vital core component of the NDN architecture, which is the forwarding strategy layer.

CCN-lite [12] is a lightweight implementation of the CCNx-NDNx protocol. It offers a simulation mode using the OMNeT++ simulation platform. CCN-lite supports scheduling, both at chunk and at packet level, and packet fragmentation. It also supports possible native deployment without any IP layer. However, this effort is mainly intended to run on resource constrained devices and is not optimized to offer high performance as its data structures rely on linked lists.

The Content Centric Networking Packet Level Simulator (CCNPL-Sim) [13] is another NDN simulator developed at Orange Labs. CCNPL-Sim makes use of the Combined Broadcast and Content-Based routing scheme (CBCB) [14] implementation within the SSim simulator to handle event management and name based forwarding and routing. Despite the effectiveness of the SSim simulation scheduler, the mandatory usage of CCNB makes the evaluation of other routing protocols, such as OSPFN [15] and NLSR [16], impossible thus limiting the experimental scope of this simulator.

Another recently introduced effort is Mini-CCNx [17]. Mini-CCNx is a fork of Mininet-HiFi specially customized to support the emulation of CCNx-NDNx

nodes. Its main goal is to add a realistic behavior to the executed tests. Mini-CCNx offers flexibility, because of the Container-Based Emulation features of Mininet, and a simple configuration GUI interface. However, it is based on the packet format of NDNx, which is an outdated version of the NDN communication model. It also mainly focuses on emulating the node hardware instead of the communication model itself.

The affluence of computing resources across the existing research network testbeds/infrastructures (e.g., GENI [18], Open Network Lab (ONL) [19], Emulab [20], etc.) also offers a valuable option for the conduction of real-time research experiments. These testbeds provide both the hardware and the software systems needed by researchers to evaluate their design. However, the complexity that is introduced in order to configure and manage all the delegated resources along with the limited experimental scale are two crucial reasons that lead researchers to resort to simulations.

5 Summary

In this release of the simulator, we have focused our efforts on providing a more realistic simulation behavior by integrating the Named Data Networking Forwarding Daemon (NFD) with ndnSIM and using directly the ndn-cxx library and the latest NDN packet format. ndnSIM provides the framework for large-scale experimentation, while its modular design offers the flexibility to the researchers to modify its components with minimal, if any, changes to other parts of its implementation. Detailed information about the current release and additional documentation is available on the ndnSIM webstite: http://ndnsim.net.

We really hope that the NDN community will find ndnSIM a valuable tool and we are looking forward to receiving the community's priceless feedback in order to further improve the simulator.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of ACM CoNEXT*, 2009.

[2] L. Zhang et al., "Named data networking (NDN) project 2010 - 2011 progress summary," PARC, http://www.named-data.net/ndn-ar2011.html, Tech. Rep., November 2011.

[3] L. Zhang et al., "Named data networking (NDN) project," PARC, Tech. Rep. NDN-0001, October 2010.

[4] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, July 2014.

[5] (2012, May) ns-3. [Online]. Available: http://www.nsnam.org/

[6] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," NDN, Technical Report NDN-0005, October 2012. [Online]. Available: http://named-data.net/techreports.html

[7] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto, C. Fan, C. Papadopoulos, D. Pesavento, G. Grassi, G. Pau, H. Zhang, T. Song, H. Yuan, H. B. Abraham, P. Crowley, S. O. Amin, V. Lehman, , and L. Wang, "NFD developers guide," NDN Project, Tech. Rep. NDN-0021, July 2014.

[8] NDN Project, "NFD - named data networking forwarding daemon," Online: http://named-data.net/doc/NFD/0.2.0/, 2014.

[9] ——, "NDN Packet Format Specification," Online: http://named-data.net/doc/ndn-tlv/, 2014.

[10] D. Rossi, G. Rossini, "Caching performance of content centric networksunder multi-path routing (and more)," Telecom ParisTech, Tech. Rep., 2011.

[11] G. Rossini and D. Rossi, "ccnSim: an highly scalable CCN simulator," in *IEEE ICC*, 2013.

[12] C. Scherb, M. Sifalakis, and C. Tschudin, "CCN-lite," Available: http://www.ccn-lite.net, 2013.

[13] L. Muscariello. (2011) Content centric networking packet level simulator. Orange Labs. [Online]. Available: http://perso.rd.francetelecom.fr/muscariello/sim.html

[14] A. Carzaniga, M.J. Rutherford, and A.L. Wolf, Ed., *A Routing Scheme for Content-Based Networking*. IEEE INFOCOM, March 2004.

[15] L. Wang, A. Hoque, C. Yi, A. Alyyan, and B. Zhang, "OSPFN: An OSPF based routing protocol for Named Data Networking," NDN, Tech. Rep NDN-0003, 2012.

[16] NDN Project, "NLSR - Named Data Link State Routing Protocol," Online: http://named-data.net/doc/NLSR/0.1.0/, 2014.

[17] C. Cabral, C. E. Rothenberg, and M. F. Magalhães, "Reproducing real NDN experiments using mini-CCNx," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, 2013.

[18] (2015, January) GENI (Global Environment for Network Innovations). [Online]. Available: http://www.geni.net

[19] (2015, January) Open Networking Lab. [Online]. Available: http://onlab.us

[20] (2015, January) Emulab - Network Emulation Testbed. [Online]. Available: http://www.emulab.net