ndnSIM: a modular NDN simulator

http://ndnsim.net

ALEX AFANASYEV
ILYA MOISEENKO
LIXIA ZHANG

University of California, Los Angeles, Computer Science Department
Goals

• Simulate basic NDN operations
• Packet-level interoperability with CCNx implementation
• Modular architecture
  – C++ classes for every NDN component
    • Face, PIT, FIB, Content store, and Forwarding strategy
• Scenario-defined module selection
  – Different management schemes for PIT
  – Different replacement policies for content store
  – Different forwarding strategies
• Ease of extensions
• Ease of use: plug and experiment
Ultimate Goal

- Establishing a *common* platform to be used by the community for all CCN/NDN simulation experimentation
  - So that people can compare/replicate results
Basic network simulation model in NS-3

Diagram showing the relationship between Application, Protocol stack, NetDevice, Channel, and Packet(s) with a Sockets-like API.
ndnSIM extension of network simulation model
Node structure overview

- Abstract interfaces of content store, PIT, FIB, and forwarding strategy.
- Each simulation run chooses specific scheme for each module.
Core NDN protocol (ndn::L3Protocol)

- aggregates and manages all communication channels (Faces)
  - adding faces and registers necessary callbacks
  - removing faces
- receives packets from Faces and direct them to a scenario-selected forwarding strategy
Faces (ndn::Face)

- Abstraction from underlying protocols
  - callback registration-deregistration
  - packet encapsulation

![Diagram]

- ndn::App
- ndn::AppFace
- ndn::L3Protocol
  - ndn::UdpFace
  - ndn::TcpFace
- ndn::NetDeviceFace
  - ndn::Ipv4Face
  - Network layer (IPv4, IPv6)
  - Link layer (PPP, 802.11, etc.)

Not yet implemented
Can be done quickly if/once the need identified
Content Store

• In-network cache abstraction
  – add item
  – lookup item

• Currently available implementations of replacement policies
  – (default) Least-recently used (ns3::ndn::cs::LRU)
  – First-in-first-out (ns3::ndn::cs::FIFO)
  – Random (ns3::ndn::cs::Random)

• A desired content store module is selected and configured in simulation scenario

```cpp
ndn::StackHelper ndnHelper;
ndnHelper.SetContentStore("ns3::ndn::cs::LRU", "MaxSize", "100");
```
Pending Interest Table (PIT)

- Abstraction to maintain state for each forwarded Interest packet
  - Create, Lookup, Erase entry

- Each PIT entry stores
  - Interest packet itself
  - list of incoming faces + associated info
  - list of outgoing faces + associated info
  - forwarding strategy tags
    - e.g., reference to a delayed processing queue

- Size of PIT can be limited in simulation scenario
  - Available policies for new PIT entry creation:
    - (default) persistent (ns3::ndn::pit::Persistent): a new entry will not be created if limit is reached
    - LRU (ns3::ndn::pit::LRU): when limit is reached, insertion of a new entry will evict the oldest entry
    - Random (ns3::ndn::pit::Random): when limit is reached, insertion will evict a random entry
Forwarding Information Base (FIB)

- Abstraction to store information about name prefixes
  - Add, Remove, LongestPrefixMatch

- Every FIB entry stores
  - prefix
  - list of (ranked) Faces
  - forwarding strategy tags
    - per-prefix limits, data-plane stats, etc.

- FIB, PIT, and Content Store implemented as a trie-like structure
  - every name component is a node in a tree
  - node’s children organized in a hash map
  - leafs contain pointers to FIB/PIT/CS entries
FIB population

• Manually

• Default route
  – all interfaces added to default route
  – forwarding strategy make a choice

• Global routing controller
  – calculate SPF
  – install a best-route for prefix

• May add support for quagga-based population
  – rely on Direct Code Execution NS-3 module
  – use real routing protocol implementations (e.g. NDN prefixes distribution by OSPF-N)
Forwarding strategies

- Abstraction for Interest and Data processing
  - OnInterest, OnData, WillErasePendingInterest, RemoveFace, FailedToCreatePitEntry, DidCreatePitEntry, WillSatisfyPendingInterest, and many other overrideable events

- Extensions
  - NACKs
  - Data plane status performance

- Available strategies
  - Flooding strategy
  - Smart flooding strategy
  - Best-Route strategy

- Several other forwarding strategies under development right now
Simulator usage by early adopters & ourselves

• Forwarding strategy experimentation
  – behavior in the presence of
    • link failures
    • prefix black-holing
    • congestion
  – resiliency of NDN to DDoS attacks (interest flooding)

• Content-store evaluation
  – evaluation different replacement policies

• NDN for car2car communication
  – Evaluations of traffic info propagation protocols

• Exploration of SYNC protocol design
  – Experimentation of multiuser chat application whose
design is based on SYNC (chronos)
NDN experimental extensions

- Interest NACKs to enable more intelligent, adaptive forwarding
- Congestion control by Limiting the number of pending Interests
  - per-face
  - per-FIB-entry
  - per-FIB-entry-per-face
- Satisfaction ratio statistics module
  - per-face (incoming/outgoing)
  - per-prefix
  - configurable time granularities
- A initial set of simple application modules
Interest NACK

- Solves dangling state problem
  - when router cannot satisfy nor forward, it sends Interest NACK
  - removes PIT entry

- Signals downstream to action
  - explore other paths to find destination
  - avoid congested paths

- Details
  - NACK code added to Interest
  - Interest NACK carries the same nonce
    - basic protection against spoofing
  - NACK codes
    - Duplicate
    - No data/no prefix
    - Congestion
    - ...
Limits on number of pending Interests

• Limit based on bandwidth-delay product
  – Assuming a know size (average) of interest packets
  – \# interests = capacity / (AvgDataSize + AvgInterestSize)

• Different granularities
  – per face (incoming/outgoing), per prefix (FIB/PIT)

• Pending Interest removed by
  – received Data
  – timeout
  – (optionally) NACK

• Features
  – prevents congestion
  – provides base for DDoS protection mechanisms
  – may result in link underutilization
An initial set of applications

- **ndn::ConsumerCbr**
  - generates Interest traffic with predefined frequency

- **ndn::ConsumerBatches**
  - generates a specified number of Interests at specified points of simulation

- **ndn::Producer**
  - Interest-sink application, which replies every incoming Interest with Data packet
Scalability numbers

- Memory overhead (on average)
  - per simulation node
    - Node without any stacks installed: **0.4 Kb**
    - Node with ndnSIM stack (empty caches and empty PIT): **1.6 Kb**
    - For reference: Node with IP (IPv4 + IPv6) stack: **5.0 Kb**
  - per PIT entry: **1.0 Kb**
  - per CS entry: **0.8 Kb**

- Processing speed: on single core 2.4 Ghz cpu
  - ~50,000 Interests per wall clock second
  - ~35,000 Interests + Data combined per wall clock second

- MPI support of NS-3
  - manual network partitioning
  - close to linear scaling with number of cores with good partitioning
Other CCN Simulators

- **ccnSim**
  - primarily focused on cache behavior research
  - smaller memory footprint
    - more abstractions and simplifications
    - simplified Interest/Data packet formats (e.g., names restricted to number vectors?)
  - Not very modular for easy extension

- **CCNPL-Sim**
  - based on custom discrete event simulator (SSim)
  - limited flexibility for extensions
    - needs a content routing scheme as inter-layer between SSim and CCNPL-Sim?
    - How to use this for forwarding strategy experimentation?

- **NS-3 Direct Code Execution + ccnd**
  - most realistic evaluation of the prototype implementation
  - high per-node overhead
  - Difficult to experiment with different design choices
    - need to be implemented in real code first
Try out ndnSIM and let us know your thought/comments/bug reports/new feature requests!

http://ndnsim.net