UCLA ENGINEERING
Computer Science

Asia Future Internet AsiaFI

ndnSIM: a modular NDN simulator

Introduction and Tutorial

http://ndnsim.net
ndnsim@lists.cs.ucla.edu

ALEX AFANASYEV
alexander.afanasyev@ucla.edu

University of California, Los Angeles, Computer Science Department

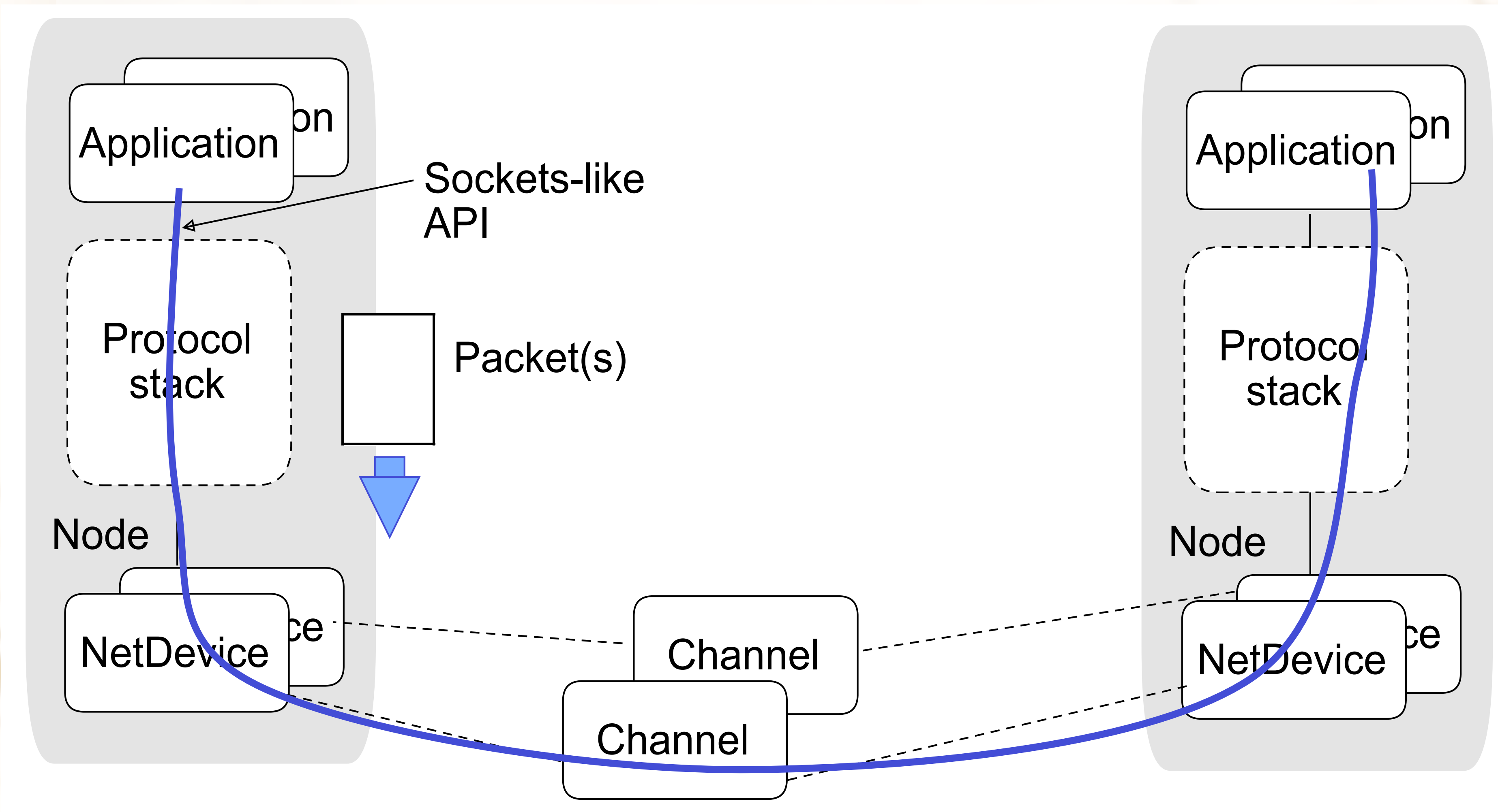
Introduction

- ndnSIM implements all basic NDN operations
- Has an option for packet-level interoperability with CCNx implementation
- Has modular architecture
  - C++ classes for every NDN component
    - Face, PIT, FIB, Content store, and Forwarding strategey
- Allows combining different implementations of core NDN components
  - Different management schemes for PIT
  - Different replacement policies for content store
  - Different forwarding strategies
- Can be easily extended
- Easy to use: plug in and experiment

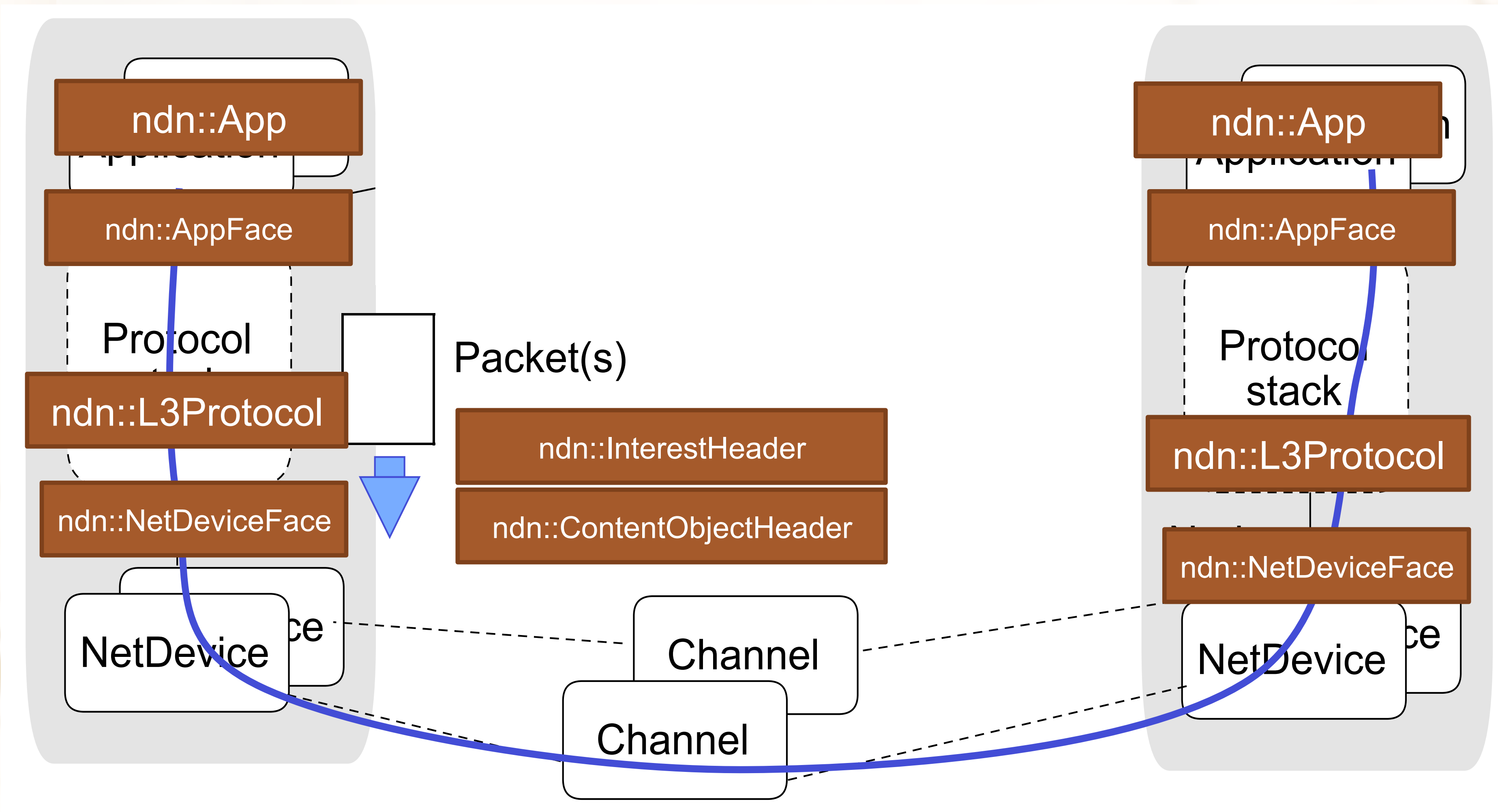
Ultimate Goal

- Establishing a *common* platform to be used by the community for all CCN/NDN simulation experimentations
  - So that people can compare/replicate results

Basic network simulation model in NS-3

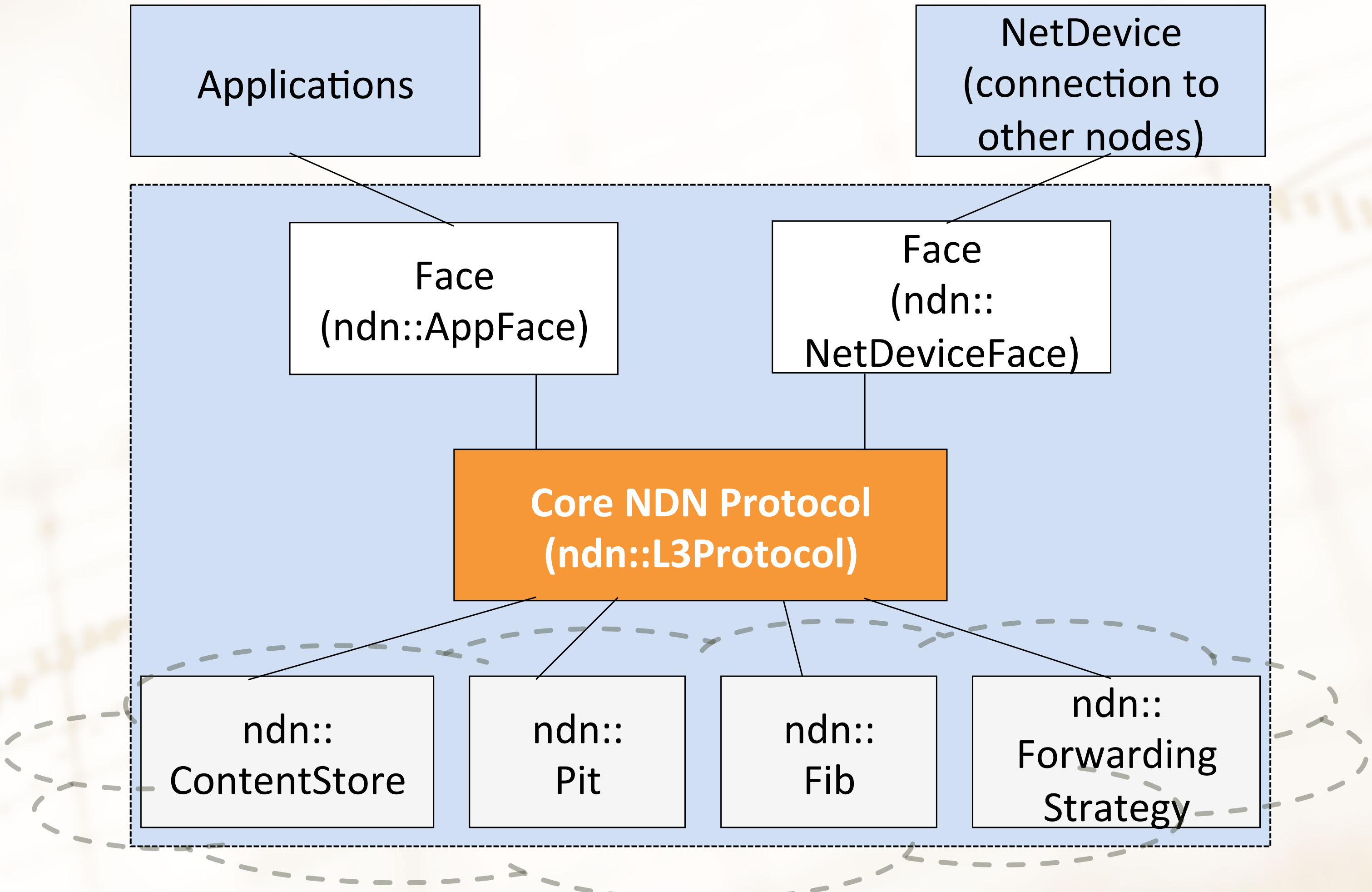


ndnSIM extension of network simulation model



University of California, Los Angeles, Computer Science Department

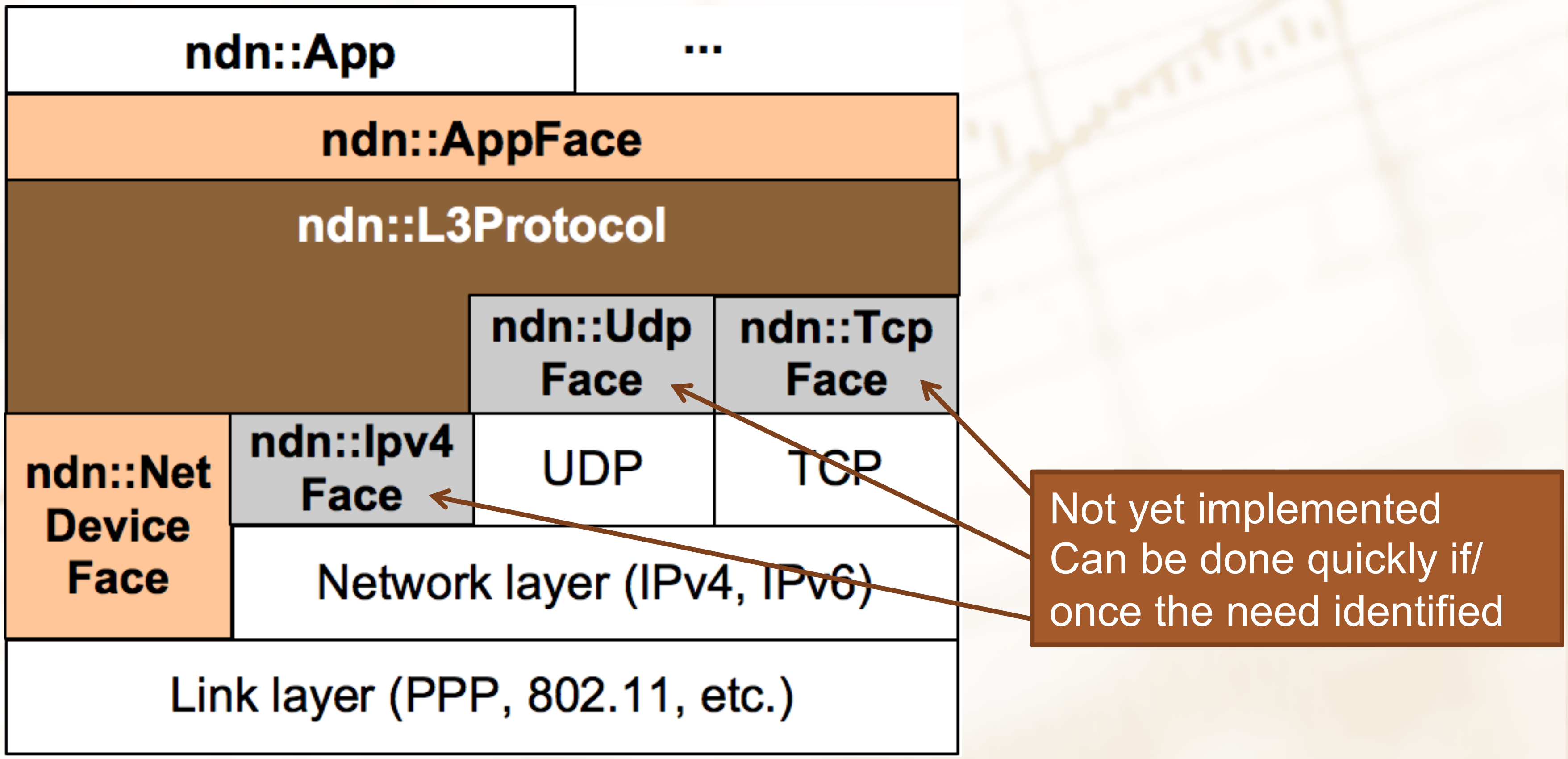
ndnSIM structure overview



- Abstract interfaces of content store, PIT, FIB, and forwarding strategy.
- Each simulation run chooses specific scheme for each module

Faces (ndn::Face)

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



ndnSIM 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)

Some 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: c

  ~50,000 Interests per w

  ~35,000 Interests + Da

Can be optimized by utilizing a simplified packet encoding.
Next release of ndnSIM will have option to choose between ccnx compatibility and processing efficiency

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

Getting started

- http://ndnsim.net/getting-started.html
- Works in OSX, Linux, FreeBSD
  - requires boost libraries > 1.48
  - visualizer module need python and various python bindings
- Download
  - mkdir ndnSIM
  - cd ndnSIM
  - git clone git://github.com/cawka/ns-3-dev-ndnSIM.git ns-3
  - git clone git://github.com/cawka/pybindgen.git pybindgen
  - git clone git://github.com/NDN-Routing/ndnSIM.git ns-3/src/ndnSIM
- Build
  - ./waf configure --enable-examples
- Run examples
  - ./waf --run=ndn-grid
  - ./waf --run=ndn-grid --vis
  - other examples: http://ndnsim.net/examples.html

General use of ndnSIM

- Define topology
  - Manually
  - Using various readers (http://ndnsim.net/examples.html#node-grid-example-using-topology-plugin)
- Create ndn::StackHelper
  - Define ContentStore size and policy
    - **ns3::ndn::cs::Lru (default size 100)**, ... Fifo, ... Random
    - ns3::ndn::cs::Stats::Lru, ...Fifo, ...Random
    - ns3::ndn::cs::Freshness::Lru, ...Fifo, ...Random
  - Define Forwarding Strategy
    - **ns3::ndn::fw::Flooding (default)**, ...SmartFlooding, ...BestRoute
- Set up routes between nodes
  - manually
  - semi-automatic
- Define and assign applications
- Collect metrics

Forwarding strategies

- Abstraction control all aspect of Interest and Data packet forwarding
  - specify where to forward Interest packets
  - track data plane performance for Data packets
- Available strategies
  - Flooding strategy (default)
    - Interests will be forwarded to all available faces available for a route (FIB entry). If there are no available GREEN or YELLOW faces, interests is dropped.
  - Smart flooding strategy
    - If GREEN face is available, Interest will be sent to the highest-ranked GREEN face. If not, Interest will be forwarded to all available faces available for a route (FIB entry)
  - Best-Route strategy
    - If GREEN face is available, Interest will be sent to the highest-ranked GREEN face. If not, Interest will be forwarded to the highest-ranked YELLOW face.
- Easy to write your own strategy or redefine aspects of the existing ones

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
- Other methods to be added later
  - Direct Code Execution based methods
    - quagga
    - ospfn

An initial set of applications

- http://ndnsim.net/applications.html

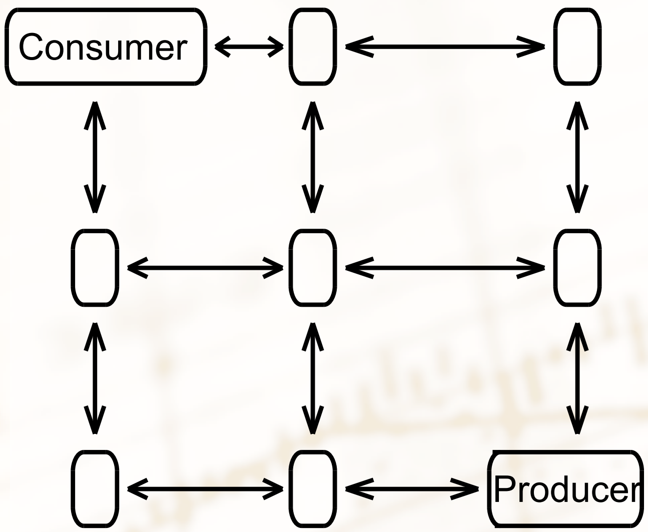
- **ndn::ConsumerCbr**
  - generates Interest traffic with predefined frequency
- **ndn::ConsumerBatches**
  - generates a specified number of Interests at specified points of simulation
- **ndn::ConsumerZipfMandelbrot**
  - (thanks to Xiaoke Jiang) requests contents (names in the requests) following Zipf-Mandelbrot distribution (number of Content frequency Distribution)
- **ndn::Producer**
  - Interest-sink application, which replies every incoming Interest with Data packet

University of California, Los Angeles, Computer Science Department

Metrics

- Packet-level trace helpers
  - L3AggregateTracer
    - track aggregate number of forwarded packets
  - L3RateTracer
    - track rate of forwarded packets
- Content store trace helper
  - CsImpTracer
    - track cache hits and cache misses
- More info: http://ndnsim.net/metric.html

Tutorial by an example



10 Mbps / 10 ms delay

- http://ndnsim.net/examples.html#node-grid-example
- Simple simulation
  - 3x3 grid topology
  - 10Mbps links / 10ms delays
  - One consumer, one producer

NS-3 101: Prepare scenario (C++)

Step 0. Create scenario.cc and place it in <ns-3>/scratch/

Step 1. Include necessary modules	#include "core-module.h" #include "ns3/n #include "ns3/etwork-module.h" #include "ns3/point-to-point-module.h" #include "ns3/point-to-point-grid.h" #include "ns3/ndnSIM-module.h" using namespace ns3;
Step 2. Define **main** function like in any other C++ program	int main (int argc, char *argv[]) {
Step 3. Set default parameters for the simulator modules. For example, define that by default all created p2p links will have 10Mbps bandwidth, 10ms delay and DropTailQueue with 20 packets	Config::SetDefault ("ns3::PointToPointNetDevice::DataRate", StringValue ("10Mbps")); Config::SetDefault ("ns3::PointToPointChannel::Delay", StringValue ("10ms")); Config::SetDefault ("ns3::DropTailQueue::MaxPackets", StringValue ("20"));
Step 4. Allow overriding defaults from command line	CommandLine cmd; cmd.Parse (argc, argv);
Step 5. Define what topology will be simulated. For example, 3x3 grid topology	PointToPointHelper p2p; PointToPointGridHelper grid (3, 3, p2p); grid.BoundingBox(100,100,200,200);
Step 6. Create and install networking stacks, install and schedule applications, define metric logging, etc.	// scenario meat
Step 7. Define when simulation should be stopped	Simulator::Stop (Seconds (20.0));
Final step. Run simulation	Simulator::Run (); Simulator::Destroy (); return 0; }

The same scenario can be also written in Python

C++

```cpp
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/point-to-point-grid.h"
#include "ns3/ndnSIM-module.h”
using namespace ns3;

int
main (int argc, char *argv[])
{
  Config::SetDefault ("ns3::PointToPointNetDevice::DataRate",
StringValue ("10Mbps"));
  Config::SetDefault ("ns3::PointToPointChannel::Delay",
StringValue ("10ms"));
  Config::SetDefault ("ns3::DropTailQueue::MaxPackets",
StringValue ("20"));

  CommandLine cmd; cmd.Parse (argc, argv);

  PointToPointHelper p2p;
  PointToPointGridHelper grid (3, 3, p2p);
  grid.BoundingBox(100,100,200,200);


  // scenario meat


  Simulator::Stop (Seconds (20.0));

  Simulator::Run ();
  Simulator::Destroy ();

  return 0;
}
```

Python

```python
from ns.core import *
from ns.network import *
from ns.point_to_point import *
from ns.point_to_point_layout import *
from ns.ndnSIM import *

Config.SetDefault ("ns3::PointToPointNetDevice::DataRate",
StringValue ("10Mbps"))
Config.SetDefault ("ns3::PointToPointChannel::Delay",
StringValue ("10ms"))
Config.SetDefault ("ns3::DropTailQueue::MaxPackets",
StringValue ("20"))

import sys; cmd = CommandLine (); cmd.Parse (sys.argv);

p2p = PointToPointHelper ()
grid = PointToPointGridHelper (3,3,p2p)
grid.BoundingBox(100,100,200,200)


# scenario meat


Simulator.Stop (Seconds (20.0))

Simulator.Run ()
Simulator.Destroy ()
```

Defining scenario in Python is easier and don’t require (re)compilation, but not all features of NS-3 and ndnSIM are available in Python interface.   The rest of the tutorial is only C++

ndnSIM 101: filling scenario meat

Step 1. Install NDN stack on all nodes (like starting ccnd on a computer)

```
ndn::StackHelper ndnHelper;
ndnHelper.InstallAll ();
```

Step 2. Define which nodes will run applications

```
// Getting containers for the consumer/producer
Ptr<Node> producer = grid.GetNode (2, 2);
NodeContainer consumerNodes;
consumerNodes.Add (grid.GetNode (0,0));
```

Step 3. “Install” applications on nodes

```
ndn::AppHelper cHelper ("ns3::ndn::ConsumerCbr");
cHelper .SetPrefix ("/prefix");
cHelper .SetAttribute ("Frequency", StringValue
("10"));
cHelper .Install (consumerNodes);

ndn::AppHelper pHelper ("ns3::ndn::Producer");
pHelper.SetPrefix ("/prefix");
pHelper.SetAttribute ("PayloadSize",
StringValue("1024"));
pHelper.Install (producer);
```

Step 2. Configure FIB

- manually
- using global routing controller (shown here)

```
ndn::GlobalRoutingHelper ndnGlobalRoutingHelper;
ndnGlobalRoutingHelper.InstallAll ();


// Add /prefix origins to ndn::GlobalRouter
ndnGlobalRoutingHelper.AddOrigins (“/prefix”,
producer);

// Calculate and install FIBs
ndnGlobalRoutingHelper.CalculateRoutes ();
```

Running the simulation (C++)

Option A: like any other program:

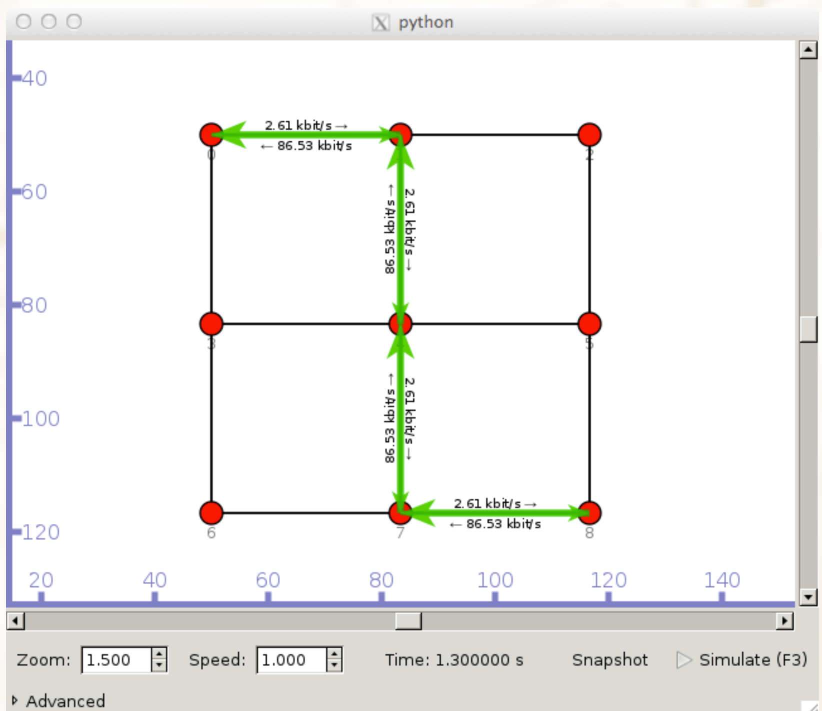
<ns-3>/build/scratch/scenario

Option B: using ./waf helper:

cd <ns-3>; ./waf --run=scenario

Option C: using ./waf helper using visualizer:

./waf --run=scenario --visualize



Result if you followed the steps

Same example is on http://ndnsim.net

Details on how to customize the scenario

- Select different forwarding strategy, configure cache (size, replacement policy):
  - ndnHelper.SetForwardingStrategy ("ns3::ndn::fw::Flooding")
    - "ns3::ndn::fw::Flooding", "ns3::ndn::fw::BestRoute" or your own
  - ndnHelper.SetContentStore ("ns3::ndn::cs::Lru", "MaxSize", "100")
    - "ns3::ndn::cs::Lru", "ns3::ndn::cs::Random", "ns3::ndn::cs::Fifo"
  - ndnHelper.SetPit ("ns3::ndn::pit::Persistent", "MaxSize", "1000")
    - "ns3::ndn::pit::Persistent", "ns3::ndn::pit::Random"

Write your own application (requester)

more http://ndnsim.net/applications.html#custom-applications

Step 1. Create a normal C++ class and derive it from ndn::App

Step 2. Define GetTypeId () function (use templates!)
Needed for NS-3 object system

Step 3. Define actions upon start and stop of the application

Step 4. Implement **OnContentObject** method to process requested data:

```
virtual void
OnContentObject (const Ptr<const
ContentObjectHeader> &contentObject,
Ptr<Packet> payload);
```

```
...
class RequesterApp : public App
{
public:
  static TypeId GetTypeId ();

  RequesterApp ();
  virtual ~RequesterApp ();

protected:
  // from App
  virtual void
  StartApplication ()
  {
    App::StartApplication ();
   // send packet for example
  }

  virtual void
  StopApplication ()
  {
    // do cleanup
    App::StopApplication ();
  }
};
```

Write your own application (producer)

Step 0. Do everything as for the requester app

Step 1. Register prefix in FIB (= set Interest filter) in StartApplication

Step 2. Implement **OnInterest** to process incoming interests

```
virtual void
OnInterest (const Ptr<const InterestHeader>
&interest, Ptr<Packet> packet);
```

```
void StartApplication ()
{
...
  Ptr<Fib> fib = GetNode ()->GetObject<Fib> ();
  Ptr<fib::Entry> fibEntry = fib->Add (m_prefix,
m_face, 0);

  fibEntry->UpdateStatus (m_face,
fib::FaceMetric::NDN_FIB_GREEN);
}
```

Write your own forwarding strategy

Step 1. Create a standard C++ class and derive it from ndn::ForwardingStrategy, one of the extensions, or one of the existing strategies

Step 2. Extend or re-implement available forwarding strategy events (for the full list refer to http://ndnsim.net/doxygen/):

- OnInterest
- OnData
- WillEraseTimedOutPendingInterest
- RemoveFace
- DidReceiveDuplicateInterest
- DidExhaustForwardingOptions
- FailedToCreatePitEntry
- DidCreatePitEntry
- DetectRetransmittedInterest
- WillSatisfyPendingInterest
- SatisfyPendingInterest
- DidSendOutData
- DidReceiveUnsolicitedData
- ShouldSuppressIncomingInterest
- TrySendOutInterest
- DidSendOutInterest
- PropagateInterest
- DoPropagateInterest

```
/**
 * \ingroup ndn
 * \brief Strategy implementing per-FIB entry limits
 */
class SimpleLimits :
    public BestRoute
{
private:
  typedef BestRoute super;

public:
  static TypeId
  GetTypeId ();

  SimpleLimits ();

  virtual void
  WillEraseTimedOutPendingInterest ...

protected:
  virtual bool
  TrySendOutInterest ...

  virtual void
  WillSatisfyPendingInterest ...

private:
  // from Object
  virtual void
  NotifyNewAggregate (); ///< @brief Even when object is
aggregated to another Object

  virtual void
  DoDispose ();
};
```

Write your own cache replacement policy

- Option A:
  - create a class derived from ndn::ContentStore, implementing all interface functions
- Option B:
  - use C++ templates of ndnSIM
    - define “policy traits” (example utils/trie/lru-policy)
      - defines what to do
        - » on insert (e.g., put in front)
        - » on update (e.g., promote to front)
        - » on delete (e.g., remove)
        - » on lookup (e.g., promote to front)
    - instantiate cache class with new policy:
      - template class ContentStoreImpl<lru_policy_traits>;
    - see examples in model/cs/content-store-impl.cc

Try out ndnSIM and let us know your thought/comments/bug reports/new feature requests !

http://ndnsim.net