ns-3 Training

Emulation
Outline

• Main emulation devices
  – Tap Bridge
  – FdNetDevice
  – NetmapNetDevice (coming soon)
Emulation support

• Support moving between simulation and testbeds or live systems
• A real-time scheduler, and support for two modes of emulation
• Linux is only operating system supported
• Must run simulator in real time
  – GlobalValue::Bind ("SimulatorImplementationType", StringValue ("ns3::RealTimeSimulatorImpl"));

• Must enable checksum calculations across models
  – GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));

• Must run as root
ns-3 emulation modes

1) ns-3 interconnects real or virtual machines

2) testbeds interconnect ns-3 stacks

ns-3 training, June 2018
Example use case: testbeds

• Support for use of Rutgers WINLAB ORBIT radio grid
Example use case: mininet

- Mininet is popular in the Software-Defined Networking (SDN) community
- Mininet uses "TapBridge" integration
- https://github.com/mininet/mininet/wiki/Link-modeling-using-ns-3
CORE emulator

Common Open Research Emulator (CORE)

The Common Open Research Emulator (CORE) is a tool for emulating networks on one or more machines. You can connect these emulated networks to live networks. CORE consists of a GUI for drawing topologies of lightweight virtual machines, and Python modules for scripting network emulation.
Emulation Devices
Device models

- File Descriptor Net Device (FdNetDevice)
  - read and write traffic using a file descriptor provided by the user
  - this file descriptor can be associated to a TAP device, to a raw socket, to a user space process generating/consuming traffic, etc.

- Tap Bridge
  - Integrate Tun/Tap devices with ns-3 devices
"TapBridge": netns and ns-3 integration

Tap device pushed into namespaces; no bridging needed
TapBridge modes

- **ConfigureLocal (default mode)**
  - ns-3 configures the tap device
  - useful for host to ns-3 interaction

- **UseLocal**
  - user has responsibility for device creation
  - ns-3 informed of device using “DeviceName” attribute

- **UseBridge**
  - TapDevice connected to existing Linux bridge
ns-3 ensures that Mac addresses are consistent.
UseLocal

Mac X spoofed to Mac Y
ns-3 devices must support SendFrom() (i.e. bridging)
FdNetDevice

- Unified handling of reading/writing from file descriptor
- Three supported helper configurations:
  - **EmuFdNetDeviceHelper** (to associate the ns-3 device with a physical device in the host machine)
  - **TapFdNetDeviceHelper** (to associate the ns-3 device with the file descriptor from a tap device in the host machine) *not the same as TapBridge*
  - **PlanetLabFdNetDeviceHelper** (to automate the creation of tap devices in PlanetLab nodes, enabling ns-3 simulations that can send and receive traffic though the Internet using PlanetLab resource.)
• Device performs MAC spoofing to separate emulation from host traffic
ns-3 over host sockets

• Two publications about how to run ns-3 applications over real hosts and sockets
  – "Simulator-agnostic ns-3 Applications", Abraham and Riley, WNS3 2012
Generic Emulation Issues

• Ease of use
  – Configuration management and coherence
  – Information coordination (two sets of state)
    • e.g. IP/MAC address coordination
  – Output data exists in two domains
  – Debugging can be more challenging

• Error-free operation (avoidance of misuse)
  – Synchronization, information sharing, exception handling
    • Checkpoints for execution bring-up
    • Inoperative commands within an execution domain
    • Deal with run-time errors
  – Soft performance degradation (CPU) and time discontinuities
Netmap NetDevice coming soon

- 2017 ESA Summer of Code project by Pasquale Imputato
- Netmap allows ns-3 to gain direct access to the network device, bypassing the host networking stack mapping the device memory in user space area.
- Support for flow control and BQL (see discussion in traffic control presentation)
- Pasquale reports that the pps achievable with the NetmapNetDevice at lowest layer (i.e., the write method) is up to 1.38 Mpps on the e1000e adapter on the i7 cpu at full frequency of 3.8 GHZ.
- Latency is also reduced due to better flow control
ns-3 Training

Distributed simulation

(some slides/images credit due to Peter Barnes)
Distributed simulation

• By default, ns-3 runs a single threaded event loop on a single CPU core
  – Multi-core machines or multiple machines not used
• Performance is limited (CPU cycles and memory)
  • $\sim 10^4$ packet receives/wall clock second/core *
  – heavyweight applications may consume memory (DCE, routing tables in very large topologies)

* Peter Barnes, WNS3 training 2016
Distributed simulation

• Decompose model into *Logical Processes*
  – Separate objects and event queues
  – Execute independently
  – Events for other LPs become messages
  – ~ MPI Ranks
Virtual time evolution

- Sometimes ahead in virtual time, sometimes behind
- Time evolution constrained by slowest LP
- No causality violations
  - Hallmark of conservative execution
Using MPI in ns-3

• Need to install MPI on the host system (OpenMPI or MPICH libraries)

• Need to pass ‘--enable-mpi’ to ‘./waf configure’
  – Followed by usual build

• Only point-to-point links may be used to separate LPs
  – needs some level of propagation delay to support lookahead in each LP
  – splitting of wireless channels not presently supported

• The ns-3 scenario has to be constructed by assigning nodes to LPs manually
Running with ‘mpirun’

- waf can’t distinguish sequential and parallel
  - Need to specify `mpirun` and number of ranks explicitly

Running Parallel Scripts with waf and mpirun

```bash
$ ./waf --run simple-distributed
Waf: Entering directory 'build/debug'
Waf: Leaving directory 'build/debug'
'build' finished successfully (2.118s)
This simulation requires 2 and only 2 logical processors.
Command ['build/debug/src/mpi/examples/ns3-dev-simple-distributed-debug'] exited with code 1

# Multiple ranks on a single computer:
$ ./waf --run simple-distributed --command-template="mpirun -np 2 %s"
Waf: Entering directory 'build/debug'
Waf: Leaving directory 'build/debug'
'build' finished successfully (2.104s)
At time 1.02264s packet sink received 512 bytes from 10.1.1.1 port 49153 total Rx 512 bytes
At time 1.0235s packet sink received 512 bytes from 10.1.2.1 port 49153 total Rx 512 bytes
At time 1.02437s packet sink received 512 bytes from 10.1.3.1 port 49153 total Rx 512 bytes
At time 1.02524s packet sink received 512 bytes from 10.1.4.1 port 49153 total Rx 512 bytes

# Multiple computers:
$ mpirun -np 2 ./waf --run simple-distributed
```
1. Include mpi-module.h
2. Same topology, split across Point-to-point link
1. Different log component name
2. Command line argument to select Null message
1. Condition on NS3_MPI
2. Null message selector
3. Initialize MPI
4. Get rank #, number of ranks
5. Check number of ranks
6. Use symbolic names for each rank
7. Create point-to-point nodes

Example

```cpp
if (verbose) {
    LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
}

NodeContainer p2pNodes;
p2pNodes.Create (2);

PointToPointHelper pointToPoint;
    pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));

    NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;
    csmaNodes.Add (p2pNodes.Get (1));
    csmaNodes.Create (nCsma);

    CsmHelper csma;
    csma.SetChannelAttribute("DataRate", StringValue("100Mbps"));
    csma.SetChannelAttribute("Delay", TimeValue (NanoSeconds (6500)));

    NetDeviceContainer csmaDevices;
    csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
```
1. Create CSMA nodes on one rank
2. Create Wifi nodes on another rank
1. Install devices, addresses and Internet stack everywhere
2. Install applications only on rank-local nodes
1. Enable PCAP tracing on local nodes?
2. Close MPI cleanly
Script Output—Identical

$ ./waf --run third
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.152s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01796s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01796s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03364s client received 1024 bytes from 10.1.2.4 port 9

$ ./waf --run third-distributed\ --command-template="mpirun -n 2 %s --tracing"
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.050s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01796s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01796s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03364s client received 1024 bytes from 10.1.2.4 port 9