Introduction to Network Simulator-3

Dr Anil Kumar Rangisetti

Dept. of Computer Science and Engineering

Indian Institute of Information Technology (IIIT) Dharwad, India

Motivation for Network Simulations

- Goals
  - build software simulation model of networking systems and to analyze/study/improve/develop network archs & protocols
- Reasons
  - real systems are expensive, complex, unavailable
- Advantages
  - relatively easy to setup, deploy and instrument
  - reproducibility, scalability
- Disadvantages
  - simplified view of complex interactions
  - could be immensely misleading

NS-3 Features

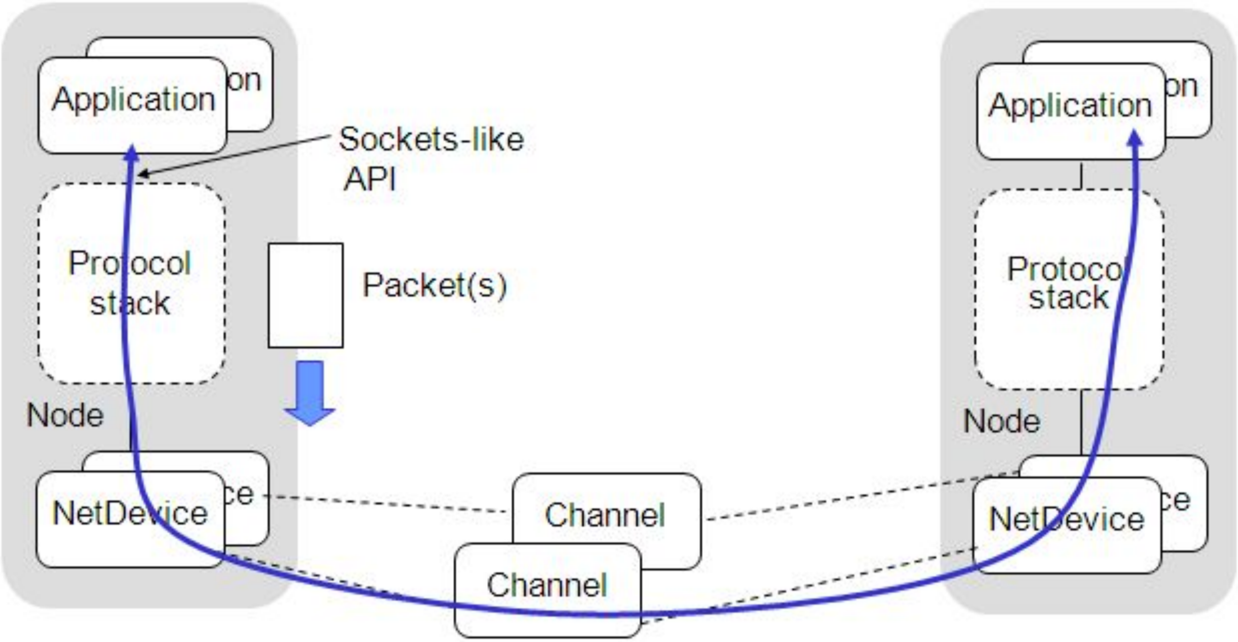
- Open Source licensing and development module
- Python scripts or C++ programs to define simulations
- Modular, documented core libraries
- Scalability
  - Nodes have optional features
    - E.g., No memory waste in IPv4 stack for nodes that don't need it
    - E.g., wired net devices do not need to know the node position at all
  - New features can be easily added in the future
    - For example, energy models
  - Memory management of Packet objects is entirely automatic and extremely efficient
- Development features
  - Modularity
  - Logging, Tracing, Debugging

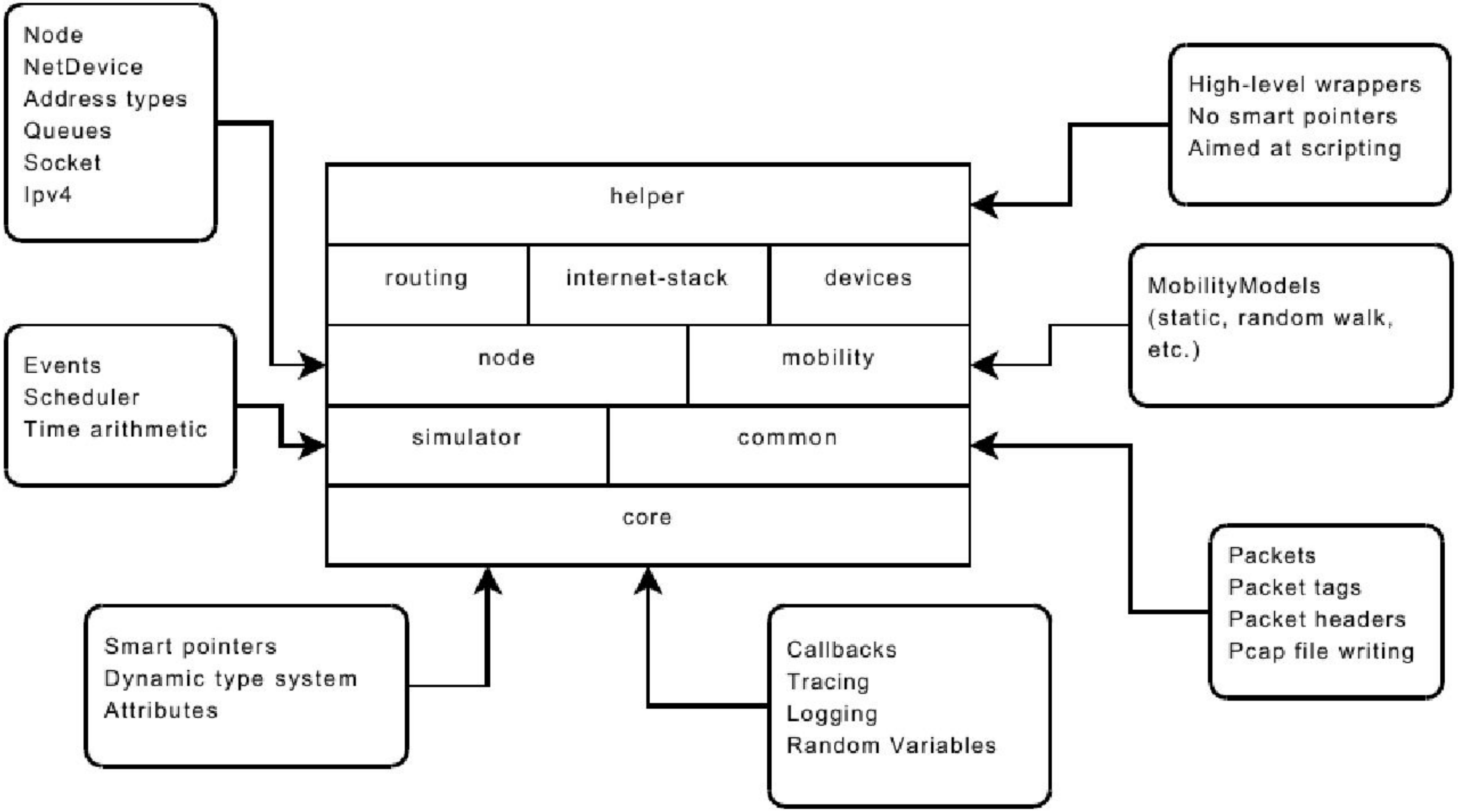
NS-3 Modules

- A set of related classes, examples, and tests, that can be combined together into an *ns-3*module so that they can be used with existing *ns-3*modules and by other researchers
- All modules are under src directory
  - Wired Network Modules
    - Point-to-Point, csma, Network, Internet, etc
  - Wireless Network Modules
    - wifi, wimax, lte, olsr, aodv, dsdv, dsr, mesh, sixlowpan, spectrum, propogation, mobility, etc
  - Generic modules
    - Core, energy, flow-monitor, stats, etc

Basic NS-3 Models

- Key objects in the simulator are Nodes, Packets, and Channels
- Node is a husk of a computer to which applications, stacks, and NICs (NetDevs) are added
- Nodes are architected for multiple interfaces





Node
NetDevice
Address types
Queues
Socket
Ipv4

Events
Scheduler
Time arithmetic

helper

routing | internet-stack | devices

node | mobility

simulator | common

core

High-level wrappers
No smart pointers
Aimed at scripting

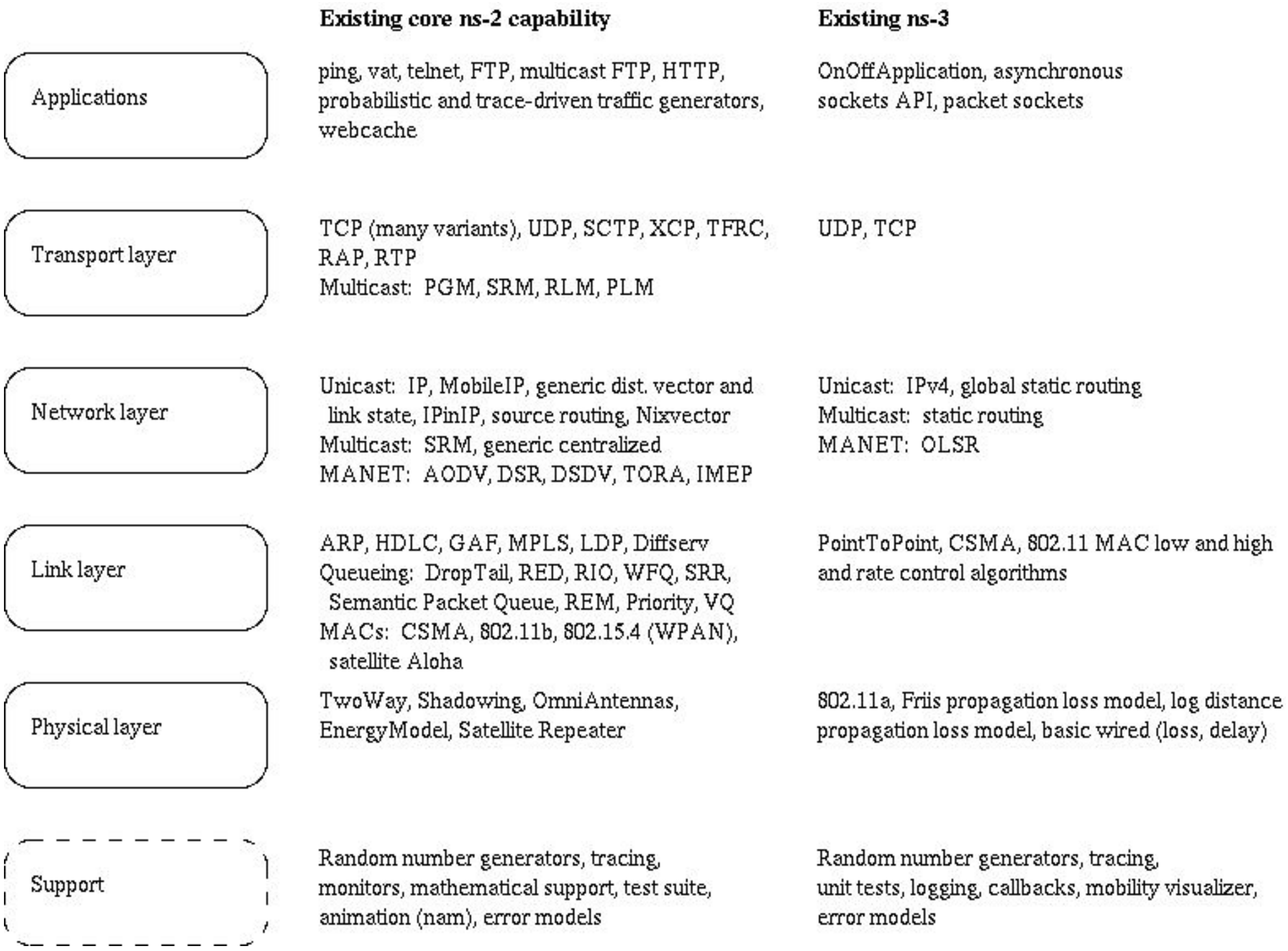
MobilityModels
(static, random walk, etc.)

Packets
Packet tags
Packet headers
Pcap file writing

Smart pointers
Dynamic type system
Attributes

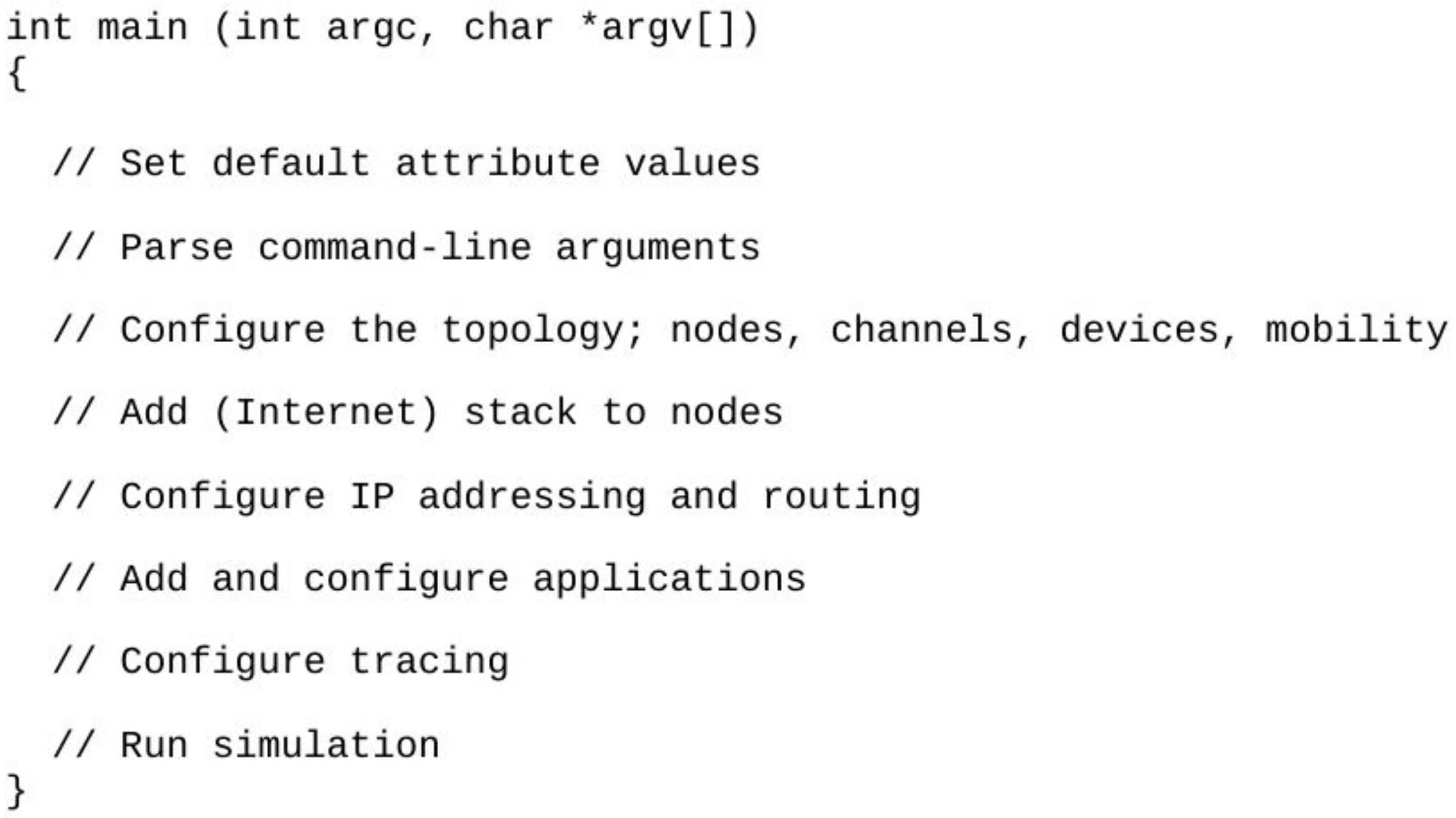
Callbacks
Tracing
Logging
Random Variables

NS-3 Protocol Stack



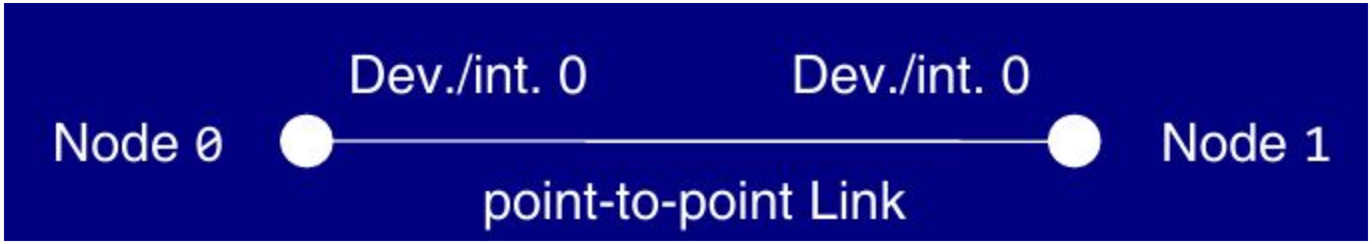
	Existing core ns-2 capability	Existing ns-3
Applications	ping, vat, telnet, FTP, multicast FTP, HTTP, probabilistic and trace-driven traffic generators, webcache	OnOffApplication, asynchronous sockets API, packet sockets
Transport layer	TCP (many variants), UDP, SCTP, XCP, TFRC, RAP, RTP Multicast: PGM, SRM, RLM, PLM	UDP, TCP
Network layer	Unicast: IP, MobileIP, generic dist. vector and link state, IPinIP, source routing, Nixvector Multicast: SRM, generic centralized MANET: AODV, DSR, DSDV, TORA, IMEP	Unicast: IPv4, global static routing Multicast: static routing MANET: OLSR
Link layer	ARP, HDLC, GAF, MPLS, LDP, Diffserv Queueing: DropTail, RED, RIO, WFQ, SRR, Semantic Packet Queue, REM, Priority, VQ MACs: CSMA, 802.11b, 802.15.4 (WPAN), satellite Aloha	PointToPoint, CSMA, 802.11 MAC low and high and rate control algorithms
Physical layer	TwoWay, Shadowing, OmniAntennas, EnergyModel, Satellite Repeater	802.11a, Friis propagation loss model, log distance propagation loss model, basic wired (loss, delay)
Support	Random number generators, tracing, monitors, mathematical support, test suite, animation (nam), error models	Random number generators, tracing, unit tests, logging, callbacks, mobility visualizer, error models

Structure of NS-3 Program



```
int main (int argc, char *argv[])
{

  // Set default attribute values

  // Parse command-line arguments

  // Configure the topology; nodes, channels, devices, mobility

  // Add (Internet) stack to nodes

  // Configure IP addressing and routing

  // Add and configure applications

  // Configure tracing

  // Run simulation
}
```

Network Simulation Example



```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE
("FirstScriptExample");
```

Topology

```
int main (intargc, char *argv[])
{
        LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
        LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);

        NodeContainer nodes; // Nodes creation
        nodes.Create(2);

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

        //Create p2p link between nodes and  device interfaces for the nodes
        NetDeviceContainer devices;
        devices = pointToPoint.Install(nodes);
```

Install Internet Stack and Assign IP addresses

```
InternetStackHelper stack;
stack.Install(nodes);

Ipv4AddressHelper address;
address.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces =
address.Assign(devices);
```

Create Applications and Install on Nodes

```
//Install UDP echo server on Node 1  and start and stop applications
UdpEchoServerHelper echoServer(9);
ApplicationContainer serverApps= echoServer.Install(nodes.Get(1));
serverApps.Start(Seconds (1.0));
serverApps.Stop(Seconds (10.0));

//Install UDP echo client on Node and start and stop applications

UdpEchoClientHelper echoClient(interfaces.GetAddress(1), 9);
echoClient.SetAttribute("MaxPackets", UintegerValue(1));
echoClient.SetAttribute("Interval", TimeValue(Seconds (1.0)));
echoClient.SetAttribute("PacketSize", UintegerValue(1024));

ApplicationContainer clientApps= echoClient.Install(nodes.Get(0));
clientApps.Start(Seconds (2.0));
clientApps.Stop(Seconds (10.0));
//Start Simulation and at the end destroy simulation
Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```

NS-3 any Simulation

- Simulation time moves in discrete jumps from event to event
- C++ functions schedule events to occur at specific simulation times
- A simulation scheduler orders the event execution
- Simulation::Run() gets it all started
- Simulation stops at a specific time or when all pending events end

Running Example

- cp yourprogam.cc to scratch/myfirst.cc
- ./waf--run /scratch/myfirst
- Sent 1024 bytes to 10.1.1.2
- Received 1024 bytes from 10.1.1.1
- Received 1024 bytes from 10.1.1.2

How to Find Flow Level Stats ?

NS-3 offers "ns3/flow-monitor-module.h" to find
flow level Throughput, Delay, Jitter, Packet Loss Ratio

Following code should be included in test program:

```
 Ptr<FlowMonitor> flowmon;
 FlowMonitorHelper flowmonHelper;
 flowmon = flowmonHelper.InstallAll ();

 Simulator::Stop (Seconds(10.1));
 Simulator::Run ();
 printStats (flowmonHelper, true); //We should write the code
 Simulator::Destroy ();
```

Example FlowStats Output

```
FlowID: 1 (UDP 10.1.1.1 / 49153 --> 10.1.1.2 / 9)
  Tx Bytes:
  Lost Packets:
  Pkt Lost Ratio:
  Throughput:
  Mean{Delay}:
  Mean{Jitter}:
FlowID: 2 (UDP 10.1.1.2 / 9 --> 10.1.1.1 / 49153)
Tx Bytes:
  Lost Packets:
  Pkt Lost Ratio:
  Throughput:
  Mean{Delay}:
  Mean{Jitter}:
Final throughput with (packets received/total time):
Total packets transmitted:
Total packets received:
Total packets dropped:
Packet Lost Ratio:
```

How to debug your program?

To identify reason for runtime errors (GDB)

run program with gdb

```
./waf --run scratch/myfirst --command-template="gdb %s"
```

using gdb commands

(list code

setting breaking points, (b #line), run, continue

inspecting variables, (print)

inspect stack frames

back trace, etc)

How to use logging?

Useful for tracing various program execution events at various levels

- LOG_LEVEL_ERROR, LOG_LEVEL_WARN, LOG_LEVEL_DEBUG
- LOG_LEVEL_LOGIC, LOG_LEVEL_INFO, LOG_LEVEL_FUNCTION
- LOG_LEVEL_ALL

How to include logging statements in your program:

e.g., NS_LOG_COMPONENT_DEFINE ("YourProgram");

include cout like statements in your program:

NS_LOG_LOGIC (“sample”<<test);

NS_LOG_WARN(“warnig”);

How to **enable to print** logging statements of **any program** in your program:

e.g., LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);

Example code

```
void printAddress(NetDeviceContainer&
devices,NodeContainer& nodes,int i){

NS_LOG_INFO(devices.Get(i)->GetAddress());
NS_LOG_INFO(nodes.Get(i)->GetObject<Ipv4>()->GetAddress(1,
0).GetLocal());
}
```

Can we inspect flows at packet level?

NS-3 Supports Packet Cpaturing using

- EnablePcap() at device level
- EnablePcapAll() across all devices

use ***wireshark*** to inspect the captured packet traces

How to enable packet capture in your program?

pointToPoint.EnablePcapAll ("filename"); //pcap file name

Simulator::Stop (Seconds(3.1));

Simulator::Run ();

after simulation

open **filename-node-interface.pacp** using wireshark

References

https://www.nsnam.org/docs/tutorial/html/