
Wireless Ad Hoc Networks

Lab 2

Network Simulator

NS3 Experiment (I) - CBR & Queue

Introduction

■ Lab Purpose

- Introducing network simulator tools
- Getting familiar with NS-3

■ Equipments

- PC / NB (Root Password : bunlab)
- Network Simulators : NS-3 (Network Simulator version 3)

Introduction

■ Network Simulator

□ NS : NS-2 & NS-3

- a discrete event simulator targeted at networking research
- NS-3 is not backward compatible with NS-2
- NS-2 is implemented using a combination of oTCL (for scripts describing the network topology) and C++ (The core of the simulator) ; However, NS-3 is written in C++ only.

□ NS-3

- Written in C++
- Bindings in Python
- Uses waf build system
 - (waf = python-based framework for compiling/installing application)
- Simulations programs are in C++ executables or python scripts

NS-3 Installation

■ Install prerequisites

- ❑ `sudo apt-get install gcc g++ python python-dev mercurial bzr gdb valgrind gsl-bin libgsl0-dev libgsl0ldbl flex bison tcpdump sqlite sqlite3 libsqlite3-dev libxml2 libxml2-dev libgtk2.0-0 libgtk2.0-dev uncrustify doxygen graphviz imagemagick texlive texlive-latex-extra texlive-generic-extra texlive-generic-recommended texinfo dia texlive texlive-latex-extra texlive-extra-utils texlive-generic-recommended texi2html python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev python-pygccxml`

■ Download source code

- ❑ `wget http://www.nsnam.org/release/ns-allinone-3.24.tar.bz2`
- ❑ `tar -xvjf ns-allinone-3.24.tar.bz2`
- ❑ `cd ns-allinone-3.24/`

■ Build

- ❑ `./build.py --enable-examples --enable-tests`

■ Configure with waf (build tools)

- ❑ `./waf -d debug --enable-examples --enable-tests configure`
- ❑ `CXXFLAGS="-Wall" ./waf configure` (if "cc1plus: warnings being treated as errors" occurs)

■ Test everything

- ❑ `./test.py`

Run NS-3 Program

- Build all your scripts

- `./waf`

- you can build and run a specific program/script

- `./waf --run <ns3-program>`

Substitute <**ns3-program**> with your own program name

- Run the script with an argument

- `./waf --run <ns3-program> --command-template="%s <args>"`

Substitute <**args**> with the arguments

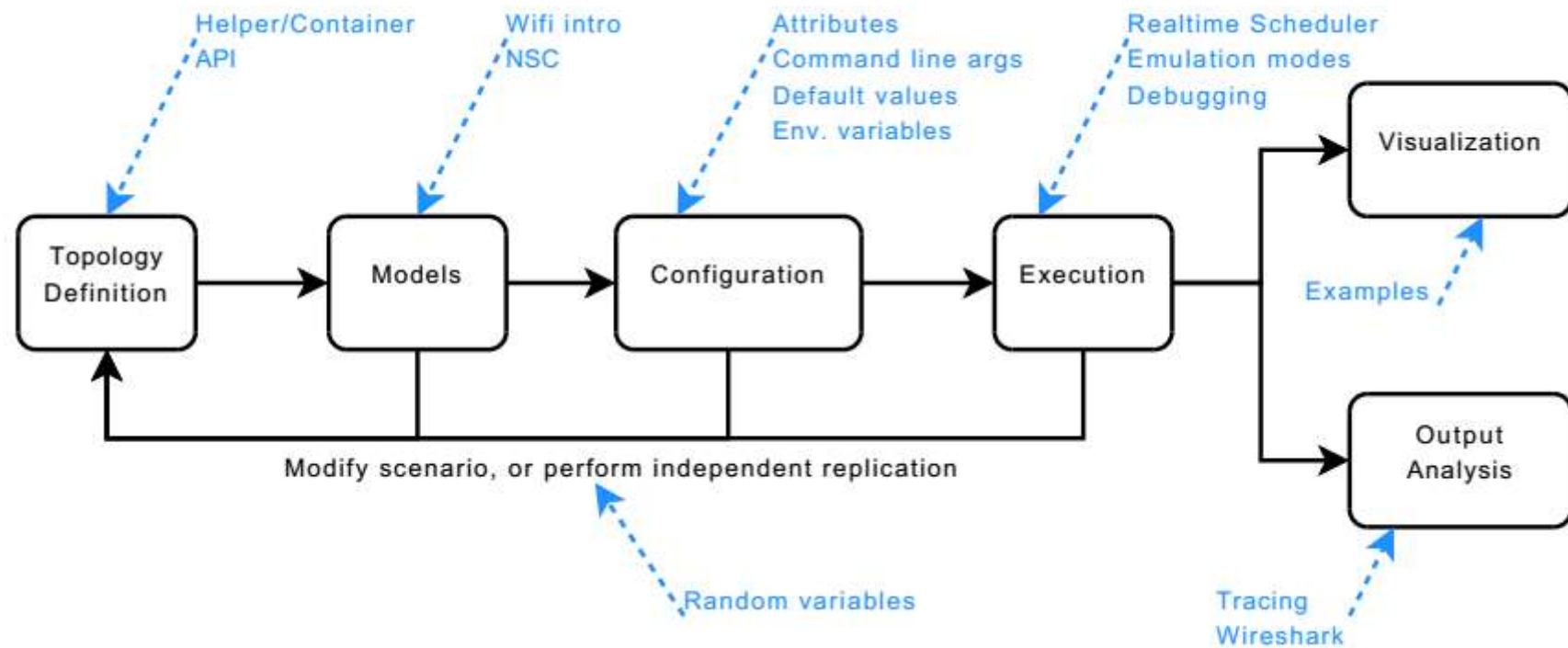
NS-3 Simulation Basics

- Simulation time moves discretely 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 specific time or when events end

Scheduling Events

```
static void random_function (MyModel *model)
{
    // print some stuff
}
int main (int argc, char **argv)
{
    MyModel model;
    Simulator::Schedule (Seconds (10.0),
                        &random_function, &model);
    Simulator::Run ();
    Simulator::Destroy ();
}
```

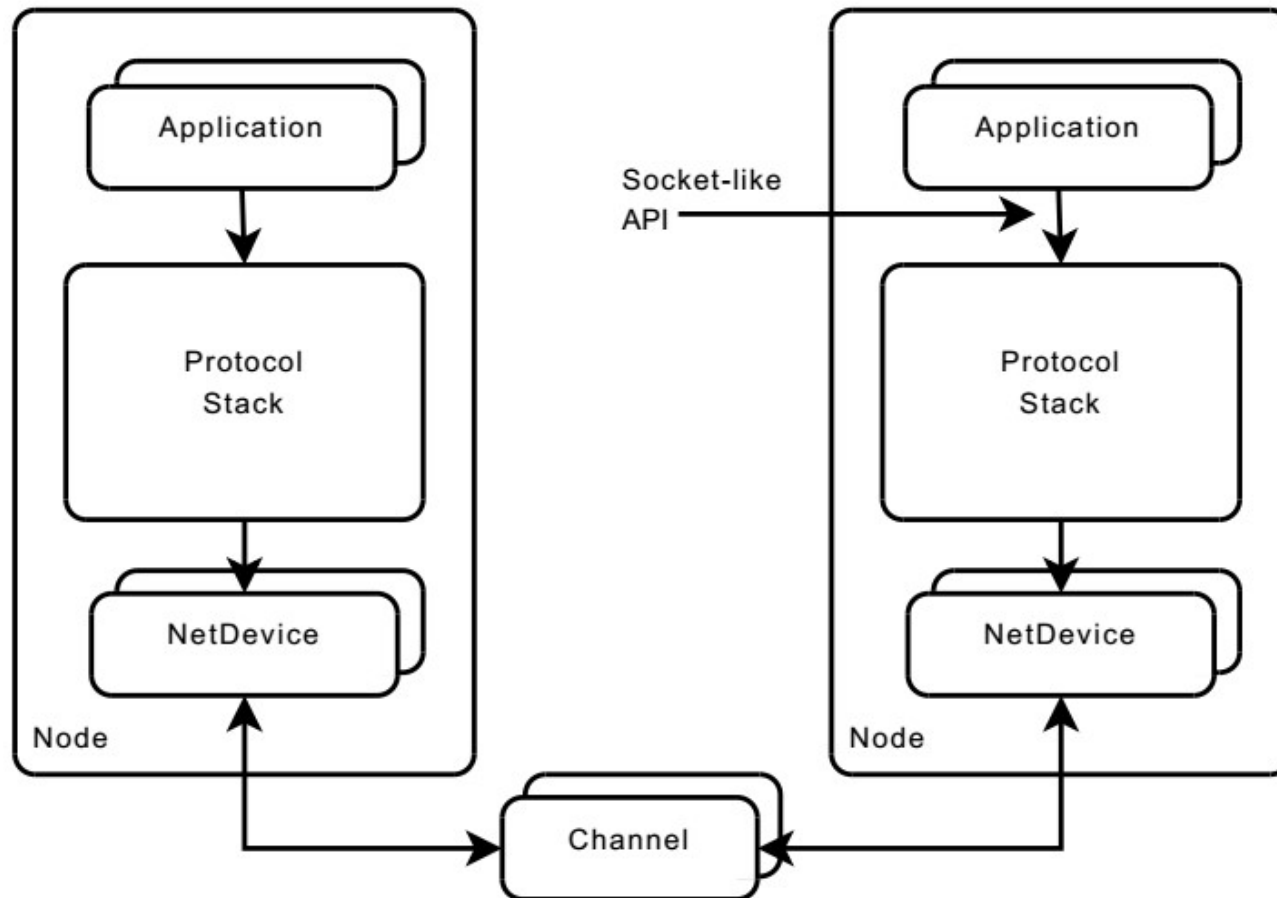
Typical Simulation Structure



What the source code look like

```
NodeContainer csmaNodes;  
csmaNodes.Create (2);  
...  
NetDeviceContainer csmaDevices;  
CsmaHelper csma;  
csma.SetChannelAttribute ("DataRate",  
                           StringValue ("5Mbps"));  
csma.SetChannelAttribute ("Delay",  
                           StringValue ("2ms"));  
csmaDevices = csma.Install (csmaNodes);
```

The Basic Model



NS-3 Fundamental Objects

- **Node**

- motherboard of a computer with RAM, CPU, and IO interfaces

- **Application**

- A packet generator and consumer, run on a Node

- **NetDevice**

- A network card which can be plugged in an IO interface of a Node

- **Channel**

- A physical connector between a set of NetDevice objects

The helper/container API (I)

- Make it easy to build topologies with repeating patterns
- Make the the topology description more high-level (easier to read and understand)
- Sets of objects are stored in **Containers**
- One operation is encoded in a **Helper** object and applies on a **Container**
- Different helpers provide different operations

The helper/container API (II)

■ Example containers

- ❑ NodeContainer
- ❑ NetDeviceContainer
- ❑ IPvAddressContainer

■ Example helper classes

- ❑ InternetStackHelper
- ❑ WifiHelper
- ❑ MobilityHelper
- ❑ etc. (Each model provides a helper class)

The helper/container API - Nodes & Devices

■ Create the nodes

```
NodeContainer csmaNodes;  
csmaNodes.Create (2);  
NodeContainer wifiNodes;  
wifiNodes.Add (csmaNodes.Get (1));  
wifiNodes.Create (3);
```



Create empty node container
Create two nodes



Create empty node container
Add existing node to it

■ Create the network device

```
NetDeviceContainer csmaDevices;  
CsmaHelper csma;  
csma.SetChannelAttribute ("DataRate",  
                          StringValue ("5Mbps"));  
csma.SetChannelAttribute ("Delay",  
                          StringValue ("2ms"));  
csmaDevices = csma.Install (csmaNodes);
```



Create empty device container
Create csma helper
Set data rate
Set delay

Create csma devices

The helper/container API - Channel & Device

■ Setup wifi channel

```
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();  
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();  
wifiPhy.SetChannel (wifiChannel.Create ());
```

■ Create wifi devices

```
NetDeviceContainer wifiDevices;  
WifiHelper wifi = WifiHelper::Default ();  
wifiDevices = wifi.Install (wifiPhy, wifiNodes);
```

The helper/container API - Mobility models

- Setup initial position

```
MobilityHelper mobility;  
mobility.SetPositionAllocator ("ns3::RandomDiscPositionAllocator",  
                               "X", StringValue ("100.0"),  
                               "Y", StringValue ("100.0"),  
                               "Rho", StringValue ("Uniform:0:30"));
```

- Setup mobility model during simulation

```
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");  
mobility.Install (wifiNodes);
```


The helper/container API - Protocol stacks

- Install ipv4, udp, tcp stacks

```
InternetStackHelper internet;  
internet.Install (NodeContainer::GetGlobal ());
```

- Assign ipv4 addresses

```
Ipv4InterfaceContainer csmaInterfaces;  
Ipv4InterfaceContainer wifiInterfaces;  
Ipv4AddressHelper ipv4;  
ipv4.SetBase ("10.1.1.0", "255.255.255.0");  
csmaInterfaces = ipv4.Assign (csmaDevices);  
ipv4.SetBase ("10.1.2.0", "255.255.255.0");  
wifiInterfaces = ipv4.Assign (wifiDevices);
```

- Setup routing tables

```
GlobalRouteManager::PopulateRoutingTables ();
```

The helper/container API - Application (I)

- Create a traffic generator

```
OnOffHelper onoff ("ns3::UdpSocketFactory",  
                  InetSocketAddress ("10.1.2.2", 1025));  
onoff.SetAttribute ("OnTime", StringValue ("Constant:1.0"));  
onoff.SetAttribute ("OffTime", StringValue ("Constant:0.0"));
```

- Install the traffic generator

```
ApplicationContainer apps;  
apps = onoff.Install (csmaNodes.Get (0));
```

- Start it

```
apps.Start (Seconds (1.0));  
apps.Stop (Seconds (4.0));
```

Example

```
OnOffHelper onoff( "ns3::UdpSocketFactory" , InetSocketAddress( address , Port ) );  
onoff.SetConstantRate(DataRate( your_rate ), packetsize );  
appcontainer.Add(onoff.Install(nodecontainer.Get(0)));  
appcontainer.Start(starttime);  
appcontainer.Stop(stoptime);
```

You can obtain Address from Ipv4interfacecontainer by method GetAddress(i,j).
For example, Ipv4interfacecontainer.GetAddress(i,j) will return the IPv4 address of the j'th address of the interface corresponding to index i.

The helper/container API - Application (II)

■ Setup a traffic sink

```
PacketSinkHelper sink ("ns3::UdpSocketFactory",  
                      InetAddress ("10.1.2.2", 1025));  
apps = sink.Install (wifiNodes.Get (1));  
apps.Start (Seconds (0.0));  
apps.Stop (Seconds (4.0));
```

Tracing in NS-3 vs NS-2

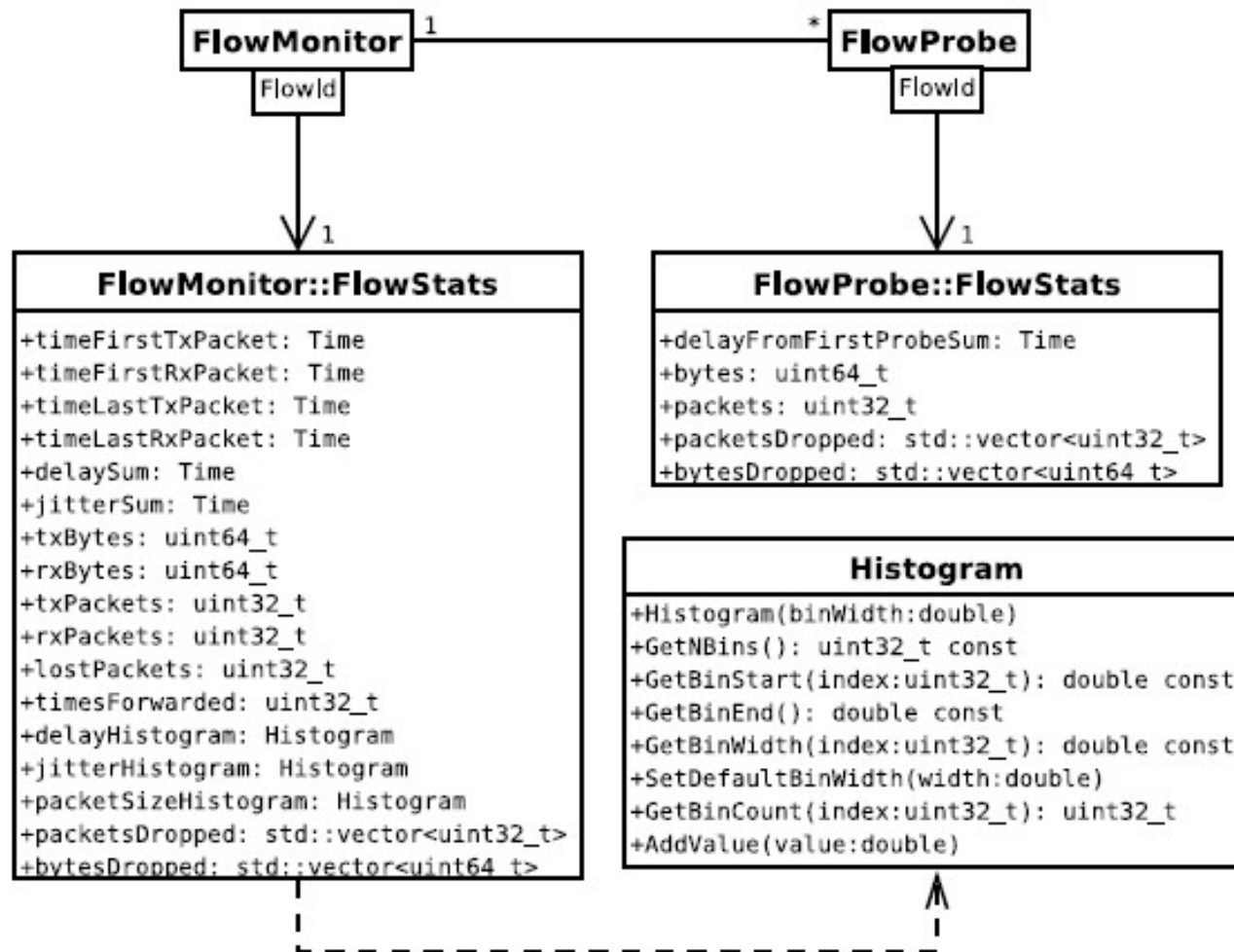
- Tracing is a structured form of simulation output
- When you open .tr file in NS-2

```
wireless1-out.tr *
s 0.053360000 _0_ AGT --- 2 cbr 210 [0 0 0 0] ----- [0:0 1:0 32 0] [1] 0 0
r 0.053360000 _0_ RTR --- 2 cbr 210 [0 0 0 0] ----- [0:0 1:0 32 0] [1] 0 0
s 0.054411734 _0_ RTR --- 1 DSR 32 [0 0 0 0] ----- [0:255 1:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
s 0.054486734 _0_ MAC --- 1 DSR 84 [0 ffffffff 0 800] ----- [0:255 1:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 0.055159205 _1_ MAC --- 1 DSR 32 [0 ffffffff 0 800] ----- [0:255 1:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 0.055184205 _1_ RTR --- 1 DSR 32 [0 ffffffff 0 800] ----- [0:255 1:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
s 0.056720000 _0_ AGT --- 5 cbr 210 [0 0 0 0] ----- [0:0 1:0 32 0] [2] 0 0
r 0.056720000 _0_ RTR --- 5 cbr 210 [0 0 0 0] ----- [0:0 1:0 32 0] [2] 0 0
s 0.060080000 _0_ AGT --- 7 cbr 210 [0 0 0 0] ----- [0:0 1:0 32 0] [3] 0 0
r 0.060080000 _0_ RTR --- 7 cbr 210 [0 0 0 0] ----- [0:0 1:0 32 0] [3] 0 0
s 0.063348966 _1_ RTR --- 4 DSR 44 [0 0 0 0] ----- [1:255 0:255 254 0] 2 [0 1] [1 1 2 0->1] [0 0 0 0->0]
s 0.063440000 _0_ AGT --- 13 cbr 210 [0 0 0 0] ----- [0:0 1:0 32 0] [4] 0 0
r 0.063440000 _0_ RTR --- 13 cbr 210 [0 0 0 0] ----- [0:0 1:0 32 0] [4] 0 0
s 0.063763966 _1_ MAC --- 0 ARP 80 [0 ffffffff 1 806] ----- [REQUEST 1/1 0/0]
r 0.064404437 _0_ MAC --- 0 ARP 28 [0 ffffffff 1 806] ----- [REQUEST 1/1 0/0]
```

NS-3 FlowMonitor

- Detect all flows passing through network
- Stores metrics for analysis
 - bitrates, duration, delays, packet sizes, packet loss ratios
- Solves the problem
 - Less programming time than callback-based tracing
 - More efficient than ascii tracing

Flow Data Structures (I)



Flow Data Structures (II)

- **timeFirstTxPacket, timeLastTxPacket, timeFirstRxPacket, timeLastRxPacket**
 - begin and end times of the flow from the point of view of the transmitter/receiver
- **delaySum, jitterSum**
 - sum of delay and jitter values
- **txBytes, txPackets, rxBytes, rxPackets**
 - number of transmitted/received bytes and packets

Flow Data Structures (III)

- **lostPackets**

- number of definitely lost packets

- **packetsDropped, bytesDropped**

- discriminates the losses by a *reason code*
 - **DROP NO ROUTE** no IPv4 route found for a packet
 - **DROP TTL EXPIRE** a packet was dropped due to an
 - IPv4 TTL field decremented and reaching zero
 - **DROP BAD CHECKSUM** a packet had a bad IPv4
 - header checksum and had to be dropped

FlowMonitor - Example

- Install FlowMonitor

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

- Print per flow statistics

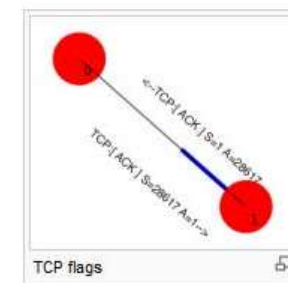
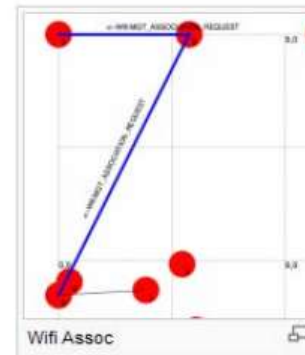
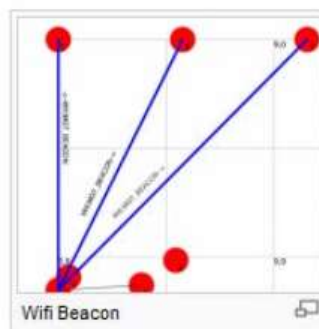
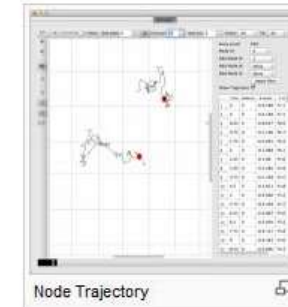
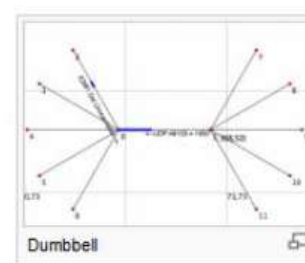


```
Flow 1 (10.0.0.1 -> 10.0.0.2)  
Tx Bytes: 3847500  
Rx Bytes: 316464  
Throughput: 0.241443 Mbps  
Flow 2 (10.0.0.3 -> 10.0.0.2)  
Tx Bytes: 3848412  
Rx Bytes: 336756  
Throughput: 0.256924 Mbps
```

```
monitor->CheckForLostPackets ();  
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());  
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();  
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i)  
{  
    // first 2 FlowIds are for ECHO apps, we don't want to display them  
    if (i->first > 2)  
    {  
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);  
        std::cout << "Flow " << i->first - 2 << " (" << t.sourceAddress << " -> " << t.destinationAddress << ")\n";  
        std::cout << " Tx Bytes: " << i->second.txBytes << "\n";  
        std::cout << " Rx Bytes: " << i->second.rxBytes << "\n";  
        std::cout << " Throughput: " << i->second.rxBytes * 8.0 / 10.0 / 1024 / 1024 << " Mbps\n";  
    }  
}
```

Visualization: NetAnim

- Animate packets over wired or wireless links
- Node position statistics with node trajectory plotting (path of a mobile node)



NS-3 Lab2 Experiment

- [Step 1] Install NS-3. (Done For You)
- [Step 2] Open the terminal and edit the NS3 source file with following settings:
 - 2 nodes placed apart 100 [m]
 - CBR packet size = 1000 [Byte]
 - CBR rate = **860 [kbps] to 960 [kbps] , 1060 [kbps] , 1160 [kbps]**
 - ifqlen = **20 to 30, 40, 50**
 - CBR traffic
 - start at 1 [sec]
 - stop at 15 [sec]
 - Simulation Time = 30 [sec]
- [Step 3] Modify the FlowMonitor section to analyze and get the following results :
 - average end-to-end delay
 - packet delivery ratio
 - total received data size

Note for NS-3 Lab1 Experiment

- average end-to-end delay =
$$\frac{\text{\#total e2e delay}}{\text{\#total packets}}$$
 - Total e2e delay = the summation end-to-end delay per packet (received time – sent time)
 -
- packet delivery ratio =
$$\frac{\text{\#total packets received}}{\text{\#total packets sent}}$$
- total received data size
 - 1 kbits = 1024 bits
 - 1 mbits = 1024 kbits = 1024*1024 bits

Questions & Report

- Draw three figures with
 - X-axis : ifqlen settings
 - Y-axis : the three network performance metrics, respectively
 - average end-to-end delay
 - packet delivery ratio
 - total received data size
 - In each figure, show the curves for the four CBR values
 - 4 curves in each figure
- You need to answer these works and then write a full report with your thoughts or analyses.
- 將實驗得到的資料整理好(製作成表格)，並且根據題目要求做出正確的圖，圖表要標示清楚，並根據課堂上學到的知識去做分析討論。
 - Deadline : **2022 / 11 / 29**