NS3 Group Presentation

B2 Group 4

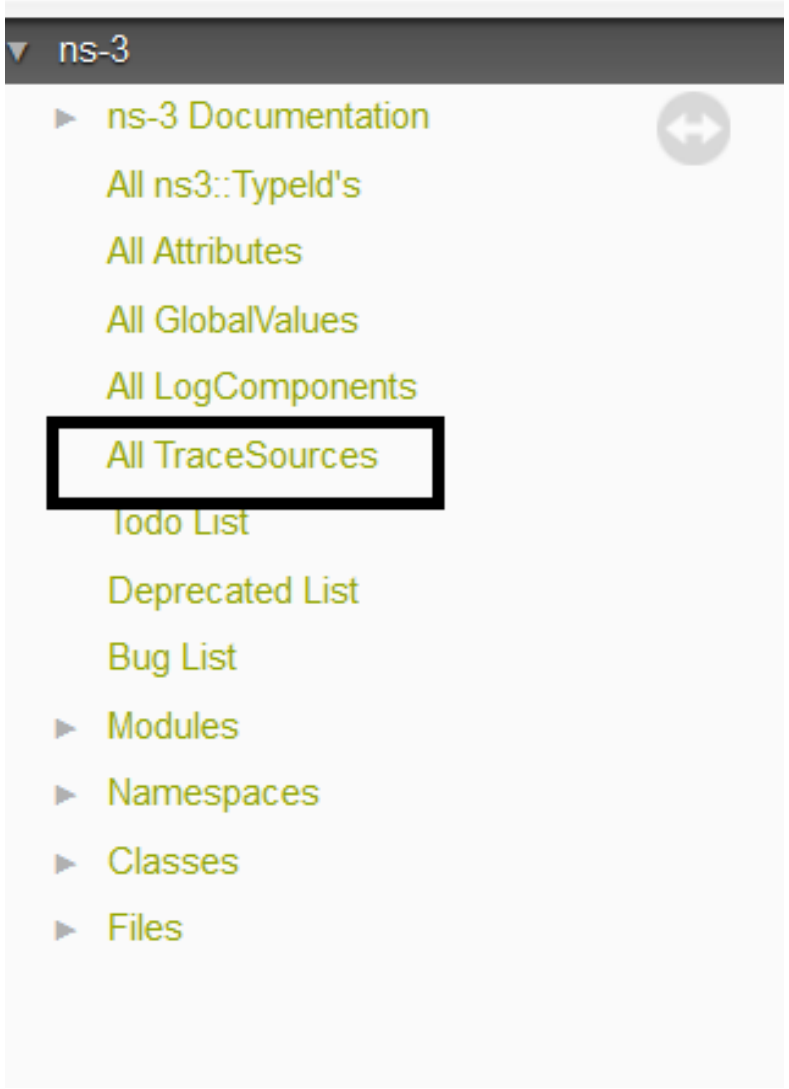
Tracing and Data Collection from Simulation Results

1705092 – Asif Ajrof
1705096 – Kazim Abrar Mahi
1705098 – Sihat Afnan
1705108 – Muhtasim Noor Alif

How to use tracing system

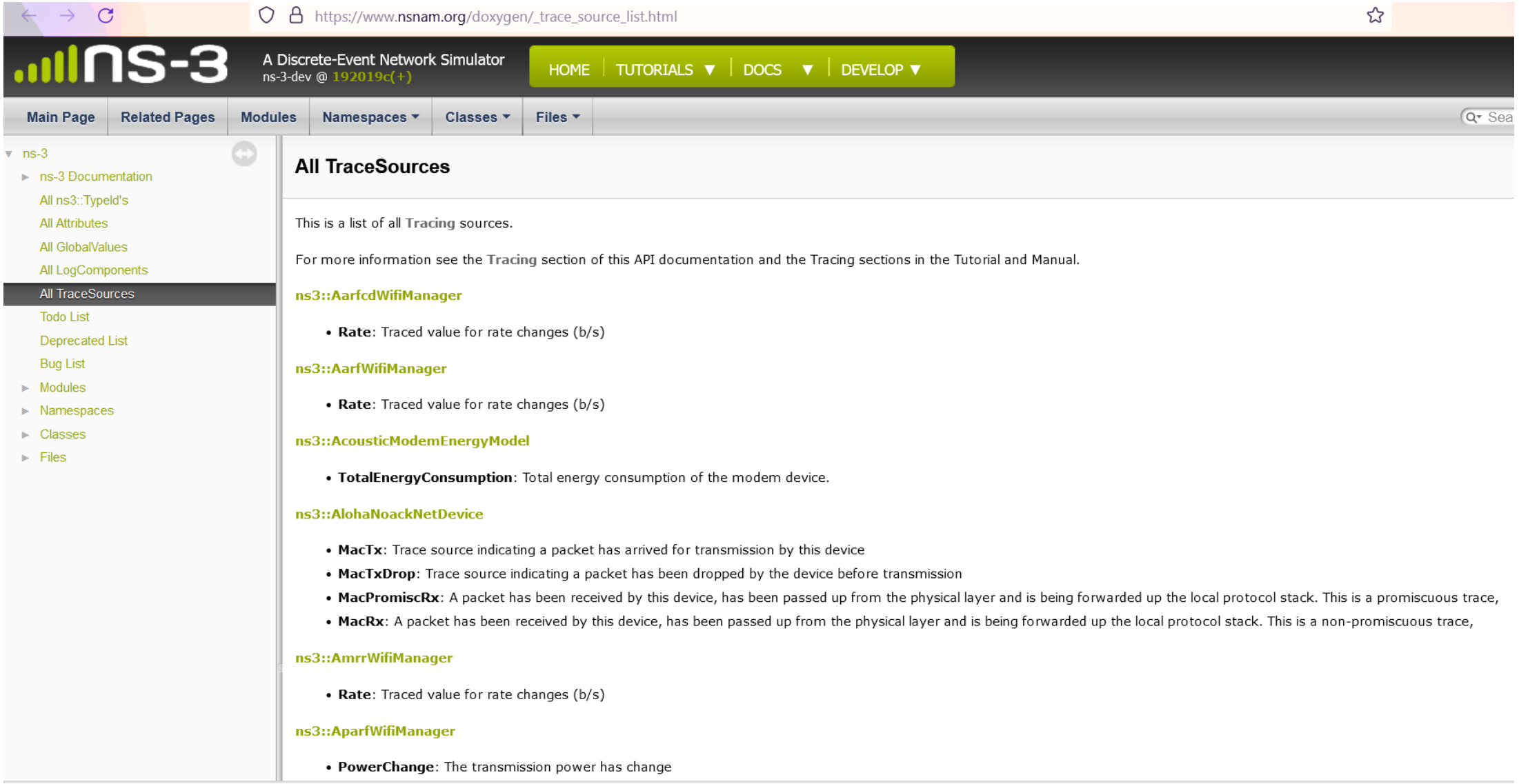
- Find trace source
- Find config path
- Find return type and formal arguments of callback function

How to find trace source

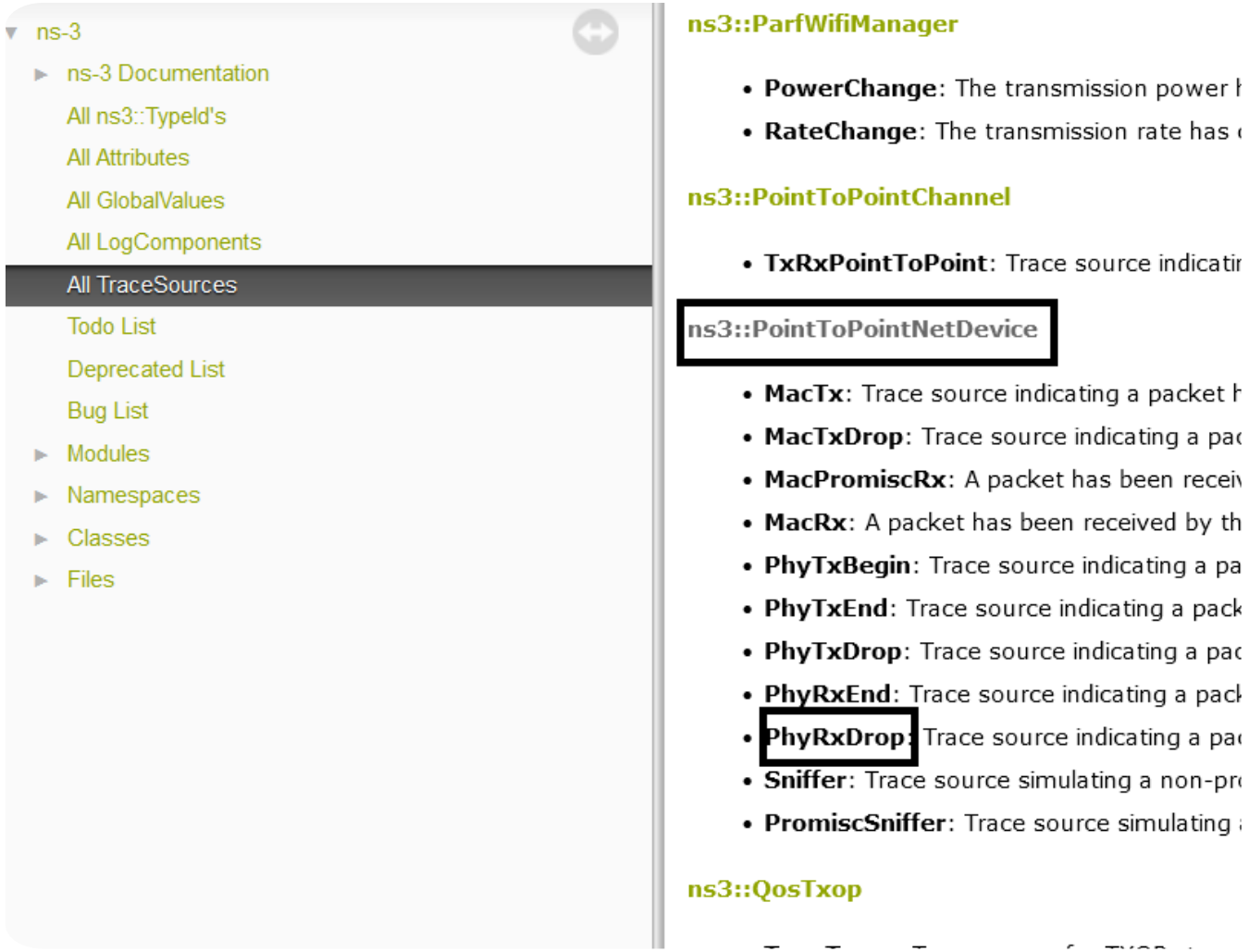


- Goto https://www.nsnam.org/doxygen/
- Click on the 'All TraceSources'
- You will see the all the available trace source in ns3

How to find trace source – List of TraceSoucres



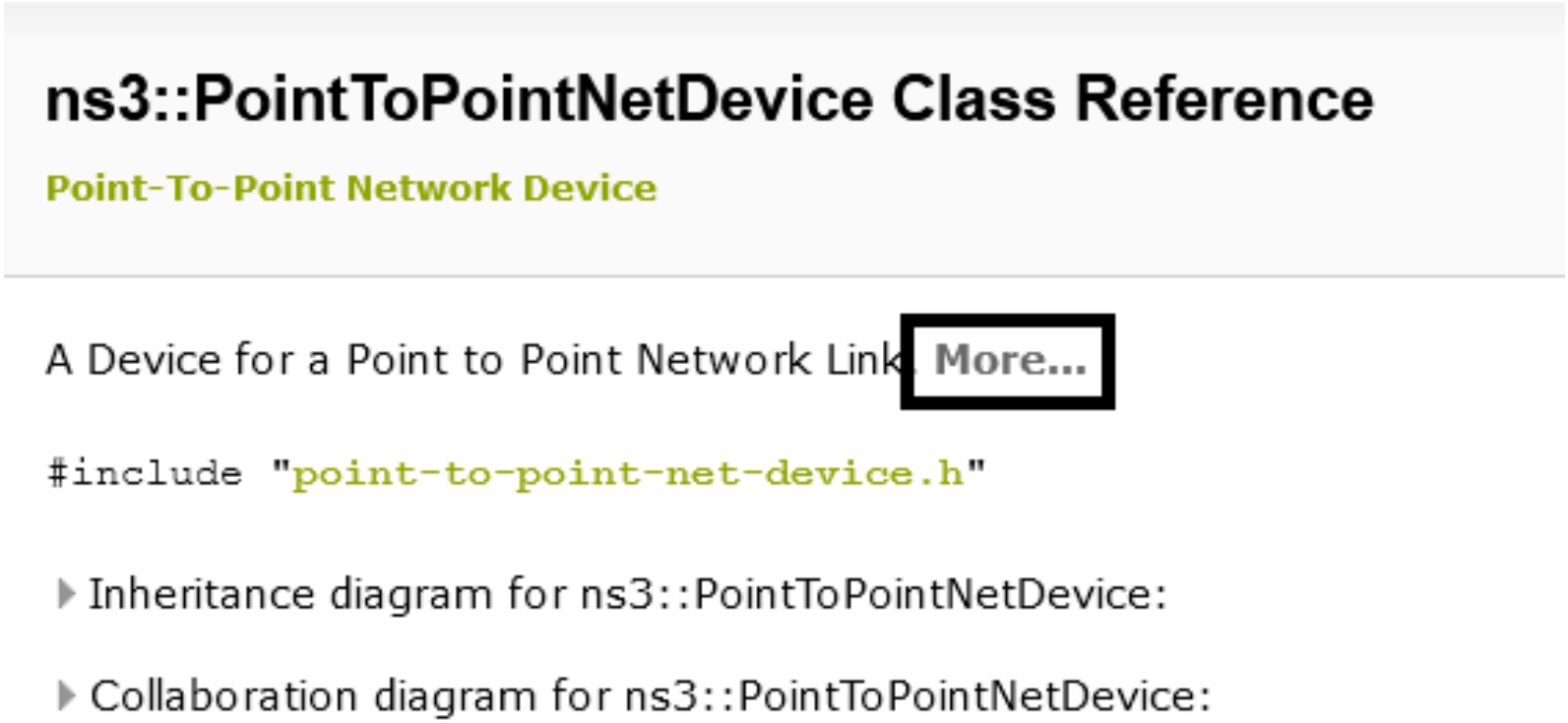
How to find config path



- Say, We want to use **PhyRxDrop trace source in ns3::PointToPointNetDevice** class

How to find config path

- Click on **ns3::PointToPointNetDevice** and click on "**More...**" Or scroll below



How to find config path

Detailed Description

A Device for a Point to Point Network Link.

This **PointToPointNetDevice** class specializes the **NetDevice** abstract base class. Together with a **PointToPointChannel** (and level of abstraction, a generic point-to-point or serial link. Key parameters or objects that can be specified for this device include propagation delay is set in the **PointToPointChannel**).

Config Paths

ns3::PointToPointNetDevice is accessible through the following paths with **Config::Set** and **Config::Connect**:

- "/NodeList/[i]/DeviceList/[i]/$ns3::PointToPointNetDevice"

Attributes

- **Mtu**: The MAC-level Maximum Transmission Unit
  - Set with class: **ns3::UintegerValue**

Find Callback Signature

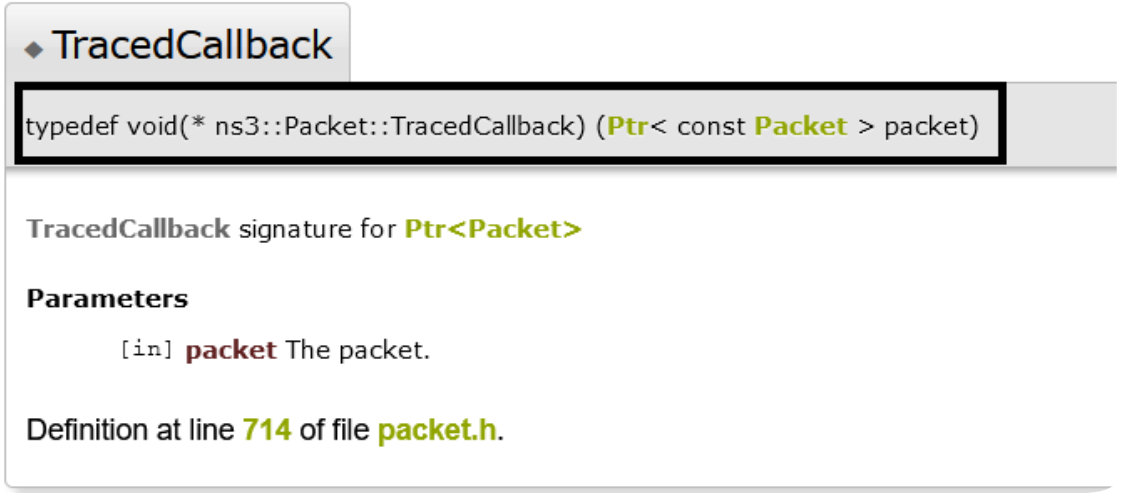
TraceSources

- **MacTx**: Trace source indicating a packet has arrived for tra
  Callback signature: **ns3::Packet::TracedCallback**
- **MacTxDrop**: Trace source indicating a packet has been dro
  Callback signature: **ns3::Packet::TracedCallback**
- **MacPromiscRx**: A packet has been received by this device
  Callback signature: **ns3::Packet::TracedCallback**
- **MacRx**: A packet has been received by this device, has bee
  Callback signature: **ns3::Packet::TracedCallback**
- **PhyTxBegin**: Trace source indicating a packet has begun tr
  Callback signature: **ns3::Packet::TracedCallback**
- **PhyTxEnd**: Trace source indicating a packet has been com
  Callback signature: **ns3::Packet::TracedCallback**
- **PhyTxDrop**: Trace source indicating a packet has been dro
  Callback signature: **ns3::Packet::TracedCallback**
- **PhyRxEnd**: Trace source indicating a packet has been com
  Callback signature: **ns3::Packet::TracedCallback**
- **PhyRxDrop**: Trace source indicating a packet has been dro
  Callback signature: **ns3::Packet::TracedCallback**
- **Sniffer**: Trace source simulating a non-promiscuous packet
  Callback signature: **ns3::Packet::TracedCallback**

- Scroll below and find **TraceSources**
- Below the line of **PhyRxDrop** find 'Callback signature'

Find Callback Signature

- Click on the link 'ns3::Packet::TracedCallback' and see typedef



If the callback signature hasn't been documented

```
TracedCallback<Ptr<const Packet> > m_macTxTrace;
TracedCallback<Ptr<const Packet> > m_macTxDropTrace;
TracedCallback<Ptr<const Packet> > m_macPromiscRxTrace;
TracedCallback<Ptr<const Packet> > m_macRxTrace;
TracedCallback<Ptr<const Packet> > m_macRxDropTrace;
TracedCallback<Ptr<const Packet> > m_phyTxBeginTrace;
TracedCallback<Ptr<const Packet> > m_phyTxEndTrace;
TracedCallback<Ptr<const Packet> > m_phyTxDropTrace;
TracedCallback<Ptr<const Packet> > m_phyRxBeginTrace;
TracedCallback<Ptr<const Packet> > m_phyRxEndTrace;
TracedCallback<Ptr<const Packet> > m_phyRxDropTrace;
TracedCallback<Ptr<const Packet> > m_snifferTrace;
TracedCallback<Ptr<const Packet> > m_promiscSnifferTrace;
```

- The return value of your callback will always be `void`
- The formal parameter list for a `TracedCallback` can be found from the template parameter list in the declaration
- For point-to-point-net-device.h, we have :
- This tells the function needs Ptr<const Packet> as parameter

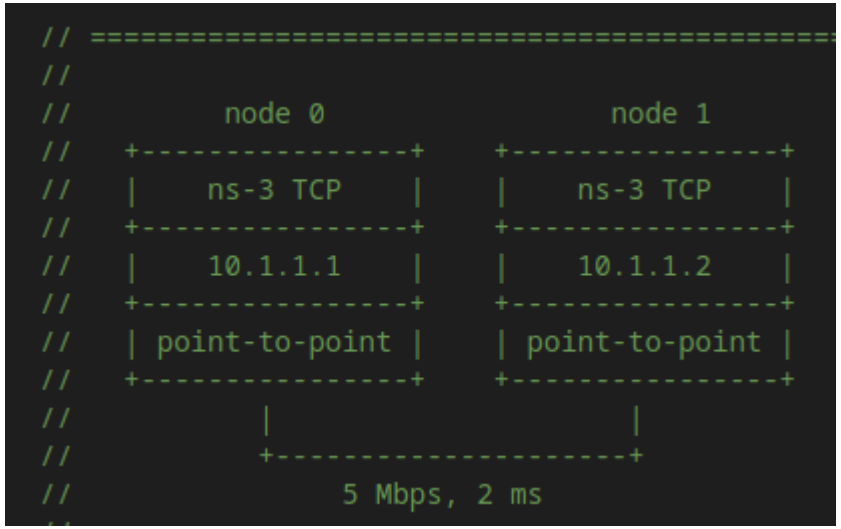
Example – Callback

```
static void
RxDrop (Ptr<const Packet> p)
{
  NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
}
```

Config && Sink-source link

```
std::string config_path = "/NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/";
Config::ConnectWithoutContext(config_path + "PhyRxDrop", MakeCallback(&RxDrop));
```

Building a Topology




```
225   NodeContainer nodes;
226   nodes.Create (2);
227
228   PointToPointHelper pointToPoint;
229   pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
230   pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
231
232   NetDeviceContainer devices;
233   devices = pointToPoint.Install (nodes);
234
235   Ptr<RateErrorModel> em = CreateObject<RateErrorModel> ();
236   em->SetAttribute ("ErrorRate", DoubleValue (0.00001));
237   devices.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));
238
239   InternetStackHelper stack;
240   stack.Install (nodes);
241
242   Ipv4AddressHelper address;
243   address.SetBase ("10.1.1.0", "255.255.255.252");
244   Ipv4InterfaceContainer interfaces = address.Assign (devices);
245
```

Declaration of Some Global Variables

```
185   //added by afnan
186   int packets_sent = 0;
187   int packets_received = 0;
188   int packets_dropped = 0;
189
190   //data file to plot graph
191   AsciiTraceHelper asciiTraceHelper;
192   Ptr<OutputStreamWrapper> result = asciiTraceHelper.CreateFileStream ("result.data");
193
```

TraceSources: RxReceive

```
210
211  static void
212  RxReceive (Ptr<PcapFileWrapper> file, Ptr<const Packet> p)
213  {
214    packets_received++;
215    NS_LOG_UNCOND ("RxReceive at " << Simulator::Now ().GetSeconds ());
216    file->Write (Simulator::Now (), p);
217  }
218
```

TxSend

```
202
203  static void
204  TxSend (Ptr<PcapFileWrapper> file, Ptr<const Packet> p)
205  {
206    packets_sent++;
207    NS_LOG_UNCOND ("TxSend at " << Simulator::Now ().GetSeconds ());
208    file->Write (Simulator::Now (), p);
209  }
210
```

RxDrop

```
193
194  static void
195  RxDrop (Ptr<PcapFileWrapper> file, Ptr<const Packet> p)
196  {
197    packets_dropped++;
198    NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
199    *result->GetStream() << Simulator::Now ().GetSeconds () << " "<<packets_dropped<< "\n";
200    file->Write (Simulator::Now (), p);
201  }
202
```

TraceConnect CallBack Methods

```
268  devices.Get (1)->TraceConnectWithoutContext ("PhyRxDrop", MakeBoundCallback (&RxDrop, file));
269  devices.Get (1)->TraceConnectWithoutContext ("PhyRxEnd", MakeBoundCallback (&RxReceive, file));
270  devices.Get (0)->TraceConnectWithoutContext ("PhyTxEnd", MakeBoundCallback (&TxSend, file));
```

Delivery and Drop Ratio

```
276
277   NS_LOG_UNCOND("----------Delivery and Drop Ratio---------------");
278   NS_LOG_UNCOND("total packets sent : "<<packets_sent);
279   NS_LOG_UNCOND("total packets dropped : "<<packets_dropped);
280   NS_LOG_UNCOND("total packets received : "<<packets_received);
281   NS_LOG_UNCOND("Packet Delivery Ratio : "<<packets_received*1.0/packets_sent);
282   NS_LOG_UNCOND("Packet Drop Ratio : "<<packets_dropped*1.0/packets_sent);
283
```

Output

```
----------Delivery and Drop Ratio---------------
total packets sent : 2013
total packets dropped : 11
total packets received : 2002
Packet Delivery Ratio : 0.994536
Packet Drop Ratio : 0.00546448
afnan@PuranPC:~/Documents/ns-allinone-3.35/ns-3.35$
```

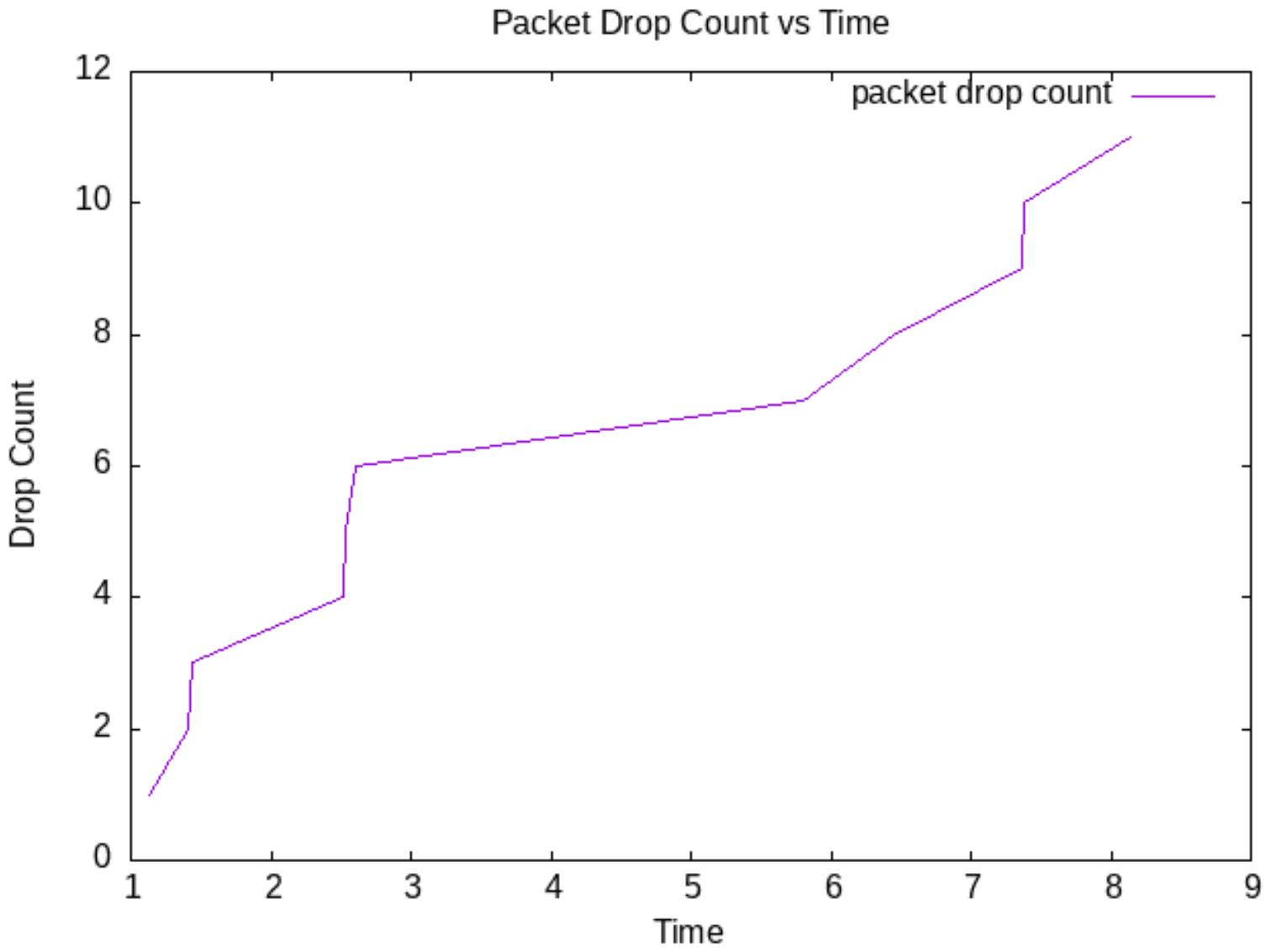
Plotting with Gnuplot

```
gnuplot> set terminal png

Terminal type is now 'png'
Options are 'nocrop enhanced size 640,480 font "arial,12.0" '
gnuplot> set output "result.png"
gnuplot> set xlabel "Time"
gnuplot> set ylabel "Drop Count"
gnuplot> plot "result.data" with lines title "packet drop count"
gnuplot>
```

Result.data

1	1.13696 1
2	1.4032 2
3	1.43648 3
4	2.5255 4
5	2.53472 5
6	2.60038 6
7	5.79616 7
8	6.44512 8
9	7.35942 9
10	7.38528 10
11	8.1415 11
12	

Drop Count vs Time



ASCII trace file (`.tr`)

- Necessary class `AsciiTraceHelper`
- Relevant function

```
void EnableAsciiAll (Ptr< OutputStreamWrapper > stream)
```

```
231    AsciiTraceHelper ascii;
232    phy.EnableAsciiAll(ascii.CreateFileStream("phy.tr"));
       .  .  .  .  .
251    Simulator::Run ();
```

ASCII trace file (`.tr`)

- Each line in the file corresponds to a trace event
- `+`: An enqueue operation occurred on the device queue;
- `-`: A dequeue operation occurred on the device queue;
- `d`: A packet was dropped, typically because the queue was full;
- `r`: A packet was received by the net device.

ASCII trace file (.tr)

- sample file example

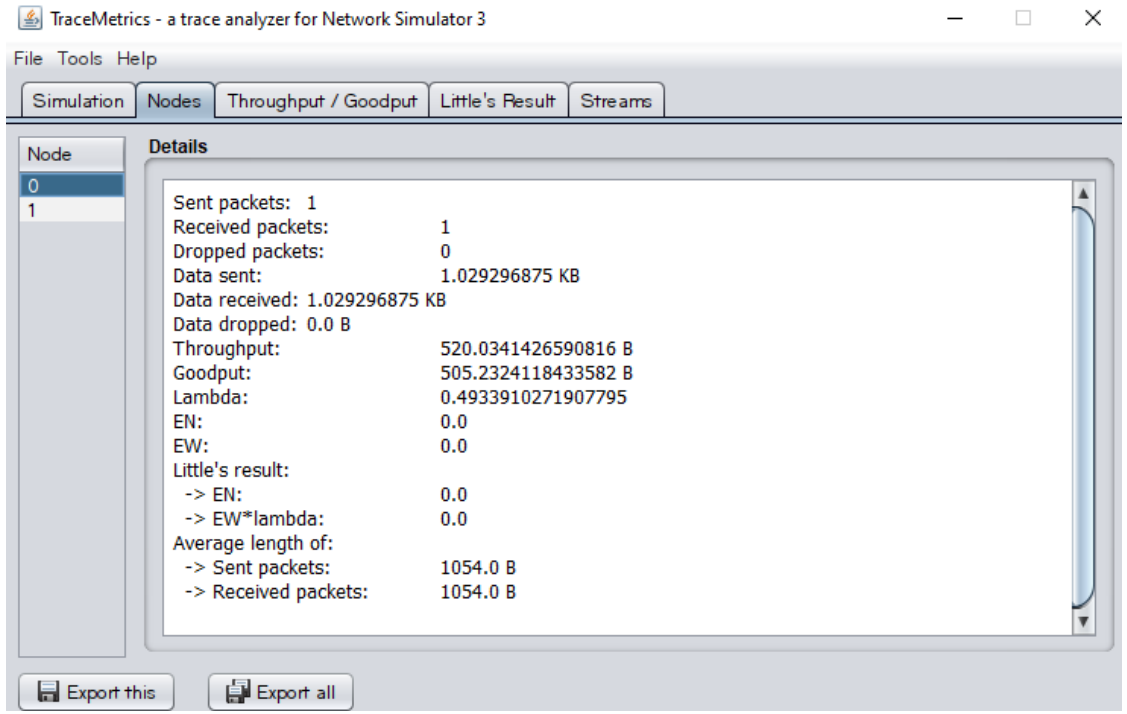
```
+
2.00819
/NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Enqueue
ns3::PppHeader (
    Point-to-Point Protocol: IP (0x0021))
    ns3::Ipv4Header (
        tos 0x0 DSCP Default ECN Not-ECT ttl 63 id 0 protocol 17 offset (bytes) 0 flags [none]
        length: 1052 10.1.3.3 > 10.1.2.4)
        ns3::UdpHeader (
            length: 1032 49153 > 9)
            Payload (size=1024)
```

```
r
2.01187
/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacRx
ns3::PppHeader (
    Point-to-Point Protocol: IP (0x0021))
    ns3::Ipv4Header (
        tos 0x0 DSCP Default ECN Not-ECT ttl 63 id 0 protocol 17 offset (bytes) 0 flags [none]
        length: 1052 10.1.3.3 > 10.1.2.4)
        ns3::UdpHeader (
            length: 1032 49153 > 9)
            Payload (size=1024)
```

```
d
1.13696
/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/PhyRxDrop
ns3::PppHeader (
    Point-to-Point Protocol: IP (0x0021))
    ns3::Ipv4Header (
        tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 35 protocol 6 offset (bytes) 0 flags [none]
        length: 556 10.1.1.1 > 10.1.1.2)
        ns3::TcpHeader (
            49153 > 8080 [ACK] Seq=17177 Ack=1 Win=32768
            ns3::TcpOptionTS(1133;1127)
            ns3::TcpOptionEnd(EOL))
            Payload Fragment [536:1040]
```

TraceMetrics - ASCII trace files analyzer

- Java-based trace file analyzer for ns-3
- run command `java -jar tracemetrics.jar`



Example file used `p2p.tr` generated from simulation of `first.cc`

Pcap trace file (.pcap)

```
233 |   pointToPoint.EnablePcapAll("myfirst");
    |   .  .  .  .  .
251 |   Simulator::Run ();
```

myfirst-0-0.pcap

myfirst-1-0.pcap

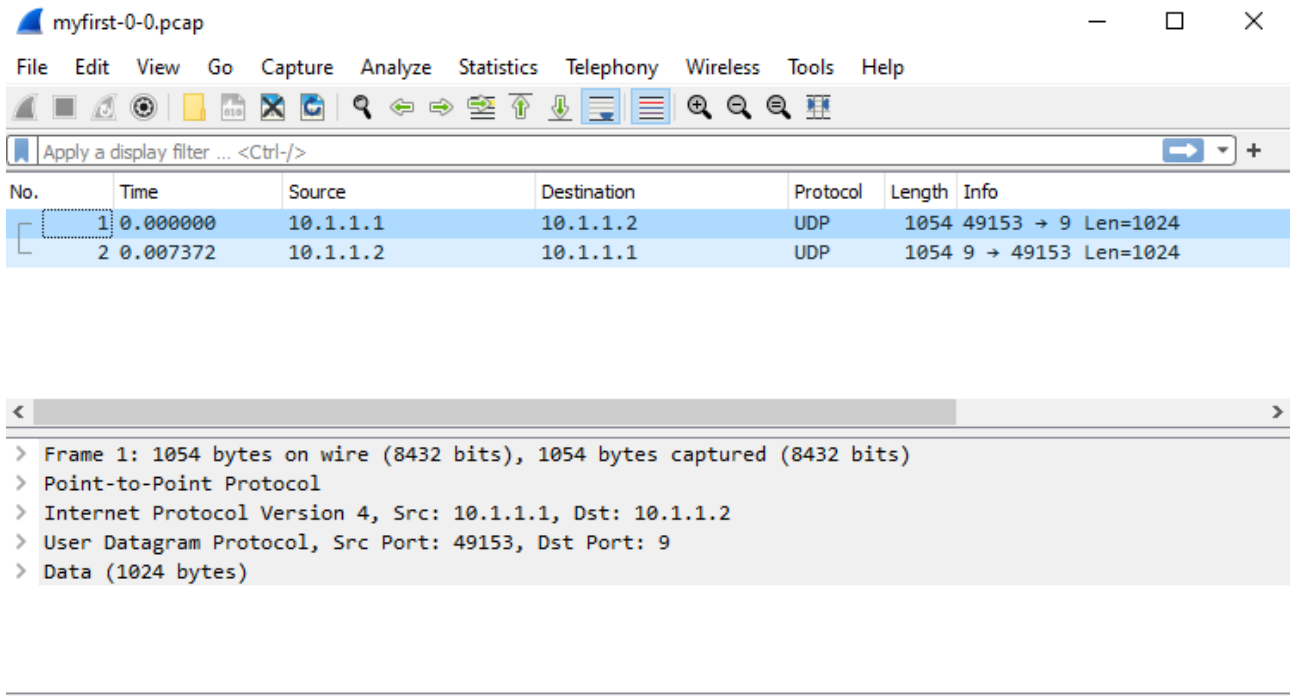
Reading `.pcap` file with tcpdump

- run command `tcpdump -nn -tt -r myfirst-0-0.pcap`

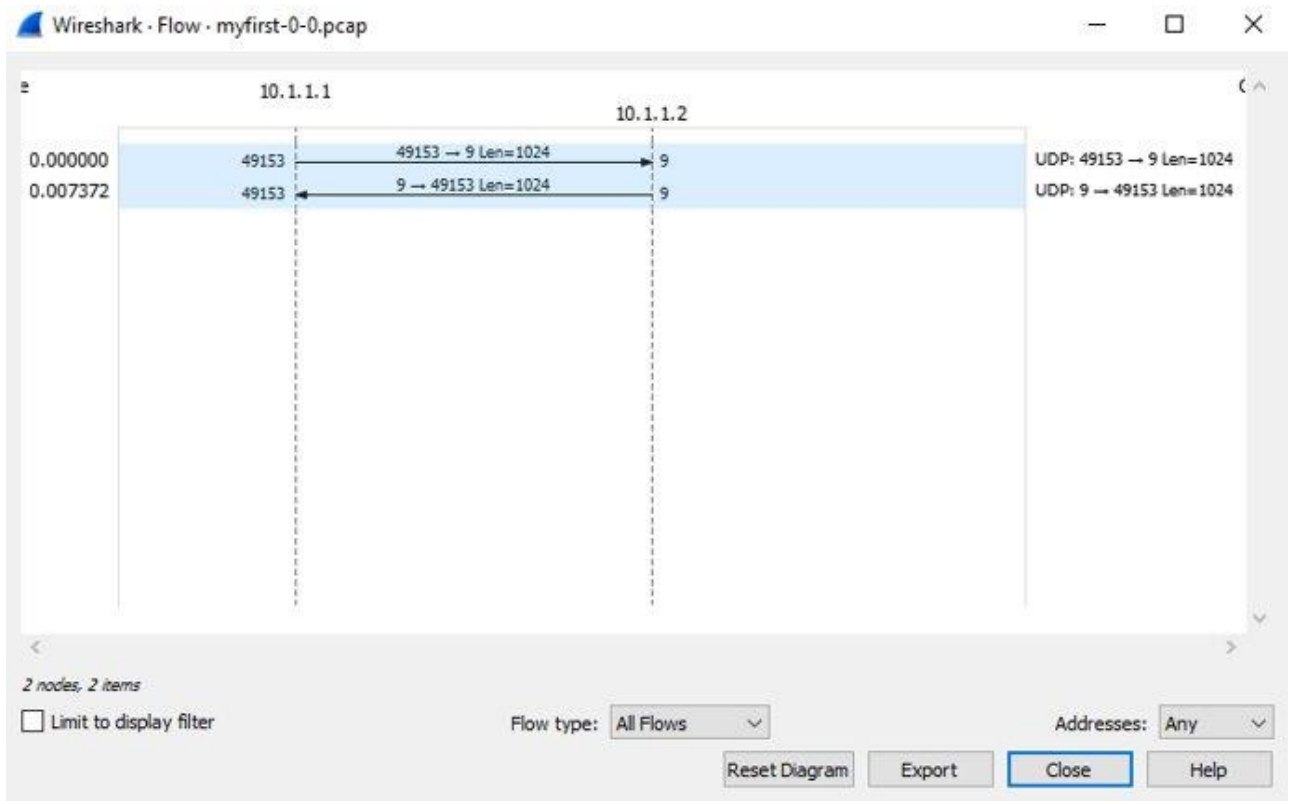
```
Reading from file myfirst-0-0.pcap, link-type EN10MB (Ethernet)
2.000000 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
2.007372 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
```

```
Reading from file myfirst-1-0.pcap, link-type EN10MB (Ethernet)
2.003686 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
2.003686 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
```

Reading .pcap file with Wireshark



Reading .pcap file with Wireshark



Flow Monitor

- Including the Header File

```
#include "ns3/flow-monitor-module.h"
```

- Initialization

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

Simulator::Run ();
```

FlowStat

/src/flow-monitor/model/flow-monitor.h

```
class FlowMonitor : public Object
{
public:

  struct FlowStats
  {
    Time     timeFirstTxPacket;

    Time     timeFirstRxPacket;

    Time     timeLastTxPacket;

    Time     timeLastRxPacket;

    Time     delaySum; // delayCount == rxPackets

    Time     jitterSum; // jitterCount == rxPackets - 1

    Time     lastDelay;

    uint64_t txBytes;
    uint64_t rxBytes;
    uint32_t txPackets;
    uint32_t rxPackets;

    uint32_t lostPackets;

    uint32_t timesForwarded;

    Histogram delayHistogram;
    Histogram jitterHistogram;
    Histogram packetSizeHistogram;

    std::vector<uint32_t> packetsDropped; // packetsDropped[reasonCode] => number of dropped packets

    std::vector<uint64_t> bytesDropped; // bytesDropped[reasonCode] => number of dropped bytes
    Histogram flowInterruptionsHistogram;
  };
```

Get Stats

```
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier());
std::map<FlowId, FlowMonitor::FlowStats> stats =  monitor->GetFlowStats();
```

/src/flow-monitor/model/flow-monitor.cc

```
const FlowMonitor::FlowStatsContainer&
FlowMonitor::GetFlowStats () const
{
  return m_flowStats;
}
```

/src/flow-monitor/model/flow-monitor.h

```
private:

  /// Structure to represent a single tracked packet data
  struct TrackedPacket
  {
    Time firstSeenTime; //!< absolute time when the packet was first seen
    Time lastSeenTime; //!< absolute time when the packet was last seen by
    uint32_t timesForwarded; //!< number of times the packet was reportedl
  };

  /// FlowId --> FlowStats
  FlowStatsContainer m_flowStats;

  /// (FlowId,PacketId) --> TrackedPacket
```

/src/flow-monitor/model/flow-monitor.h

```
  // --- methods to get the results ---

  /// Container: FlowId, FlowStats
  typedef std::map<FlowId, FlowStats> FlowStatsContainer;
  /// Container Iterator: FlowId, FlowStats
  typedef std::map<FlowId, FlowStats>::iterator FlowStatsContainerI;
  /// Container Const Iterator: FlowId, FlowStats
  typedef std::map<FlowId, FlowStats>::const_iterator FlowStatsContainerCI;
  /// Container: FlowProbe
  typedef std::vector< Ptr<FlowProbe> > FlowProbeContainer;
  /// Container Iterator: FlowProbe
  typedef std::vector< Ptr<FlowProbe> >::iterator FlowProbeContainerI;
  /// Container Const Iterator: FlowProbe
  typedef std::vector< Ptr<FlowProbe> >::const_iterator FlowProbeContainerCI;
```

Five Tuple

```
Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(iter->first);
```

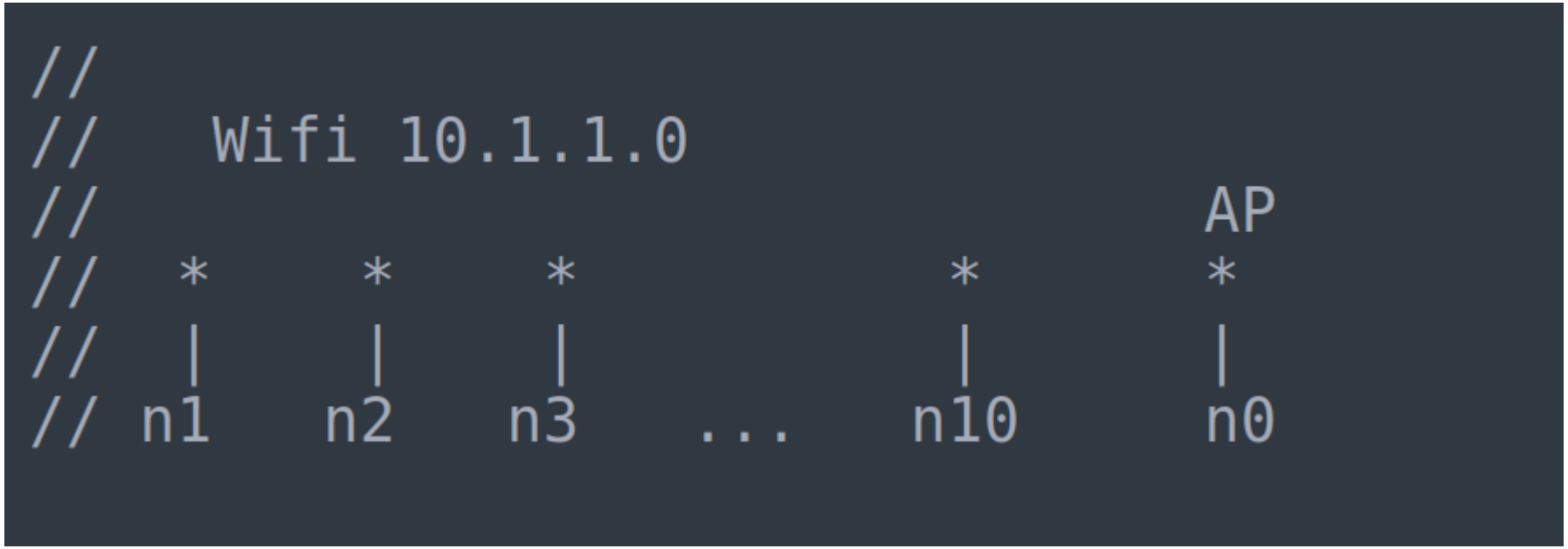
./src/flow-monitor/model/ipv4-flow-classifier.h

```
class Ipv4FlowClassifier : public FlowClassifier
{
public:

  /// Structure to classify a packet
  struct FiveTuple
  {
    Ipv4Address sourceAddress;      //!< Source address
    Ipv4Address destinationAddress; //!< Destination address
    uint8_t protocol;               //!< Protocol
    uint16_t sourcePort;            //!< Source port
    uint16_t destinationPort;       //!< Destination port
  };
```

Calculation

```
// ----------------------- Network Performance Calculation ----------------------- //

uint32_t sentPackets = 0;
uint32_t receivedPackets = 0;
uint32_t lostPackets = 0;

int j = 0;
float avgThroughput = 0;
Time jitter;
Time delay;

Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier());
std::map<FlowId, FlowMonitor::FlowStats> stats =  monitor->GetFlowStats();

for(std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin(); iter != stats.end(); iter++)
{
  Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(iter->first);

  NS_LOG_UNCOND("\nFlow Id: " << iter->first);
  NS_LOG_UNCOND("Src Addr: " << t.sourceAddress);
  NS_LOG_UNCOND("Dst Addr: " << t.destinationAddress);
  NS_LOG_UNCOND("Sent Packets: " << iter->second.txPackets);
  NS_LOG_UNCOND("Received Packets: " << iter->second.rxPackets);
  NS_LOG_UNCOND("Lost Packets: " << iter->second.txPackets - iter->second.rxPackets);
  NS_LOG_UNCOND("Packet Delivery Ratio: " << iter->second.rxPackets*100/iter->second.txPackets << "%");
  NS_LOG_UNCOND("Packet Loss Ratio: " << (iter->second.txPackets - iter->second.rxPackets)*100/iter->second.txPackets << "%");
  NS_LOG_UNCOND("Delay: " << iter->second.delaySum);
  NS_LOG_UNCOND("Jitter: " << iter->second.jitterSum);
  NS_LOG_UNCOND("Throughput: " << iter->second.rxBytes * 8.0 /(iter->second.timeLastRxPacket.GetSeconds() - iter->second.timeFirstTxPacket.GetSeconds())/1024 << " kbps");


  sentPackets += iter->second.txPackets;
  receivedPackets += iter->second.rxPackets;
  lostPackets += (iter->second.txPackets - iter->second.rxPackets);
  avgThroughput += iter->second.rxBytes * 8.0 /(iter->second.timeLastRxPacket.GetSeconds() - iter->second.timeFirstTxPacket.GetSeconds())/1024;
  delay += iter->second.delaySum;
  jitter += iter->second.jitterSum;

  j++;
}

avgThroughput = avgThroughput/j;
NS_LOG_UNCOND("\n--------------- Simulation Stats ------------"<<std::endl);
NS_LOG_UNCOND("Total Sent packets: " << sentPackets);
NS_LOG_UNCOND("Total Received Packets: " << receivedPackets);
NS_LOG_UNCOND("Total Lost Packets: " << lostPackets);
NS_LOG_UNCOND("Packet Loss Ratio: " << lostPackets*100/sentPackets << "%");
NS_LOG_UNCOND("Packet Delivery Ratio: " << receivedPackets * 100 /sentPackets << "%");
NS_LOG_UNCOND("Average Throughput: " << avgThroughput << " kbps");
NS_LOG_UNCOND("End to end delay: "<< delay);
NS_LOG_UNCOND("End to end jitter delay: "<< jitter);
NS_LOG_UNCOND("Total Flow ID: " << j);

monitor->SerializeToXmlFile("test_flow.xml", true, true);

Simulator::Destroy ();
return 0;
}
```

Topology



Output

```
Lost Packets: 0
Packet Delivery Ratio: 100%
Packet Loss Ratio: 0%
Delay: +6.14911e+07ns
Jitter: +4.68838e+07ns
Throughput: 20.2195 kbps

Flow Id: 18
Src Addr: 10.1.1.11
Dst Addr: 10.1.1.9
Sent Packets: 5
Received Packets: 5
Lost Packets: 0
Packet Delivery Ratio: 100%
Packet Loss Ratio: 0%
Delay: +7.64864e+07ns
Jitter: +4.5061e+07ns
Throughput: 20.3434 kbps

Flow Id: 19
Src Addr: 10.1.1.11
Dst Addr: 10.1.1.7
Sent Packets: 5
Received Packets: 5
Lost Packets: 0
Packet Delivery Ratio: 100%
Packet Loss Ratio: 0%
Delay: +9.45941e+07ns
Jitter: +6.57116e+07ns
Throughput: 20.2099 kbps

---------- Simulation Stats ------------

Total Sent packets: 95
Total Received Packets: 90
Total Lost Packets: 5
Packet Loss Ratio: 5%
Packet Delivery Ratio: 94%
Average Throughput: 19.1216 kbps
End to end delay: +1.9497e+09ns
End to end jitter delay: +1.05261e+09ns
Total Flow ID: 19
```

Xml File

```
monitor->SerializeToXmlFile("test_flow.xml", true, true);
```

test_flow.xml

```
<?xml version="1.0" ?>
<FlowMonitor>
  <FlowStats>
    <Flow flowId="1" timeFirstTxPacket="+2e+09ns" timeFirstRxPacket="+2.00974e+09ns" timeLastTxPacket="+6e+09ns" timeLastRxPacket="
    +6.02364e+09ns" delaySum="+1.73126e+08ns" jitterSum="+1.13757e+08ns" lastDelay="+2.36378e+07ns" txBytes="10380" rxBytes="10380"
    txPackets="5" rxPackets="5" lostPackets="0" timesForwarded="0">
      <delayHistogram nBins="74" >
        <bin index="9" start="0.009" width="0.001" count="1" />
        <bin index="20" start="0.02" width="0.001" count="1" />
        <bin index="23" start="0.023" width="0.001" count="1" />
        <bin index="45" start="0.045" width="0.001" count="1" />
        <bin index="73" start="0.073" width="0.001" count="1" />
      </delayHistogram>
      <jitterHistogram nBins="50" >
        <bin index="11" start="0.011" width="0.001" count="1" />
        <bin index="24" start="0.024" width="0.001" count="1" />
        <bin index="28" start="0.028" width="0.001" count="1" />
        <bin index="49" start="0.049" width="0.001" count="1" />
      </jitterHistogram>
      <packetSizeHistogram nBins="104" >
        <bin index="103" start="2060" width="20" count="5" />
      </packetSizeHistogram>
      <flowInterruptionsHistogram nBins="5" >
        <bin index="3" start="0.75" width="0.25" count="1" />
        <bin index="4" start="1" width="0.25" count="3" />
      </flowInterruptionsHistogram>
    </Flow>
    <Flow flowId="2" timeFirstTxPacket="+2e+09ns" timeFirstRxPacket="+2.02433e+09ns" timeLastTxPacket="+6e+09ns" timeLastRxPacket="
    +6.05061e+09ns" delaySum="+1.77678e+08ns" jitterSum="+4.51575e+07ns" lastDelay="+5.06098e+07ns" txBytes="10380" rxBytes="10380"
    txPackets="5" rxPackets="5" lostPackets="0" timesForwarded="0">
      <delayHistogram nBins="51" >
        <bin index="14" start="0.014" width="0.001" count="1" />
        <bin index="24" start="0.024" width="0.001" count="1" />
        <bin index="39" start="0.039" width="0.001" count="1" />
        <bin index="48" start="0.048" width="0.001" count="1" />
        <bin index="50" start="0.05" width="0.001" count="1" />
      </delayHistogram>
      <jitterHistogram nBins="25" >
        <bin index="2" start="0.002" width="0.001" count="1" />
        <bin index="9" start="0.009" width="0.001" count="2" />
        <bin index="24" start="0.024" width="0.001" count="1" />
      </jitterHistogram>
      <packetSizeHistogram nBins="104" >
        <bin index="103" start="2060" width="20" count="5" />
      </packetSizeHistogram>
      <flowInterruptionsHistogram nBins="5" >
        <bin index="3" start="0.75" width="0.25" count="1" />
        <bin index="4" start="1" width="0.25" count="3" />
      </flowInterruptionsHistogram>
```

Parsing the XML file

flowmon-parse-results.py

```
238
239
240     for sim in sim_list:
241         for flow in sim.flows:
242
243             t = flow.fiveTuple
244             proto = {6: 'TCP', 17: 'UDP'} [t.protocol]
245             print("FlowID: %i (%s %s/%s --> %s/%i)" % \
246                 (flow.flowId, proto, t.sourceAddress, t.sourcePort, t.destinationAddress, t.destinationPort))
247
248             print("\tTransmitted Packets: %i" % flow.txPackets)
249             print("\tReceived Packets: %i" % flow.rxPackets)
250
251             if flow.txBitrate is None:
252                 print("\tTX bitrate: None")
253             else:
254                 print("\tTX bitrate: %.2f kbit/s" % (flow.txBitrate*1e-3,))
255
256             if flow.rxBitrate is None:
257                 print("\tRX bitrate: None")
258             else:
259                 print("\tRX bitrate: %.2f kbit/s" % (flow.rxBitrate*1e-3,))
260
261             if flow.delayMean is None:
262                 print("\tMean Delay: None")
263             else:
264                 print("\tMean Delay: %.2f ms" % (flow.delayMean*1e3,))
265
266             if flow.packetLossRatio is None:
267                 print("\tPacket Loss Ratio: None")
268             else:
269                 print("\tPacket Loss Ratio: %.2f %%" % (flow.packetLossRatio*100))
270
271             if flow.packetDeliveryRatio is None:
272                 print("\tPacket Delivery Ratio: None")
273             else:
274                 print("\tPacket Delivery Ratio: %.2f %%" % (flow.packetDeliveryRatio*100))
275
276             print("\n")
277
278     print("\n")
279     print("-------------- Simulation Stats ---------------")
280     print("Total Transmitted Packets: %d" % total_tx)
281     print("Total Receieved Packets: %d" % total_rx)
282     print("Total Lost Packets: %d" % total_ls)
283     print("Total Packet Delivery Ratio: %.2f %%" % (total_rx/(total_rx + total_ls)*100))
284     print("Total Packet Drop Ratio: %.2f %%" % (total_ls/(total_rx + total_ls)*100))
285
286
```

$ python3 flowmon-parse-results.py test_flow.xml

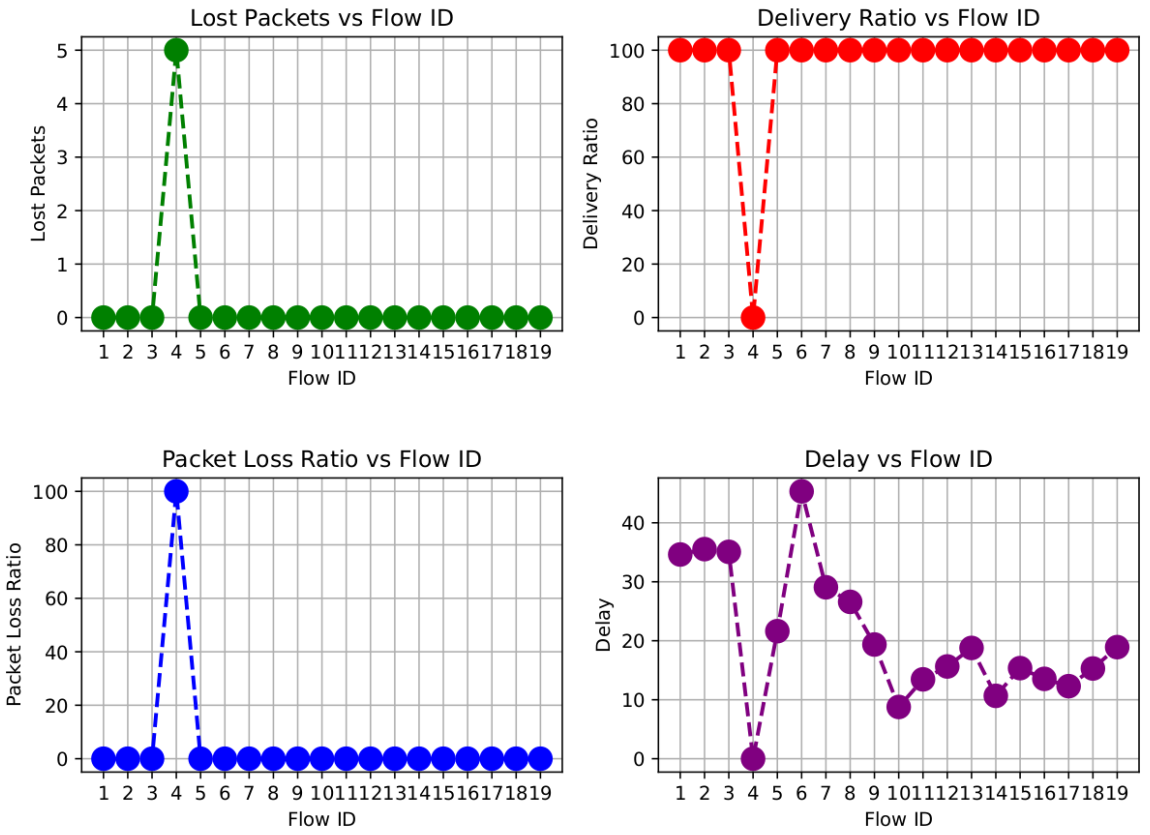
```
        Packet Loss Ratio: 0.00 %
        Packet Delivery Ratio: 100.00 %


FlowID: 17 (UDP 10.1.1.11/1234 --> 10.1.1.6/49153)
        Transmitted Packets: 5
        Received Packets: 5
        TX bitrate: 20.80 kbit/s
        RX bitrate: 20.89 kbit/s
        Mean Delay: 12.30 ms
        Packet Loss Ratio: 0.00 %
        Packet Delivery Ratio: 100.00 %


FlowID: 18 (UDP 10.1.1.11/1234 --> 10.1.1.9/49153)
        Transmitted Packets: 5
        Received Packets: 5
        TX bitrate: 20.88 kbit/s
        RX bitrate: 20.98 kbit/s
        Mean Delay: 15.30 ms
        Packet Loss Ratio: 0.00 %
        Packet Delivery Ratio: 100.00 %


FlowID: 19 (UDP 10.1.1.11/1234 --> 10.1.1.7/49153)
        Transmitted Packets: 5
        Received Packets: 5
        TX bitrate: 20.77 kbit/s
        RX bitrate: 20.85 kbit/s
        Mean Delay: 18.92 ms
        Packet Loss Ratio: 0.00 %
        Packet Delivery Ratio: 100.00 %



-------------- Simulation Stats ---------------
Total Transmitted Packets: 95
Total Receieved Packets: 90
Total Lost Packets: 5
Total Packet Delivery Ratio: 94.74 %
Total Packet Drop Ratio: 5.26 %
alif@alif-X510UQR:~/Work/CSE_322_Networking/NS3/ns-allinone-3.35/ns-3.35$
```

Plotting Graph

plot_flow.py

```
    if tpt.get('destinationPort')== 654 :
        continue

    sent_packets = int(flow.get('txPackets'))
    received_packets = int(flow.get('rxPackets'))
    received_delay_total = float(flow.get('delaySum')[:-2])*1e-9

    flow_id.append(flow.get('flowId'))
    lost_packets.append(sent_packets - received_packets)
    delays.append(float(received_delay_total / (received_packets + 0.00001)) * 1000)    # 0.00001 is added to avoid division by
    delivery_ratio.append(float(received_packets / sent_packets) * 100)
    packet_loss_ratio.append(float((sent_packets - received_packets) / (sent_packets)) * 100)

fig, axs = plt.subplots(2, 2, figsize=(10, 7))

axs[0, 0].plot(flow_id, lost_packets, color='green', marker='o', linestyle='dashed', linewidth=2, markersize=12)
axs[0, 0].grid()
axs[0, 0].set_title('Lost Packets vs Flow ID')
axs[0, 0].set(xlabel="Flow ID", ylabel="Lost Packets")

axs[0, 1].plot(flow_id, delivery_ratio, color='red', marker='o', linestyle='dashed', linewidth=2, markersize=12)
axs[0, 1].grid()
axs[0, 1].set_title('Delivery Ratio vs Flow ID')
axs[0, 1].set(xlabel="Flow ID", ylabel="Delivery Ratio")

axs[1, 0].plot(flow_id, packet_loss_ratio, color='blue', marker='o', linestyle='dashed', linewidth=2, markersize=12)
axs[1, 0].grid()
axs[1, 0].set_title('Packet Loss Ratio vs Flow ID')
axs[1 ,0].set(xlabel="Flow ID", ylabel="Packet Loss Ratio")

axs[1, 1].plot(flow_id, delays, color='purple', marker='o', linestyle='dashed', linewidth=2, markersize=12)
axs[1, 1].grid()
axs[1, 1].set_title('Delay vs Flow ID')
axs[1, 1].set(xlabel="Flow ID", ylabel="Delay")

plt.subplots_adjust(hspace=0.5)
plt.savefig("stat.pdf")

print("Stat.pdf generated")
```

$ python3 plot_flow.py test_flow.xml



Thank you