DELETE CODE

Activities Text Editor Apr 21 16:04

Open 2th.cc ~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch Save

2th.cc lab2.cc

```
NetDeviceContainer devices;
devices = pointToPoint.Install(nodes);

InternetStackHelper stack;
stack.Install(nodes);

TrafficControlHelper tch;
tch.SetRootQueueDisc("ns3::RedQueueDisc");
QueueDiscContainer qdiscs = tch.Install(devices);

Ptr<QueueDisc> q = qdiscs.Get(1);
q->TraceConnectWithoutContext("PacketsInQueue", MakeCallback(&TcPacketsInQueueTrace));
Config::ConnectWithoutContext(
    "/NodeList/1/$ns3::TrafficControlLayer/RootQueueDiscList/0/SojournTime",
    MakeCallback(&SojournTimeTrace));

Ptr<NetDevice> nd = devices.Get(1);
Ptr<PointToPointNetDevice> ptpnd = DynamicCast<PointToPointNetDevice>(nd);
Ptr<Queue<Packet>> queue = ptpnd->GetQueue();
queue->TraceConnectWithoutContext("PacketsInQueue", MakeCallback(&DevicePacketsInQueueTrace));

Ipv4AddressHelper address;
address.SetBase("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign(devices);

// Flow
uint16_t port = 7;
Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
PacketSinkHelper packetSinkHelper(socketType, localAddress);
ApplicationContainer sinkApp = packetSinkHelper.Install(nodes.Get(0));

sinkApp.Start(Seconds(0.0));
sinkApp.Stop(Seconds(simulationTime + 0.1));

uint32_t payloadSize = 1448;
Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(payloadSize));
```

C++ Tab Width: 8 Ln 151, Col 1 INS

Activities Text Editor Apr 21 16:05

Open *2th.cc ~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch Save

*2th.cc lab2.cc

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

Simulator::Stop(Seconds(simulationTime + 5));
Simulator::Run();

Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets << " / " << stats[1].txBytes
          << std::endl;
std::cout << "  Offered Load: "
          << stats[1].txBytes * 8.0 /
                (stats[1].timeLastTxPacket.GetSeconds() -
                 stats[1].timeFirstTxPacket.GetSeconds()) /
                1000000
          << " Mbps" << std::endl;
std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets << " / " << stats[1].rxBytes
          << std::endl;
uint32_t packetsDroppedByQueueDisc = 0;
uint64_t bytesDroppedByQueueDisc = 0;
if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE_DISC)
{
    packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
    bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
}
std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc << " / "
          << bytesDroppedByQueueDisc << std::endl;
uint32_t packetsDroppedByNetDevice = 0;
uint64_t bytesDroppedByNetDevice = 0;
if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE)
{
    packetsDroppedByNetDevice = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE];
    bytesDroppedByNetDevice = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE];
}
std::cout << "  Packets/Bytes Dropped by NetDevice:   " << packetsDroppedByNetDevice << " / "
          << bytesDroppedByNetDevice << std::endl;
```

C++ Tab Width: 8 Ln 185, Col 39 INS

Activities Text Editor Apr 21 16:06

Open *2th.cc ~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch Save

*2th.cc lab2.cc

```cpp
    Simulator::Run();

    Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
    std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
    std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
    std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets << " / " << stats[1].txBytes
              << std::endl;

    std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets << " / " << stats[1].rxBytes
              << std::endl;
    uint32_t packetsDroppedByQueueDisc = 0;
    uint64_t bytesDroppedByQueueDisc = 0;
    if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE_DISC)
    {
        packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
        bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
    }
    std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc << " / "
              << bytesDroppedByQueueDisc << std::endl;
    uint32_t packetsDroppedByNetDevice = 0;
    uint64_t bytesDroppedByNetDevice = 0;
    if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE)
    {
        packetsDroppedByNetDevice = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE];
        bytesDroppedByNetDevice = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE];
    }
    std::cout << "  Packets/Bytes Dropped by NetDevice:   " << packetsDroppedByNetDevice << " / "
              << bytesDroppedByNetDevice << std::endl;
    std::cout << "  Throughput: "
              << stats[1].rxBytes * 8.0 /
                     (stats[1].timeLastRxPacket.GetSeconds() -
                      stats[1].timeFirstRxPacket.GetSeconds()) /
                     1000000
              << " Mbps" << std::endl;
    std::cout << "  Mean delay:   " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
              << std::endl;
    std::cout << "  Mean jitter:   " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
```

C++ Tab Width: 8 Ln 200, Col 55 INS

Activities Text Editor Apr 21 16:07

Open *2th.cc ~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch Save

*2th.cc lab2.cc

```cpp
    std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
    std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
    std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets << " / " << stats[1].txBytes
              << std::endl;

    std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets << " / " << stats[1].rxBytes
              << std::endl;

    std::cout << "  Throughput: "
              << stats[1].rxBytes * 8.0 /
                     (stats[1].timeLastRxPacket.GetSeconds() -
                      stats[1].timeFirstRxPacket.GetSeconds()) /
                     1000000
              << " Mbps" << std::endl;
    std::cout << "  Mean delay:   " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
              << std::endl;
    std::cout << "  Mean jitter:   " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
              << std::endl;
    auto dscpVec = classifier->GetDscpCounts(1);
    for (auto p : dscpVec)
    {
        std::cout << "  DSCP value:   0x" << std::hex << static_cast<uint32_t>(p.first) << std::dec
                  << "  count:   " << p.second << std::endl;
    }

    Simulator::Destroy();

    std::cout << std::endl << "*** Application statistics ***" << std::endl;
    double thr = 0;
    uint64_t totalPacketsThr = DynamicCast<PacketSink>(sinkApp.Get(0))->GetTotalRx();
    thr = totalPacketsThr * 8 / (simulationTime * 1000000.0); // Mbit/s
    std::cout << "  Rx Bytes: " << totalPacketsThr << std::endl;
    std::cout << "  Average Goodput: " << thr << " Mbit/s" << std::endl;
    std::cout << std::endl << "*** TC Layer statistics ***" << std::endl;
    std::cout << q->GetStats() << std::endl;
    return 0;
}
```

C++ Tab Width: 8 Ln 200, Col 1 INS

Activities Text Editor Apr 21 16:07

Open *2th.cc ~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch Save

*2th.cc lab2.cc

```cpp

    Simulator::Stop(Seconds(simulationTime + 5));
    Simulator::Run();

    Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
    std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
    std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
    std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets << " / " << stats[1].txBytes
              << std::endl;

    std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets << " / " << stats[1].rxBytes
              << std::endl;

    std::cout << "  Throughput: "
              << stats[1].rxBytes * 8.0 /
                     (stats[1].timeLastRxPacket.GetSeconds() -
                      stats[1].timeFirstRxPacket.GetSeconds()) /
                     1000000
              << " Mbps" << std::endl;
    std::cout << "  Mean delay:   " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
              << std::endl;
    std::cout << "  Mean jitter:   " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
              << std::endl;
    auto dscpVec = classifier->GetDscpCounts(1);

    Simulator::Destroy();

    std::cout << std::endl << "*** Application statistics ***" << std::endl;
    double thr = 0;
    uint64_t totalPacketsThr = DynamicCast<PacketSink>(sinkApp.Get(0))->GetTotalRx();
    thr = totalPacketsThr * 8 / (simulationTime * 1000000.0); // Mbit/s
    std::cout << "  Rx Bytes: " << totalPacketsThr << std::endl;
    std::cout << "  Average Goodput: " << thr << " Mbit/s" << std::endl;
    std::cout << std::endl << "*** TC Layer statistics ***" << std::endl;
    std::cout << q->GetStats() << std::endl;
    return 0;
}
```

C++ Tab Width: 8 Ln 205, Col 45 INS

CHANGES IN CODE

```
std::string transportProt = "Udp";

nodes.Create(4);

NetDeviceContainer devices01;
devices01 = pointToPoint.Install(nodes.Get(0),nodes.Get(1));

NetDeviceContainer devices12;
devices12 = pointToPoint.Install(nodes.Get(1),nodes.Get(2));

NetDeviceContainer devices23;
devices23 = pointToPoint.Install(nodes.Get(2),nodes.Get(3));

//Ipv4AddressHelper address;
//address.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces01 = address.Assign(devices01);
address.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces12 = address.Assign(devices12);
address.SetBase("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces23 = address.Assign(devices23);
Ipv4GlobalRoutingHelper::PopulateRoutingTables();   // type these global routing



// Flow   ---> rename as UDP flow, copy these same code and paste after apps.stop() and make changes

//UDP Flow
InetSocketAddress rmt(interfaces01.GetAddress(0), port);      //change interfaces01
   apps.Add(onoff.Install(nodes.Get(3)));                    //change .Get(3)

// TCP Flow
uint16_t porttcp = 9;                                                                    //change to 9
socketType = "ns3::TcpSocketFactory";                                                   //add these line
Address localAddresstcp(InetSocketAddress(Ipv4Address::GetAny(), porttcp ));
PacketSinkHelper packetSinkHelpertcp (socketType, localAddresstcp);
ApplicationContainer sinkApptcp = packetSinkHelpertcp.Install(nodes.Get(0));

sinkApptcp.Start(Seconds(0.5));
sinkApptcp.Stop(Seconds(simulationTime + 0.1));
```

Delete 2 line code from here

```
OnOffHelper onofftcp(socketType, Ipv4Address::GetAny());
onofftcp.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
onofftcp.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
onofftcp.SetAttribute("PacketSize", UintegerValue(payloadSize));
onofftcp.SetAttribute("DataRate", StringValue("50Mbps")); // bit/s
ApplicationContainer appstcp;
```

```
InetSocketAddress rmttcp(interfaces01.GetAddress(0), porttcp);                    //change to interfaces01
rmttcp.SetTos(0xb8);
AddressValue remoteAddresstcp(rmttcp);
onofftcp.SetAttribute("Remote", remoteAddresstcp);
appstcp.Add(onofftcp.Install(nodes.Get(2)));                          //change to .Get(2)
appstcp.Start(Seconds(1.5));                                          // change to Seconds(1.5)
appstcp.Stop(Seconds(simulationTime + 0.1));



// write these code in the flow monitor statistics section
std::cout << "  Dropped Packets/Bytes:   " << stats[1].lostPackets
        << " / " << stats[1].rxBytes << std::endl;



//std::cout << "  Mean jitter:   " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
//         << std::endl;
//    auto dscpVec = classifier->GetDscpCounts(1);
//Add these for loop
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin (); iter != stats.end (); ++iter)
   {
     Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);
     std::cout << "Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst Addr " <<
t.destinationAddress<< std::endl;
     std::cout << "Tx Packets   = " << iter->second.txPackets<< std::endl;
     std::cout << "Rx Packets   = " << iter->second.rxPackets<< std::endl;
     std::cout << "Lost Packets = " << iter->second.lostPackets<< std::endl;
     std::cout << "Throughput   = " << iter->second.rxBytes * 8.0 / (iter->second.timeLastRxPacket.GetSeconds()-
iter->second.timeFirstTxPacket.GetSeconds()) / 1000000  << " Kbps"<< std::endl;
   }
```