DELETE CODE

```
    csmaInterfaces = address.Assign(csmaDevices);

    address.SetBase("10.1.3.0", "255.255.255.0");
    address.Assign(staDevices);
    address.Assign(apDevices);

    UdpEchoServerHelper echoServer(9);

    ApplicationContainer serverApps = echoServer.Install(csmaNodes.Get(nCsma));
    serverApps.Start(Seconds(1.0));
    serverApps.Stop(Seconds(10.0));

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

    ApplicationContainer clientApps = echoClient.Install(wifiStaNodes.Get(nWifi - 1));
    clientApps.Start(Seconds(2.0));
    clientApps.Stop(Seconds(10.0));

    Ipv4GlobalRoutingHelper::PopulateRoutingTables();

    Simulator::Stop(Seconds(10.0));

    if (tracing)
    {
        phy.SetPcapDataLinkType(WifiPhyHelper::DLT_IEEE802_11_RADIO);
        pointToPoint.EnablePcapAll("third");
        phy.EnablePcap("third", apDevices.Get(0));
        csma.EnablePcap("third", csmaDevices.Get(0), true);
    }

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

Delete these in traffic control and then paste it in lab3.cc after Simulator::Run() (it will be explained in next page)

```
    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;
    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;
```

CHANGES IN CODE

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

//copy code from traffic-control.cc and paste in lab3.cc and make changes
  // Flow
   uint16_t port = 7;
   Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
   PacketSinkHelper packetSinkHelper("ns3::UdpSocketFactory", localAddress);
   ApplicationContainer sinkApp = packetSinkHelper.Install(csmaNodes.Get(nCsma));

   sinkApp.Start(Seconds(0.0));
   sinkApp.Stop(Seconds(10 + 0.1));

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

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

   InetSocketAddress rmt(csmaInterfaces.GetAddress(nCsma), port);
   rmt.SetTos(0xb8);
   AddressValue remoteAddress(rmt);
   onoff.SetAttribute("Remote", remoteAddress);
   apps.Add(onoff.Install(wifiStaNodes.Get(nWifi-1)));
   apps.Start(Seconds(1.0));
   apps.Stop(Seconds(10 + 0.1));

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

//Again copy code from traffic-control.cc and paste after 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;
//add these line
   std::cout << "  lost Packets/Bytes:   " << stats[1].lostPackets
       << 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;
```