DELETE CODE

Activities Text Editor Apr 21 13:05

Open 1st.cc ~/Desktop Save

1st.cc *Untitled Document 1 traffic-control.cc

```
// in the device layer.

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("TrafficControlExample");

/**
 * Number of packets in TX queue trace.
 *
 * \param oldValue Old velue.
 * \param newValue New value.
 */
void
TcPacketsInQueueTrace(uint32_t oldValue, uint32_t newValue)
{
    std::cout << "TcPacketsInQueue " << oldValue << " to " << newValue << std::endl;
}

/**
 * Packets in the device queue trace.
 *
 * \param oldValue Old velue.
 * \param newValue New value.
 */
void
DevicePacketsInQueueTrace(uint32_t oldValue, uint32_t newValue)
{
    std::cout << "DevicePacketsInQueue " << oldValue << " to " << newValue << std::endl;
}

/**
 * TC Soujoun time trace.
 *
 * \param sojournTime The soujourn time.
 */
void
```

C++ Tab Width: 8 Ln 102, Col 1 INS

Activities Text Editor Apr 21 13:07

Open *1st.cc ~/Desktop Save

*1st.cc *Untitled Document 1 traffic-control.cc

```
    pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
    pointToPoint.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("1p"));

    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));
```

C++ Tab Width: 8 Ln 114, Col 1 INS

Activities Text Editor Apr 21 13:08

Open *1st.cc ~/Desktop Save

*1st.cc *Untitled Document 1 traffic-control.cc

```cpp
    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;
    auto dscpVec = classifier->GetDscpCounts(1);
```

C++ Tab Width: 8 Ln 170, Col 55 INS

Activities Text Editor Apr 21 13:08

Open *1st.cc ~/Desktop Save

*1st.cc *Untitled Document 1 traffic-control.cc

```cpp
              << 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;

    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 169, Col 6 INS

Activities Text Editor Apr 21 13:09

Open *1st.cc ~/Desktop Save

*1st.cc *Untitled Document 1 traffic-control.cc

```
    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;

    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;

    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 174, Col 45 INS

CHANGES IN CODE

```
std::string transportProt = "Udp"; // Changed transport protocol to UDP

 nodes.Create(4); // Modified network topology to include 4 nodes


 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();            //add these line after the following


//Flow
 ApplicationContainer sinkApp = packetSinkHelper.Install(nodes.Get(3));        // change it to .Get(3)


 InetSocketAddress rmt(interfaces23.GetAddress(1), port); // Set UDP destination address

 apps.Add(onoff.Install(nodes.Get(0))); // Configure UDP traffic
```