به نام خدا

پاسخ تمرینات عملی لایه دسترسی چندگانه

سعیدرضا زواشکیانی

۴۰۰۲۰۶۲۶۲

Analyzing CTS/RTS

Question 1)

The function *experiment* first creates 3 nodes and places them randomly. Then we create a propagation loss matrix in which the nodes that don't have a link have a loss of 200 dB and the links node0 to node1 and node1 to node2 both have a loss of 50 dB. After that we create a WIFI channel object and sets the created propagation matrix to its *SetPropagationLossModel* attribute. we use IEEE 802.11b standard for WIFI and sets the channel on the physical layer of WIFI. We also set the MAC type to adhoc mode. Then we create a *NetDeviceContainer* to install the MAC and physical layer on the nodes.

We then install TCP/IP stack on the nodes and assign IP addresses to the net devices.

We then send two flows from node0 to node1 and from node2 to node1 with the *OnOffHelper* application and setting its attributes respectively. We also send a single packet before the CBR flows start to get around the lack of perfect ARP. To explain the situation, we sometimes want a "ping" application to measure network RTT. We're not interested in another layer having to do ARP here, we just want to know how long ping packets take to get through the network. By disabling ARP, we can be sure that there is no variability introduced by the ARP mechanism.
After that we use *FlowMonitor* to measure the statistics.
In printing the flow statistics, we don't display the mentioned ping apps and only measure statistics for the two UDP flows, firstly created. The duration that the apps start and end is 9 seconds, which is the reason that we divide by 9 in measuring throughput.
After that we run the *experiment* with different RTS thresholds to enable and disable RTS/CTS.

The following are the output of the code. In the IEEE 802.11 protocol the RTS/CTS method was initially proposed to help with the hidden station problem. As is seen from the results the throughput is improved very much when RTS/CTS is enabled.

```
Hidden station experiment with RTS/CTS disabled:
Flow 1 (10.0.0.1 -> 10.0.0.2)
  Tx Packets: 2410
  Tx Bytes:   3441480
  TxOffered:  3.05909 Mbps
  Rx Packets: 384
  Rx Bytes:   548352
  Throughput: 0.487424 Mbps
Flow 2 (10.0.0.3 -> 10.0.0.2)
  Tx Packets: 2411
  Tx Bytes:   3442908
  TxOffered:  3.06036 Mbps
  Rx Packets: 101
  Rx Bytes:   144228
  Throughput: 0.128203 Mbps
```

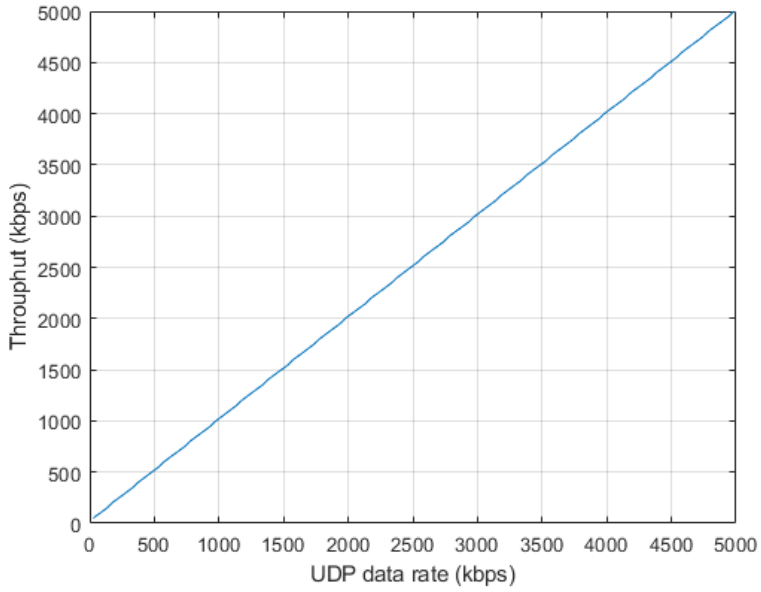
```
Hidden station experiment with RTS/CTS enabled:
Flow 1 (10.0.0.1 -> 10.0.0.2)
  Tx Packets: 2410
  Tx Bytes:   3441480
  TxOffered:  3.05909 Mbps
  Rx Packets: 1272
  Rx Bytes:   1816416
  Throughput: 1.61459 Mbps
Flow 2 (10.0.0.3 -> 10.0.0.2)
  Tx Packets: 2411
  Tx Bytes:   3442908
  TxOffered:  3.06036 Mbps
  Rx Packets: 1000
  Rx Bytes:   1428000
  Throughput: 1.26933 Mbps
```

Question 2

RTS/CTS scheme does not help with the exposed station problem. Unlike the hidden node problem that increases the number of collisions, the exposed node problem increases the delay of nodes by preventing them from transmitting data. As a result of increasing the delay, the throughput decreases. That is, we can achieve a higher throughput when we are having an exposed station problem.

Simulating a Wireless Network

I completed the code with the help of the guides in the homework and from other examples in the NS3 examples. At first, I was having trouble receiving packets, which I thought was a result of misallocating the IP addresses, but it was because I used a bad mobility model. I had set the *deltaX* and *deltaY* parameters to 20 each, and the nodes would get so far that there would be no packets received. So, I decreased those parameters and got the results. I put most of the code in a new function called experiment, and used the *gnuplot-example.cc* to plot throughput vs data rate. I used data rates of 50kbps-5000kbps with a step size of 50kbps. The *.plt* output was plotted with matlab. The throughput is approximately the same as the data rate as seen in the figure.



Data Link Layer Switching

Question 1)

By going to the file *ns3/src/bridge/model/bridge-net-device.cc*, we can see the attributes of the *BridgeNetDevice as below:*

```
34  TypeId
35  BridgeNetDevice::GetTypeId (void)
36  {
37    static TypeId tid = TypeId ("ns3::BridgeNetDevice")
38      .SetParent<NetDevice> ()
39      .SetGroupName("Bridge")
40      .AddConstructor<BridgeNetDevice> ()
41      .AddAttribute ("Mtu", "The MAC-level Maximum Transmission Unit",
42                     UintegerValue (1500),
43                     MakeUintegerAccessor (&BridgeNetDevice::SetMtu,
44                                           &BridgeNetDevice::GetMtu),
45                     MakeUintegerChecker<uint16_t> ())
46      .AddAttribute ("EnableLearning",
47                     "Enable the learning mode of the Learning Bridge",
48                     BooleanValue (true),
49                     MakeBooleanAccessor (&BridgeNetDevice::m_enableLearning),
50                     MakeBooleanChecker ())
51      .AddAttribute ("ExpirationTime",
52                     "Time it takes for learned MAC state entry to expire.",
53                     TimeValue (Seconds (300)),
54                     MakeTimeAccessor (&BridgeNetDevice::m_expirationTime),
55                     MakeTimeChecker ())
56    ;
57    return tid;
58  }
```

There are three attributes which are Mtu, EnableLearning, ExpirationTime. The description of the attributes is also given in front of them. The *ExpirationTime* is the time it takes for learned MAC state entry to expire. The backward learning algorithm used in bridges should be dynamic to adapt to topology changes. An expiration timer is used to remove very old entries. Setting it to an extremely short time can cause too much flooding and congestion, and setting it to an extremely long time can cause the adaptation to be delayed and unaware of the new changes in the topology.

Question 2, 3)

I implemented the bridges as was shown in the bridge examples of NS3. I used the code *csma-bridge-one-hop.cc* and changed it according to the question. The statistics of the flows are shown in the figures below:

```
Bridge topology without the link from bridge 1 to bridge 3 with 1 Mbps links
Flow 4294967295 (192.168.62.1 -> 192.168.62.6)
  Tx Packets: 75
  Tx Bytes:   1127100
  TxOffered:  1.00187 Mbps
  Rx Packets: 69
  Rx Bytes:   1036932
  Throughput: 0.921717 Mbps
-------------------------------------------------
Bridge topology with the link from bridge 1 to bridge 3 with 1 Mbps links
Flow 4294967295 (192.168.62.1 -> 192.168.62.6)
  Tx Packets: 75
  Tx Bytes:   1127100
  TxOffered:  1.00187 Mbps
  Rx Packets: 17
  Rx Bytes:   255476
  Throughput: 0.22709 Mbps
```

```
Bridge topology without the link from bridge 1 to bridge 3 with 100 Mbps links
Flow 4294967295 (192.168.62.1 -> 192.168.62.6)
  Tx Packets: 75
  Tx Bytes:   1127100
  TxOffered:  1.00187 Mbps
  Rx Packets: 73
  Rx Bytes:   1097044
  Throughput: 0.97515 Mbps
-----------------------------------------------
Bridge topology with the link from bridge 1 to bridge 3 with 100 Mbps links
Flow 4294967295 (192.168.62.1 -> 192.168.62.6)
  Tx Packets: 75
  Tx Bytes:   1127100
  TxOffered:  1.00187 Mbps
  Rx Packets: 1
  Rx Bytes:   15028
  Throughput: 0.0133582 Mbps
```

The pcap/tr files of the first case which is 1 Mbps links without the link from bridge 1 to bridge 3 are provided in a zip file to decrease the size. The other files can be generated by running the code, which generates and outputs the results for all of the cases, but due to large size of files they are not uploaded.

When there is not a link between the bridge 1 and bridge 3, increasing the link data rate, improves the throughput, but with the existence of the link, there will be a loop in the topology and the links will be congested, thus decreasing the throughput. When the link data rate is 100 Mbps and there is a loop, the packets all get in the loop and the links will be congested even more, thus decreasing the throughput more. Also, the size of the pcap/tr files will be extremely large in case of 100 Mbps links and existence of the loop, which is a result of the packets running around in a loop.