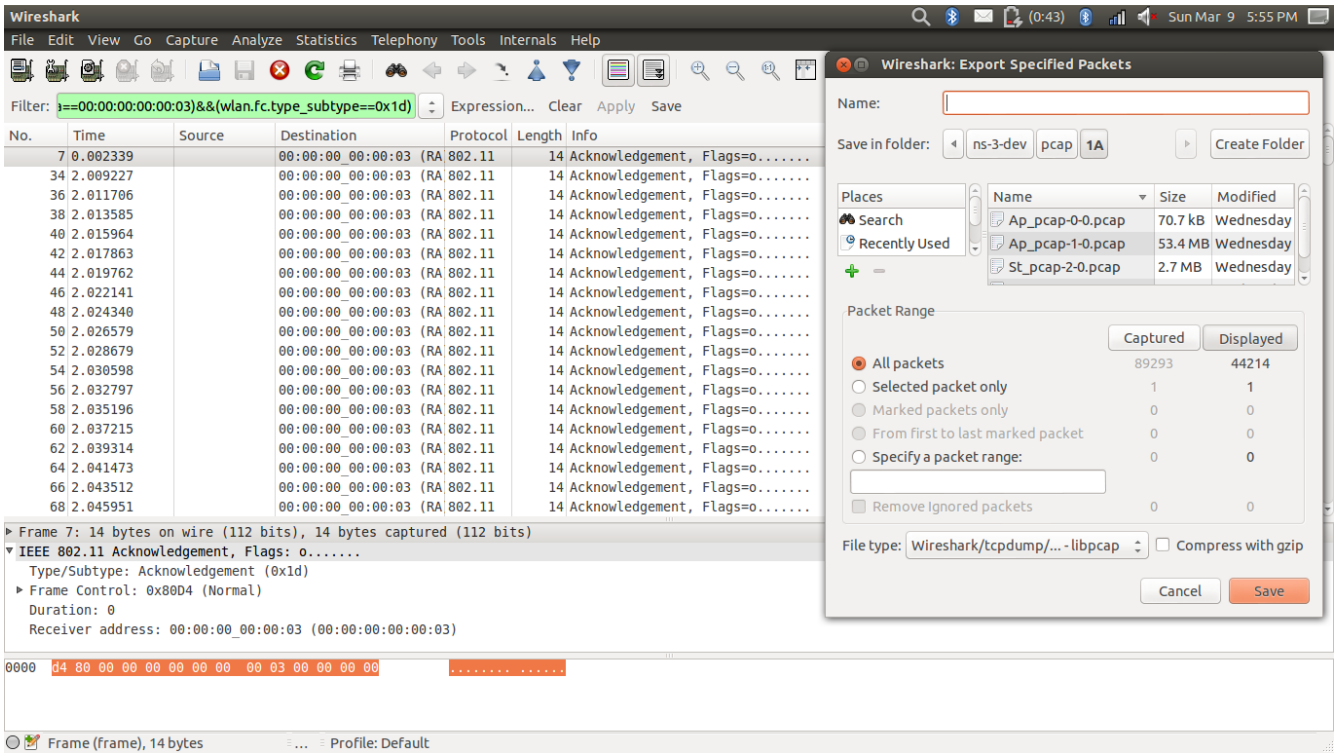
NS-3 Project Report Group-10

Ganapathy Raman Madanagopal
<grma@kth.se>
900423-3335

Rakesh Varudu
<varudu@kth.se>
880405-1491

Problem:1A

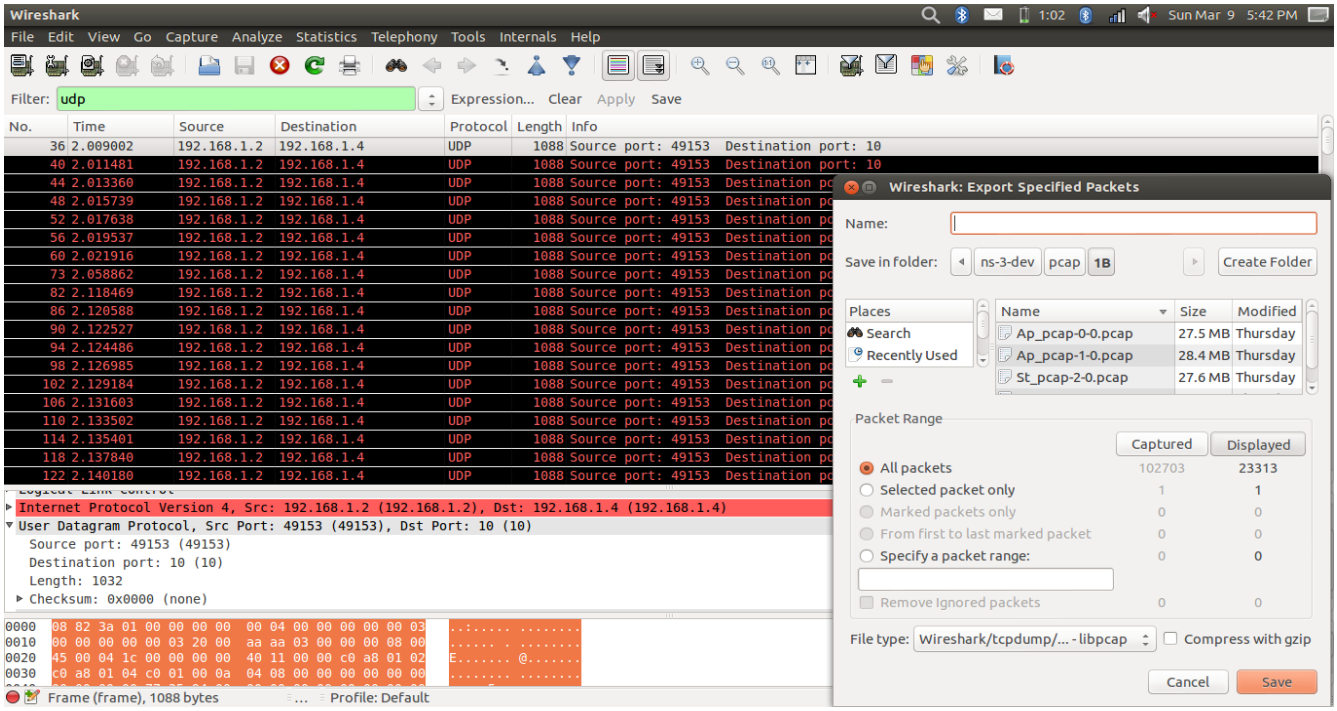
The topology is configured as mentioned in the problem, where station 'a' (ST-a) can sense and listen all frames that station b (ST-b) sends. All device have the sensing range of 10m and they are separated by the distance of 7m. But both AP's cannot sense each other, since they are far away from their sensing range. Also ST-a cannot sense AP-B and ST-b cannot sense AP-A. For every successful transmission of data frame from AP-B to ST-b, there will be 802.11b MAC layer ACK's from ST-b to AP-B. This frames also includes ARP messages, RTS/CTS. To calculate rate at which ACK's is sent from ST-B, we apply the following filter on the capture file from ST-A (Ap_pcap_2-0.pcap), (wlan.ra == 00:00:00:00:00:03) && (wlan.fc.type_subtype == 0x1d). After running the stimulation for 99 seconds we get **44214 frames**, i.e. AP- B approximately sends **447 frames per second.**



Problem 1B

To achieve symmetry in the flow, activate flow between AP-A to ST-A and there is an existing flow AP-B to St-B. We should check rate at which both AP's are sending in this symmetrical flow, we need to check the capture files of both AP-A and AP-B in order to find the rate at which they are sending. This topology address the problem of hidden terminal problem. As the Auto Rate Fallback protocol is disabled, AP-A and AP-B uses fixed rate to send the data at constant rate and for same amount of time they have transmitted. So the flows should be symmetrical. Also CSMA/CA protocol is employed by which they transmit RTS/CTS to avoid collision. When we apply the

filter “UDP” on both the capture files of AP-A and AP-B, we get **22555** and **23313** UDP packets respectively. So their rates are **228** and 235 frames per second respectively, which indicates that both AP-A and AP-B sends at same rate approximately.



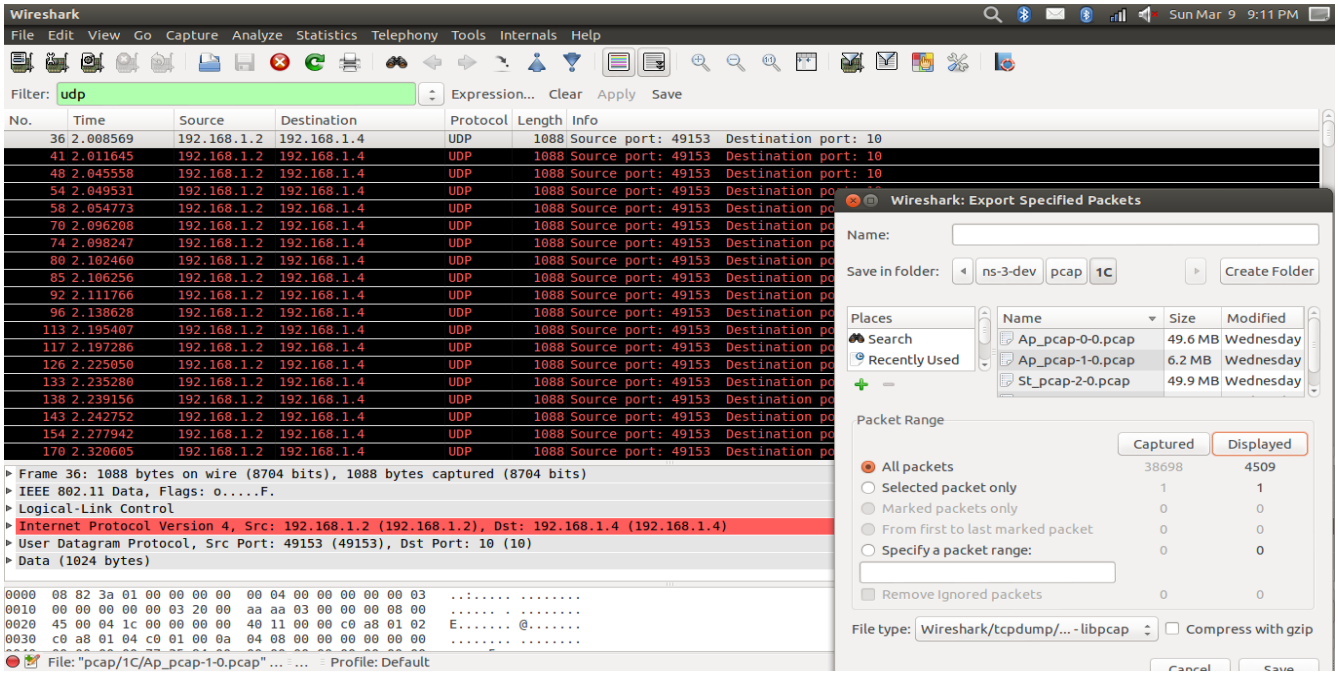
Problem 1-C

Now the direction of the flow is from ST-A to AP-A, which is the reverse of the previous problem, which create asymmetry in the flows (a-A and B-b). In this case ST-A can sense whether ST-B is transmitting or not by CSMA/CA protocol. But on the other flow, AP-B sense that channel is always free. As a result of this AP-B send packets to ST-B at constant rate. But ST-B can send acknowledgment back only when the flow (a-A) is not active. In this way the congestion window of AP-B grows exponentially and when it reaches its maximum value, the frame is discarded from the retransmission queue (during the back-off interval of ST-A) and the next frame is sent out. On the other hand the congestion window of ST-A will be always equal to its minimum value unlike that of AP-B even when the channel is not free (back-off stage). Because of this AP-B has to wait for a long time until it can gain access to the channel which is only possible during the short back-off time of ST-A. By this way flow between AP-B to ST-B suffers from starving. It can be viewed from the capture files of ST-A and AP-B, where the ST-A has sent **41291** UDP data frames (~**498** data frames per second) and AP-B has sent **4509** UDP data frames (~**46** data frames per second).

The following are important code snippet from Problem-1C (Vector Position & Client Server Part)

```
Ptr<ListPositionAllocator> StAlloc = CreateObject<ListPositionAllocator>();
StAlloc->Add(Vector(7.0, 0.0, 0.0));  /* Placing ST_A */
StAlloc->Add(Vector(14.0, 0.0, 0.0)); /* Placing ST_B */

Ptr<ListPositionAllocator> ApAlloc = CreateObject<ListPositionAllocator>();
ApAlloc->Add(Vector(0.0, 0.0, 0.0));  /* Placing AP_A */
ApAlloc->Add(Vector(21.0, 0.0, 0.0)); /* Placing AP_B */

/* Server and Client part. AP_A sends data packets to ST_A */
```

```
  UdpServerHelper Server1 (10); /* Starting Server at Port 10 */
  ApplicationContainer ServerApp1 = Server1.Install (ApNodes.Get (0)); /* AP_A as Server */
  ServerApp1.Start (Seconds (1.0));
  ServerApp1.Stop (Seconds (100.0));
/* Binding the Client to Sever and opening Port 10 for communication with server */
   UdpClientHelper Client1 (ApInterfaces.GetAddress (0), 10);
  Client1.SetAttribute ("MaxPackets", UintegerValue (100000000));
  Client1.SetAttribute ("Interval", TimeValue (Seconds (0.0001)));
  Client1.SetAttribute ("PacketSize", UintegerValue (1024));
  ApplicationContainer ClientApp1 = Client1.Install (StNodes.Get (0)); /* ST_A as Client */
  ClientApp1.Start (Seconds (2.0)); /* Starting Client after Server has started */
  ClientApp1.Stop (Seconds (100.0)); /* Stopping time of Server */
   /* Server and Client part.  AP_B sends data packets to ST_B */
  UdpServerHelper Server2 (10); /* Starting Server at Port 10 */
  ApplicationContainer ServerApp2 = Server2.Install (StNodes.Get (1)); /* ST_B as Server */
  ServerApp2.Start (Seconds (1.0)); /* Starting time of Server */
  ServerApp2.Stop (Seconds (100.0)); /* Stopping time of Server */
/* Binding the Client to Sever and opening Port 10 for communication with server */
  UdpClientHelper Client2 (StInterfaces.GetAddress (1), 10);
  Client2.SetAttribute ("MaxPackets", UintegerValue (100000000));
  Client2.SetAttribute ("Interval", TimeValue (Seconds (0.0001)));
  Client2.SetAttribute ("PacketSize", UintegerValue (1024));
  ApplicationContainer ClientApp2 = Client2.Install (ApNodes.Get (1)); /* AP_B as Client */
  ClientApp2.Start (Seconds (2.0)); /* Starting Client after Server has started */
  ClientApp2.Stop (Seconds (100.0)); /* Stopping time of Client */
```

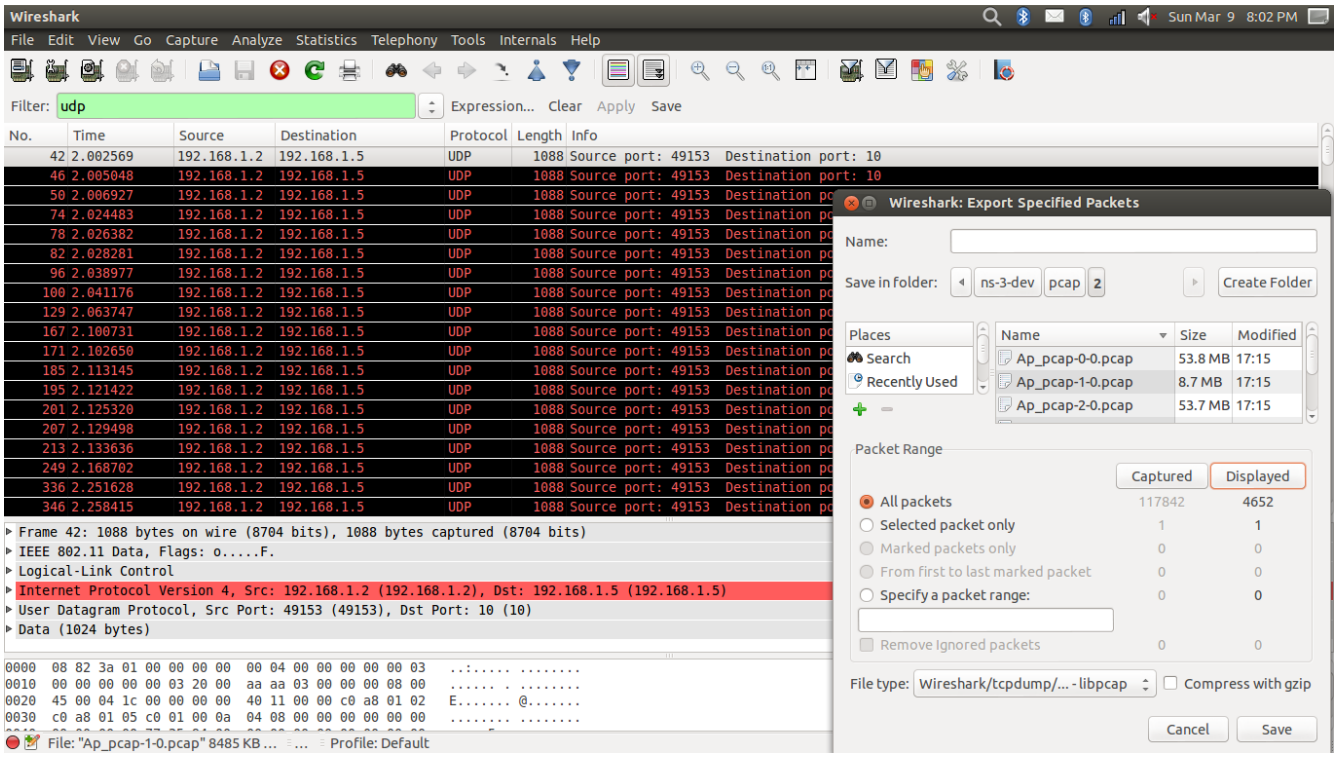


Problem 2

In the topology AP-A can sense AP-B, AP-C can sense AP-B and AP-B can sense both AP-A and AP-C. So in this topology AP-B faces the problem of "flow in the middle", which means AP-B has to compete with both AP-

A and AP-C for accessing the channel, whereas AP-A and AP-C has to compete only with AP-B. This denotes that AP-C can transmit data frames even when AP-A is sending and vice-versa, since both doesn't sense each other. So AP-B will send less number of frames when compared to AP-A and AP-C. This is evident from the capture files of AP's. The UDP packets on AP-A, AP-B and AP-C are **44689, 4652 and 44668 respectively** for a duration of 99 seconds. So they send data frames at the rate **451, 47, and 451** per second respectively.

The major difference between problem 1c and problem 2 is that in problem 1c, flow AP-B-ST-b has to starve as it had no information about (AP-A) and other flow ST-a-AP-A. So AP-B always sense the channel to be free. But in problem 2, AP-B has AP-A and AP-C in its sensing range. So it has to wait for its turn to transmit. So the probability of AP-B transmitting is less than that of AP-A and AP-C.



Problem 3-A

In this topology we have one Access Point (AP-A) and we varied the sources (from N = 2 to 10). All the Nodes in the single cell network can sense each other. It is known that, as we increase the number of nodes in the network, the collision probability also increases. As the number of nodes increase in the network, the probability that they will try to transmit at the same time also increase, which in turn leads to increase in collision probability. We calculated the collision probability by logging different parameters required by enabling the following command in a separate shell script which captures the required logs into a file.

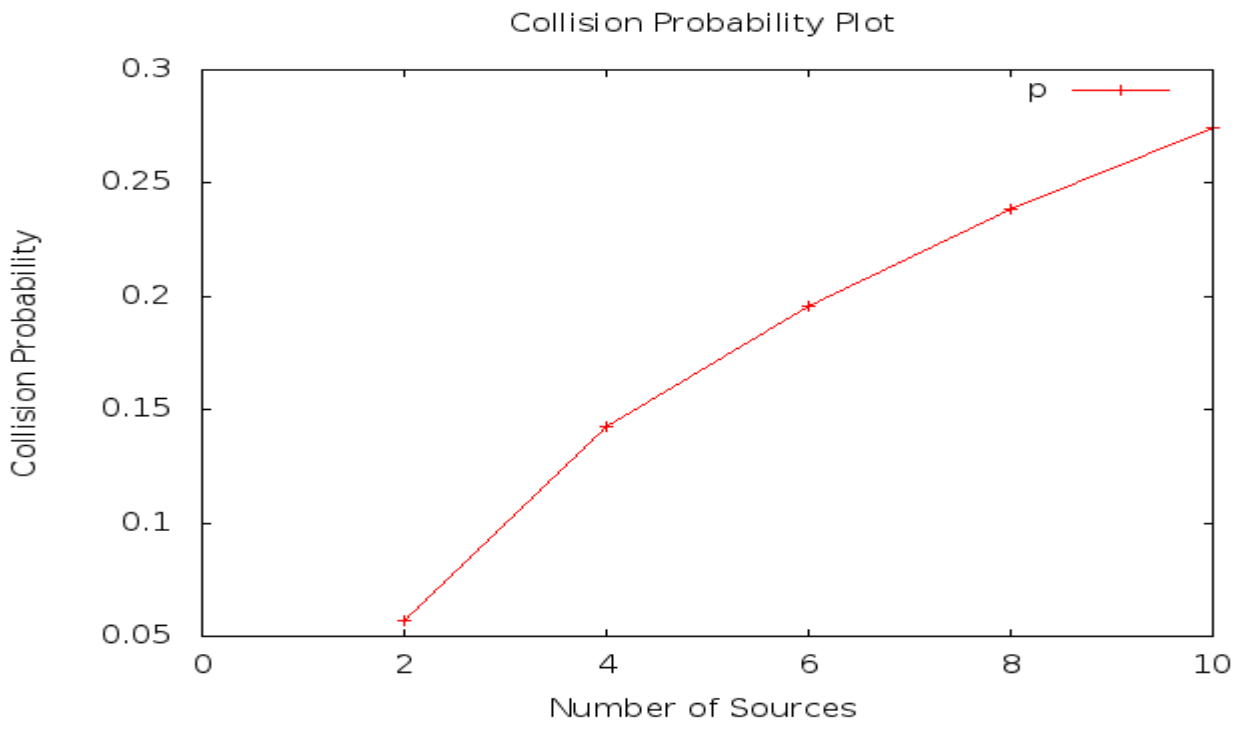
Using a Shell scripts which fetches the values of “Missed CTS” and “Total RTS” based on which collision probability is defined. **(Collision Probability = #Missed CTS / #RTS)**.

```
# export 'NS_LOG=DcaTxop=level_all|prefix_func|prefix_time'

# ./waf --run "scratch/3a --nSt=$i" > output.txt 2>&1

# rts=`grep rts output.txt |wc -l`
```

```
# mcts=`grep 'missed cts' output.txt |wc -l`
# cprob = `echo "scale=6;$cts/$rts" | bc`
```

#Sources	Collision Probability
2	0.057072
4	0.142836
6	0.195476
8	0.238880
10	0.274693

From the below graph it is evident that as the number of nodes in the network increase, the collision probability has also increased.



The following are important code snippet from Problem-3A

```
/* Setting maximum transmission range to 10m and Stations are within the range of coverage
*/Channel.AddPropagationLoss("ns3::RangePropagationLossModel", "MaxRange", DoubleValue(10.0));
Phy.SetChannel (Channel.Create ());
```

```
/* Create Mobility and setting the vector position for Stations and Access Points */
Ptr<ListPositionAllocator> StAlloc = CreateObject<ListPositionAllocator>();
double Pos = 0.0;
for(i=0; i<nSt; i++)
{
   StAlloc->Add(Vector(Pos, 0.0, 0.0));  /* Placing ST_i */
  Pos = Pos + (7.0/nSt);
}
Ptr<ListPositionAllocator> ApAlloc = CreateObject<ListPositionAllocator>();
ApAlloc->Add(Vector(3.5, 3.5, 0.0));  /* Placing AP_A */
/* Server and Client part. ST_i sends data packets to AP_A */
UdpServerHelper Server (10); /* Starting Server at Port 10 */
/* AP_A as Server */
ApplicationContainer ServerApp = Server.Install (ApNodes.Get (0));
ServerApp.Start (Seconds (1.0));  /* Starting time of Server */
ServerApp.Stop (Seconds (100.0)); /* Stopping time of Server */
/* Binding Client to Sever and opening Port 10 for communication with server */
for(i=0; i<nSt; i++)
{
  UdpClientHelper Client (ApInterfaces.GetAddress (0), 10);
  Client.SetAttribute ("MaxPackets", UintegerValue (100000000));
  Client.SetAttribute ("Interval", TimeValue (Seconds (0.0001)));
  Client.SetAttribute ("PacketSize", UintegerValue (1024));
  /* Installing Client*/
  ApplicationContainer ClientApp = Client.Install (StNodes.Get (i));
  ClientApp.Start (Seconds (2.0));/* Starting Client after Server has started */
  ClientApp.Stop (Seconds (100.0)); /* Stopping time of Client *
}
```

Problem 3-B

In this topology we have one Access Point (AP-A) and we varied the sources (from N = 2 to 10). All the Nodes in the single cell network can sense each other. It is known that, as we increase the number of nodes in the network, the collision probability also increases. With increase in number of nodes, the throughput gradually increases but starts decreasing once the congestion sets in.

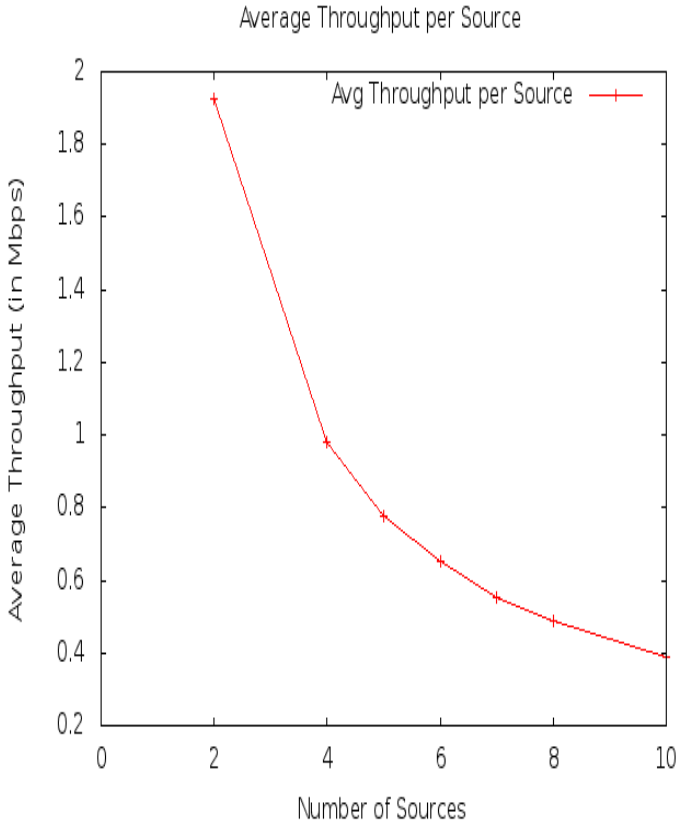
We calculated the average total throughput and average per flow throughput using the FlowMonitorHelper.

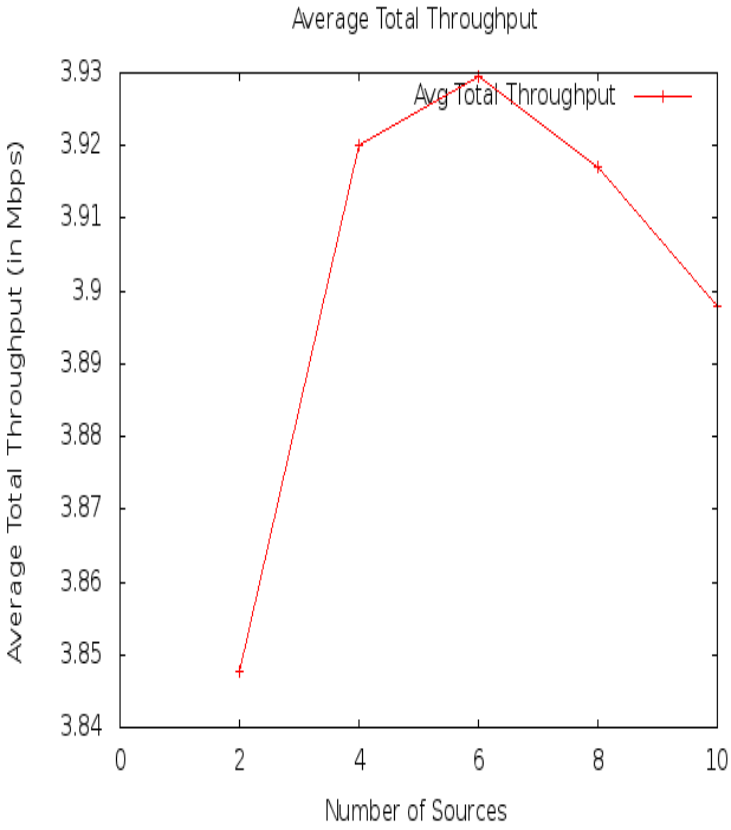
 # Average total throughput (in Mbps) = (rxBytes * 8.0) / (Total Time in seconds) / 1024 /1024

 # Average per source throughput (in Mbps) = Average total throughput / (No of sources)

#Sources	Total Throughput (in Mbps)	Average Throughput per source (in Mbps)
2	3.8476	1.9238
4	3.920128	0.980032
6	3.929508	0.654918
8	3.91696	0.48962

10	3.89805	0.389805





Generally, Back-off Interval is defined by Random
(0, CW), where CW is the contention window. In our case, back-off interval should vary according to the number of nodes in the network. Initially set the back-off Interval to random value, # Back-off Interval = Random (CW min, Cmax).

Let CWmin is directly proportional to 'n'-Nodes in System. So, as the number of nodes in the system increase, let the minimum contention window size also increase which ensures the back-off interval to be at least some value which is more than 0. This Back-off interval allows the system to recover and reduces the congestion in the network due to increased number of nodes in the network. If we have used original Back-off interval, there is a chance that it takes the any value ranging from 0 to CW which can be sometimes possibly 0 also, in which case the node waits only for 0 seconds, but in our case, Back-off should at least take a value of not less than CW min, which in our case is defined as CWmin + n *delta; where n is the number of nodes. As the number of nodes in the system increase, the CW min also increase by a factor of n*delta. This enables to wait for a decent back-off interval of time which allows the system to recover from the collisions.

Note:

Since, we cannot upload all the files related to project in moodle, all the NS-3 scripts, wireshark captures, report and some Unix shell scripts that we used to automate are placed in a google doc whose link is shared below:

https://drive.google.com/folderview?id=0B_zNSL9cwhx_QWVrcndIc3dVR3c&usp=sharing