Assignment four

2021 Summer ELG 5142 Ubiquitous Sensing and Smart City

Team members-Group 8:

- Abdelrhman Gaber Youssef Saad Rezkallah
- Eman Metwally Mohammed Abood
- Basma Reda Shaban Abd-Elsalam Abd-Elwahab

Introduction: **ns-3** is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. ns-3 is free, open-source software, licensed under the GNU GPLv2 license, and maintained by a worldwide community.

Question 1 implementation:

1- Create 3 vehicles nodes.

```
void
WaveNetDeviceExample::CreateWaveNodes (void)
{
  //creating sender and receiver
  nodes = NodeContainer ();
  nodes.Create (3);
```

Question 2 implementation

2- The vehicles are mobile on a rectangular area according to RandomWalk2dMobility model:
•The area is a rectangle with bounds 0<x<500 and 0<y<500.
•The speed of the vehicles is a uniform random number between 8 m/s and 13 m/s.
•Vehicles are initially positioned randomly on the rectangular area.

Step 1: This to define a movement in a rectangle and random walk and we will use the end position = 50, because **when we try 500 the distance is so high, and it will lead to not sending any backet.**

```
// Q 2  this to define a movement in a rectangle and random walk and we will use the end position = 50
because when we try 500 the distance is so high and it will lead to not sending any backet
*/
  //installing mobility on the nodes since wifi nodes needs mobility to be installed
  MobilityHelper mobility;
  mobility.SetPositionAllocator ("ns3::RandomRectanglePositionAllocator",
                                 "X", StringValue ("ns3::UniformRandomVariable[Min=0|Max=50]"),
                                 "Y", StringValue ("ns3::UniformRandomVariable[Min=0|Max=50]")
                                );
```

Step 2: Limiting the time speed to be between 8 to 13 m/s and limiting the boundary of the rectangle to be between 0 to 500 for x and y.

```
mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                           "Speed", StringValue ("ns3::UniformRandomVariable[Min=8|Max=13]"),
                           "Bounds", StringValue ("0|50|0|50"));
mobility.Install (nodes);
YansWifiChannelHelper waveChannel = YansWifiChannelHelper::Default ();
YansWavePhyHelper wavePhy =  YansWavePhyHelper::Default ();
wavePhy.SetChannel (waveChannel.Create ());
```

Step 3: Set the data link type of PCAP traces to be used. This function has to be called before EnablePcap(), so that the header of the pcap file can be written correctly. Here we set it to IEEE 802.11 Wireless LAN headers.

```
wavePhy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11);
QosWaveMacHelper waveMac = QosWaveMacHelper::Default ();
WaveHelper waveHelper = WaveHelper::Default ();
devices = waveHelper.Install (wavePhy, waveMac, nodes);
```

Step 4:

```
for (uint32_t i = 0; i != devices.GetN (); ++i)
  {
    //we use DynamicCast to use a subclass of a base class
    Ptr<WaveNetDevice> device = DynamicCast<WaveNetDevice> (devices.Get (i));
    //Set the callback to be used to notify higher layers when a packet has been received.
    device->SetReceiveCallback (MakeCallback (&WaveNetDeviceExample::Receive, this));

  }
// Tracing
wavePhy.EnablePcap ("wave-simple-device", devices);
}
```

Question 3 implementation:

The simulation time is 20 seconds.

- We used the time to be 20.1 seconds instead of 20 seconds and we noticed that, if we set it at 20 Seconds, the last packets (at 20) wouldn't be sent or received, so we set it at 20.1 Seconds.

```
Simulator::Schedule (Seconds (20.1), &WaveNetDevice::StopSch, sender1, SCH1);
Simulator::Stop (Seconds (20.1));
Simulator::Run ();
Simulator::Destroy ();
```

Question 4, Question 5, Question 6, and Question 8 implementation:

1- At second1, vehicle1(vehicle with index 0) sends a broadcast message (wave short message) via control channel (CCH):
•The packet payload of the CCH message is 500 Bytes.
•The ethernet type protocol is set to 0x88dc which correspond to WSMP.
•The transmission charact eristics of this CCH broadcast message are as follows:
- The transmission data rate (wifiMode) is OfdmRate12MbpsBW10MHz.
- The priority of packets is 7(the packets priority is a number between 0 and 7, and 7 is the lowest priority)

- Replace the constants to be variables that we would pass to the "**SendOneWsmpPacket**" function.

2- At intervals of 5 seconds (5, 10, 15, 20), vehicle 1, vehicle2 and vehicle 3 broadcast messages of size 1000 bytes via service channel 1 (SCH1):

- The priority of the packets sent from vehicle 1 is 0 (highest priority)
- The rate of the broadcast for vehicle 1 is 27Mbps.
- The priority of the packets sent from vehicle 2 is 5.
- The rate of the broadcast for vehicle 2 is 9Mbps.
- The priority of the packets sent from vehicle 3is 7.
- The rate of the broadcast for vehicle 3is 6 Mbps.

3- Avoid the redundancy of the code by using loops as required in Q8.

```
void
WaveNetDeviceExample::SendOneWsmpPacket  (uint32_t channel,
                                          uint32_t d ,
                                          uint32_t seq,
                                          uint32_t prio,
                                          uint32_t packsize,
                                          const char* drate)
{
  Ptr<WaveNetDevice>  sender1 = DynamicCast<WaveNetDevice> (devices.Get (d));
  //WSM packets received from the lower layers with an Ethernet Type of 0x88DC are delivered to the WSM protocol.
  const static uint16_t WSMP_PROT_NUMBER = 0x88DC;
  Mac48Address bssWildcard = Mac48Address::GetBroadcast ();
  TxInfo txInfo = TxInfo (channel, prio, WifiMode(drate)); // the default powerLevel = 8
  Ptr<Packet> p  = Create<Packet> (packsize);   //Create a packet with a zero-filled payload of size 100.
  SeqTsHeader seqTs;       //SeqTsHeader: Packet header to carry sequence number and timestamp
  seqTs.SetSeq (seq);      //Set the sequence number in the header
  p->AddHeader (seqTs);    //Add header to a packet
  sender1->SendX  (p, bssWildcard, WSMP_PROT_NUMBER, txInfo);
}
```

The first packet will be queued currently and be transmitted in next SCH interval

```
void
WaveNetDeviceExample::SendWsmpExample ()
{
  CreateWaveNodes ();
  const SchInfo schInfo = SchInfo (SCH1, false, EXTENDED_ALTERNATING);   //alternating access to CCH and SCH
  // An important point is that the receiver should also be assigned channel
  // access for the same channel to receive packets.
  Ptr<WaveNetDevice>  sender1 = DynamicCast<WaveNetDevice> (devices.Get (0));
  /*
 // Q4 this for loop to send the backet via CCH in question two at t = 1
 and it will send a broadcast message and at t = 5,10,15,20 it will send a message via SCH1 with the different paiority and different rate
 */
  int i= -1;
```

```
for(uint32_t t=1; t<=20;t++)
{

  if(t==1)
  {
    // the first packet will be queued currently and be transmitted in next SCH interval
     Simulator::Schedule (Seconds (t-1), &WaveNetDevice::StartSch, sender1, schInfo);
    Simulator::Schedule (Seconds (t), &WaveNetDeviceExample::SendOneWsmpPacket,  this, CCH,0, 1,7,500,"OfdmRate12MbpsBW10MHz");
  }

  else if(t%5==0)
  {
    i+=2;
    for (uint32_t d=0;d<=2;d++)
    {
      Ptr<WaveNetDevice>  sender2 = DynamicCast<WaveNetDevice> (devices.Get (d));
      Simulator::Schedule (Seconds (t-1), &WaveNetDevice::StartSch, sender2, schInfo);
      switch(d)
      {
        case 0:
        Simulator::Schedule (Seconds (t), &WaveNetDeviceExample::SendOneWsmpPacket,  this, SCH1,0, t-d-i,0,1000,"OfdmRate27MbpsBW10MHz");
        break;

        case 1:
        Simulator::Schedule (Seconds (t), &WaveNetDeviceExample::SendOneWsmpPacket,  this, SCH1,1, t-d-i,5,1000,"OfdmRate9MbpsBW10MHz");
        break;

        case 2:
        Simulator::Schedule (Seconds (t), &WaveNetDeviceExample::SendOneWsmpPacket,  this, SCH1,2, t-d-i,7,1000,"OfdmRate6MbpsBW10MHz");
        break;

      }

    }
  }
}
```

Question 7 implementation:

A callback function should be defined in the Simulation wherein supposed to print out the sender and receiver's MAC address, the sequence number and the time stamp.

```
bool
WaveNetDeviceExample::Receive (Ptr<NetDevice> dev, Ptr<const Packet> pkt, uint16_t mode, const Address &sender)
{
  SeqTsHeader seqTs;
  pkt->PeekHeader (seqTs);
  std::cout << "receive a packet: " << std::endl
            << "  reciever = " << dev->GetAddress() << "," << std::endl
            << "  sender = " << sender << "," << std::endl
            << "  sequence = " << seqTs.GetSeq () << "," << std::endl
            << "  sendTime = " << seqTs.GetTs ().As (Time::S) << "," << std::endl
            << "  recvTime = " << Now ().As (Time::S) << "," << std::endl
            << "  protocol = 0x" << std::hex << mode << std::dec  << std::endl;
  return true;
}
```

Results of the implementation:

Run the boundaries between 1 and 50:

```
abdelrhman@abdelrhman-VirtualBox:~/Desktop/ns-allinone-3.35/ns-3.35$ ./waf --ru
n Assigment4
Waf: Entering directory `/home/abdelrhman/Desktop/ns-allinone-3.35/ns-3.35/buil
d'
[2824/2876] Compiling scratch/Assigment4.cc
[2825/2876] Compiling scratch/subdir/scratch-simulator-subdir.cc
[2826/2876] Compiling scratch/scratch-simulator.cc
[2835/2876] Linking build/scratch/Assigment4
[2836/2876] Linking build/scratch/subdir/subdir
[2837/2876] Linking build/scratch/scratch-simulator
Waf: Leaving directory `/home/abdelrhman/Desktop/ns-allinone-3.35/ns-3.35/build
'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (8.805s)
run WAVE WSMP routing service case:
```

```
Build commands will be stored in build/compile_commands.json
'build' finished successfully (8.805s)
run WAVE WSMP routing service case:
receive a packet:
  reciever = 02-06-00:00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:00:08,
  sequence = 1,
  sendTime = +1s,
  recvTime = +1.00484s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:00:08,
  sequence = 1,
  sendTime = +1s,
  recvTime = +1.00484s,
  protocol = 0x88dc
```

```
receive a packet:
  reciever = 02-06-00:00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:00:18,
  sequence = 2,
  sendTime = +5s,
  recvTime = +5.0555s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:00:08,
  sender = 02-06-00:00:00:00:00:00:18,
  sequence = 2,
  sendTime = +5s,
  recvTime = +5.0555s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:00:10,
  sequence = 3,
  sendTime = +5s,
  recvTime = +5.05702s,
  protocol = 0x88dc
r Terminal packet:
  reciever = 02-06-00:00:00:00:00:00:08,
  sender = 02-06-00:00:00:00:00:00:10,
  sequence = 3,
  sendTime = +5s,
  recvTime = +5.05702s,
  protocol = 0x88dc
receive a packet:
```

```
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 4,
  sendTime = +5s,
  recvTime = +5.05857s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 4,
  sendTime = +5s,
  recvTime = +5.05857s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:18,
  sequence = 5,
  sendTime = +10s,
  recvTime = +10.0555s,
  protocol = 0x88dc
```

```
receive a packet:
  reciever = 02-06-00:00:00:00:00:08,
  sender = 02-06-00:00:00:00:00:10,
  sequence = 3,
  sendTime = +5s,
  recvTime = +5.05702s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 4,
  sendTime = +5s,
  recvTime = +5.05857s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 4,
  sendTime = +5s,
  recvTime = +5.05857s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:18,
  sequence = 5,
  sendTime = +10s,
  recvTime = +10.0555s,
  protocol = 0x88dc
```

```
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:18,
  sequence = 5,
  sendTime = +10s,
  recvTime = +10.0555s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:08,
  sender = 02-06-00:00:00:00:00:18,
  sequence = 5,
  sendTime = +10s,
  recvTime = +10.0555s,
  protocol = 0x88dc
```

```
receive a packet:
  reciever = 02-06-00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:10,
  sequence = 6,
  sendTime = +10s,
  recvTime = +10.057s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:08,
  sender = 02-06-00:00:00:00:00:10,
  sequence = 6,
  sendTime = +10s,
  recvTime = +10.057s,
  protocol = 0x88dc
receive a packet:
```

```
abdelrhman@abdelrhman-VirtualBox: ~/Desktop/ns-allinon...
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 7,
  sendTime = +10s,
  recvTime = +10.0586s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 7,
  sendTime = +10s,
  recvTime = +10.0586s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:18,
  sequence = 8,
  sendTime = +15s,
  recvTime = +15.0555s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:08,
  sender = 02-06-00:00:00:00:00:18,
  sequence = 8,
  sendTime = +15s,
  recvTime = +15.0555s,
  protocol = 0x88dc
receive a packet:
```

```
receive a packet:
  reciever = 02-06-00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:10,
  sequence = 9,
  sendTime = +15s,
  recvTime = +15.057s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:08,
  sender = 02-06-00:00:00:00:00:10,
  sequence = 9,
  sendTime = +15s,
  recvTime = +15.057s,
  protocol = 0x88dc
```

```
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 10,
  sendTime = +15s,
  recvTime = +15.0586s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 10,
  sendTime = +15s,
  recvTime = +15.0586s,
  protocol = 0x88dc
receive a packet:
```

```
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:18,
  sequence = 11,
  sendTime = +20s,
  recvTime = +20.0555s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:08,
  sender = 02-06-00:00:00:00:00:18,
  sequence = 11,
  sendTime = +20s,
  recvTime = +20.0555s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:08,
  sender = 02-06-00:00:00:00:00:10,
  sequence = 12,
  sendTime = +20s,
  recvTime = +20.057s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:10,
  sequence = 12,
  sendTime = +20s,
  recvTime = +20.057s,
  protocol = 0x88dc
receive a packet:
```

```
receive a packet:
  reciever = 02-06-00:00:00:00:00:10,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 13,
  sendTime = +20s,
  recvTime = +20.0586s,
  protocol = 0x88dc
receive a packet:
  reciever = 02-06-00:00:00:00:00:18,
  sender = 02-06-00:00:00:00:00:08,
  sequence = 13,
  sendTime = +20s,
  recvTime = +20.0586s,
  protocol = 0x88dc
```

Conclusion:

In this assignment, we learned how to create an adhoc vehicular topology in NS3 using WaveMacHelper, sending a broadcast message, via the control channel, set a priority of the packets sent from the vehicles, and control all over it.

References:

https://www.nsnam.org/doxygen/classns3_1_1_random_walk2d_mobility_model.html

https://www.nsnam.org/doxygen/classns3_1_1_random_rectangle_position_allocator.html