MoReV2X Documentation

Index

Main script

The reference script is highwayprova.cc, contained in the /scratch folder.
In the first part of the script, you can find the code for the overall configuration. Since it is basic c++ programming, we are not going to discuss it. This documentation starts by discussing the constructs that are typical of ns-3 and so new.
The goal of the main script is to create a network of nodes featuring the NR-V2X stack. These nodes are vehicles that move at a constant velocity and can transmit both periodic and aperiodic traffic. The sender application is defined by the NrV2XUdpClient class while the receiving application is a simple packet sink application.
Creation of the node container for the responders through the command:

```
NodeContainer ueResponders;
```

Creation of the node container [1].
With a for loop, nodes are created and added to the previously declared container:

```
for (uint32_t t=0; t<ueCount; ++t)    //MATTIA: for loop to crea
{
  Ptr<Node> lteNode = CreateObject<Node> ();  //MATTIA: creation
  // LTENodeState is a new class developed from scratch. Defined
  Ptr<LTENodeState> nodeState = CreateObject<LTENodeState> ();
  nodeState -> SetNode(lteNode);      //MATTIA: want the state o
  lteNode -> AggregateObject(nodeState);  // Aggregate two objec
  ueResponders.Add(lteNode); //MATTIA: finally, node added to th
}
```

The mobility helper is created (used later). The MobilityHelper is a helper class used to assign positions and mobility models to nodes [2]:

```
MobilityHelper mobilityUE;
```

The lane number and the x position of the vehicles are created as random variables. The lane width is set to 4 m:

```
Ptr<UniformRandomVariable> laneNumber = CreateObject<UniformRandomVariable>(); /
Ptr<UniformRandomVariable> Xposition = CreateObject<UniformRandomVariable>(); //
double laneWidth = 4.0; //MATTIA: width of the lane, double variable
```

The next step is the creation of a position allocator (vector that contains the positions of the vehicles)

```
Ptr<ListPositionAllocator> positionAlloc = CreateObject <ListPositionAllocator>();
```

A for loop is used to define the x position and the y position of each vehicle. For the y position, the lane width is multiplied for an integer chosen between 1 and 6. For the x position, a value is chosen between 0 and the highway length (initialized to 5000 m before the main)

```
for (NodeContainer::Iterator L = ueResponders.Begin(); L != ueResponders.End(); ++L)
{
  double yPos = laneNumber->GetInteger(1,6)*laneWidth; //MATTIA: generating y coordi
  double xPos = Xposition->GetValue(0,highwayLength); //MATTIA: generating x coordin
  positionAlloc ->Add(Vector(xPos, yPos, 0)); //MATTIA: adding position vector for t
}
```

The position allocator and the mobility model are set on the mobilityUE helper created before. The chosen model is a constant velocity mobility model.

```
mobilityUE.SetPositionAllocator(positionAlloc); //MATTIA: vector po
mobilityUE.SetMobilityModel("ns3::ConstantVelocityMobilityModel");
```

Now, the installation on the responding UEs takes place:

```
mobilityUE.Install (ueResponders);
```

With a for loop, the nodes are scanned and the constant velocity is set. Two steps: first, the mobility model is extracted from the node (from this model we retrieve the x position - mob->GetPosition().x - and the y position - mob->GetPosition().y -). Secondly, the velocity is set - VelMob->SetVelocity(Vector(x,y,z) -).

```
for (NodeContainer::Iterator L = ueResponders.Begin(); L != ueResponders.End(); ++L)
{
  Ptr<Node> node = *L;
  Ptr<MobilityModel> mob = node->GetObject<MobilityModel> (); //MATTIA: from here, e
  Ptr<ConstantVelocityMobilityModel> VelMob = node->GetObject<ConstantVelocityMobili
  if (mob->GetPosition().y > 13)
    VelMob->SetVelocity(Vector(19.44, 0, 0));
    // VelMob->SetVelocity(Vector(0, 0, 0));
  else
    VelMob->SetVelocity(Vector(-19.44, 0, 0));
    // VelMob->SetVelocity(Vector(0, 0, 0));
}
```

Velocity management for each node in the node container.

```
for (NodeContainer::Iterator L = ueResponders.Begin(); L != ueResponders.End(); ++L)
{
  int ID;
  Ptr<Node> node = *L;
  ID = node->GetId ();

  Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
  if (! mob) continue; // Strange -- node has no mobility model installed. Skip.
  Vector pos = mob->GetPosition ();

  PrevX.push_back(pos.x);
  PrevY.push_back(pos.y);
  PrevZ.push_back(pos.z);

  VelX.push_back(0);
  VelY.push_back(0);
  VelZ.push_back(0);
```

With the next lines, the VehicleTrafficType vector is filled based on the type of traffic that we desire for our simulation. The possibilities are:

1) **Aperiodic traffic**;
2) **Periodic traffic**;
3) **Mixed traffic** (50% aperiodic and 50% periodic);
4) **Mixed traffic** (10% aperiodic and 90% periodic);

These vectors will be used to retrieve values for each node and assign them to attributes of the NrV2XUdpClient class (discussed in the next section).

```
  if ((AperiodicTraffic))
  // 0x00 for periodic traffic. 0x01 for aperiodic traffic
    VehicleTrafficType.push_back(0x01);
  else if (PeriodicTraffic)
    VehicleTrafficType.push_back(0x00);
  else if (PeriodicPercentage == 10)
  {
    VehicleTrafficType.push_back(0x01);
    if (ID % 10 == 0)
        VehicleTrafficType.push_back(0x00);
  }
  else if (PeriodicPercentage == 50)
  {
    VehicleTrafficType.push_back(0x01);
    if (ID % 2 == 0)
        VehicleTrafficType.push_back(0x00);
  }
  else if (PeriodicPercentage == 90)
  {
    VehicleTrafficType.push_back(0x00);
    if (ID % 10 == 0)
        VehicleTrafficType.push_back(0x01);
  }
}
```

```
Sl3GPPChannelMatrix->InitChannelMatrix(ueResponders); //MATTIA: Matrix with 3GPP Channel Models

 // Define the packets inter-arrival time and size
 if ((MixedTraffic) || (AperiodicTraffic) || (PeriodicTraffic))
 {
   for (NodeContainer::Iterator L = ueResponders.Begin(); L != ueResponders.End(); ++L)
   {
     int ID;
     Ptr<Node> node = *L;
     ID = node->GetId ();
     if (ID % 5 != 0)
     {
       if (inputPDB != 0)
         PDB_Aperiodic.push_back(inputPDB);
       else
         PDB_Aperiodic.push_back(50);
       Aperiodic_Tgen_c.push_back(50);
     }
     else
     {
       if (inputPDB != 0)
         PDB_Aperiodic.push_back(inputPDB);
       else
         PDB_Aperiodic.push_back(50);
       Aperiodic_Tgen_c.push_back(50);
     }
   }
```

With these lines: the matrix containing channel models from 3GPP is initialized. The vectors PDB_Aperiodic and Aperiodic_Tgen_c are filled.

```
for (NodeContainer::Iterator L = ueResponders.Begin(); L != ueResponders.End(); ++L)
{
  int ID;
  Ptr<Node> node = *L;
  ID = node->GetId ();
  if (ID % 2 == 0)
  {
    if (inputPDB != 0)
      PDB_Periodic.push_back(inputPDB);
    else
      PDB_Periodic.push_back(100);
    Periodic_Tgen.push_back(100);
  }
  else
  {
    if (inputPDB != 0)
      PDB_Periodic.push_back(inputPDB);
    else
      PDB_Periodic.push_back(100);
    Periodic_Tgen.push_back(100);
  }
}
```

Here, the vectors PDB_Periodic and Periodic_Tgen are filled.
All these vectors will be used to retrieve values to assign to the attributes of the NrV2XUdpClient class.

```
MeasInterval = ((double)TrepPrint)/1000;
```

Measurement interval set.

```
NS_LOG_INFO ("Installing UE network devices...");
NetDeviceContainer ueDevs = lteHelper->InstallUeDevice (ueResponders); // MATTIA: ins

for (NodeContainer::Iterator L = ueResponders.Begin(); L != ueResponders.End(); ++L)
{
  Ptr<Node> node = *L;
  NS_LOG_INFO("Node " << node->GetId() << ", " << node->GetNDevices()); // MATTIA: Th
  Ptr<NetDevice> dev = node->GetDevice(0); // MATTIA: Retrieve the index-th NetDevice
  Ptr<NistLteUeNetDevice> netDev = dev->GetObject<NistLteUeNetDevice>();
  Ptr<NrV2XUeMac> mac = netDev->GetMac();

  if (FrequencyReuse)
    mac->CopySubchannelsMap(SubchannelIndexMap);

  if (PeriodicTraffic)
    mac->PushNewRRIValue(100);
  if (AperiodicTraffic)
  {
    if (avgRRI)
    {
      mac->PushNewRRIValue(50*2);
    }
    else
    {
      mac->PushNewRRIValue(50);
    }
  }
}
```

Now, we create a container for the network devices installed on the different nodes (ueDevs). Each node can have more network devices. You can think of network devices as interfaces.
Then, with a for loop we go through the node container created previously. For each node we retrieve the network device of interest (0 in our case, since it is the LTE

interface, and so ->GetDevice(0)) and then from the latter we retrieve the NIST net device (->GetObject<NistLteUeNetDevice>). NIST is a group that modified some portions of the stack.
From the stack that we have extracted we retrieve Mac specifications (netDev->GetMac()). If the traffic is periodic, we push an RRI value equal to 100ms. If the traffic is aperiodic, we push an RRI value equal to 100ms (if the option avgRRI is enabled) or to 50ms (if the option avgRRI is not enabled).
To install the IP stack, the following lines are used:

```
NS_LOG_INFO ("Installing IP stack...");
InternetStackHelper internet;        //MA
internet.Install (ueResponders); //MATTI
```

Also here, an helper (InternetStackHelper) is used. After is all set, the IP addresses are allocated and a network route is created as well.

```
NS_LOG_INFO ("Allocating IP addresses and setting up network route...");

Ipv4StaticRoutingHelper ipv4RoutingHelper;
Ipv4AddressHelper ipv4;
NS_LOG_INFO ("Assign IP Addresses.");

ipv4.SetBase ("10.1.0.0", "255.255.0.0");
Ipv4InterfaceContainer intcont = ipv4.Assign (ueDevs);
```

Two helpers are used (Ipv4StaticRoutingHelper and Ipv4AddressHelper). With the Setbase() function, we set the base network number, network mask and base address.

```
// Set up the IP network seen from the eNB
for (uint32_t u = 0; u < ueResponders.GetN (); ++u) // Could have used an iterator also here. Anyway, the resu
  {
    Ptr<Node> ueNode = ueResponders.Get (u);
      std::cout << ueNode -> GetObject<Ipv4> () -> GetAddress(1,0).GetLocal() << std::endl;
    // Set the default gateway for the UE
    Ptr<Ipv4StaticRouting> ueStaticRouting = ipv4RoutingHelper.GetStaticRouting (ueNode->GetObject<Ipv4> ());
    ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress (), 1);
   // ueStaticRouting->SetDefaultRoute (Ipv4Address::GetAny(), 1);
    //ueStaticRouting->SetDefaultMulticastRoute(1);
  }
```

An helper (namely, lteHelper) is used to connect the UEs to the lte network.
The role of such helper [3] is to create various LTE objects and arrange them together to make the whole LTE system (downlink spectrum channel, uplink spectrum channel, eNBs etc.).

```
NS_LOG_INFO ("Attaching UE's to LTE network...");
lteHelper->Attach (ueDevs);

BuildingsHelper::Install (ueResponders);
BuildingsHelper::MakeMobilityModelConsistent ();
```

```
NS_LOG_INFO ("Installing applications...");
// UDP application: in our case, this is the application that generates CAMs and DENMs

TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");

groupAddress = "225.0.0.1";
std::cout << "Group address " << groupAddress << std::endl;
NrV2XUdpClientHelper nrv2xudpClient (groupAddress , 8000); //set destination IP address

nrv2xudpClient.SetAttribute ("MaxPackets", UintegerValue (100000));
nrv2xudpClient.SetAttribute ("EnableTx", BooleanValue (true));

ApplicationContainer clientApps = nrv2xudpClient.Install(ueResponders);
Ptr<UniformRandomVariable> rand = CreateObject<UniformRandomVariable>();
```

Here, applications are installed on the UEs. An helper called NrV2XUdpClientHelper is used.

```
//@OLD  clientApps.Get(0) -> SetStartTime(Seconds (rand -> GetValue(0.001, 1.0)));
  Ptr<UniformRandomVariable> random_index = CreateObject<UniformRandomVariable>();
  for(uint32_t i = 0; i < ueResponders.GetN(); i++) //MATTIA: Application transmits CAM messages
  {
      //@DOC: Randomize the generation time of CAM messages
      Ptr<NrV2XUdpClient> yy = clientApps.Get(i)->GetObject<NrV2XUdpClient> ();
      uint32_t nodeID =  clientApps.Get(i)->GetNode()->GetId();

      //######## @MATTIA: Setting attributes for each node in the node container ueResponders ########
      //######## Done because maybe we want different behaviours for different nodes ########
      yy->m_VehicleTrafficType = VehicleTrafficType[nodeID-1];
      yy->m_LargestCAMSize = LargestCAMSize; //@MATTIA: SAME FOR ALL

      if(VehicleTrafficType[nodeID-1] == 0x00){ //---- IT'S PERIODIC TRAFFIC ----
        yy->m_LargestPeriodicSize = LargestPeriodicSize;
        yy->m_PeriodicTGen = Periodic_Tgen[nodeID-1];
        yy->m_PDBPeriodic = PDB_Periodic[nodeID-1];
        //yy->m_PeriodicPKTSize = PeriodicPKTs_Size;
        for(uint16_t j = 0; j < PeriodicPKTs_Size.size(); j++)
        {
           yy->PeriodicSizePushValue(PeriodicPKTs_Size[j]);
        }
        yy->m_PatternIndex = random_index->GetInteger(0,yy->m_PeriodicPKTSize.size()-1);


      }
```

Here, we are setting attributes for the NrV2XUdpClient class. We distinguish between periodic traffic (code above) and aperiodic traffic (code below).

```
    if(VehicleTrafficType[nodeID-1] == 0x01){ //---- IT'S APERIODIC TRAFFIC ----
      yy->m_LargestAperiodicSize = 1200;
      yy->m_AperiodicTGen = Aperiodic_Tgen_c[nodeID-1];
      if(nodeID == 1){ //INITIALIZE ONLY ONE TIME (AT THE FIRST PACKET)
        yy->InitExpRndVariable(double(Aperiodic_Tgen_c[nodeID-1]));
      }
      yy->m_PDBAperiodic = PDB_Aperiodic[nodeID-1];


      // uint16_t quantizationStep = 100;
      uint16_t quantizationStep = 200;
      LargestAperiodicSize = 1200; // Largest packet size for aperiodic traffic
      // LargestAperiodicSize = 100; // Largest packet size for aperiodic traffic
      for(uint16_t k = 1; k <= LargestAperiodicSize/quantizationStep; k++)
      {
         if(VariablePacketSize){ // Aperiodic traffic has variable packet size
             yy->AperiodicSizePushValue(k*quantizationStep-35);
         }
         else{ // Aperiodic traffic has fixed packet size
             yy->AperiodicSizePushValue(200-35);  // Valid packet sizes from 100 to 1000 bytes with 100 bytes quantiz
         }
      }

    }
    //######## @MATTIA: End of setting attributes for each node

    clientApps.Get(i) -> SetStartTime(Seconds (rand -> GetValue(0.001, 1.0)));
    clientApps.Get(i) -> SetStopTime (Seconds (simTime + 0.5)); // The simulation ends at simTime + 1, so we have a b
}

// Application to receive traffic
PacketSinkHelper clientPacketSinkHelper ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), 8000));
ApplicationContainer clientRespondersSrvApps = clientPacketSinkHelper.Install(ueResponders);
clientRespondersSrvApps.Start (Seconds (0.001));  // Start ASAP
clientRespondersSrvApps.Stop (Seconds (simTime+0.9));

// At this point, each V-UE mounts both a Client and a Server (Sink) application
```

Receiver side, we install a packet sink application. So, an application that only receives packets from the transmitting UEs (vehicles).

```
NS_LOG_INFO ("Creating sidelink configuration...");
uint32_t groupL2Address = 0x00;

Ptr<NistSlTft> tftRx = Create<NistSlTft> (NistSlTft::BIDIRECTIONAL, groupAddress, groupL2Address);
proseHelper->ActivateSidelinkBearer (Seconds(0.001), ueDevs, tftRx); // Set accordingly with clientApps StartTime! 0.

Ptr<LteUeRrcSl> ueSidelinkConfiguration = CreateObject<LteUeRrcSl> ();
ueSidelinkConfiguration->SetSlEnabled (true);

NistLteRrcSap::SlPreconfiguration preconfiguration;
preconfiguration.preconfigGeneral.carrierFreq = 54900; //not important
preconfiguration.preconfigGeneral.slBandwidth = channelBW_RBs;    //original: 50 PRBs = 10MHz
preconfiguration.preconfigComm.nbPools = 1; // the number of pools, not relevant for V2V

NistSlPreconfigPoolFactory pfactory;
NistSlResourcePoolFactory commfactory;
```

With the following code, the PSSCH bitmap value is built. Basically, we are deciding if we want to use the overall grid or not.

```
//build PSCCH bitmap value
uint64_t pscchBitmapValue = 0x0;
for (uint32_t i = 0 ; i < pscchLength; i++) {
  pscchBitmapValue = pscchBitmapValue >> 1 | 0x8000000000;
}
std::cout << "bitmap=" << std::hex << pscchBitmapValue << '\n'; // this is the PSCCH subframe poo

pfactory.SetControlBitmap (pscchBitmapValue);
pfactory.SetControlPeriod (period);
pfactory.SetDataOffset (pscchLength);

commfactory.SetControlBitmap (pscchBitmapValue);

pfactory.SetHaveUeSelectedResourceConfig(false); // Mode1 --> done in this way, it does not work

preconfiguration.preconfigComm.pools[0] = pfactory.CreatePool ();

ueSidelinkConfiguration->SetSlPreconfiguration (preconfiguration);

NS_LOG_INFO ("Installing sidelink configuration...");
lteHelper->InstallSidelinkConfiguration (ueDevs, ueSidelinkConfiguration);
```

Finally, the simulation is started and then stopped after simTime+1 seconds.

```
NS_LOG_INFO ("Starting simulation...");
Simulator::Stop (Seconds (simTime+1));
Simulator::Run ();
/*
  Put code to evaluate KPIs here
*/
Simulator::Destroy ();

NS_LOG_INFO ("Done.");

return 0;
```

Nr V2x Udp Client

The reference files are: “nr-v2x-udp-client.cc” and “nr-v2x-udp-client.h”.
Let’s start from “nr-v2x-udp-client.cc”.
At the beginning of the file, some header files and libraries are included:
A virtual constructor is also needed.

```
NrV2XUdpClient::~NrV2XUdpClient ()
{
  NS_LOG_FUNCTION (this);
}
```

Each node can produce more than one packet. So, you need a vector for each node to model the traffic. A vector for periodic traffic and a vector for aperiodic traffic have been created. They contain the size of the packets generated.
You can experience some troubles if you add a vector directly as an attribute. So, what you can do is to initialize a null vector in the constructor (see above) and then

create a function to fill the vector element by element. The function can be called in the main script.
In the following, you find the two functions to add new values to the vectors m_AperiodicPKTSize and m_PeriodicPKTSize. Both the two functions take as an input the new integer value that you want to add.

```
void
NrV2XUdpClient::AperiodicSizePushValue (uint16_t Avalue)
{
  NS_LOG_FUNCTION(this);
  m_AperiodicPKTSize.push_back(Avalue);
}

void
NrV2XUdpClient::PeriodicSizePushValue (uint16_t Pvalue)
{
  NS_LOG_FUNCTION(this);
  m_PeriodicPKTSize.push_back(Pvalue);
}
```

The reference files are nr-v2x-udp-client.cc and nr-v2x-udp-client.h.
The following function is used in order to set the remote address. As parameters, an ipv4 address and a port number are needed. The casting, Address(ip), is needed because the variable m_peerAddress of the class has an Address type [4].

```
void
NrV2XUdpClient::SetRemote (Ipv4Address ip, uint16_t port)
{
  NS_LOG_FUNCTION (this << ip << port);
  m_peerAddress = Address(ip);
  m_peerPort = port;
}
```

The following function is used to set the remote address. As parameters, an ipv6 address and a port number are needed. The casting, Address(ip), is needed because the variable m_peerAddress of the class has an Address type [4].

```
void
NrV2XUdpClient::SetRemote (Ipv6Address ip, uint16_t port)
{
  NS_LOG_FUNCTION (this << ip << port);
  m_peerAddress = Address(ip);
  m_peerPort = port;
}
```

The following function is used to set the remote address. As parameters, an address and a port number are needed. Here, the assignment is direct without casting [4].

```
void
NrV2XUdpClient::SetRemote (Address ip, uint16_t port)
{
  NS_LOG_FUNCTION (this << ip << port);
  m_peerAddress = ip;
  m_peerPort = port;
}
```

The following function is a destructor [5].

```
void
NrV2XUdpClient::DoDispose (void)
{
  NS_LOG_FUNCTION (this);
  Application::DoDispose ();
}
```

The following function is used to start the application. If the socket does not exist, it is created. If m_peerAddress is an ipv4 address, the method Bind() is used. If m_peerAddress is an ipv6 address, the method Bind6() is used [6].

```
void
NrV2XUdpClient::StartApplication (void)
{
  NS_LOG_FUNCTION (this);

  if (m_socket == 0)
    {
      TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
      m_socket = Socket::CreateSocket (GetNode (), tid);
      if (Ipv4Address::IsMatchingType(m_peerAddress) == true)
        {
          m_socket->Bind ();
          m_socket->Connect (InetSocketAddress (Ipv4Address::ConvertFrom(m_peerAddress), m_peerPort));
        }
      else if (Ipv6Address::IsMatchingType(m_peerAddress) == true)
        {
          m_socket->Bind6 ();
          m_socket->Connect (Inet6SocketAddress (Ipv6Address::ConvertFrom(m_peerAddress), m_peerPort));
        }
    }

  m_socket->SetRecvCallback (MakeNullCallback<void, Ptr<Socket> > ());
  m_sendEvent = Simulator::Schedule (Seconds (0.0), &NrV2XUdpClient::Send, this);
}
```

The following function is used to stop the application [7].

```
void
NrV2XUdpClient::StopApplication (void)
{
  NS_LOG_FUNCTION (this);
  Simulator::Cancel (m_sendEvent);
}
```

Now, we are going to discuss the most interesting method of the class: the Send method. This is modified with respect to the original Send method of the udp client class of ns3.

```
void
NrV2XUdpClient::Send (void)
{
  NS_LOG_FUNCTION (this);
  NS_ASSERT (m_sendEvent.IsExpired ());

  int TrepPrint = 100;

  Ptr<Node> currentNode = GetNode();   //@Mattia: creating a node
  uint32_t nodeId = currentNode -> GetId();    //@Mattia: getting the ID of the current node
  uint32_t ReservationSize;
  double numHops = 0;
  double T_gen = 0;
  bool insideTX = false;
  bool avgRRI = false;
  Ipv4Address groupAddress;
  groupAddress = "225.0.0.1";
```

At the beginning, we define some useful variables: nodeId contains the ID of the current node, ReservationSize, numHops and T_gen are used later on.
A uniform random variable called packetSize_index is also created.

```
Ptr<UniformRandomVariable> packetSize_index = CreateObject<UniformRandomVariable>();
//NS_LOG_INFO("Packet size index: "<<packetSize_index);
//std::cin.get();

//NS_LOG_INFO("Transmission enabled? " << m_EnableTX);
if(m_EnableTX)
{
  //NS_LOG_INFO("Transmission enabled");
  SeqTsHeader seqTs;  // Packet header for UDP Client/Server application
  seqTs.SetSeq (m_sent); // The packet header must be sequentially increased. Use the pa
  NrV2XTag v2xTag;
  v2xTag.SetGenTime (Simulator::Now ().GetSeconds ());
  v2xTag.SetMessageType (0x00); //CAM, @LUCA just want to work with CAMs
  v2xTag.SetTrafficType (m_VehicleTrafficType); // Coexistence of periodic and aperiodi
  v2xTag.SetPPPP (0x00); // PPPP for CAM
  v2xTag.SetPrsvp ((double) (1000 * m_interval.GetSeconds ())); // the required PHY res
  v2xTag.SetNodeId ((uint32_t) nodeId); // Encapsulate the nodeId. It will be used at MA
  v2xTag.SetReselectionCounter((uint16_t)10000); //safe value for standard-compliant Cr
```

An if else construct is used to understand if the transmission is enabled. In our case, m_EnableTX is fixed to TRUE and so we will always enter the if branch and we will never enter the else branch.
The sequence number and the timestamp are attached to the transmitted packet thanks to the variable seqTs.
Next, a tag is created (see the next section for more details). Through this tag, we set the generation time, the message type (0x00 and so CAM messages in this specific case), the traffic type, the PPPP, the Prsvp, the node ID and the reselection counter.
If the traffic type is aperiodic (0x01), the generation time T_gen is given by a constant (m_Aperiodic_TGen = 50) plus a random contribution. The random contribution is given by RndExp which is a static variable (defined in the header file). The reason for that is we want a unique “random container” for all nodes. From this “random container” we will extract random values. The mean value of this random variable is set to 50 from the main script (highwayprova.cc) but can be changed based on the type of simulation that we want to perform.

Next, the PDB is added to the tag. The Prsvp and the reservation size are updated. The packet size is also added to the tag. For aperiodic traffic the packet size can be fixed or variable (with quantization steps). You can choose to set or not the variable packet size with --VariableSize from the command prompt, when you launch the simulation. For periodic traffic, the size is always fixed.

```
if(m_VehicleTrafficType == 0x01) //It's aperiodic traffic
{

  //NS_LOG_INFO("Random 1: "<<RndExp);
  //NS_LOG_INFO("Random 2: "<<RndExp_1);

  if (nodeId % 2 == 0)
    {
      T_gen = m_AperiodicTGen + NrV2XUdpClient::RndExp->GetValue ();
      NS_LOG_INFO("Aperiodic inter-arrival time constant = " << m_AperiodicTGen << ", Exp mean = " << NrV2XUdpClient::
    }
    else
    {
      T_gen = m_AperiodicTGen + NrV2XUdpClient::RndExp->GetValue ();
      NS_LOG_INFO("Aperiodic inter-arrival time constant = " << m_AperiodicTGen << ", Exp mean = " << NrV2XUdpClient::

    }
    //v2xTag.SetPdb ((double)Aperiodic_Tgen_c[nodeId-1]); // @LUCA modified later
    v2xTag.SetPdb ((double) m_PDBAperiodic); // @LUCA modified later



    m_size = m_AperiodicPKTSize[packetSize_index->GetInteger(0,m_AperiodicPKTSize.size()-1)];



    //m_size = PacketSizeDistribution();
    //ReservationSize = LargestAperiodicSize;
    ReservationSize = m_size;


    if (avgRRI)
      v2xTag.SetPrsvp ((double) m_AperiodicTGen*2); // average RRI
    else
      v2xTag.SetPrsvp ((double) m_AperiodicTGen); // minimum RRI

    v2xTag.SetReservationSize((uint16_t) ReservationSize + 35);
    NS_LOG_UNCOND("Udp: UE " << nodeId << " transmitting packet with size: " << m_size+35 << " B and reserving resourc
    v2xTag.SetPacketSize((uint16_t) m_size + 35);

    fileprova.open("results/simulazioniMattia/TGenAndPktsSize.txt", std::ios_base::app);
    fileprova << Simulator::Now().GetSeconds() << " " << T_gen << " " << v2xTag.GetPacketSize() << std::endl;
    fileprova.close();

 }
```

```
else //It's periodic traffic
{

    T_gen = m_PeriodicTGen;


    v2xTag.SetPrsvp ((double) m_PeriodicTGen);
    //v2xTag.SetPdb ((double) v2xTag.GetPrsvp ()); // @LUCA modified later


    v2xTag.SetPdb (m_PDBPeriodic); // @LUCA modified later


    m_size = m_PeriodicPKTSize[m_PatternIndex%5];

    //ReservationSize = LargestPeriodicSize;
    ReservationSize = m_size;




    v2xTag.SetPacketSize((uint16_t) m_size + 35);
    v2xTag.SetReservationSize((uint16_t) ReservationSize + 35);
    NS_LOG_UNCOND("Udp: UE " << nodeId << " transmitting packet with size: " << m_size+35 << " B and reserving resourc


}
```

For periodic traffic, the generation time of the packets is fixed to 50 and does not have a random contribution.

```
NrV2XUdpClient::packetID = NrV2XUdpClient::packetID + 1; // increment the packet ID number
v2xTag.SetIntValue(NrV2XUdpClient::packetID);
v2xTag.SetPacketId(NrV2XUdpClient::packetID);
v2xTag.SetDoubleValue(Simulator::Now().GetSeconds());
Ptr<MobilityModel> mobility = GetNode()->GetObject<MobilityModel>();
Vector currentPos = mobility -> GetPosition();
double xPosition = currentPos.x;
double yPosition = currentPos.y;
// Set generation position in tag
v2xTag.SetGenPosX(xPosition);
v2xTag.SetGenPosY(yPosition);
v2xTag.SetNumHops(numHops); // 0 by default
/*  NOW CREATE THE PACKET */
Ptr<Packet> p = Create<Packet> (m_size-(8+4)); // 8+4 : the size of the seqTs header

p->AddHeader (seqTs);  // Add the header to the packet

p->AddByteTag (v2xTag); // Attach the tag
```

```
std::stringstream peerAddressStringStream;
if (Ipv4Address::IsMatchingType (m_peerAddress))
{
  peerAddressStringStream << Ipv4Address::ConvertFrom (m_peerAddress);
}
else if (Ipv6Address::IsMatchingType (m_peerAddress))
{
  peerAddressStringStream << Ipv6Address::ConvertFrom (m_peerAddress);
}
// my custom socket to send data
TypeId UDPtid = TypeId::LookupByName ("ns3::UdpSocketFactory");
// This method wraps the creation of sockets that is performed on a giver
Ptr<Socket> sendSocket = Socket::CreateSocket(currentNode, UDPtid);
sendSocket -> SetAllowBroadcast(true);  // Configure whether broadcast tr
Ipv4Address destAddress = groupAddress; // Important! The SlTft will chec
//Ipv4Address destAddress ("225.255.255.255");
uint16_t destPort = 8000;
```

These lines are directly taken from the Send method of the standard udp client class of ns3. They are used to convert ip addresses and to set the destination port to the value 8000.These lines are directly taken from the Send method of the standard udp client class of ns3. They are used to convert ip addresses and to set the destination port to the value 8000.

```
          if (Ipv4Address::IsMatchingType(destAddress) == true)
          {
            sendSocket->Bind ();
            sendSocket->Connect (InetSocketAddress (Ipv4Address::ConvertFrom(destAddress), destPort));
                //sendSocket->ShutdownRecv();
          }
          else if (Ipv6Address::IsMatchingType(destAddress) == true)
          {
            sendSocket->Bind6 ();
            sendSocket->Connect (Inet6SocketAddress (Ipv6Address::ConvertFrom(destAddress), destPort));
          }
          // Send data on the socket and check if returns an error or not (if -> Send fails it returns -1)
          if ((sendSocket->Send (p)) >= 0)
          {
            ++m_sent;
           // NS_LOG_INFO ("TraceDelay TX " << m_size << " bytes to " << peerAddressStringStream.str () << " Uid: " << p
            NS_LOG_INFO ("TraceDelay TX " << p->GetSize() << " bytes to " << peerAddressStringStream.str () << " Uid: " <
          }
          else
          {
            NS_LOG_INFO ("Error while sending " << m_size << " bytes to "
                                                << peerAddressStringStream.str ());
          }
      //    std::cin.get();

          //Ptr<UniformRandomVariable> rand = CreateObject<UniformRandomVariable>();
          NS_LOG_INFO("m_sent: " << m_sent << " m_count: " << m_count);
          if (m_sent < m_count)
          {
             if (m_VehicleTrafficType == 0x00)  //--- If it's PERIODIC traffic ---
             {
               m_PatternIndex++;
               //m_sendEvent = Simulator::Schedule (m_interval, &UdpClient::Send, this);
               m_sendEvent = Simulator::Schedule (MilliSeconds(T_gen), &NrV2XUdpClient::Send, this);
             }
             else //--- If it's APERIODIC traffic ---
             {
                 // NS_LOG_UNCOND("Node ID " << nodeId << " Tgen " << T_gen);
                 // std::cin.get();
                 m_sendEvent = Simulator::Schedule (MilliSeconds(T_gen), &NrV2XUdpClient::Send, this);
             }
          }
   } // end if Enable
```

Here, we create the socket and send packets.

```
    else
    {
      NS_LOG_INFO("Transmission not enabled");
      m_sendEvent = Simulator::Schedule (MilliSeconds(TrepPrint), &NrV2XUdpClient::Send, this);
    }

    //std::cin.get();
}

} // Namespace ns3
```

This last else statement is connected to m_EnableTX. If it is FALSE, then a message is displayed to advise that the transmission is not enabled.

In the header file nr-v2x-udp-client.h, you can find the declaration of all the attributes and the methods of the class. Below, the corresponding screenshots are reported.

```cpp
#ifndef NR_V2X_UDP_CLIENT_H
#define NR_V2X_UDP_CLIENT_H

#include "ns3/application.h"
#include "ns3/event-id.h"
#include "ns3/ptr.h"
#include "ns3/ipv4-address.h"
#include "ns3/random-variable-stream.h"

namespace ns3 {

class Socket;
class Packet;

/**
 * \ingroup udpclientserver
 * \class UdpClient
 * \brief A Udp client. Sends UDP packet carrying sequence number and time stamp
 *  in their payloads
 *
 */
class NrV2XUdpClient : public Application
{
public:
  /**
   * \brief Get the type ID.
   * \return the object TypeId
   */
  static TypeId GetTypeId (void);

  NrV2XUdpClient ();

  void Send (void);

  /**
   * \brief push back a new value in the AperiodicPKTSize vector
   * \param the new value to add
   */
  void AperiodicSizePushValue(uint16_t Avalue);

  /**
   * \brief push back a new value in the PeriodicPKTSize vector
   * \param the new value to add
   */
  void PeriodicSizePushValue(uint16_t Pvalue);
```

```
virtual ~NrV2XUdpClient ();

void InitExpRndVariable(double meanValue);

/**
 * \brief set the remote address and port
 * \param ip remote IPv4 address
 * \param port remote port
 */
void SetRemote (Ipv4Address ip, uint16_t port);
/**
 * \brief set the remote address and port
 * \param ip remote IPv6 address
 * \param port remote port
 */
void SetRemote (Ipv6Address ip, uint16_t port);
/**
 * \brief set the remote address and port
 * \param ip remote IP address
 * \param port remote port
 */
void SetRemote (Address ip, uint16_t port);

bool m_EnableTX; //!< Check if the transmission is enabled
uint8_t m_VehicleTrafficType; //!< Traffic type of the UE
std::vector<uint16_t> m_AperiodicPKTSize; //!< Size of packets for aperiodic traffic
std::vector<uint16_t> m_PeriodicPKTSize; //!< Size of packets for periodic traffic
//uint64_t m_packetID; //!< ID of the packet
uint16_t m_LargestPeriodicSize; //!< Largest size for periodic traffic
uint16_t m_LargestAperiodicSize; //!< Largest size for aperiodic traffic
uint16_t m_LargestCAMSize; //!< Largest size for CAM messages
bool m_enableUDPfiles; //!< Enable UDP files
uint16_t m_PatternIndex; //!< Pattern index for the node
uint16_t m_PeriodicTGen; //!< Generation period for periodic traffic
uint16_t m_AperiodicTGen; //!< Generation period for aperiodic traffic
double m_PDBPeriodic; //!< PDB for periodic traffic
double m_PDBAperiodic; //!< PDB for aperiodic traffic
std::string m_FilePath; //!< File path
```

```
  static uint64_t packetID;
  static Ptr<ExponentialRandomVariable> RndExp;

protected:
  virtual void DoDispose (void);

private:

  virtual void StartApplication (void);
  virtual void StopApplication (void);

  /**
   * \brief Send a packet
   */


  uint32_t m_count; //!< Maximum number of packets the application will send
  Time m_interval; //!< Packet inter-send time
  uint32_t m_size; //!< Size of the sent packet (including the SeqTsHeader)

  uint32_t m_sent; //!< Counter for sent packets
  Ptr<Socket> m_socket; //!< Socket
  Address m_peerAddress; //!< Remote peer address
  uint16_t m_peerPort; //!< Remote peer port
  EventId m_sendEvent; //!< Event to send the next packet

};

} // namespace ns3

#endif /* NR_V2X_UDP_CLIENT_H */
```

Concetto di Tag
nr-v2v-tag.cc/.h

Let's start with the file "nr-v2x-tag.cc".
The function Serialize takes as an input the TagBuffer i variable [8].
A tag is created for each packet sent (so, for each call of the NrV2XUdpClient::send() function).
Basically, the class TagBuffer allows the creation of subclasses of the ns3::Tag class. These subclasses are used to serialize and deserialize data through a stream-like API. This class keeps track of the current point in the buffer. This current point is advanced every time that a new data is written.
With the Serialize function, we write some data in the TagBuffer such as: the generation time of the packet, the X and Y coordinates of the UE, the ID of the packet, the ID of the node, the message type, the traffic type (periodic/aperiodic), the packet size, the reservation size and the value of the reselection counter. Also, you write the ProSe Per-Packet Priority (pppp) value, the packet delay budget (pdb) value, the rsvp value and the Modulation and Coding Scheme (mcs) value.

```
void
NrV2XTag::Serialize (TagBuffer i) const
{
  i.WriteU8 (m_simpleValue);
  i.WriteU64 (m_intValue);
  i.WriteDouble(m_doubleValue);

  i.WriteDouble(m_genTime);
  i.WriteDouble(m_genPosX);
  i.WriteDouble(m_genPosY);
  i.WriteU64 (m_packetId);
  i.WriteU32 (m_nodeId);
  i.WriteU32 (m_numHops);

  i.WriteU8 (m_messageType);
  i.WriteU8 (m_trafficType);
  i.WriteU16 (m_Cresel);
  i.WriteU16 (m_PacketSize);
  i.WriteU16 (m_ReservationSize);

  i.WriteU8 (m_pppp);
  i.WriteDouble (m_pdb);
  i.WriteU32 (m_p_rsvp);
  i.WriteU32 (m_mcs);

}
```

The function Desrialize is basically a dual function of Serialize. With this function you read the content stored in the TagBuffer i. So, instead of i.Write, you use i.Read.

```
void
NrV2XTag::Deserialize (TagBuffer i)
{
  m_simpleValue = i.ReadU8 ();
  m_intValue = i.ReadU64 ();
  m_doubleValue = i.ReadDouble();

  m_genTime = i.ReadDouble();
  m_genPosX = i.ReadDouble();
  m_genPosY = i.ReadDouble();
  m_packetId = i.ReadU64();
  m_nodeId = i.ReadU32();
  m_numHops = i.ReadU32();
  m_messageType = i.ReadU8();
  m_trafficType = i.ReadU8();
  m_Cresel = i.ReadU16();
  m_PacketSize = i.ReadU16();
  m_ReservationSize = i.ReadU16();
  m_pppp = i.ReadU8();
  m_pdb = i.ReadDouble();
  m_p_rsvp = i.ReadU32 ();
  m_mcs = i.ReadU32 ();


}
```
OBJ

Now, you can see a list of **Setters**. Setters are functions that allow you to assign a value (passed by the main) to a variable of the class.
If you look at the .cc file you can see that the Setters available are:

1) SetSimpleValue (only for test purposes)
2) SetIntValue;
3) SetDoubleValue;
4) SetGenTime (to set the generation time of the packet);
5) SetGenPosX (to set the x coordinate of the generation position);
6) SetGenPosY (to set the y coordinate of the generation position);
7) SetPacketID (to set the ID of the generated packet);
8) SetNodeID (to set the ID of the node - vehicle-)
9) SetNumHops;
10) SetMessageType (to set the type of the message transmitted);
11) SetTrafficType (to set the type of traffic generated - Aperiodic, Periodic, Mixed);
12) SetReselectionCounter (to set the value of the reselection counter);
13) SetPacketSize (to set the size of the packet);
14) SetReservationSize (to set the size of the reservation made);
15) SetPPPP;
16) SetPdb (to acquire the packet delay budget);
17) SetPrsvp;
18) SetMcs;

OBJ

As we have Setters, we have also **Getters**. Getters are functions that allow you to retrieve the value assigned to a variable of the class.

If you look at the .cc file you can see that the Getters available are:

19) GetSimpleValue (only for test purposes)
20) GetIntValue;
21) GetDoubleValue;
22) GetGenTime (to acquire the generation time of the packet);
23) GetGenPosX (to acquire the x coordinate of the generation position);
24) GetGenPosY (to acquire the y coordinate of the generation position);
25) GetPacketID (to acquire the ID of the generated packet);
26) GetNodeID (to acquire the ID of the node - vehicle-)
27) GetNumHops;
28) GetMessageType (to acquire the type of the message transmitted);
29) GetTrafficType (to acquire the type of traffic generated - Aperiodic, Periodic, Mixed);
30) GetReselectionCounter (to acquire the value of the reselection counter);
31) GetPacketSize (to acquire the size of the packet);
32) GetReservationSize (to acquire the size of the reservation made);
33) GetPPPP;
34) GetPdb (to acquire the packet delay budget);
35) GetPrsvp;
36) GetMcs;

In the file “nr-v2x-tag.h”, you can find the definition of all the attributes and the methods of the class.

```
#ifndef NR_V2X_TAG_H
#define NR_V2X_TAG_H

#include "ns3/tag.h"

namespace ns3 {

class Tag;

class NrV2XTag : public Tag
{
public:
  static TypeId GetTypeId (void);
  virtual TypeId GetInstanceTypeId (void) const;
  virtual uint32_t GetSerializedSize (void) const;
  virtual void Serialize (TagBuffer i) const;
  virtual void Deserialize (TagBuffer i);
  virtual void Print (std::ostream &os) const;

  // these are our accessors to our tag structure

  //Setters
  void SetSimpleValue (uint8_t value);
  void SetIntValue (uint64_t value);
  void SetDoubleValue (double value);

  void SetGenTime(double value);
  void SetPacketId(uint64_t value);
  void SetGenPosX(double value);
  void SetGenPosY(double value);
  void SetNodeId(uint32_t value);
  void SetNumHops(uint32_t value);

  void SetMessageType(uint8_t value);
  void SetTrafficType(uint8_t value);
  void SetReselectionCounter(uint16_t value);
  void SetPacketSize(uint16_t value);
  void SetReservationSize(uint16_t value);

  void SetPPPP (uint8_t value);
  void SetPdb (double value);
  void SetPrsvp (uint32_t value);
  void SetMcs (uint32_t value);
```

```
// Getters
  uint8_t GetSimpleValue (void) const;
  uint64_t GetIntValue (void) const;
  double GetDoubleValue (void) const ;

  double GetGenTime(void) const;
  uint64_t GetPacketId(void) const;
  double GetGenPosX(void) const;
  double GetGenPosY(void) const;
  uint32_t GetNodeId(void) const;
  uint32_t GetNumHops(void) const;

  uint8_t GetMessageType(void) const;
  uint8_t GetTrafficType(void) const;
  uint16_t GetReselectionCounter(void) const;
  uint16_t GetPacketSize(void) const;
  uint16_t GetReservationSize(void) const;

  uint8_t GetPPPP (void) const;
  double GetPdb (void) const;
  uint32_t GetPrsvp (void) const;
  uint32_t GetMcs (void) const;
```

```
private:
  uint8_t m_simpleValue;
  uint64_t m_intValue;
  double m_doubleValue;

  double m_genTime;
  uint64_t m_packetId;
  double m_genPosX;
  double m_genPosY;
  uint32_t m_nodeId;
  uint32_t m_numHops;

  uint8_t m_messageType;  // 0x00 = CAM
                          // 0x01 = DENM

  uint8_t m_trafficType;  // 0x00 = Periodic
                          // 0x01 = Aperiodic

  uint16_t m_Cresel;  // 0x00 = Periodic
                      // 0x01 = Aperiodic

  uint16_t m_PacketSize;
  uint16_t m_ReservationSize;

  uint8_t m_pppp; // the ProSe Per-Packet Priority
  double m_pdb; // the Packet Delay Budget (over the single hop)
  uint32_t m_p_rsvp; // the reservation period for this packet
  uint32_t m_mcs; // the Modulation and Coding Scheme to be used by lower layers for transmission
};

} //namespace ns3
#endif /* NR_V2X_TAG_H */
```

UE RLC

UE MAC

The reference files are “nr-v2x-ue-mac.cc” and “nr-v2x-ue-mac.h”.

The type NrV2XUeMac has some attributes.

1) SlGrantMcs: to define the Modulation and Coding Scheme
2) RandomV2VSelection: to define if the resources for V2V transmissions are randomly selected or not. The default value is false and so the resources are not randomly selected
3) SlSharedChUeScheduling: to obtain information about the sidelink shared channel
4) ListL2Enabled: to enable list L2 used in the SSPS algorithm of Mode 4 (LTE-V2X). Since L2 is not used in NR-V2X Mode 2, the default value is false
5) UseRxCresel: to use the received value of reselection counter contained in the SCI
6) DynamicScheduling: used to enable Mode 2 dynamic scheduling. The default value is false and so Mode 2 dynamic scheduling is not enabled
7) MixedTraffic: used to enable mixed traffic scheduling. The default value is false and so mixed traffic scheduling is not enabled
8) OneShot: to use a one-shot transmission when a reselection is triggered. Default value set to false
9) RSRPthreshold: to set the RSRP threshold used for excluding reserved resources. The default value is -128.0 dBm
10) EnableReTx: used to enable retransmissions. The default value is false and so retransmissions are not enabled
11) SubchannelSize: to define the subchannel size (in Resource Blocks). The default value is 10. Remember that in the resource grid, on the vertical axis we have frequencies and the elementary unit is the resource block which is 180 KHz wide. A group of adjacent resource blocks is a subchannel
12) RBsBandwidth: used to define the bandwidth at the MAC Sublayer (expressed in number of RBs). The default value is 50 and so the bandwidth is: 180 KHz * 50 = 9000 KHz = 9 MHz
13) SlotDuration: to define the time slot duration. The default value is 1.0 ms. Remember that in the resource grid, on the horizontal axis we have time and the elementary unit is the subframe which has a duration of 1 ms
14) NumerologyIndex: to define the numerology index
15) AllowReEvaluation: to enable the reevaluation mechanism. The default value is false and so the reevaluation mechanism is not enabled
16) AllSlotsReEvaluation:
17) UMHReEvaluation:
18) FrequencyReuse: used to enable frequency reuse scheduling. The default value is false and so frequency reuse scheduling is not enabled
19) OutputPath: used to specify the output path (folder) in which to save the results
20) SavingPeriod: used to configure the period for saving data. The default value is 1 s

Next, we have the constructor in which some variables are initialized.

```
NrV2XUeMac::NrV2XUeMac ()
:
//   m_RRIvalues ({3, 11, 20, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000})
//   m_RRIvalues ({20, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000})
   m_RRIvalues ({}),
   m_cphySapProvider (0),
   m_bsrPeriodicity (MilliSeconds (1)), // ideal behavior
   m_bsrLast (MilliSeconds (0)),
   m_freshUlBsr (false),
   m_harqProcessId (0),
   m_rnti (0),
   m_rachConfigured (false),
   m_waitingForRaResponse (false),
   m_slBsrPeriodicity (MilliSeconds (1)),
   m_slBsrLast (MilliSeconds (0)),
   m_freshSlBsr (false),
   m_alreadyUeSelectedSlBsr (false), //to check again whether there is a fresh SL BSR
   m_absSFN (0), //the absolute SFN cycle number --> incremented every 1024 frames
   m_millisecondsFromLastAbsSFNUpdate (0),
//   m_nsubCHsize (10),
   m_L_SubCh (1),
//   m_BW_RBs (50),
   m_maxPDB(110.0),
   m_keepProbability (0.0),
   m_sizeThreshold (0.2),
   m_sensingWindow (1100),
   m_oneShotGrant (false)
{
   NS_LOG_FUNCTION (this);

   NS_ASSERT_MSG(m_RRIvalues.size() < 16, "Maximum size of the RRI list is 16");
   NS_ASSERT_MSG(m_keepProbability >= 0, "Keep probability must be non-negative");

   m_debugNode = 0;

   ReservationsInfo initEntry;
   initEntry.UnutilizedSubchannelsRatio = {};
   initEntry.UnutilizedReservations = 0;
   initEntry.Reservations = 0;
   initEntry.LatencyReselections = 0;
   initEntry.SizeReselections = 0;
   initEntry.CounterReselections = 0;
   initEntry.TotalTransmissions = 0;
```

m_RRIvalues({}) is a vector that contains RRI values. To fill this vector from the main scripti, a method PushNewRRIValue is defined later on.
m_sensingWindow is initialized to 1100 ms (dimension of the sensing window).
With the following lines, we initialize the entry of the map.
ReservationsInfo initEntry;
initEntry.UnutilizedSubchannelsRatio = {};
initEntry.UnutilizedReservations = 0;
initEntry.Reservations = 0;
initEntry.LatencyReselections = 0;
initEntry.SizeReselections = 0;
initEntry.CounterReselections = 0;
initEntry.TotalTransmissions = 0;

Setting the Service Access Point (SAP) to communicate from the Ue Radio Resource Control (RRC) layer to the Ue Physical layer. (Control plane) [10]. (TX).

```
void
NrV2XUeMac::SetNistLteUeCphySapProvider (NistLteUeCphySapProvider * s)
{
  NS_LOG_FUNCTION (this << s);
  m_cphySapProvider = s;
}
```

Getting the SAP that is used to communicate from the Ue Physical layer to the Ue MAC sublayer. (Data plane) [10]. (RX).

```
NistLteUePhySapUser*
NrV2XUeMac::GetNistLteUePhySapUser (void)
{
        return m_uePhySapUser;
}
```

Setting the SAP to communicate from the Ue MAC sublayer to the Ue Physical layer. (Data plane) [10]. (TX).

```
void
NrV2XUeMac::SetNistLteUePhySapProvider (NistLteUePhySapProvider* s)
{
       m_uePhySapProvider = s;
}
```

Getting the SAP to communicate between the Radio Link Control (RLC) sublayer and the MAC sublayer. (Data plane) [10]. (TX).

```
NistLteMacSapProvider*
NrV2XUeMac::GetNistLteMacSapProvider (void)
{
        return m_macSapProvider;
}
```

Setting the SAP to communicate from the Ue MAC sublayer to the Ue RRC sublayer. (Control plane) [10]. (RX).

```
void
NrV2XUeMac::SetNistLteUeCmacSapUser (NistLteUeCmacSapUser* s)
{
       m_cmacSapUser = s;
}
```

Getting the SAP to communicate from the Ue RRC sublayer to the Ue MAC sublayer. (Control plane) [10]. (TX).

```
NistLteUeCmacSapProvider*
NrV2XUeMac::GetNistLteUeCmacSapProvider (void)
{
        return m_cmacSapProvider;
}
```

NEW
V2X Select resources

```
NrV2XUeMac::V2XSidelinkGrant
NrV2XUeMac::V2XSelectResources (uint32_t frameNo, uint32_t subframeNo, double pdb, double p_rsvp, uint8_t v2xMessageType,
{
   NS_LOG_FUNCTION(this);

   V2XSidelinkGrant V2XGrant;

   NS_ASSERT_MSG(pdb < m_maxPDB, "Current implementation allows only PDB values smaller than 110 ms");

   frameNo --;
   subframeNo --;

   std::vector<uint16_t>::iterator RRIit;
   RRIit = find(m_RRIvalues.begin(), m_RRIvalues.end(), p_rsvp);
   NS_ASSERT_MSG(RRIit != m_RRIvalues.end(), "RRI not included in the list of allowed RRI values");
//   NS_LOG_UNCOND(*RRIit);

//   NS_ASSERT_MSG(ReservationSize >= PacketSize, "Reservation size must be larger than the packet size (in bytes)"); //R
   NS_ASSERT_MSG(pdb <= p_rsvp, "Packet Delay Budget (PDB) must be lower or equal than the reservation period");

   V2XGrant.m_mcs = m_slGrantMcs;
   V2XGrant.m_RRI = p_rsvp;



   if (ReselectionCounter == 0)
   {
     NS_ASSERT_MSG(false, "Reselection counter = 0, check the CAM trace! Node ID " << m_rnti);
   }
   if (p_rsvp == 0)
   {
     NS_ASSERT_MSG(false, "Reservation period = 0, check the CAM trace! Node ID " << m_rnti);
   }
```

V2XGrant → what is a grant? A grant can be viewed as a sort of permission to transmit. If you have a valid grant, you are allowed to transmit something at a specific subframe. If the grant is expired, you cannot transmit. For example, a grant can expire when the reselection counter goes to zero and you have to select new resources.
The grant is implemented as a data structure that contains several information such as the node id, the RRI value and so on.
With NS_ASSERT_MSG → check if the PDB (parameter passed to the function) exceeds the max PDB or not.
Decrease the frame number and the subframe number (parameters passed to the function). ???? MAYBE BECAUSE THE FRAME NUMBER AND THE SUBFRAME NUMBER GOES FROM 1 TO 10 AND FROM 1 TO 1024 AND WE NEED TO DECREASE THEM BY 1.

Iterate through the possible RRI values and find if p_rsvp (reservation period) is contained. If it is not contained, display an error message because you are not using one of the allowed RRI values.
Next, with another NS_ASSERT_MSG → check if the PDB is lower with respect to the reservation period. It must be like that. Otherwise, the booked resource will arrive after the packet is expired. If the condition is not satisfied, display an error message.
Assign the Modulation and Coding Scheme (MCS) → V2XGrant.m_mcs = m_slGrantMcs.
Assign the RRI → V2XGrant.m_RRI = p_rsvp.
If the reselection counter is zero → display message. Need to select a new resource with probability 1 - P.
If the reservation period is zero → display message. Means that you will not use the same resource for the next transmission.

```
// Assigning the reselection counter value
if (ReselectionCounter == 10000)
{
  V2XGrant.m_Cresel = GetCresel (p_rsvp);
}
else
  V2XGrant.m_Cresel = ReselectionCounter;

if (m_mixedTraffic)
{
  if ((m_dynamicScheduling) && (v2xTrafficType == 0x01))
    V2XGrant.m_Cresel = 1;
}
else
{
  if (m_dynamicScheduling)
    V2XGrant.m_Cresel = 1;
}
```

If the value in tag is 10000 → use the standard value for the reselection counter. 10000 is a sort of dummy value.
Check the traffic type.
If you have mixed traffic, if you are using dynamic scheduling and the traffic is aperiodic, set the reselection counter to 1. Nodes that are producing periodic traffic use the SSPS, so the reselection counter will not be one for such nodes.
If you do not have mixed traffic, the reselection counter is set to 1 for all nodes, weather the traffic is periodic or aperiodic.
This can be changed based on the type of simulations that you have to do.

```
SidelinkCommResourcePool::SubframeInfo currentSF;
currentSF.frameNo = frameNo;
currentSF.subframeNo = subframeNo;
```

Getting current frame and subframe number.

```
uint16_t nsubCHsize = m_nsubCHsize; // [RB]
//uint16_t startRBSubchannel = 0;
uint16_t NSubCh; //the total number of subchannels
NSubCh = std::floor(m_BW_RBs / nsubCHsize);
uint16_t L_SubCh = m_L_SubCh, L_RBs; // the number of subchannels for the reservation
```

The total number of subchannels is found by dividing the total bandwidth (measured in RBs) and the size of a subchannel (measured in RBs). std::floor to round up the result.
Initialize the number of subchannels for the reservation.

```
uint32_t AdjustedPacketSize, AdjustedReservationSize;
AdjustedPacketSize = m_NRamc->GetSlSubchAndTbSizeFromMcs (PacketSize, V2XGr
L_SubCh/= m_nsubCHsize;
NS_LOG_DEBUG("Reserving resources for packet size: " << PacketSize << " B, 

AdjustedReservationSize = m_NRamc->GetSlSubchAndTbSizeFromMcs (ReservationS
L_SubCh/= m_nsubCHsize;
NS_LOG_DEBUG("Actual reservation size is: " << ReservationSize << " B, adju
```

AdjustedPacketSize = m_NRamc->GetSlSubchAndTbSizeFromMcs(): Get the Transport Block Size for a selected MCS and number of (Physical Resource Block) PRB.
Call the method two times for the AdjustedPacketSize and the AdjustedReservationSize. Divide by 8 for conversion from bytes to bits.

```
//  uint16_t T_1 = 4; // should be T1 = 4 (see 3GPP)
  double T_2 = pdb - m_slotDuration;
//   uint16_t T_1_slots = T_1/m_slotDuration;
  uint32_t T_2_slots = T_2/m_slotDuration;

  // these values should be converted in slot values, now they rely on the assumpt
//  uint16_t nbRb_Pssch = L_SubCh*nsubCHsize - 2; //FIXME: this applies to the adj
  uint16_t nbRb_Pssch = L_SubCh*nsubCHsize ; //FIXME: Mode 2 PSSCH
  uint16_t nbRb_Pscch = nsubCHsize ; //FIXME: Mode 2 PSCCH (Occupy only the first

  Ptr<UniformRandomVariable> uniformRnd = CreateObject<UniformRandomVariable> ();

  // Second Option to build the list
  std::map<uint16_t,std::list<SidelinkCommResourcePool::SubframeInfo> > Sa, L1;
  std::vector<CandidateCSRl2> finalL2;
  uint32_t iterationsCounter = 0;
  double psschThresh = m_rsrpThreshold;
```

Set T2 (Nr-V2X Mode 2). Measured in number of slots.

Create a uniform random variable-

Create the resource pools → Sa collects resources from selection window, L1 is the list after the removal of some resources.
Setting iterations counter to zero.
Setting threshold to PSSCH.

```
Sa = SelectionWindow (currentSF, T_2_slots, NSubCh - L_SubCh + 1);

NS_LOG_DEBUG("Initial list Sa size: " << ComputeResidualCSRs (Sa) );

NS_LOG_DEBUG("Saving initial Sa");
if (m_rnti == m_debugNode)
{
  std::ofstream SaFileAlert;
  SaFileAlert.open (m_outputPath + "SafileAlert.txt", std::ios_base::app);
  SaFileAlert << "-----Initial Sa------ At time " << Simulator::Now().GetSeco
  for(std::map<uint16_t,std::list<SidelinkCommResourcePool::SubframeInfo> >::
  {
    for(std::list<SidelinkCommResourcePool::SubframeInfo>::iterator FrameIT =
      SaFileAlert << "CSR index " << SaIt->first << " Frame " << FrameIT->fra
  }
  SaFileAlert.close ();
}

uint32_t nCSRinitial = ComputeResidualCSRs (Sa);
uint32_t nCSRpastTx, nCSRfinal;
```

Sa = SelectionWindow() → select candidate resources from selection window.
Saving in a text file the result.

```
 if (!m_randomSelection)
 {
   L1 = Mode2Step1 (Sa, currentSF, V2XGrant, T_2, NSubCh, L_SubCh, &iterationsCo
 }
 else
 {
   //In case of random selection, L1 is filled with all CSRs
   NS_FATAL_ERROR("Random selection is not implemented!");
//   L1 = Sa; // Already assigned
 }
/  std::cin.get();
 nCSRfinal = ComputeResidualCSRs (L1);

 NS_LOG_INFO("Initial list size = " << nCSRinitial << ". After removing past Tx
```

If there is not a random selection, implement the first step of Mode 2 algorithm and find the resource list L1.
Else, random selection is not implemented.
Candidate Single-Subframe Resources —> compute the residual from list L1

```
if (m_List2Enabled)
{
  NS_ASSERT_MSG(false, "List L2 is temporarily not available");
}
else
{
  std::vector<CandidateCSRl2> L2EquivalentVector;
  CandidateCSRl2 FinalL2tmpItem;
  for(std::map<uint16_t,std::list<SidelinkCommResourcePool::SubframeInfo> >::iterator L1it = L1.begin(); L1it != L1.end
  {
    //  NS_LOG_DEBUG("CSR index " << L1it->first);
    for(std::list<SidelinkCommResourcePool::SubframeInfo>::iterator FrameIT = L1it->second.begin(); FrameIT != L1it->se
    {
      FinalL2tmpItem.CSRIndex = L1it->first;
      FinalL2tmpItem.subframe = (*FrameIT);
      FinalL2tmpItem.rssi = 0;
      finalL2.push_back (FinalL2tmpItem);
    }
    //   NS_LOG_DEBUG("Frame " << FrameIT->frameNo << " subframe " << FrameIT->subframeNo);
  }

  if (m_rnti == m_debugNode)
  {
    std::ofstream L2fileAlert;
    L2fileAlert.open (m_outputPath + "L2fileAlert.txt", std::ios_base::app);
    L2fileAlert << "At " << Simulator::Now ().GetSeconds () << " SF(" << currentSF.frameNo << "," << currentSF.subframe
    std::vector<CandidateCSRl2>::iterator L2ItDebug;
    for (L2ItDebug = finalL2.begin (); L2ItDebug != finalL2.end (); L2ItDebug++)
    {
      L2fileAlert << "CSRindex: " << (int) (*L2ItDebug).CSRIndex << ", SF(" <<  (*L2ItDebug).subframe.frameNo << "," <<
  //       NS_LOG_DEBUG("CSRindex: " << (int) (*L2ItDebug).CSRIndex << ", SF(" <<  (*L2ItDebug).subframe.frameNo << "
    }
    L2fileAlert.close ();
 //      std::cin.get();
  }

}
```

If List 2 is enabled → display alert message because List L2 is not used in Mode 2.
Else, iterate through the list.
Rnti stands for Radio Network Temporary Identifier.
If we are at a debug node, fill a txt file with the corresponding information. This allow you to debug a specific node. When m_debugNode is set to zero, it means that you are not debugging any node.

```
uint16_t firstSelectedCSR;
SidelinkCommResourcePool::SubframeInfo firstSelectedSF;

uint16_t secondSelectedCSR;
SidelinkCommResourcePool::SubframeInfo secondSelectedSF;

if (m_randomSelection)
{
  NS_FATAL_ERROR("Random selection not implemented");
}
else // if list L2 Enabled
{
  Ptr<UniformRandomVariable> selectFromL2 = CreateObject<UniformRandomVariable> ();
  CandidateCSRl2 FirstSelectedResource = finalL2[selectFromL2 -> GetInteger (0, finalL2.size () - 1)];

  firstSelectedCSR = FirstSelectedResource.CSRIndex;
  firstSelectedSF = FirstSelectedResource.subframe;

  if ((m_dynamicScheduling) && (m_FreqReuse))
  {
    NS_LOG_DEBUG("Frequency-reuse scheduling is enabled");
    NodeContainer GlobalContainer = NodeContainer::GetGlobal();
    Ptr<Node> Node;
    Vector posNode;
    Ptr<MobilityModel> mobNode;
    for (NodeContainer::Iterator L = GlobalContainer.Begin(); L != GlobalContainer.End(); ++L)
    {
      Node = *L;
      if (Node->GetId() == m_rnti)
      {
        mobNode = Node->GetObject<MobilityModel>();
        break;
      }
    }
    posNode = mobNode->GetPosition();
    NS_LOG_DEBUG("Node " << m_rnti << " at X = " << posNode.x << " meters");
```

Random selection is not implemented. Select randomly a resource from L2 and store it into FirstSelectedResource. Now, extract the CSR index and store it into firstSelectedCSR. Then, extract the subframe and store it into firstSelectedSF.
If there is dynamic scheduling and frequency reuse.
Ignore parts with frequency reuse.
If dynamic scheduling and frequency reuse are enabled, iterate through the node container until you find the node rnti. Here, you extract the mobility model.

```
  for (std::map < uint16_t, std::vector < std::pair <double, double>>>::iterator mapIT = m_subchannelsMap.begin(); m
  {
    std::vector < std::pair <double, double>> GeoCellsVector = mapIT->second;
    for(std::vector < std::pair <double, double>>::iterator GeoCellsIT = GeoCellsVector.begin(); GeoCellsIT != GeoCe
    {
      if ((posNode.x >= GeoCellsIT->first) && (posNode.x < GeoCellsIT->second))
        firstSelectedCSR = mapIT->first;
    }
  }

  firstSelectedSF.frameNo = frameNo;
  firstSelectedSF.subframeNo = subframeNo + 1;
  if (firstSelectedSF.subframeNo > 9)
  {
    firstSelectedSF.subframeNo = firstSelectedSF.subframeNo % 10;
    firstSelectedSF.frameNo++;
  }
  firstSelectedSF.frameNo = firstSelectedSF.frameNo % 1024;

} // end      if ((m_dynamicScheduling) && (m_FreqReuse))

NS_LOG_DEBUG("Now: UE " << m_rnti << " at SF(" << currentSF.frameNo << "," << currentSF.subframeNo << ") Selected CS

V2XGrant.m_TxNumber = 1;
V2XGrant.m_TxIndex = 1;

V2XSchedulingInfo firstSelection, secondSelection;
firstSelection.m_rbLenPssch = nbRb_Pssch;
firstSelection.m_rbLenPscch = nbRb_Pscch;
firstSelection.m_nextReservedSubframe = firstSelectedSF.subframeNo+1;
firstSelection.m_nextReservedFrame = firstSelectedSF.frameNo+1;
firstSelection.m_rbStartPscch = firstSelectedCSR * nsubCHsize; // (Mode 2)
firstSelection.m_rbStartPssch = firstSelectedCSR * nsubCHsize; // (Mode 2)

SidelinkCommResourcePool::SubframeInfo firstReEvaluationSF = ComputeReEvaluationFrame(firstSelection.m_nextReservedF
firstSelection.m_ReEvaluationFrame = firstReEvaluationSF.frameNo;
firstSelection.m_ReEvaluationSubframe = firstReEvaluationSF.subframeNo;
firstSelection.m_EnableReEvaluation = true;
firstSelection.m_SelectionTrigger = V2Xtrigger;
firstSelection.m_announced = false;
```

NOTE: Remember that the resource grid is divided into subframes. Each subframe corresponds to 1ms. A frame is defined as a group of 10 subframes. The subframe number can assume values between 0 and 9. The frame number can assume values between 0 and 1023. As a consequence, each time that you increase the frame number or the subframe number, you have to check if you exceed these limits. The modulo operator can be employed in this case: by doing the modulo 10 operation you obtain always values between 0 and 9. By doing the modulo 1024 operation you obtain always values between 0 and 1023.

firstSelection / secondSelection → when you have retransmissions. If you do not have retransmissions you use firstSelection. If you have retransmissions you use secondSelection.

```
    NS_LOG_INFO("Slots difference = " << EvaluateSlotsDifference(firstSelectedSF, secondSelectedSF, m_maxPDB/m_slotD

    V2XGrant.m_grantTransmissions = UnimoreSortSelections(firstSelection, secondSelection, m_maxPDB/m_slotDuration);

    if (!enableSecondReEvaluation)
      V2XGrant.m_grantTransmissions[2].m_EnableReEvaluation = false;
  }
  else
  {
    NS_LOG_INFO("Not enough resources for selecting a re-transmission");
    V2XGrant.m_grantTransmissions.insert(std::pair<uint16_t, V2XSchedulingInfo> (1, firstSelection));
    NS_ASSERT_MSG(false, "Not enough resources for selecting a re-transmission");
  }
}
else
{
  V2XGrant.m_grantTransmissions.insert(std::pair<uint16_t, V2XSchedulingInfo> (1, firstSelection));
}
```

```
  V2XGrant.m_tbSize = 0; //computed later

  /* std::ofstream selectedFile;
  selectedFile.open(m_outputPath + "selectedFile.txt", std::ios_base::app);
  selectedFile << "Now true: SF(" << currentSF.frameNo+1 << "," << currentSF.subframeNo+1 << ", Selected true: SF(" << V
  selectedFile.close (); */
  if (m_randomSelection)
  {
    NS_FATAL_ERROR("Random selection not yet implemented");
  }
  else
  {
//    V2XGrant.m_rbLenPscch = 2; //fixed and standardized (Mode 4)
//    V2XGrant.m_rbStartPscch = firstSelectedCSR * nsubCHsize; // (Mode 4)
 //   V2XGrant.m_rbStartPscch = firstSelectedCSR * nsubCHsize; // (Mode 2)
  //  V2XGrant.m_rbStartPssch = firstSelectedCSR * nsubCHsize; // (Mode 2)
  }

//  SidelinkCommResourcePool::SubframeInfo ReEvaluationSF = ComputeReEvaluationFrame(V2XGrant.m_nextReservedFrame, V2XGr
//  V2XGrant.m_ReEvaluationFrame = ReEvaluationSF.frameNo;
//  V2XGrant.m_ReEvaluationSubframe = ReEvaluationSF.subframeNo;

/   V2XGrant.m_EnableReEvaluation = true;
/   V2XGrant.m_SelectionTrigger = V2Xtrigger;

  NS_LOG_DEBUG(Simulator::Now ().GetSeconds () << " UE " << m_rnti << " selected " << V2XGrant.m_grantTransmissions.size
  for (std::map<uint16_t, V2XSchedulingInfo>::iterator grantsIT = V2XGrant.m_grantTransmissions.begin(); grantsIT != V2X
  {
    NS_LOG_DEBUG("Selection " << grantsIT->first << " at SF(" << grantsIT->second.m_nextReservedFrame << "," << grantsIT
    << grantsIT->second.m_rbStartPssch << ", rbLen (PSCCH) " << grantsIT->second.m_rbLenPscch << ", rbLen (PSSCH) " << g
    << " at SF(" << grantsIT->second.m_ReEvaluationFrame << "," << grantsIT->second.m_ReEvaluationSubframe << ")");
  }

  UeSelectionInfo tmp;
  tmp.selGrant = V2XGrant;
  tmp.RSRPthresh = psschThresh-3;
  tmp.iterations = iterationsCounter;
  tmp.time = Simulator::Now ().GetSeconds();
  tmp.selFrame.frameNo = currentSF.frameNo;
  tmp.selFrame.subframeNo = currentSF.subframeNo;
  tmp.nodeId = m_rnti;
  tmp.nCSRfinal = nCSRfinal;
  tmp.nCSRpastTx = nCSRpastTx;
  tmp.nCSRinitial = nCSRinitial;
  tmp.pdb = pdb;
```

Some debugging information about the selected resources.
Setting tmp attributes. Push back tmp in the selected grants vector. ???? What is tmp and for what is it used.

```
NrV2XUeMac::SelectedGrants.push_back(tmp);

if (Simulator::Now ().GetSeconds() - NrV2XUeMac::prevPrintTime_selection > m_savingPeriod)
{
  NrV2XUeMac::prevPrintTime_selection = Simulator::Now ().GetSeconds();
  std::ofstream SSPSlog;
  SSPSlog.open (m_outputPath + "SSPSlog.txt", std::ios_base::app);
  for(std::vector<UeSelectionInfo>::iterator selIT = NrV2XUeMac::SelectedGrants.begin(); selIT != NrV2XUeMac::Selected
  {
    for (std::map<uint16_t, V2XSchedulingInfo>::iterator grantsIT =  selIT->selGrant.m_grantTransmissions.begin(); gra
    {
    SSPSlog << selIT->nodeId << "," << selIT->time << "," << selIT->selFrame.frameNo+1 << "," << selIT->selFrame.subfr
    << selIT->nCSRfinal << "," << selIT->nCSRpastTx << "," << selIT->nCSRinitial << "," << selIT->selGrant.m_Cresel <<
    << selIT->pdb << "," << selIT->selGrant.m_grantTransmissions.size() << "," << grantsIT->first << "," << grantsIT->
    << grantsIT->second.m_nextReservedSubframe << "," << grantsIT->second.m_rbStartPssch << "," << grantsIT->second.m_
    << grantsIT->second.m_ReEvaluationFrame << "," << grantsIT->second.m_ReEvaluationSubframe << std::endl;
    }
  }
  SSPSlog.close();
```

With this part of the code, if you are above the saving period, you store the information about the SSPS algorithm in the text file SSPSlog.txt.
Some of the most important information that are stored in the file are: nodeID, time, frame number, subframe number, number of iterations, RSRP threshold (always -128 dBm), nCSRfinal (final resources), nCSRpastTx (resources used previously for transmission), nCSRinitial (initial resources selected from the selection window Sa), reselection counter value, RRI value, ????, PDB, grant size, next reserved frame, next reserved subframe, enable reevaluation, reevaluation frame and reevaluation subframe.

```
      NrV2XUeMac::SelectedGrants.clear();
    }

//    if (m_rnti == m_debugNode)
//    std::cin.get();

    return V2XGrant;
}
```

Return the V2XGrant at the end of the function.

Another important function to understand how the MAC sublayer works is DoSubframeIndication.

```
void
NrV2XUeMac::DoSubframeIndication (uint32_t frameNo, uint32_t subframeNo)
{
   NS_LOG_FUNCTION (this << " Frame no. " << frameNo << " subframe no. " << subframeNo << " UE " << m_rnti);
   m_frameNo = frameNo;
   m_subframeNo = subframeNo;

   // TODO FIXME New for V2V

   bool aperiodicTraffic = false;
   uint32_t nodeId = 0;
//   uint32_t RNGframe, RNGsubframe;
   uint16_t TB_RBs = 0;

   RefreshHarqProcessesPacketBuffer ();
   if ((Simulator::Now () >= m_bsrLast + m_bsrPeriodicity) && (m_freshUlBsr == true))
   {
        SendReportBufferNistStatus ();
        m_bsrLast = Simulator::Now ();
        m_freshUlBsr = false;
        m_harqProcessId = (m_harqProcessId + 1) % HARQ_PERIOD; // HARQ_PERIOD = 7
   }
```

The parameters that are passed to the function are the frame number (frameNo) and the subframe number (subframeNo).
The first step is to assign the parameters passed to the function to the variables m_frameNo and m_subframeNo, that are attributes of the class.
The boolean variable aperiodicTraffic is set to false, meaning that we are dealing with periodic traffic.
The node identifier (nodeId) is initialized to zero and the transport block (measured in terms of resource blocks) is set to zero.
Then, a function named RefreshHarqProcessesPacketBuffer(), which is an internal method of the class, is called. Buffers are refreshed in this way. (HARQ not utilized).

```
//Sidelink Processes
//There is a delay between the MAC scheduling and the transmission so we assume that we are ahead
subframeNo += UL_PUSCH_TTIS_DELAY;
if (subframeNo > 10)
{
  ++frameNo;
  if (frameNo > 1024)
  {
    frameNo = 1;
    if (Simulator::Now ().GetSeconds () * 1000 > m_millisecondsFromLastAbsSFNUpdate + 1000.0)  //Useless
    //check whether the SFN cycle has not just been updated
    {
      m_absSFN ++;
      m_millisecondsFromLastAbsSFNUpdate = Simulator::Now ().GetSeconds () * 1000;
    }
  }
  subframeNo -= 10;
}
```

Since there is a delay between the MAC scheduling and the transmission, we assume that we are ahead. So, we add to the subframe number a global variable called UL_PUSCH_TTIS_DELAY. (UL → Uplink, PUSCH → Physical Uplink Shared Channel, TTI → Transmission Time Interval).
Then, a control on the adjusted subframe number is made. If the subframe number is greater than 10, it's value is increased by one. If the subframe number is greater than 1024, it's value is set back to one. Finally, the subframe number is decremented by 10.

```
//   NS_LOG_INFO (this << " Adjusted Frame no. " << frameNo << " subframe no. " << subframeNo);
   SidelinkCommResourcePool::SubframeInfo currentSF, estSF;
   estSF = SimulatorTimeToSubframe(Simulator::Now (), m_slotDuration);
   currentSF.frameNo = frameNo;
   currentSF.subframeNo = subframeNo;
```

Resource pools are created. The first one, currentSF, is used to identify the resources in the current subframe. With currentSF.frameNo = frameNo and currentSF.subframeNo = subframeNo, the current frame number and the current subframe number are set. The second one, estSF, is used to store the conversion from the simulator time to the subframe time. (estSF for debugging purposes). SimulatorTimeToSubframe useful also in other parts (nr-v2x-utils).

```
if (SubtractFrames(frameNo, m_prevListUpdate.frameNo, subframeNo, m_prevListUpdate.subframeNo) >= 1000 )
{
//  NS_LOG_DEBUG("----------------------------");
  m_prevListUpdate.frameNo = frameNo;
  m_prevListUpdate.subframeNo = subframeNo;
  UpdatePastTxInfo(frameNo, subframeNo);
  UpdateSensedCSR(frameNo, subframeNo);
 // std::cin.get();
}
```

If the difference between the current and the previous frame number and subframe number is greater than 1000, update the information. Reset delle mappe delle trasmissioni passate (se no crescono all'infinito e con tanti utenti diventa pesante).

```
// V2X Stuff
//Communication (D2D)
if ((Simulator::Now () >= m_slBsrLast + m_slBsrPeriodicity) && (m_freshSlBsr == true))
{
  SendSidelinkReportBufferStatus ();
  m_slBsrLast = Simulator::Now ();
  m_freshSlBsr = false;
  //m_harqProcessId = (m_harqProcessId + 1) % HARQ_PERIOD; //is this true?
}

std::map <uint32_t, PoolInfo>::iterator poolIt;
```

Useless

```
std::map <uint32_t, PoolInfo>::iterator poolIt;

for (poolIt = m_sidelinkTxPoolsMap.begin() ; poolIt != m_sidelinkTxPoolsMap.end() ; poolIt++) // Actually there is
{
  if ((poolIt -> second.m_pool -> IsV2XEnabled() || true) && m_slGrantMcs != 0)  // If the pool is V2X enabled
  {
    //TODO V2X stuff
    Ptr<PacketBurst> emptyPburst = CreateObject <PacketBurst> ();
    poolIt->second.m_miSlHarqProcessPacket = emptyPburst;
    //Get the BSR for this pool
    //If we have data in the queue
    //find the BSR for that pool (will also give the SidelinkLcIdentifier)
    std::map <SidelinkLcIdentifier, NistLteMacSapProvider::NistReportBufferNistStatusParameters>::iterator itBsr;
    for (itBsr = m_slBsrReceived.begin () ;  itBsr != m_slBsrReceived.end () ; itBsr++)
    {

      if (itBsr->first.dstL2Id == poolIt->first)
      {
        //This is the BSR for the pool
        if (itBsr->second.V2XTrafficType == 0X00)
        {
          NS_LOG_DEBUG (this << " Periodic traffic");
        }
        else if (itBsr->second.V2XTrafficType == 0X01)
        {
          NS_LOG_DEBUG (this << " Aperiodic traffic");
          aperiodicTraffic = true;
        }
        nodeId = itBsr->second.V2XNodeId;
      //  NS_LOG_DEBUG("Node ID = " << nodeId);

        break;
      }//end if (itBsr->first.dstL2Id == poolIt->first)
    }// end for (itBsr = m_slBsrReceived.begin () ;  itBsr != m_slBsrReceived.end () ; itBsr++)
```

If the pool poolIt is V2X enabled, then you create a packet burst object.
Check the traffic type to understand if you are dealing with periodic traffic or aperiodic traffic. If the traffic is aperiodic, then the boolean variable aperiodicTraffic (previously initialized to false) is set to true. Then, the identifier of the node is extracted.

```
if (itBsr == m_slBsrReceived.end () || (*itBsr).second.txQueueSize == 0)  // If there is no data to transmit
{
  NS_LOG_DEBUG (this << " no BSR received. Assume no data to transfer. Valid grant? " << poolIt->second.m_V2X_grant_recei
  if (poolIt->second.m_V2X_grant_received) // If the UE has a valid reservation
  {
    if (poolIt->second.m_currentV2XGrant.m_Cresel > 0 && poolIt->second.m_currentV2XGrant.m_grantTransmissions[1].m_nextR
        && poolIt->second.m_currentV2XGrant.m_grantTransmissions[1].m_nextReservedSubframe == subframeNo)
    {
      // Update grant reselection parameters anyway, even if higher layers did not request Tx
      poolIt->second.m_currentV2XGrant.m_Cresel--;
      NS_LOG_INFO("Cresel decremented to: " << poolIt->second.m_currentV2XGrant.m_Cresel);
      if (poolIt->second.m_currentV2XGrant.m_Cresel == 0 ) //TODO FIXME
      {
        poolIt->second.m_V2X_grant_received = false; //GRANT HAS EXPIRED, next time will be recomputed
      }

      for (std::map<uint16_t, V2XSchedulingInfo>::iterator grantsIT = poolIt->second.m_currentV2XGrant.m_grantTransmissio
      {
        SidelinkCommResourcePool::SubframeInfo updatedSF = UnimoreUpdateReservation (grantsIT->second.m_nextReservedFrame

        grantsIT->second.m_nextReservedFrame = updatedSF.frameNo;
        grantsIT->second.m_nextReservedSubframe = updatedSF.subframeNo;

        updatedSF = UnimoreUpdateReservation (grantsIT->second.m_ReEvaluationFrame, grantsIT->second.m_ReEvaluationSubfra

        grantsIT->second.m_ReEvaluationFrame = updatedSF.frameNo;
        grantsIT->second.m_ReEvaluationSubframe = updatedSF.subframeNo;

        NS_LOG_UNCOND("Now: SF(" << frameNo << "," << subframeNo << "). Grant index " << grantsIT->first << ": just updat
        NS_LOG_INFO("Now: SF(" << frameNo << "," << subframeNo << "). Grant index " << grantsIT->first << ": just updated
```

If you are arrived at the end of the iterator, it means that there is no data to transmit. So, you have to check if the UE has a valid reservation. (poolIt -> second.m_V2X_grant_received, because poolIt is a map and you have to take the second element. Then you have to extract the attribute m_V2X_grant_received).
If poolIt->second.m_currentV2XGrant.m_Cresel > 0 (meaning that the reselection counter has not reached zero) AND poolIt->second.m_currentV2XGrant.m_grantTransmissions[1].m_nextReservedFrame == frameNo AND poolIt->second.m_currentV2XGrant.m_grantTransmissions[1].m_nextReservedSubframe == SubframeNo,
This means that higher layers have not requested a transmission but you update anyway. So, you decrement the reselection counter by one and if the counter reaches zero, you make the grant expire. So, the next time, it will be recomputed.
Next, iterate through poolIt->second.m_currentV2XGrant.m_grantTransmissions.
Update reservations with UnimoreUpdateReservation(), shown later on. You store the result in updatedSF.
With updatedSF.frameNo → you access the frame number.
With updatedSF.subframeNo → you access the subframe number.
Assign these values to grantsIT->second.m_nextReservedFrame and to grantsIT->second.m_nextReservedSubframe.
Update for reevaluations. Call again the method UnimoreUpdateReservation(). Store again the result in updatedSF.
Assign these values to:
grantsIT->second.m_ReEvaluationFrame and
grantsIT->second.m_ReEvaluationSubframe.

```
        NS_LOG_UNCOND("Now: SF(" << frameNo << "," << subframeNo << "). Grant index " << grantsIT->first << ": just updat
        NS_LOG_INFO("Now: SF(" << frameNo << "," << subframeNo << "). Grant index " << grantsIT->first << ": just updated

        if (grantsIT->first == 1)
        {
          NS_LOG_INFO("This is initial transmission, enable re-evaluation check");
          grantsIT->second.m_EnableReEvaluation = true; // Re-evaluate next selected resource
        }
        else
        {
          SidelinkCommResourcePool::SubframeInfo previousSF, currSF;
          previousSF.frameNo = poolIt->second.m_currentV2XGrant.m_grantTransmissions[grantsIT->first-1].m_nextReservedFra
          previousSF.subframeNo = poolIt->second.m_currentV2XGrant.m_grantTransmissions[grantsIT->first-1].m_nextReserved

          currSF.frameNo = grantsIT->second.m_nextReservedFrame;
          currSF.subframeNo = grantsIT->second.m_nextReservedSubframe;

          uint32_t slotsDiff = EvaluateSlotsDifference(previousSF, currSF, m_maxPDB/m_slotDuration);
          if (slotsDiff < 32)
            grantsIT->second.m_EnableReEvaluation = false; // Re-evaluate next selected resource
          else
            grantsIT->second.m_EnableReEvaluation = true; // Re-evaluate next selected resource
          NS_LOG_INFO("This is not the initial transmission. Previous SF(" << previousSF.frameNo << "," << previousSF.sub
          << currSF.subframeNo << "), slots difference = " << slotsDiff << ". Enabled ? " << grantsIT->second.m_EnableReE
        }
      }
        NrV2XUeMac::ReservationsStats[m_rnti].UnutilizedReservations += 1;
      NrV2XUeMac::ReservationsStats[m_rnti].UnutilizedReservations += poolIt->second.m_currentV2XGrant.m_grantTransmissio
      //NS_LOG_UNCOND("sono qui");
    //std::cin.get();
    } //end if (poolIt->second.m_currentV2XGrant.m_Cresel > 0 && poolIt->second.m_currentV2XGrant.m_nextReservedFrame ==
  } //end if (poolIt->second.m_V2X_grant_received)
} //end if (itBsr == m_slBsrReceived.end () || (*itBsr).second.txQueueSize == 0)
```

```
    if (poolIt->second.m_pool->GetSchedulingType() == SidelinkCommResourcePool::UE_SELECTED)
    {
      if (!(*itBsr).second.alreadyUESelected && !(itBsr == m_slBsrReceived.end () || (*itBsr).second.txQueueSize == 0))
      {
        NS_LOG_INFO (this << " SL BSR size=" << m_slBsrReceived.size ());
        if (!poolIt->second.m_V2X_grant_received || ((*itBsr).second.V2XMessageType == 0x01 && (*itBsr).second.isNewV2X
        {
          (*itBsr).second.isNewV2X = false;
          V2XSidelinkGrant processedV2Xgrant;
          NS_LOG_DEBUG("TxQueue: " << (*itBsr).second.txQueueSize);
          SidelinkCommResourcePool::SubframeInfo SFpkt;
          SFpkt = SimulatorTimeToSubframe(Seconds(itBsr->second.V2XGenTime), m_slotDuration);
          NS_LOG_DEBUG("Packet generated at: " << itBsr->second.V2XGenTime << " = SF(" << SFpkt.frameNo << "," << SFpkt

            if (m_rnti == m_debugNode)
            std::cin.get();

          processedV2Xgrant = V2XSelectResources (frameNo, subframeNo, itBsr->second.V2XPdb, itBsr->second.V2XPrsvp, i

            NrV2XUeMac::ReservationsStats[m_rnti].CounterReselections += 1;
          NrV2XUeMac::ReservationsStats[m_rnti].CounterReselections += processedV2Xgrant.m_grantTransmissions.size();

          if ((*itBsr).second.V2XMessageType == 0x01)
          {
            processedV2Xgrant.m_Cresel = 0;
          }
          // Now assign the subframe!
          poolIt->second.m_currentV2XGrant = processedV2Xgrant;
          poolIt->second.m_V2X_grant_received = true;
          poolIt->second.m_V2X_grant_fresh = true;

      //    poolIt->second.m_currentV2XGrant.m_tbSize = m_amc->GetUlTbSizeFromMcs ((int) poolIt->second.m_currentV2XGra
          NS_LOG_UNCOND("UE MAC: just made UE selection. Now: F:" << frameNo << ", SF:" << subframeNo);
        } //end if (!poolIt->second.m_V2X_grant_received || ((*itBsr).second.V2XMessageType == 0x01 && (*itBsr).second
      } //END OF if (!(*itBsr).second.alreadyUESelected && !(itBsr == m_slBsrReceived.end () || (*itBsr).second.txQueue
    } //END OF if (poolIt->second.m_pool->GetSchedulingType() == SidelinkCommResourcePool::UE_SELECTED)
```

UE needs to transmit new data. Call V2XSelectResources to obtain a grant.

ì

```
SidelinkCommResourcePool::SubframeInfo
NrV2XUeMac::UnimoreUpdateReservation (uint32_t frameNo, uint32_t subframeNo, uint16_t RRI_slots)
{
  NS_LOG_FUNCTION(this);

  SidelinkCommResourcePool::SubframeInfo updatedSF;

  uint32_t SFtemp = subframeNo;
  SFtemp--;
  subframeNo = (SFtemp + RRI_slots) % 10 + 1;
  frameNo--;
  frameNo = (frameNo + (SFtemp + RRI_slots) / 10) % 1024 +1;

  updatedSF.frameNo = frameNo;
  updatedSF.subframeNo = subframeNo;

  return updatedSF;
}
```

Method used to update reservations.

```
  if (( (ReservationDelay > itBsr->second.V2XPdb) || (ReTxReservationDelay > itBsr->second.V2XPdb) ) && poolIt->second
  {
    NS_ASSERT_MSG(aperiodicTraffic, "Periodic traffic should not generate latency reselections");
      NrV2XUeMac::ReservationsStats[m_rnti].LatencyReselections += 1;
      if (poolIt->second.m_currentV2XGrant.m_rbLenPssch < TBLen_RBs)
    if (poolIt->second.m_currentV2XGrant.m_grantTransmissions[1].m_rbLenPssch < TBLen_RBs)
    {
        NrV2XUeMac::ReservationsStats[m_rnti].SizeReselections += 1;
      NS_LOG_DEBUG("Now: Latency and Size Reselection");
    }
    else
      NS_LOG_DEBUG("Now: Latency Reselection");
      std::cin.get();
    V2XSidelinkGrant newV2Xgrant;

    if (m_oneShot)
    {
      NS_FATAL_ERROR("One shot strategy is deprecated and not working");
   /*   if (poolIt->second.m_currentV2XGrant.m_rbLenPssch < TBLen_RBs)
        newV2Xgrant = V2XSelectResources (frameNo, subframeNo, itBsr->second.V2XPdb, itBsr->second.V2XPrsvp, itBsr->se
      else
        newV2Xgrant = V2XSelectResources (frameNo, subframeNo, itBsr->second.V2XPdb, itBsr->second.V2XPrsvp, itBsr->se
      m_tmpV2XGrant = poolIt->second.m_currentV2XGrant;
      m_oneShotGrant = true;*/
    }
    else
    {
      if (poolIt->second.m_currentV2XGrant.m_grantTransmissions[1].m_rbLenPssch < TBLen_RBs)
      {
        newV2Xgrant = V2XSelectResources (frameNo, subframeNo, itBsr->second.V2XPdb, itBsr->second.V2XPrsvp, itBsr->se
        NrV2XUeMac::ReservationsStats[m_rnti].SizeReselections += newV2Xgrant.m_grantTransmissions.size();
      }
      else
        newV2Xgrant = V2XSelectResources (frameNo, subframeNo, itBsr->second.V2XPdb, itBsr->second.V2XPrsvp, itBsr->se
      NrV2XUeMac::ReservationsStats[m_rnti].LatencyReselections += newV2Xgrant.m_grantTransmissions.size();
    }
    poolIt->second.m_V2X_grant_fresh = true;
    poolIt->second.m_currentV2XGrant = newV2Xgrant;
  }
```

Decide if you have a latency and size reselection or a latency reselection only.

```
else if (poolIt->second.m_currentV2XGrant.m_grantTransmissions[1].m_rbLenPssch < TBLen_RBs && poolIt->second.m_currentV
{
//  NS_ASSERT_MSG(aperiodicTraffic, "Periodic traffic should not generate size reselections");
  NS_LOG_DEBUG("Now: Size Reselection");
//  std::cin.get();
    NrV2XUeMac::ReservationsStats[m_rnti].SizeReselections += 1;
  V2XSidelinkGrant newV2Xgrant;
/    SidelinkCommResourcePool::SubframeInfo nextReservedSF;
/    nextReservedSF.frameNo = poolIt->second.m_currentV2XGrant.m_nextReservedFrame;
/    nextReservedSF.subframeNo = poolIt->second.m_currentV2XGrant.m_nextReservedSubframe;

  if (m_oneShot)
  {
  /*  newV2Xgrant = V2XSelectResources (frameNo, subframeNo, itBsr->second.V2XPdb, itBsr->second.V2XPrsvp, itBsr->secon

    NS_LOG_DEBUG("New reservation at SF(" << newV2Xgrant.m_nextReservedFrame << "," << newV2Xgrant.m_nextReservedSubfra
    << "). Old reservation at SF(" << nextReservedSF.frameNo << "," << nextReservedSF.subframeNo << ")");
    NS_LOG_DEBUG("New diff = " << SubtractFrames(newV2Xgrant.m_nextReservedFrame, frameNo, newV2Xgrant.m_nextReservedSu

    if ( SubtractFrames(newV2Xgrant.m_nextReservedFrame, frameNo, newV2Xgrant.m_nextReservedSubframe, subframeNo) >= Su
    {
      NS_LOG_INFO("Manually update the current grant");
      NONONONONOUnimoreUpdateReservation (poolIt); //Now deprecated, does not work any more

      nextReservedSF.frameNo = poolIt->second.m_currentV2XGrant.m_nextReservedFrame;
      nextReservedSF.subframeNo = poolIt->second.m_currentV2XGrant.m_nextReservedSubframe;

      NS_LOG_DEBUG("New reservation at SF(" << newV2Xgrant.m_nextReservedFrame << "," << newV2Xgrant.m_nextReservedSubf
      NS_LOG_DEBUG("New diff = " << SubtractFrames(newV2Xgrant.m_nextReservedFrame, frameNo, newV2Xgrant.m_nextReserved
//      std::cin.get();
    }

    m_tmpV2XGrant = poolIt->second.m_currentV2XGrant;
    m_oneShotGrant = true;*/
  }
  else
  {
    newV2Xgrant = V2XSelectResources (frameNo, subframeNo, itBsr->second.V2XPdb, itBsr->second.V2XPrsvp, itBsr->second.
    NrV2XUeMac::ReservationsStats[m_rnti].SizeReselections += newV2Xgrant.m_grantTransmissions.size();
  }

  poolIt->second.m_V2X_grant_fresh = true;
  poolIt->second.m_currentV2XGrant = newV2Xgrant;

}
```

Else if → here you have a size reselection. OneShot transmission is deprecated and not working, hence commented. Call the function V2XSelectResources to select new resources and update reservation stats.
m_V2X_grant_fresh = true and m_currentV2XGrant is set to newV2Xgrant.

```
  else
  {
    NS_LOG_DEBUG("Packet respects the current reservation");
  }
```

Else → no need for size reselections or latency reselections, hence the packet respects the current reservation.

Now choosing if you want to do re-evaluations.

```
if (m_reEvaluation && SubtractFrames(poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedFram
    poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe, subframeNo) >= GetTproc1 (m
{
  if (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_ReEvaluationFrame == frameNo && poolIt->second.
  {
    NS_LOG_DEBUG("Re-evaluation now SF(" << frameNo << "," << subframeNo << ")");
     if (m_reEvaluation && poolIt->second.m_V2X_grant_fresh)
    if (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_EnableReEvaluation)
    {
        std::cin.get();
        ReEvaluateResources(poolIt->second.m_currentV2XGrant, itBsr->second);
      ReEvaluateResources(currentSF, poolIt, itBsr->second);
    }

  }
  else if (SubtractFrames(poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_ReEvaluationFrame, frameNo,
          poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_ReEvaluationSubframe, subframeNo) >= 0 &&
  {
    NS_LOG_DEBUG("Re-evaluation now SF(" << frameNo << "," << subframeNo << ")");
     if (m_reEvaluation && poolIt->second.m_V2X_grant_fresh)
    if (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_EnableReEvaluation)
    {
        std::cin.get();
        ReEvaluateResources(poolIt->second.m_currentV2XGrant, itBsr->second);
      ReEvaluateResources(currentSF, poolIt, itBsr->second);
    }
  }
}
else
{
  if(SubtractFrames(poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedFrame, frameNo,
    poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe, subframeNo) < GetTproc1 (m_
  {
    NS_LOG_DEBUG("Packet arrived too late for re-evaluation");
  }
  else
  {
    NS_LOG_DEBUG("Re-evaluation not enabled");
  }
}
```

First, check if the packet is arrived too late for a re-evaluation or not.

The decision to execute again step 1 (and how often it is executed) is left to UE implementation. The UE can execute again step 1 at slot m-T 3 or before. T 3 is the maximum time allowed for a UE (in slots) to complete the resource selection process and is equal to T proc,1 .

That's why you do → poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe, subframeNo) >= GetTproc1 (m_numerologyIndex))

First if → one slot re-evaluation, meaning re-evaluation only of the next resource. Subsequent else if → all slots re-evaluations. Re-evaluations of the next resources / slots. In both cases, call the function ReEvaluateResources if the re-evaluation mechanism is enabled (m_EnableReEvaluation == 1). The function ReEvaluateReources takes as parameters the current subframe (currentSF), the resource pool (poolIt) and the second element of the map itBsr. First element of this map is the SidelinkLcIdentifier, second element of this map is an object of the class NistLteMacSapProvider (service access point between the RLC and the MAC), which reports the status of the receiver buffer. (????????).

```
if (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedFrame == frameNo &&
    poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe == subframeNo)
{
  NS_LOG_INFO("Received grant for SF(" << frameNo << ", " << subframeNo << ")");

  m_updateReservation = true;
  m_validReservation = true;
  NrV2XUeMac::ReservationsStats[m_rnti].Reservations += 1;
    NrV2XUeMac::ReservationsStats[m_rnti].Reservations += poolIt->second.m_currentV2XGrant.m_grantTransmissions.size();

  if (m_oneShotGrant)
    m_updateReservation = false;

  // Change also the fresh grant
*   if (poolIt->second.m_V2X_grant_fresh)
  {
   // poolIt->second.m_currentV2XGrant = poolIt->second.m_currentV2XGrant;
    m_aperiodicV2XGrant =  poolIt->second.m_currentV2XGrant;
    // Need to make sure that no other transmission occurs before the first
  }
  else
  {
    m_aperiodicV2XGrant = poolIt->second.m_currentV2XGrant;
  }*/

  NS_LOG_DEBUG("Current SF(" << frameNo << "," << subframeNo << ")");
  NS_LOG_DEBUG("Send the packet at SF(" << poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_next
  << poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe << ")");

  (*itBsr).second.alreadyUESelected = true; //FIXME please change the name of this member
  //Compute the TB size
  uint16_t TBlen_subCH, TBLen_RBs;
  poolIt->second.m_currentV2XGrant.m_tbSize = m_NRamc->GetSlSubchAndTbSizeFromMcs ((*itBsr).second.txQueueSize, poolIt->second.m_currentV2XGrant.m_
//   poolIt->second.m_currentV2XGrant.m_tbSize = m_amc->GetUlTbSizeFromMcs ((int) poolIt->second.m_currentV2XGrant.m_mcs, (int) poolIt->second.m_cur
  NS_LOG_DEBUG("Allocated " << poolIt->second.m_currentV2XGrant.m_tbSize << " B, with " <<  (*itBsr).second.txQueueSize << " B in the queue. PSSCH

  TxPacketInfo tmpInfo;
  tmpInfo.packetID = itBsr->second.V2XPacketID;
  tmpInfo.txTime = Simulator::Now().GetSeconds();
  tmpInfo.selTrigger = poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_SelectionTrigger;
```

Now, updated reservation (m_updatedReservation = true) and also valid reservation (m_validReservation = true). So, update reservation stats (increase by one).
Some debug info printed on the terminal.
Also, set the size (in TBs) of the current V2X grant. (WHY????).
Set temporary information of the packet (TxPacketInfo tmpInfo): namely, the packet ID, the transmission time and the selection trigger.
TBlen_subCH → numero di subchannel prenotati
TBlen_RBs → numero di RBs utilizzati in realtà (con queste due metriche vedi quanto sprechi)
Dal MCS si può ricavare la dimensione del singolo subchannel e del TB. Non si può prenotare meno di un subchannel. Devi prenotare sempre multipli interi di subchannel. Quindi TBlen_subCH sarà sempre un numero intero. Invece TBLen_RBs può corrispondere ad un numero non intero di subchannels ma frazionario.

```
struct TxPacketInfo
{
  uint32_t packetID;
  double txTime;
  uint16_t selTrigger;
  bool announced;
};
```

Structure of TxPacketInfo defined in the header (nr-v2x-ue-mac.h). The structure contains four attributes: the ID of the packet (packetID), the transmission time (txTime), the selection trigger (selTrigger, slot at which new resources must be

selected) and the boolean variable announced. 0 → packet not announced, 1 → packet announced.

```
f (poolIt->second.m_V2X_grant_fresh)

NS_LOG_INFO("Fresh Grant");
poolIt->second.m_v2xTx = poolIt->second.m_pool->GetV2XSlTransmissions (frameNo, subframeNo, poolIt->second.m_currentV2XGrant.m_grantTransmiss
poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbStartPssch, poolIt->second.m_currentV2X
poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbLenPscch, poolIt->second.m_currentV2XGr
poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe, 0);
```

Now, if you have a fresh grant (reservation not respected), you need to get V2X sidelink transmissions. Pass all the parameters such as the frameNo, subframeNo, the resource block at which the Pssch start (.m_rbStartPssch), the length (in resource blocks) of the Pssch (.m_rbLenPssch), the resource block at which the Pscch start (.m_rbStartPscch) and the length of the Pscch (.m_rbLenPscch).
Remember that **Pssch** is the **Physical Sidelink Shared Channel**, while the **Pscch** is the **Physical Sidelink Control Channel**.

```
if (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_EnableReEvaluation)
{
  // add check if its second transmission, second transmission is announced
  NS_LOG_DEBUG("Packet " << itBsr->second.V2XPacketID << " transmitted without being announced");
  tmpInfo.announced = false;
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_EnableReEvaluation = false;
    PKTsType << itBsr->second.V2XPacketID << ",0," << poolIt->second.m_currentV2XGrant.m_SelectionTrigger << "," << Simulator::Now().GetSeconds
}
```

First transmission with a fresh grant is not announced. Disable Re-Evaluations (already done previously).

```
else
{
  NS_LOG_DEBUG("Packet " << itBsr->second.V2XPacketID << " transmitted after being announced");
  tmpInfo.announced = true;
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_EnableReEvaluation = true;
    PKTsType << itBsr->second.V2XPacketID << ",1," << poolIt->second.m_currentV2XGrant.m_SelectionTrigger << "," << Simulator::Now().GetSeconds() << std
  NS_ASSERT_MSG(poolIt->second.m_currentV2XGrant.m_TxIndex != 1, "Newly selected packets should always be re-evaluated");
}
  std::cin.get();
```

Else, the subsequent transmissions are announced. Re-Evaluations enabled. If the transmission index is not 1, newly selected packets should always be re-evaluated.

```
        std::cin.get();
      if (poolIt->second.m_currentV2XGrant.m_TxIndex == poolIt->second.m_currentV2XGrant.m_TxNumber)
        poolIt->second.m_V2X_grant_fresh = false; //we have just used this grant for the first time
      poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_EnableReEvaluation = false
    }
```

```
else // If the packet respects the reservation
{
  NS_LOG_INFO("Reused Grant, now: SF(" << frameNo << "," << subframeNo << ")");
  poolIt->second.m_v2xTx = poolIt->second.m_pool->GetV2XSlTransmissions (frameNo, subframeNo, poolIt->second.m_currentV2XGrant.m_grantTrans
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbStartPssch, poolIt->second.m_curren
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbLenPscch, poolIt->second.m_currentV
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe, 0);
  if (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_EnableReEvaluation)
  {
    NS_LOG_DEBUG("Packet " << itBsr->second.V2XPacketID << " transmitted without being announced");
    tmpInfo.announced = false;
    poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_announced = false;
      PKTsType << itBsr->second.V2XPacketID << ",0," << poolIt->second.m_currentV2XGrant.m_SelectionTrigger << "," << Simulator::Now().GetS
  }
  else
  {
    NS_LOG_DEBUG("Packet " << itBsr->second.V2XPacketID << " transmitted after being announced");
    tmpInfo.announced = true;
    poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_announced = true;
      PKTsType << itBsr->second.V2XPacketID << ",1," << poolIt->second.m_currentV2XGrant.m_SelectionTrigger << "," << Simulator::Now().GetS
  }
    std::cin.get();
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_EnableReEvaluation = false;
}
```

The packet respects the reservation and so you do not need a fresh grant but you can reuse the same one.
Also here, you make the same check as before for the transmission without being announced and the transmission with being announced.

```
if (m_updateReservation)
{
  if (poolIt->second.m_currentV2XGrant.m_Cresel > 0)
  {
    if (poolIt->second.m_currentV2XGrant.m_TxIndex == 1)
      poolIt->second.m_currentV2XGrant.m_Cresel--;
    //NS_LOG_DEBUG("Decreasing the reselection counter. Node " << m_rnti);
//    std::cin.get();
  }

  SidelinkCommResourcePool::SubframeInfo updatedSF = UnimoreUpdateReservation (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_c
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe, poolIt->second.m_currentV2X

  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedFrame = updatedSF.frameNo;
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe = updatedSF.subframeNo;

  updatedSF = UnimoreUpdateReservation (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_ReEvaluatio
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_ReEvaluationSubframe, poolIt->second.m_currentV2X

  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_ReEvaluationFrame = updatedSF.frameNo;
  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_ReEvaluationSubframe = updatedSF.subframeNo;

  poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_EnableReEvaluation = false;

  NS_LOG_INFO("Just updated re-evaluation: SF(" << poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_
  << poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_ReEvaluationSubframe << ")");
  //Re-evaluation should not be updated since it is performed only on selected resources
}

NS_LOG_UNCOND("Just updated reservation SF(" << poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nex
<< poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReservedSubframe << ")");
```

ONE SHOT deprecated → ignore

```
for (std::list<SidelinkCommResourcePool::V2XSidelinkTransmissionInfo>::iterator txIt = poolIt->second.m_v2xTx.begin (); txIt != poolIt->se
{
  txIt->subframe.frameNo++;
  txIt->subframe.subframeNo++;
}
```

Iterating through the list of V2X Pssch and Pscch transmissions to increase the frame number and the subframe number (meaning, advance in time).

```
    NS_LOG_INFO("poolIt->second.m_currentV2XGrant.m_Cresel: " << poolIt->second.m_currentV2XGrant.m_Cresel);
    if ((poolIt->second.m_currentV2XGrant.m_Cresel == 0) && (poolIt->second.m_currentV2XGrant.m_TxIndex == poolIt->second.m_currentV2XGrant.m_gra
    {
      if (m_evalKeepProb->GetValue() > (1-m_keepProbability))
      {
        NS_LOG_UNCOND("Keep the same resources");
        std::cin.get();
        poolIt->second.m_currentV2XGrant.m_Cresel = GetCresel(poolIt->second.m_currentV2XGrant.m_RRI);
      }
      else
      {
        // Next time higher layers request TB transmission, a new reservation or Scheduling Request will be needed
        poolIt->second.m_V2X_grant_received = false; //grant has expired
        NS_LOG_INFO("Grant has expired");
      }
    }
/     std::cin.get();  //Pause the program for debugging purposes
  } // else if (poolIt->second.m_currentV2XGrant.m_nextReservedFrame == frameNo && poolIt->second.m_currentV2XGrant.m_nextReservedSubframe == sub
  else
  {
    NS_LOG_DEBUG("Not yet ready to transmit: SF(" << frameNo << "," << subframeNo << ")");
  }
} // if (poolIt->second.m_V2X_grant_received && !(*itBsr).second.alreadyUESelected && !(itBsr == m_slBsrReceived.end () || (*itBsr).second.txQueu
```

First part, if the reselection counter value is zero or the transmission index of the current grant (.m_currentV2XGrant.m_TxIndex) is equal to the grant transmissions size → keep the same resource with probability 1-P, as specified in the standard. If you do not keep the same resources, the grant expires.

Else “Not yet ready to transmit” meaning that this is not you subframe / frame to transmit, so you have to wait !!!!

```
std::list<SidelinkCommResourcePool::V2XSidelinkTransmissionInfo>::iterator allocIter;
//check if we need to transmit PSCCH and PSSCH
allocIter = poolIt->second.m_v2xTx.begin();
```

Initialize to iterate through the list of v2x Pssch and Pscch transmissions.

```
std::list<SidelinkCommResourcePool::V2XSidelinkTransmissionInfo>::iterator allocIter;
//check if we need to transmit PSCCH and PSSCH
allocIter = poolIt->second.m_v2xTx.begin();

if (allocIter != poolIt->second.m_v2xTx.end() && allocIter->subframe.frameNo == frameNo && allocIter->subframe.subframeNo  == subframeNo)
{
  NS_LOG_UNCOND("Now: " << Simulator::Now().GetSeconds()*1000 << " ms: Ok, now I should transmit data, Frame no. " << frameNo << ", Subframe no
  NistV2XSciListElement_s sci1;
  sci1.m_rnti = m_rnti;
  sci1.m_genTime = itBsr->second.V2XGenTime;
  sci1.m_packetID = itBsr->second.V2XPacketID;
  sci1.m_announcedTB = poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_announced;
  // The m_validReservation flag triggers the packet drop down at PHY layer. its content is conveyed at PHY layer by the SCI hopping field
  // The hopping field is unused. TODO create a new dedicated SCI field.
  if (m_validReservation)
  {
    sci1.m_hopping = 0x01;
    NrV2XUeMac::ReservationsStats[m_rnti].TotalTransmissions += 1;
  }
  else
  {
    sci1.m_hopping = 0x00;
  }
```

If you do not arrive at the end of your resources and the frame number and the subframe number corresponds, transmit the packet. SCI hopping field to deliver the information at the physical layer. Choose the value of this field based on the m_validReservation flag.
In this case, focusing on the SCI (Sidelink Control Information).

```
sci1.m_rbStartPssch = poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbStartPssch;
sci1.m_rbLenPssch = poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbLenPssch;
sci1.m_rbLenPssch_TB = TB_RBs;
NS_ASSERT_MSG(poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbLenPssch >= TB_RBs, "Reservat
sci1.m_rbStartPscch = poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbStartPscch;
sci1.m_rbLenPscch = poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbLenPscch;
sci1.m_reservedSubframe.frameNo = poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextReserve
sci1.m_reservedSubframe.subframeNo = poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_nextRese

NrV2XUeMac::ReservationsStats[m_rnti].UnutilizedSubchannelsRatio.push_back((double) (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt

NS_LOG_INFO("Used RBs = " << TB_RBs << ", reserved RBs = " << poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGr
", unused RBs = " << (poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_rbLenPssch - TB_RBs));

sci1.m_selectionTrigger = poolIt->second.m_currentV2XGrant.m_grantTransmissions[poolIt->second.m_currentV2XGrant.m_TxIndex].m_SelectionTrigger;
 //ReTransmissions stuff
sci1.m_TxIndex = poolIt->second.m_currentV2XGrant.m_TxIndex;
sci1.m_TxNumber = poolIt->second.m_currentV2XGrant.m_TxNumber;
sci1.m_announceNextTxion = false;
```

Set all the parameters of the SCI field.

```
// Print the MAC debugger counters (for ETSI traffic)
if (Simulator::Now ().GetSeconds() - NrV2XUeMac::prevPrintTime_reservations > m_savingPeriod)
{
  NrV2XUeMac::prevPrintTime_reservations = Simulator::Now ().GetSeconds();
  std::ofstream ResLOG;
  ResLOG.open (m_outputPath + "ReservationsLog.txt", std::ios_base::app);
  for (std::map<uint32_t, ReservationsInfo>::iterator ResIT = NrV2XUeMac::ReservationsStats.begin(); ResIT != NrV2XUeMac::ReservationsStats.end(); Re
  {
    double AvgUSR = 0;
    for (std::vector<double>::iterator USit = ResIT->second.UnutilizedSubchannelsRatio.begin(); USit != ResIT->second.UnutilizedSubchannelsRatio.end(
    {
      AvgUSR += (*USit);
    }
    if (ResIT->second.UnutilizedSubchannelsRatio.size() > 0)
      AvgUSR /= ResIT->second.UnutilizedSubchannelsRatio.size();
    else
      AvgUSR = -1;

    ResLOG << ResIT->first << "," << Simulator::Now ().GetSeconds() << ","  << AvgUSR << "," << ResIT->second.UnutilizedReservations << "," << ResIT
    << "," << ResIT->second.LatencyReselections << "," << ResIT->second.SizeReselections << "," << ResIT->second.CounterReselections << "," << ResIT

    ResIT->second.UnutilizedSubchannelsRatio.clear();
    ResIT->second.UnutilizedReservations = 0;
    ResIT->second.Reservations = 0;
    ResIT->second.LatencyReselections = 0;
    ResIT->second.SizeReselections = 0;
    ResIT->second.CounterReselections = 0;
    ResIT->second.TotalTransmissions = 0;
  }
  ResLOG.close ();
  //std::cin.get();
}
```

Filling the ReservationsLog.txt file. So, printing the MAC debugger counters.

```
if (!m_updateReservation)
{
  sci1.m_reservation = 0; // Notify other neighbors that these resource won't be reserved
}
NS_LOG_DEBUG("SCI >> Node ID: " << nodeId << " transmitting packet " << sci1.m_packetID  << " at SF(" << sci1.m_receiv
<< sci1.m_rbLenPssch_TB << " rbStart PSCCH: " << sci1.m_rbStartPscch << " rbLen PSCCH: " << sci1.m_rbLenPscch << " ann
<< sci1.m_reservedSubframe.subframeNo << ") Reservation: " << sci1.m_reservation << " ms. Transmission " << sci1.m_TxI
```

By changing the sci1.m_reservation to 0, you notify the neighbors that the current resource will not be reserved.

```
if (sci1.m_announceNextTxion)
{
  NS_LOG_DEBUG("SCI >> Node ID: " << nodeId << " also announcing reservation at SF(" << sci1.m_secondSubframe.frameNo << "," <<
  sci1.m_secondRbStartPssch << " rbLen PSSCH (reserved): " << sci1.m_secondRbLenPssch << " rbStart PSCCH: " << sci1.m_secondRbS
  NS_LOG_DEBUG("SCI >> Node ID: " << nodeId << " also announcing reservation at SF(" << sci1.m_secondReservedSubframe.frameNo <
  sci1.m_secondRbStartPssch << " rbLen PSSCH (reserved): " << sci1.m_secondRbLenPssch << " rbStart PSCCH: " << sci1.m_secondRbS

}
```

Announcing the next transmission.

```
msg->SetSci (sci1);
m_uePhySapProvider->SendNistLteControlMessage (msg);
// Transmit data
if (true) //FIXME check if this is the first transmission
{
  NS_LOG_INFO (this << " New PSSCH transmission");
  Ptr<PacketBurst> emptyPb = CreateObject <PacketBurst> ();
  poolIt->second.m_miSlHarqProcessPacket = emptyPb;
  //get the BSR for this pool
  //if we have data in the queue
  //find the BSR for that pool (will also give the SidleinkLcIdentifier)
  std::map <SidelinkLcIdentifier, NistLteMacSapProvider::NistReportBufferNistStatusParameters>::iterator itBsr;
  //  NS_ASSERT(m_slBsrReceived.size () < 2);
  for (itBsr = m_slBsrReceived.begin () ;  itBsr != m_slBsrReceived.end () ; itBsr++)
  {
    if (itBsr->first.dstL2Id == poolIt->first)
    {
      //this is the BSR for the pool
      std::map <SidelinkLcIdentifier, NistLcInfo>::iterator it = m_slLcInfoMap.find (itBsr->first);
      //for sidelink we should never have retxQueueSize since it is unacknowledged mode
      //we still keep the process similar to uplink to be more generic (and maybe handle
      //future modifications)
```

IF (TRUE) è per una cosa esistente prima e poi sistemata

```
if (m_oneShotGrant)
{
  NS_FATAL_ERROR("One shot temporarily disabled!");

  m_oneShotGrant = false;
  poolIt->second.m_currentV2XGrant = m_tmpV2XGrant;
  if (poolIt->second.m_currentV2XGrant.m_Cresel ==
  {
    if (m_evalKeepProb->GetValue() > (1-m_keepProba
    {
      NS_LOG_UNCOND("Keep the same resources");
      std::cin.get();
      poolIt->second.m_currentV2XGrant.m_Cresel = G
    }
    else
    {
      // Next time higher layers request TB transmi
      poolIt->second.m_V2X_grant_received = false;
      NS_LOG_INFO("Grant has expired");
    }
  }
  NS_LOG_DEBUG("Next reservation: SF(" << poolIt->s

}
  std::cin.get();
```

One shot is not enabled → so commented

```
if (((*itBsr).second.statusPduSize > 0) && (bytesForThisLc > (*itBsr).second.statusPduSize))
{
  NS_FATAL_ERROR("This should never happen: if (((*itBsr).second.statusPduSize > 0) && (bytesForThisLc > (*itBsr).second.statusPduSize))
  (*it).second.macSapUser->NotifyTxOpportunity ((*itBsr).second.statusPduSize, 0, 0);
  if ( (*itBsr).second.alreadyUESelected)
  {
    (*itBsr).second.alreadyUESelected = false;
  }
  bytesForThisLc -= (*itBsr).second.statusPduSize; //decrement size available for data
  NS_LOG_DEBUG (this << " serve STATUS " << (*itBsr).second.statusPduSize);
  (*itBsr).second.statusPduSize = 0;
}
else
{
  if ((*itBsr).second.statusPduSize > bytesForThisLc)
  {
    NS_FATAL_ERROR ("Insufficient Tx Opportunity for sending a status message");
  }
}
```

USELESS

```
if ((bytesForThisLc > 7) && (((*itBsr).second.retxQueueSize > 0) || ((*itBsr).second.txQueueSize > 0)))
{
  if ((*itBsr).second.retxQueueSize > 0)
  {
    NS_LOG_DEBUG (this << " serve retx DATA, bytes " << bytesForThisLc);
    NS_FATAL_ERROR("This should never happen: if ((*itBsr).second.retxQueueSize > 0)");
    (*it).second.macSapUser->NotifyTxOpportunity(bytesForThisLc, 0, 0);
    if ((*itBsr).second.retxQueueSize >= bytesForThisLc)
    {
      (*itBsr).second.retxQueueSize -= bytesForThisLc;
    }
    else
    {
      (*itBsr).second.retxQueueSize = 0;
    }
  }
  else if ((*itBsr).second.txQueueSize > 0)
  {
    // minimum RLC overhead due to header
    uint32_t rlcOverhead = 2;
    NS_LOG_DEBUG (this << " serve tx DATA, bytes " << bytesForThisLc << ", RLC overhead " << rlcOverhead);
    if (poolIt->second.m_currentV2XGrant.m_TxIndex < poolIt->second.m_currentV2XGrant.m_grantTransmissions.size())
    {
      NS_LOG_INFO("Do not clean the RLC buffer");
      (*it).second.macSapUser->NotifyTxOpportunity (bytesForThisLc, 0, 1);
    }
    else
    {
      NS_LOG_INFO("Clean the RLC buffer");
      (*it).second.macSapUser->NotifyTxOpportunity (bytesForThisLc, 0, 0);
    }
    // added for V2X
    if ((*itBsr).second.alreadyUESelected)
    {
      NS_LOG_LOGIC("already UE SELECTED");
      (*itBsr).second.alreadyUESelected = false;
    }
```

USELESS except “else if ((*itBsr).second.txQueueSize > 0) → perché questa è la coda che si riempie con dati (bytes) provenienti dal livello applicazione. NotifyTxOpportunity() → per notificare al livello fisico che ci sono bytes da trasmettere.

```
            if ((*itBsr).second.txQueueSize >= bytesForThisLc - rlcOverhead)
            {
              NS_FATAL_ERROR("NrV2XUeMac: (*itBsr).second.txQueueSize >= bytesForThisLc - rlcOverhead should never happen");
              (*itBsr).second.txQueueSize -= bytesForThisLc - rlcOverhead;
            }
            else
            {
              if (poolIt->second.m_currentV2XGrant.m_TxIndex < poolIt->second.m_currentV2XGrant.m_grantTransmissions.size())
              {
                poolIt->second.m_currentV2XGrant.m_TxIndex += 1;
                NS_LOG_INFO("Not clearing the txQueue, there is a blind re-transmission " << (*itBsr).second.txQueueSize);
              }
              else
              {
                poolIt->second.m_currentV2XGrant.m_TxIndex = 1;
                (*itBsr).second.txQueueSize = 0;
                NS_LOG_INFO("Clearing the txQueue " << (*itBsr).second.txQueueSize);
              }
            }
          }
        } //end if ((bytesForThisLc > 7..
        else
        {
          if ( ((*itBsr).second.retxQueueSize > 0) || ((*itBsr).second.txQueueSize > 0))
          {
            if (poolIt->second.m_pool->GetSchedulingType() == SidelinkCommResourcePool::SCHEDULED)
            {
              // resend BSR info for updating eNB peer MAC
              m_freshSlBsr = true;
              NS_FATAL_ERROR("This should never happen");
            }
          }
        }
        NS_LOG_LOGIC (this << " RNTI " << m_rnti << " Sidelink Tx " << bytesForThisLc << " new queues " << (uint32_t)(*it).first.lcId << " sta
      } // end if ( ((*itBsr).second.statusPduSize > 0) || ((*itBsr).second.retxQueueSize >0) || ((*itBsr).second.txQueueSize > 0))
      break;
    } // end if (itBsr->first.dstL2Id == poolIt->first)
  } // end for (itBsr = m_slBsrReceived.begin () ;  itBsr != m_slBsrReceived.end () ; itBsr++)
    if (m_rnti == 2)
      std::cin.get();
 } //end if (true)
 else
 {
   NS_FATAL_ERROR("This should never happen");
 }
```

```
        NS_FATAL_ERROR("This should never happen");
      }
      poolIt->second.m_v2xTx.erase (allocIter); //clear the transmission
    }// end if (allocIter != poolIt->second.m_v2xTx.end() && (*allocIter).subframe.frameNo == frameNo && (*allocIter).subframe.subf

  } // end if ( IsV2XEnabled() )
  else
  {
    NS_ASSERT_MSG(false,"Non-V2X enabled pool!");
  }
}
```

After the constructor, we have the virtual constructor:

```
NrV2XUeMac::~NrV2XUeMac ()
{
        NS_LOG_FUNCTION (this);
}
```

After the virtual constructor, we have the destructor:

```
void
NrV2XUeMac::DoDispose ()
{
        NS_LOG_FUNCTION (this);
        m_miUlHarqProcessesPacket.clear ();
        delete m_macSapProvider;
        delete m_cmacSapProvider;
        delete m_uePhySapUser;
        Object::DoDispose ();
}
```

In the header file all the attributes and methods used are declared.

UE PHY

The reference files are “nr-v2x-ue-phy.cc” and “nr-v2x-ue-phy.h”.

SPECTRUM (channel)

.
.
.
.

Basic notions about ns3

NOMENCLATURE
Class::method()

PRINT SOMETHING ON THE TERMINAL
NS_LOG_INFO(“Print an output text”)
To see all the levels of log messages (7 in total), check the following tutorial [9].

INCLUDE HEADER FILES
#include <file-name.h>, includes an header file that is in the same directory as the one of the script.
#include <ns3/file-name.h>, with the prefix ns3/ you are able to include an header file that can also be in a different directory with respect to the one of the script.

HEADER FILES
The general structure for an header file is:

```
#ifndef NAME_HEADERFILE_H
#define NAME_HEADERFILE_H

#include … (include all the files and libraries that you need)

namespace ns3{ //Start namespace ns3

class classname:
{
  friend class …    (here, you define friend classes, if they are needed)
  friend class …
  friend class …
  public:        (here, you define all the public functions and all the public variables,
                  since they are public → they can be seen also by the main
  private:       (here, you define all the private functions and all the private variables,
                  since they are private → they can be seen only by the class  and by
                  friend classes but not by the main
};

} //End namespace ns3
#endif /* NR_V2X_TAG_H */
```

STOP THE SIMULATION FOR DEBUGGING

`std::cin.get()`, use this command if you want to stop the simulation at any point. The simulation is stopped until the enter key is pressed.

CREATE TEXT FILES FOR DEBUGGING OR FOR STORING DATA

To manage files, you have to include the proper library:

```
#include <fstream>
```

`std::string out = 'filepath/';`	Declaration of a string that represents the filepath
`system(("rm -r " + out).c_str());`	Remove the folder specified by the output path
`system(("mkdir " + out).c_str());`	Create the folder specified by the output path

By removing and creating the folder, you obtain a new folder for every simulation. The new folder is overwritten to the old one.
Now, we need to add the textfile in the folder.

`std::ofstream filename;`	Declaration of the file
`filename = open(out + "filename.txt");`	Opening of the file
`filename << "Hello World!" << std::endl;`	Writing inside the file something (and starting a new line)

filename.close(); Closing the file

Maybe you want to append something later on and so you can use the following commands:

filename.open(out + “filename.txt”, std::ios_base::app);
filename << “Append something inside the file”;
filename.close();

RUNNING A SCRIPT FROM THE TERMINAL
./waf --run filename, use this terminal command to run a script. Note that you have to type in filename and not filename.cc. The bar that you see before “run” is a double bar and not a single bar.

In the specific case of our script:
./waf --run 'scratch/highwayprova --Vehicles=50 --noIBE --seed=1/2 --runNo=1/2 --Periodic/Aperiodic --ChannelBW=20 --simTime=18 --SubChannel=12 --Numerology=1 --ReEvaluation --Dynamic'

You can change the number of vehicles.
You can change the seed and the runNo (four possible combinations: 1 1, 1 2, 2 1, 2 2).
You can select if you want periodic or aperiodic traffic.
You can select if you want the Dynamic scheme or not.

To debug the physical layer, you need to produce the txt file phyDebugAll.txt (nr-v2x-ue-phy.cc)

To debug the Medium Access Control (MAC) sublayer, you need to produce the txt file

CREATING STATIC VARIABLES
How to create a static variable: important because a static variable can be seen by all the objects of a class and by all the parent classes with that class. Put in the public or private section based on the needs.

1) In the header file of the class, declare **static <variable type> variableName**;
2) In the .cc file of the class, even before the constructor, initialize the static variable with the following line: **ClassName::variableName = value to initialize**;
3) Now, you can use and modify the variable in all the methods of the class and all the objects will see the changes. Remember just to use ClassName::variableName;

An example can be found in the nr-v2x-udp-client.h and nr-v2x-udp-client.cc with the static variable packetID. In this case, there is the need to create a static variable because we want a unique ID for each packet and this ID must be sequential (each node that implements a NrV2XUdpClient object must keep track of the current value of packetID).

REFERENCES

[1] https://www.nsnam.org/doxygen/classns3_1_1_node_container.html

[2] https://www.nsnam.org/doxygen/classns3_1_1_mobility_helper.html

[3] https://www.nsnam.org/doxygen/classns3_1_1_lte_helper.html#details

[4] https://www.nsnam.org/doxygen/classns3_1_1_udp_client.html#a0a9f35e331cd7ff8301ef536f8008c69

[5] https://www.nsnam.org/doxygen/classns3_1_1_udp_client.html#a414e67a17b0d38c0567a6ffe9a3cb1a3

[6] https://www.nsnam.org/doxygen/classns3_1_1_udp_client.html#a18c3a451f2c18d2edc6a634a913f1474

[7] https://www.nsnam.org/doxygen/classns3_1_1_udp_client.html#affb385b7a19c3dc83359787d55226712

[8] https://www.nsnam.org/doxygen/classns3_1_1_tag_buffer.html

[9] https://www.nsnam.org/docs/release/3.7/tutorial/tutorial_21.html

[10] https://www.nsnam.org/docs/models/html/lte-design.html