EURECOM
Department of Mobile Communications
Campus Sophia Tech
Les Templiers
450 route des Chappes
B.P. 193
06410 Biot
FRANCE

Research Report RR-12-270

Implementation and Validation of an IPv6 Stack on ns-3 for iTETRIS

September 20th, 2012
Last update November 30th, 2012

Panagiotis Matzakos, and Jérôme Härri

Tel : (+33) 4 93 00 81 00
Fax : (+33) 4 93 00 82 00
Email : {Panagiotis.Matzakos,Jerome.Haerri}@eurecom.fr

[1]EURECOM's research is partially supported by its industrial members: BMW Group, Cisco, Monaco Telecom, Orange, SAP, SFR, STEricsson, Swisscom, Symantec.

Acknowledgements

This work has been conducted with the support of France Télécom for the "Contrat de Préstation de Recherche Externalisé" (CRE) N 00 500 123 10 C0406. The authors would like to acknowledge the support of France Télécom for this work.

Panagiotis Matzakos, and Jérôme Härri

Abstract

The current document is a detailed specification of the modifications which needed to be implemented for the integration of a complete IPv6 stack in the ns-3 version (i.e. ns-3.7) of iTETRIS simulation platform for Vehicular Communications. The IPv6 equipped technologies are UMTS and 802.11p.

Index Terms

Vehicular Communications, Heterogeneous Networks, IPv6 stack, iTetris, ns3, Wave, 802.11p.

Contents

List of Figures

1 Introduction

1.1 Objectives

In the context of Eurecom's collaboration with Orange Labs for designing, implementing and simulating Cooperative Heterogenous Communications for Inteligent Transportation Systems (ITS), as well as enhancing the iTETRIS platform with more capabilities regarding the simulation of suitable mobility scenarios, we decided that one of the first basic reqirements is the implementation and integration of a complete IPv6 stack within the ns-3 supported version of iTETRIS.

Having a fully functional IPv6 stack in our disposal will give us more options for testing large scale heterogeneous communications' scenarios and will act as the intermediate step for integrating mobility management such as the implementation of Mobile IPv6 and IPv6 over geonetworking.

1.2 Outline

In the first section we will give a brief description of iTetris architecture and its components-simulators (i.e. iCS, NS-3, SUMO) focusing on the interface between iCS (iTETRIS core) and ns-3. Then we will start presenting analytically the modifications which were made for IPv6 integration. Finally we will present some test scenarios to verify the right operation of the IPv6 stack within the NS-3 extended version for iTETRIS.

As you will notice, this integration is split in three parts. The first part is about the Facilities-Application Layer of ns-3, the second part includes the necessary enhancements in the main communication stack, starting from the application layer and going down to the different network device types, and the third part includes the necessary modifications regarding the scenarios configuration and the device installers.

An important helper for this integration was the implementation of IPv6 in the current version of ns-3 (i.e. ns-3.14) especially regarding the part of the main communication stack. Thus, our effort for this part was to adjust this implementation in our iTETRIS version of ns-3.7.

2 iTETRIS Architecture

iTetris is a platform for vehicular communication simulations which permits to define realistic road and network traffic scenarios and simulate them through the integrated Network Simulator, NS-3, and the road traffic simulator, SUMO.

It aims in evaluating, through accurate simulations of Intelligent Transportation Systems, the performances of traffic management strategies and thus improve the traffic efficiency. iTetris Control System, iCS, is the heart of iTetris which acts as an interface between the two simulators and the Application that we define.

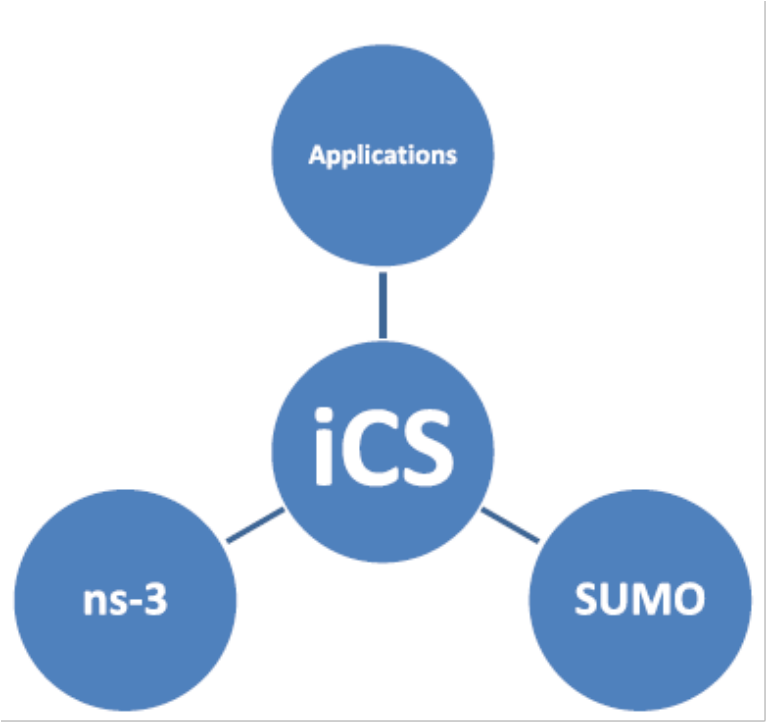

Figure 1: iTETRIS Modules

2.1 iCS

We can notice that iCS is the heart of the system since every interaction has the iCS as source or destination. iCS requests the application for new services (the subscriptions) and applies them to the simulation blocks of NS-3 and SUMO within the respective step of the operation cycle as you can see in the figure. The interactions between the application and NS-3/SUMO are always done through the iTETRIS Control System, never direct.

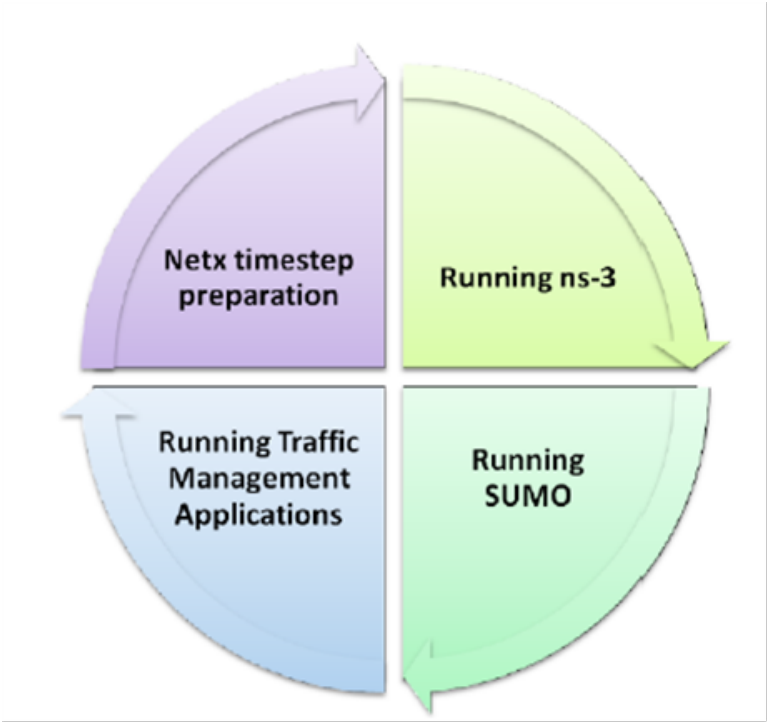

Figure 2: iTETRIS Cycle

2.2 NS-3

NS-3 is a network traffic simulator that recreates the vehicular communications generated by iTetris. Thus, it can simulate for example the transmission of an emergency message from a Vehicle to another Vehicle (V2V), or of road information sent to the Vehicle from an Infrastructure Node (e.g. RSU, Base Station or TMC). The asset of NS-3 is the range of radio access technologies it provides. It permits iTetris to simulate vehicular to vehicular or vehicular to infrastructure communications through different technologies such as WLAN 802.11p, WiMax or UMTS.

2.3 Applications

Running on top of iCS, the applications can specify particular communication and traffic scenarios. The communication specifications are passed to the iCS which can contact the traffic simulator using the iCS-NS3 Control Interface (iNCI socket).

In figure 3 we can see the INCI interface as well as the Facilities-Applications Layer of NS-3, which implement the interoperability with the iTETRIS applications to define V2X messages transmission parameters for the iTETRIS communication stack. Part of the ns-3 Facilities is the Management Layer which integrates

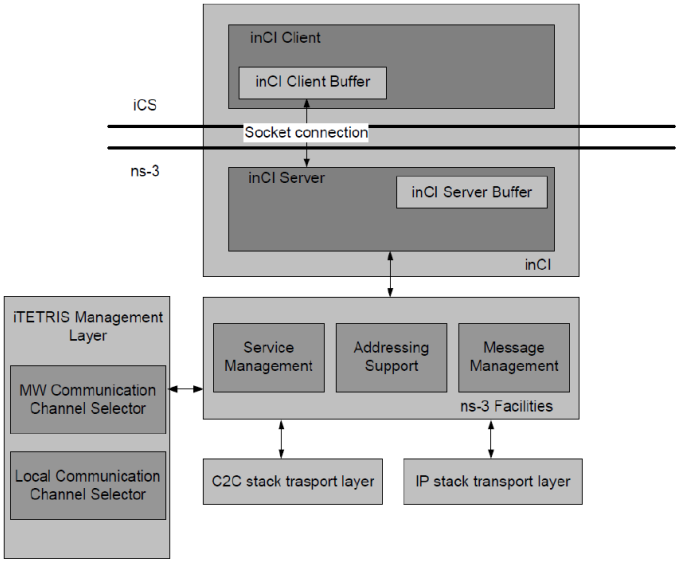

Figure 3: iNCI Socket Interface and Facilities

the technology selection part (e.g. Network Device choice, C2C or TCP-UDP/IP stack selection), based on particular Communication profiles, and this is where we will need to work for implementing Heterogeneous Communications for specific use cases.

3 IPv6 Integration

3.1 Facilities/Application Layer

3.1.1 Facilities (IdBasedTxon)

Let's now start following the path of a transmission command arriving from iCS to NS-3 through the iNCI socket interface. In the following UML class diagram (for the case of an IdBasedTxon) we highlight with red the new functions which have to be added and with blue the functions which need to be modified for IPv6 support.

For testing reasons we have introduced an additional **STACK** parameter in ns3Server::StartTransmission(STACK). This parameter is carried within the sequence of function calls until we reach the point where a choice is made between IPv4 and IPv6 stack. This parameter is currently defined from the iTETRIS application and it can be either IPv4 or IPv6. However in the future this choice can be done through the selection of an appropriate communication profile in the application.

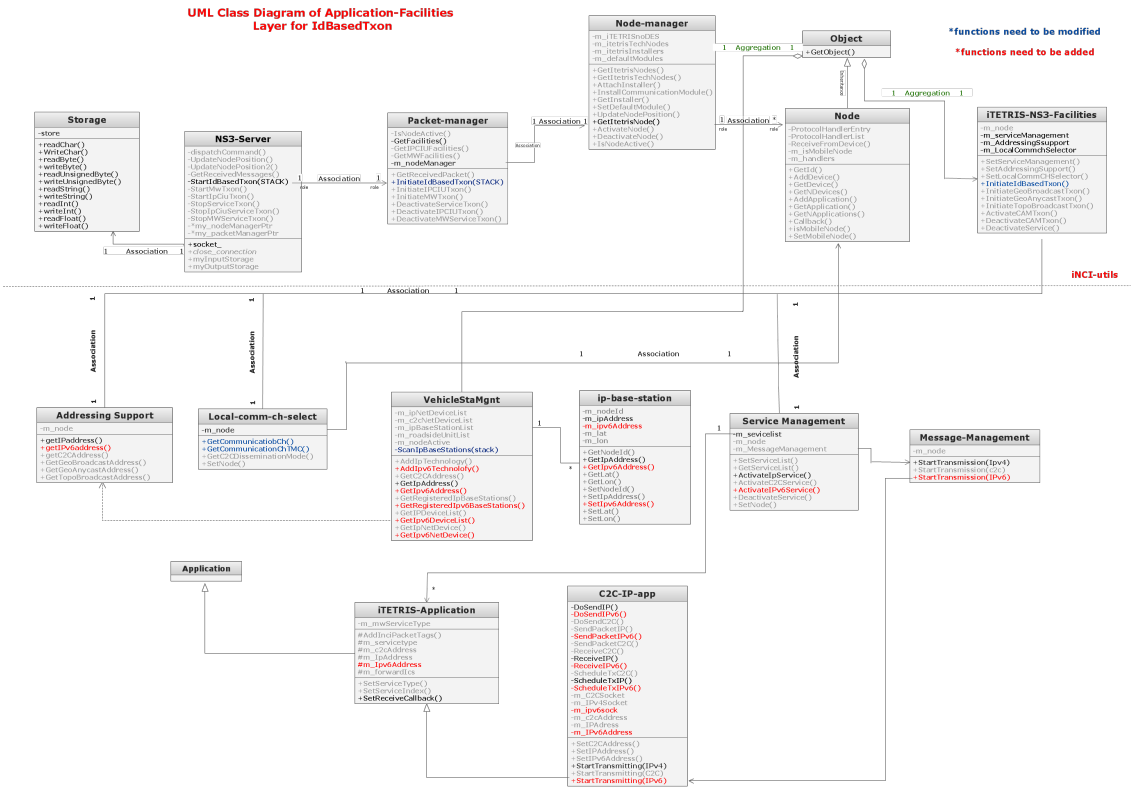

Figure 4: Facilities Layer Class Diagram

- **PacketManager::InitiateIdBasedTxon(nodeID, STACK):** The Packet Manager is part of the iNCI utilities. Within this function we first get the facilities object of the node with the particular nodeID and then call the equivalent function of iTETRISNS3Facilities.

- **iTETRISNS3Facilities::InitiateIdBasedTxon(nodeID, STACK):** At this point, the Local Communication Channel Selector is first called in order to decide the parameters of the Transmission, based on the inputs (i.e. Communication profile, STACK and the list of available technologies).

- **LocalCommChSelector::GetCommunicationCh()/ GetCommunicationChannelTMC():** Based on the communication profile, the STACK and the list of technologies, the appropriate stack to destination parameters are chosen and returned by that fumction to iTETRIS-NS3-Facilities. In the case of GetCommunicationChannelTMC() function the best base station per technology is chosen with the help of Vehicle Station Management class (installed in the node). **At this point we have introduced some parts which make use of the introduced STACK parameter to make the selection between IPv4 and IPv6 stack.** Thus, these two functions return some trans-

mission parameters within the **stacktodest** variable such as the stack type (i.e. IPv4, IPv6 or C2C) which permit us to activate the appropriate service from the facilities (e.g. call ServiceManagement:: ActivateIpv6Service() for IPv6 type of stack).

- **ServiceManagement:: ActivateIpv6Service():** This function retrieves the application with a specific id from its service list, then it assigns it some extra parameters and finally it calls MessageManagement::StartTransmission (application, destinationaddress), which now also supports IPv6 type of destination address. Message Management then calls C2C-IP-app::StartTransmitting() function.
- **AddressingSupport::GetIpAddress() / GetIpv6Address() / GetC2CAddress():** The call to that type of functions permits us to retrieve the specific address of the destination, based on its id and of course the choice of the communication stack.

3.1.2 Facilities (MWBasedTxon)

For the case of MiddleWare based transmissions (i.e. the transmissions are triggered by the TMC), things are pretty much the same until the point where we retrieve the **MWFacilities** Object from the Packet Manager (5).

- **MWFacilities::InitiateMWGeoBasedTxon(STACK, TechnologyList, CircularGeoAddress):** Within this function, we retrieve the dissemination profile, by calling the communication channel selector, as we will show next. Then, based on the stack-specific address of the destination and the disseminator node (technology specific infrastructure node) we activate the appropriate Service by calling the Service Management facility. At this point we added an IPv6 part, where we call ServiceManagement::ActivateIPv6Service() to trigger the IPv6 transmission.
- **MWCOMMchSelector::GetDisseminationProfile(STACK, TechnologyList, CircularGeoAddress):** Inside this function, the TMC, given its lists of infrastructure nodes per technology (Base Stations for Umts, RSUs for Wave), selectcs the best serving one, based on the minimum distance from the destination area (we can retrieve the coordinates from the CircularGeoAddress parameter). *We note here that this is a simplistic decision that is currently being made and which is expected to be enriched with multiple other criteria, when we are going to investigate heterogeneous communications in large-scale scenarios.*

3.1.3 Application

After having specified the transmission parameters within the Facilities Layer we can now pass to the Application level which will generate the packet and pass

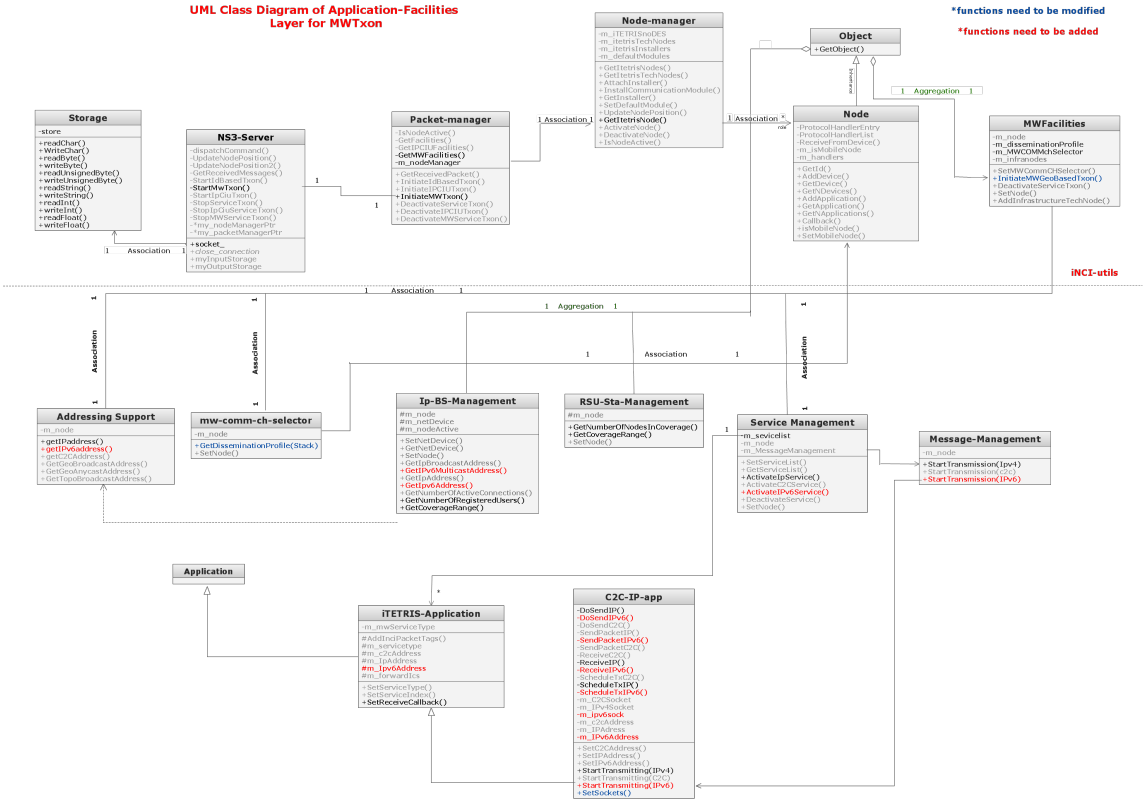

Figure 5: Facilities Layer class diagram (MWBasedTxon)

it to the chosen Communication stack. Let's have a look at the new elements for C2C-IP-app here.

- **m_IPv6socket, m_IPv6port**: We introduced a new socket for the IPv6 stack here. For this reason we also had to modify the **SetSockets** function to bind the new socket to a specific address and port (i.e. m_IPv6port). We also set the Receive callback of the m_IPv6socket to the new function **ReceiveIPv6()**, which is the application's receiving function for IPv6 packets.

- **StartTransmitting(IPv6address), DoSendIp(IPv6address):** StartTransmitting function sets the new m_IPv6Address equal to the IPv6 destination address specified by its input parameter and it also creates an Inet6SocketAddress from the set of the destination address and the IPv6 port (m_portIPv6). Then, with those two parameters, it calls the DoSendIp() function, which creates the new packet, connects the socket to the Inet6Socket destination address and then forwards the packet to the Socket Layer (TCP or UDP based on our configuration). ScheduleTxIP(IPv6) is finally called for the scheduling of the IPv6 based Txon.

- **ScheduleTxIP(IPv6), SendPacketIPv6(), m_sendEventIPv6:** Within the first function, the scheduler is called with a callback to SendPacketIPv6() and its return EventId type value is assigned to m_sendEventIPv6 to specify the state of the new IPv6 event. Finally SendPacketIPv6() function defines the transmission instances based on the value of message regenaration parameter.

- **ReceiveIpv6:** As stated above, this is the function which is called from the socket layer so that the application layer of the receiver can get the IPv6 packet by listening to the socket where it is connected.

- **StopTransmitting:** We had to modify this function to include the cases of the termination of IPv6 events (m_sendEventIPv6) and the closing of m_Ipv6Socket.

3.1.4 Configure Application Flow for umts

For the umts case, an addition which had to be made for the C2C-IP application was to configure a new flow for it, within the RRC and RLC layers of the umts stack. To do this, we added the function **ConfigureNode()** in the C2C-IP app., which basically calls the function **NewApplicationConfiguration()** of the user or base station equipment manager (depending on the type of the node who is transmitting) with parameters such as the transmission type (i.e. unicast, multicast), the new application's index tag and the destination address of the receiver. Then, within this function, the **(rrclayer::ConfigureNewFlow())** is called to create and add the new application flow in the list of application flows which is kept in the

node's RRC layer. Subsequently, the new flow is also added in the RLC Layer of the node.

After these modifications, the generated application packets will not be blocked in the RRC layer of the transmitting node, since their application tag will fit the respective flow id in the applications' list. **We note here that these modifications need to be made for every new application which needs to use the umts technology**.

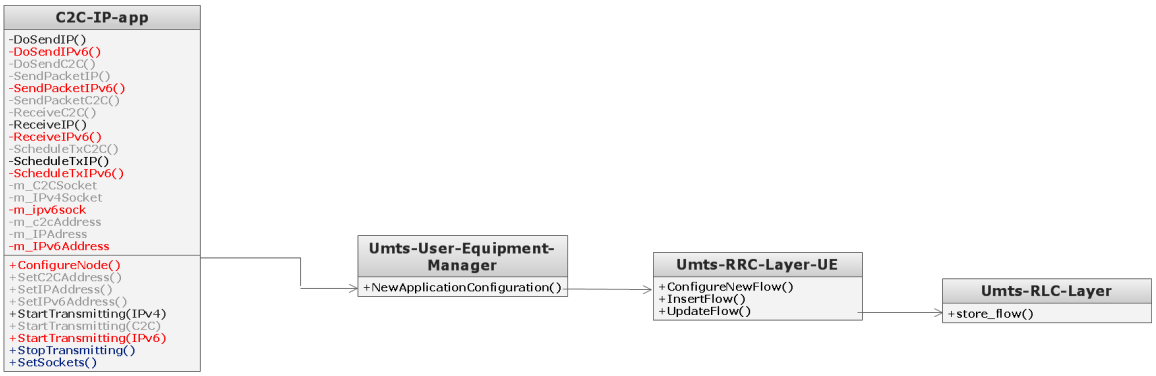

Figure 6: umts application flow configuration

3.1.5 Vehicle Station Management

At this point it would be useful to focus on some details regarding the Vehicle Station Management operation and the modifications needed there, as well as in its dependent classes, for IPv6 integration. Let's take the example of UMTS that you can see in the following figure.

- **Vehicle Station Management:** The important function here is ScanIpBaseStations(STACK). This function is called from GetCommunicationChannelTMC() for an infrastructure (TMC) based Transmission and from GetIpAddress(), which returns the IPv4 address of a BaseStation with a given nodeID. It is used to find the Best Serving Base Station for each technology by using the technology specific Vehicle Scan Manager which is available in the Vehicle. As usual, here we added a **GetIpv6Address()** function which returns the IPv6 address of a BaseStation or RSU given its nodeID. This function is also called from the addressing support facility and it triggers the Base Staions' and RSUs' scanning. We note here that the scanning procedure is different for the RSUs, supporting 802.11p technology however, since it is based on geolocalization mechanisms. In the next section, we will

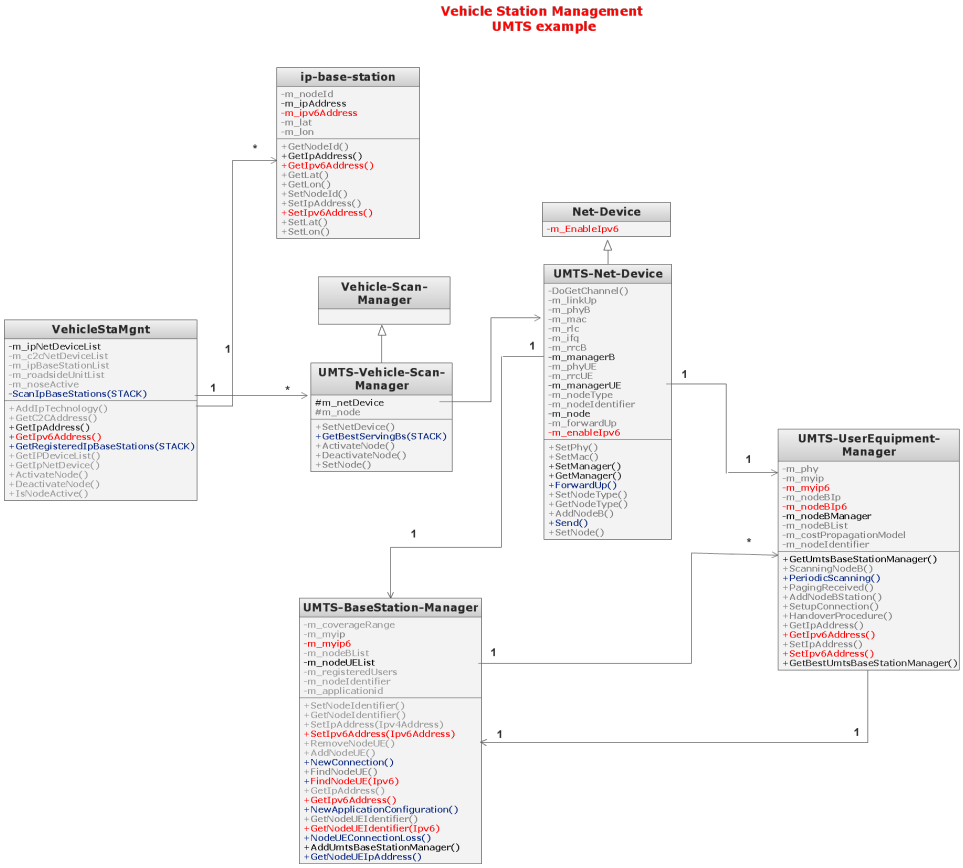

Figure 7: Vehicle Station Management and dependencies

show how we made use of these C2C-stack mechanisms to retrieve the IPv6 address of the destination RSUs.

- **IpBaseStation:** We make the logical assumption here that each IpBaseStation can support both Ipv4 and Ipv6 Addresses, so we didn't introduce a new type of Base station, but added some Ipv6 supporting functions to the existing one as you can see in the UML.

- **Umts Vehicle Scan Manager:** Here the **GetBestServingBs(STACK)** function is called from the Vehicle station management to find the best serving UMTS Base station Manager as we stated above by calling the function *UmtsUserEquipmentManager:: GetBestUmtsBaseStationManager(STACK)*. **After getting the appropriate base station manager it creates an IpBaseStation and it assigns it the IPv4 or IPv6 address of the base station manager, depending on the value of the STACK parameter.**

- **Umts User Equipment Manager:** The role of the calls to GetBestUmtsBaseStationManager(STACK), PeriodicScanning(STACK) here is to return to the Vehicle Scan Manager the Base Station Manager that corresponds to the NodeB with the best coverage in the area. To do this, a control packet is generated through the PeriodicScanning function and its transmission is initiated by passing it to UmtsBaseStationManager:: NewConnection(controlpacket) to establish a connection with a new base station. **Here, we have introduced the STACK parameter in the PeriodicScanning function in order to define whether the source address of the generated control packet will be IPv4 or IPv6**.

- **Umts Net Device:** Regarding the Umts-Net-Device, we introduced a boolean variable (i.e. *m_enableIpv6*) within the father class (i.e. Net-Device) which is enabled from higher layers (Ipv6Interface in particular) to define an IPv6 transmission. This variable is used for packet header deserialization, to make clear whether we have an IPv4 or an IPv6 packet header. Based on this header separation, we did the necessary modifications in functions: Send() (which sends the packets to lower layers) and ForwardUp (which forwards the received packets to higher layers).

3.1.6 RSU scanning and IPv6 address retrieval from the Vehicle nodes

As we mentioned above, the RSU scanning is based on a geolocalization procedure. In particular, each vehicle integrating the C2C stack holds a location table where it keeps track of the nodes in its neighborhood. This location table is being updated by the messages received, from a node's neighbors, within the C2C layer3 Protocol. To take advantage of that for the IPv6 case, we introduced an additional source IPv6 address field in the C2C-Common-Header. The C2C-Common-Header is generated within the C2C-L3-Protocol::Send() function of the transmitter. Thus, when the C2C-Common header is deserialized in the C2C-L3-Protocol of the receiver to add a new entry in the vehicle's location table, the IPv6 field is also added there.

As a result, when we need the destination's IPv6 address, within the addressing support facility, we just have to search through the location table for the particular entry of the location table with the specified node id and read the respective IPv6 field.

The described process is depicted in figure 8.

3.2 Main IPv6 Stack

As stated in the Introduction, for this part of the IPv6 integration we tried to adjust our iTETRIS ns-3.7 version to the implementation of the recent ns-3.14 version. One of the main decisions made in ns-3.14 was to use one Socket Implementation as well as one Transport Layer per each type (i.e. TCP, UDP) for

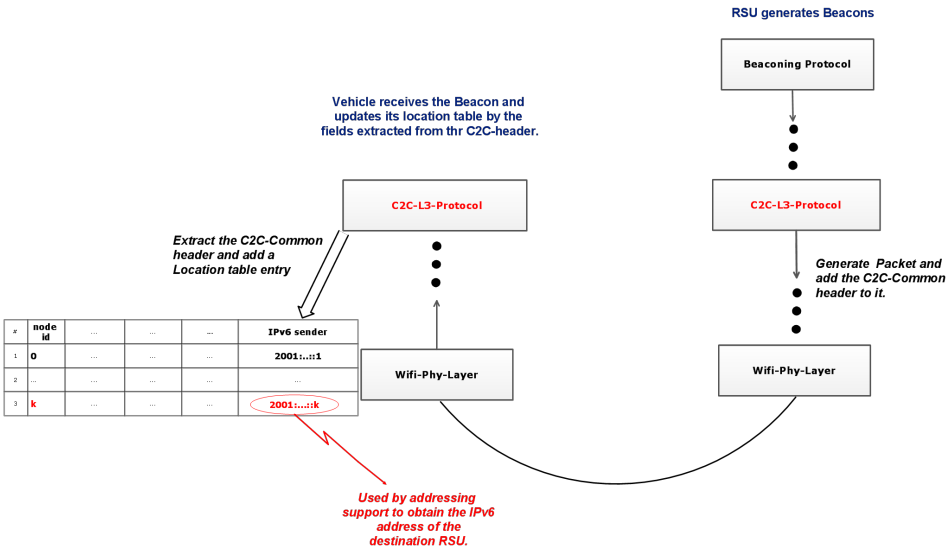

Figure 8: Location Table update and IPv6 address extraction

both IPv4 and IPv6 cases. Thus, in our case we had to enrich our socket and transport layers implementations with all the necessary functions and attributes to fully support IPv6 as you can see in the following UML.

Beyond that, the main IPv6L3Protocol, the ICMPv6 protocol as well as IPv6 static routing were already implemented but not integrated with the rest of the stack.

3.2.1 Socket/Transport Layer

We will focus now on all the modifications needed, regarding the UDP case. The TCP case implementation is similar but not really compatible with vehicular communications since it is difficult to assume that we have stream based communications due to high mobility conditions.

UDP-Socket-Implementation

- **Connect(Address), m_endpoint6, Bind(Address) Bind6(), FinishBind():** For the connection of the socket with the destination InetSocket Address, we modified the Connect function so that at the beginning it makes a check to see if the destination address is of type Inet4SocketAddress or Inet6SocketAddress. Then it sets its destination address and port equal to the respective fields of the InetSocketAddress of type IPv4 or IPv6.

  For the communication of the Transport Layer (i.e. UdpL4Protocol in our case) with the UDP-Socket, we introduced an IPv6Endpoint (m_endpoint6)

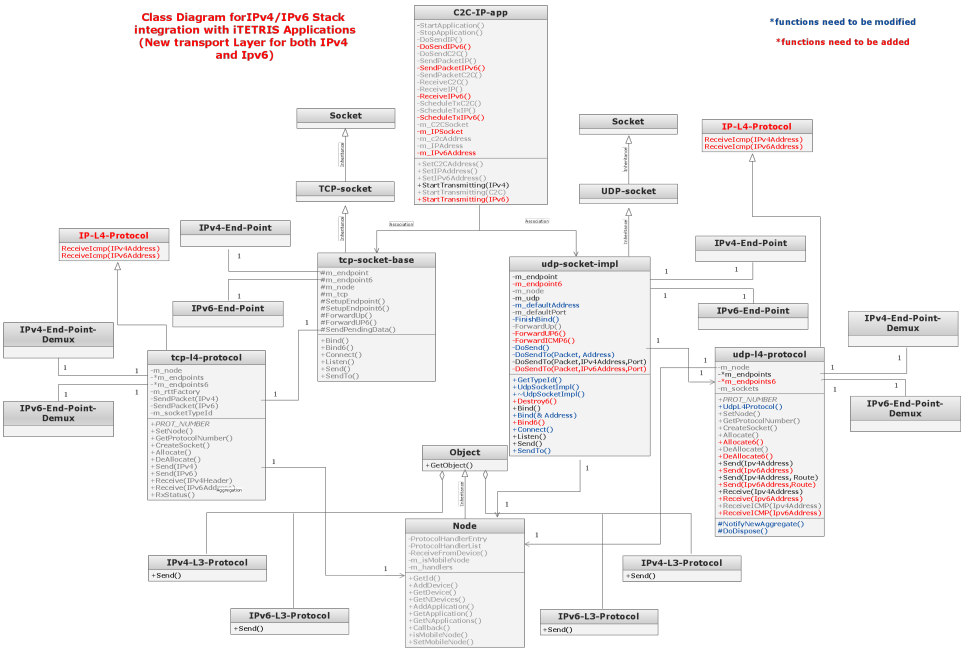

Figure 9: ns3-14 approach of socket, transport layers

and then we modified FinishBind() to associate its receive callbacks to appropriate functions of the UDP-Socket-Implementation object (e.g. SetRxCallback() to ForwardUp6(), SetIcmpCallback to ForwardIcmp6). Thus, when a packet is received from the transport layer the appropriate endpoint will be chosen from a list of IPv6 endpoints (held in UdpL4Protocol) and the packet will be forwarded through the appropriate function to the Socket Layer as you can see in figure 10.

- **DoSend(), DoSendTo(Address), DoSendTo(IPv6Address,Port):** These functions are called sequentially to send a packet to Layer 4. For the two first, we made the necessary modifications to implement the differentiation between IPv4 and IPv6 cases and we introduced the last one for IPv6 destination addresses. Inside that function a valid route is also extracted by contacting the routing protocol of m_node's IPv6 object as shown in the transmission side of figure 10. (We note here that, in this figure, it is shown that it is the L4 Protocol that extracts the route, which is the case for the TCP implementation. For UDP this task is performed by the socket layer but the mechanism remains the same).

- **ForwardUp6, ForwardIcmp6, Destroy6 :** The two first introduced functions are called from the IPv6 endpoints to forward regular or ICMP packets up to the socket layer, as mentioned above, whereas the last one is called to deallocate the IPv6 endpoints and the associated UdpL4Protocol.

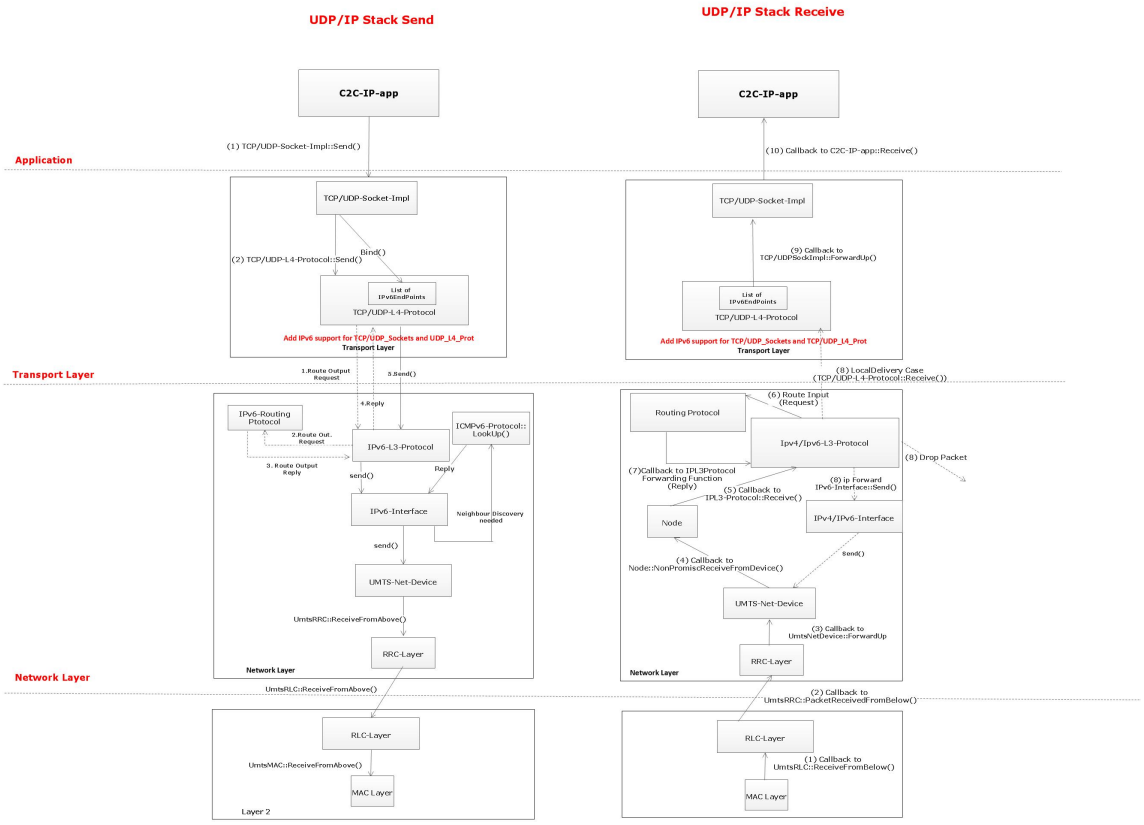

Figure 10: TCP-UDP/IP Stack flow for a UMTS Device

UDP-L4-Protocol

- **m_endpoints6, Allocate6(), DeAllocate6():** UdpL4Protocol has to keep a list of endpoints which are the links to the Socket Layer as we mentioned before. For that reason we introduced here a list of IPv6 endpoints (i.e. m_endpoints6). Through the call to Allocate6() function we can create a new endpoint and add it to the endpoint list. This is done during the binding procedure with the UDP socket. DeAllocate6(IPv6Endpoint) is used when we need to free an IPv6 endpoint.

- **Send(IPv6Addresses),Send(IPv6Addresses, Route):** Here we introduced two new functions for forwarding the packets to IPv6L3Protocol, after inserting the UdpHeader with the source and destination ports. In the second case, the Route is also provided by the socket layer as we explained above.

- **Receive(IPv6Address), ReceiveICMP(IPv6Address)**: These two new functions provide the support for receiving regular and Icmp IPv6 packets at layer

4 protocol and forward them to the appropriate endpoint, in order to reach the socket layer.

3.2.2 Network Layer

As we mentioned above, the IPv6L3Protocol as well as its dependent classes(i.e. IPv6Interface, IPv6InterfaceAddress, Ipv6RoutingProtocol, IPv6StaticRouting) were already implemented and we just had to connect them with the rest of the stack. Thus, in the following UML you can see this connection, with the **node** class being the linking point between Socket and Transport Layers and the IPv6 Layer 3 (IPv6 object aggregated to the Node).

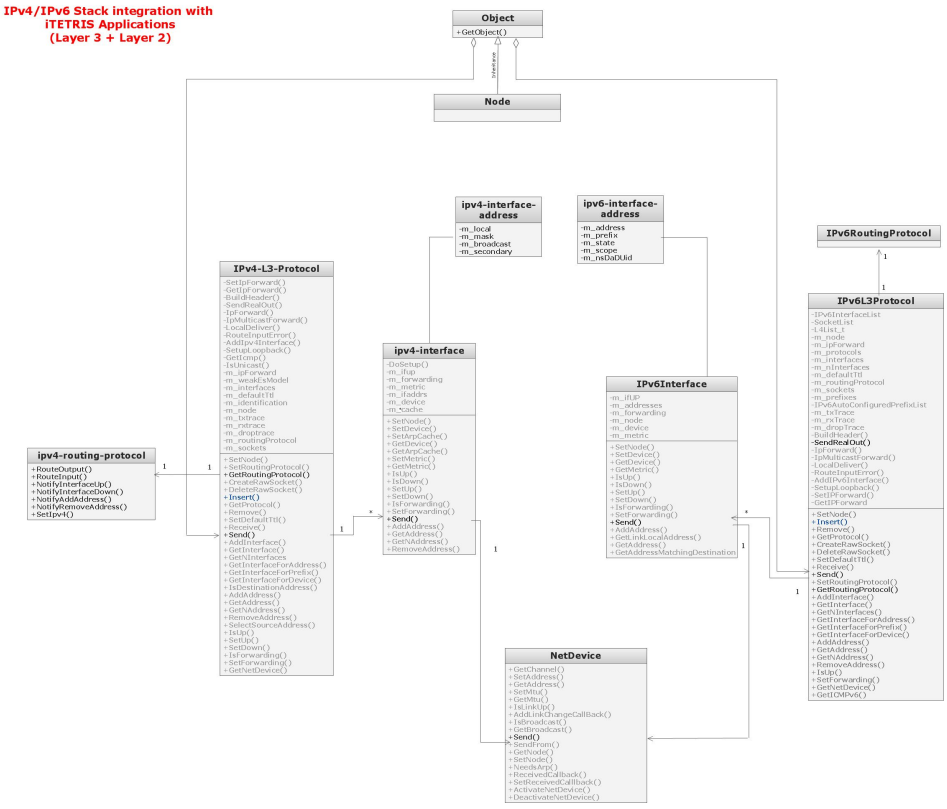

Figure 11: IPv6 Network Layer

3.3 Configuration and Installers

At this point, we will point out the modifications needed regarding the Device installers and the configuration of the Vehicles and the Base stations. Until now, we have dealt with the UMTS technology and this is where we will focus our analysis.

However, the modifications needed for the other technologies will not differ too much.

The starting point of any ns-3 simulation is a master xml file, which specifies the appropriate installers and the configuration files for each specific technology and node type. This file (i.e. fileConfTechnologies) looks like the following:

```
<installers>
      <installer type="ns3::UmtsVehInstaller" name="UmtsVeh" file="confUV.xml" default="false" />
      <installer type="ns3::UmtsBsInstaller" name="UmtsBs" file="confUB.xml" default="false" />
      <installer type="ns3::ItetrisNetworkTransportInstaller" name="NetTrans" default="true" />
</installers>
```

and it is parsed by **Configuration-Manager-Xml** in the way that is shown below:

```
ConfigurationManagerXml confManager (fileConfTechnologies);
iTETRISNodeManager* nodeManager = new iTETRISNodeManager ();
confManager.ReadFile(nodeManager);
```

Thus, inside `ConfigurationManagerXml::ReadFile(nodeManager)`, we first create an installer of the type specified in the master configuration file and then attach it to the nodemanager:

```
  nodeManager->AttachInstaller((char*)name, installer)
```

Attaching here means that we add an entry to the m_iTetrisInstallers, which is a list of installers and their respective module name (e.g. "UmtsVehicle" for UmtsVehicleInstaller). Appart from that, we create a NodeContainer for each module type (e.g.UmtsVehicle) and then insert the pair of module type and its corresponding NodeContainer in m_iTetrisTechnodes list.

The next step for the actual installation of the communication modules is to call:

```
nodeManager->CreateItetrisNode (Vector(700,0,0));
nodeManager->InstallCommunicationModule ("UmtsVehicle", IPv6);
```

The first line is creating an iTETRISNode with the particular coordinates and install any possible default communication modules to it (they are specified with `default=true` in the configuration file). The second line is called to install the specified communication module at the node we just created.

As we can see in figure 12 our modifications at this part start from the NodeManger, where we have modified InstallCommunicationModule() function for the Umts case by introducing the additional STACK parameter. The new version of the function is called only for the umts case. The reason for this is that, at this point, we want to avoid possible conflicts between IPv4 and IPv6 addressing when we add entries to the IPInterfaceLists (e.g. it may occur that there are two umts entries: one with IPv4 address and one with IPv6). Thus, we currently make a choice between them.

As we can see, in the Umts Installer there is a new **IPv6AddressHelper**. In the constructor of both installers we introduced a call to:

```
m_ipv6AddressHelper.NewNetwork ("2001::", 64)
```

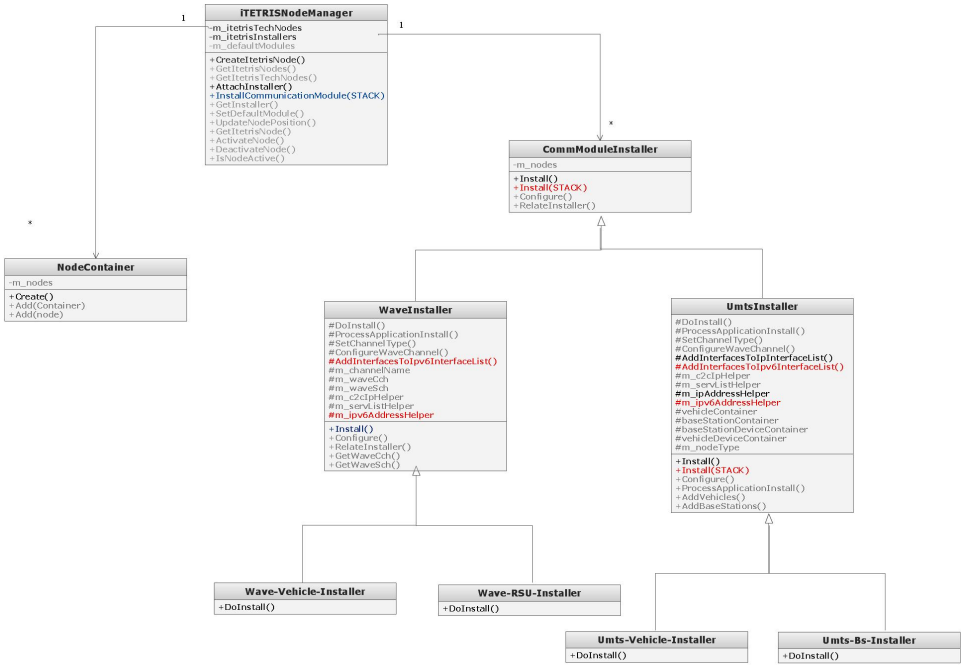

Figure 12: Modifications in the installers

so that we can allocate a new network from which we will create the addresses that we need for each specific technology node.

Then, inside umts-installer we call:

```
m_ipv6AddressHelper.Assign(netDevices);
AddInterfacesToIpv6InterfaceList (container);
```

to assign a valid IPv6 address from the Network for each device's IPv6 interface. Finally, we call: `AddInterfacesToIpv6InterfaceList (container)` to add the new IPv6 interface in the Ipv6InterfaceList held in the Node Container (in this case single node container).

At this point, we have made an important modification for the case of the mobile nodes. So far, within UMTS nodes configuration, the global IPv4 addresses were assigned to each node from the configuration file in the same way as we described above and which is still valid for the case of the IPv6 supporting Base Stations. For IPv6 simulations, however, each mobile node needs to perform the ICMPv6 autoconfiguration procedure to obtain a valid global IPv6 address. As a result, for the case of the mobile nodes (vehicles) this part of the umts-installer should differ in the following way:

```
m_ipv6AddressHelper.AssignWithoutAddress(netDevices);
```

Through this call, we just create and insert a new IPv6 interface to the node but without assigning it any IPv6 address. Also, we do not add this interface to the list of IPv6Interfaces (i.e. call to **AddInterfacesToIpv6InterfaceList**) since it does not have a valid address yet. Another modification that needs to be made, for the same purpose, is within the umts-vehicle-installer where we need to remove the line that assigns the IPv6 address to the umts-net-device. We will show in the next section when this call has to be made now.

3.3.1 Network Transport Installer

Let's go back to our master configuration file. From there, we call the Network Transport Installer. The main attribute of this class is an Internet-stack-helper, which is responsible for installing the enabled Communication stacks to the specified nodes. **Thus, here to provide IPv6 support we introduced a new boolean variable (i.e. m_ipv6enable) and a function to activate it (i.e. SetIpv6StackInstall).** As a result, in the new case of enabled IPv6 stack, the following protocols are created and aggregated to the node:

- *Ipv6L3Protocol*
- *Icmpv6L4Protocol*
- *UdpL4Protocol*
- *Ipv6StaticRouting (part of Ipv6L3Protocol)*

We mention here that, currently, we enable only IPv6 and C2C stacks in the Internet-Stack-Helper constructor, in order to avoid, once again, possible conflicts with IPv4.

3.3.2 Integration of ICMPv6-related Operations

As described above, the basic modifications that had to be made at this point were related to the integration of Address autoconfiguration procedure. For this process, a router advertisement daemon application (radvd) was already available in our NS-3 version, which has the task of periodically transmitting multicast router advertisements. **In this context, the radvd application should be installed in every IPv6 Base Station or RSU node that we configure and running in our simulations together with the C2C-IP application**.

The radvd installation takes place in the **umts-installer::Install()** function for the case of the Base station nodes which support IPv6.

As we can see from the following code, in line 1 we assign a valid global IPv6 address for the interface corresponding to the umts-netdevice of the Base Station.

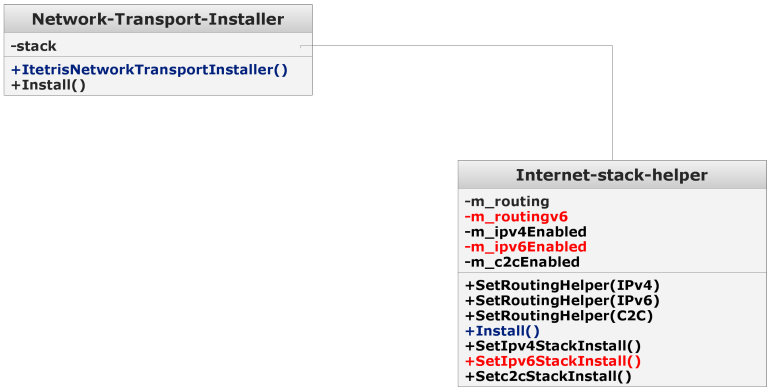

Figure 13: Network Transport Installer

This address comes from the Network prefix we defined in the Umts Base Station's configuration file. In line 2 we define that this is a router interface. In line 5 we create an instance of the radvd daemon. In lines 6-8, we create a RadvdInterface for the Router (Base Station) and add our network prefix to this interface. Then, we add a new configuration, corresponding to our router interface, in the list of configurations which are held in the radvd object. The third and fourth parameter of RadvdPrefix define the preferred and the valid life time of the autoconfigured address. In line 10, we finally add the radvd application to the Base Station and in lines 11-13 we set some application parameters (i.e. application tag, start, and stop time). Finally, in line 14 we add the umts interface in the list of IPv6 interfaces for the Base Station node.

```
1   Ipv6InterfaceContainer iicr = m_ipv6AddressHelper.Assign(
        netDevices);
2   iicr.SetRouter (0, true);
3   Ipv6Address prefix = m_ipv6AddressHelper.GetPrefix();
4   uint32_t indexRouter = iicr.GetInterfaceIndex (0);
5   Ptr<Radvd> radvd = CreateObject<Radvd> ();
6   Ptr<RadvdInterface> routerInterface = Create<RadvdInterface
        > (indexRouter, 5000, 1000);
7   Ptr<RadvdPrefix> routerPrefix = Create<RadvdPrefix> (prefix
        , 64, 3, 5);
8   routerInterface->AddPrefix (routerPrefix);
9   radvd->AddConfiguration (routerInterface);
10  uint32_t app_index = container.Get(0)->AddApplication(radvd
        );
11  radvd->SetAppIndex(app_index);
12  radvd->SetStartTime (Seconds (1.0));
13  radvd->SetStopTime (Seconds (3.0));
14  AddInterfacesToIpv6InterfaceList (container);
```

In figure 14, we can view the steps of the address autoconfiguration procedure. Starting from the right, the Base Station periodically sends multicast ICMPv6 Router Advertisement packets. The mobile station receives the Router Advertisements and, within the ICMPv6L4Protocol, the IPv6L3Protocol is triggered to perform the address autoconfiguration procedure. After the new address is generated, it is added in the list of addresses, which is kept in the umts IPv6 interface. Finally, the addition of the IPv6 address in the UmtsNetDevice (which was removed from the umts-vehicle-installer as we stated before) is also performed here.

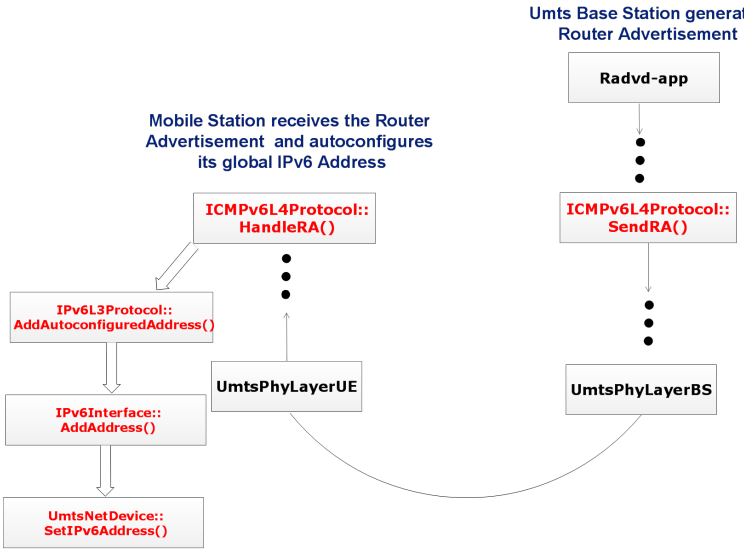

Figure 14: IPv6 Address Autoconfiguration Procedure

We note here that, as we described in section 3.1.3, we had to define and add a new application flow for the rrc and rlc layers of the transmitter (in this case Base Station). The same action needs to be done for the ICMPv6 Neighbor and Router Solicitations messages sent by the Mobile nodes (i.e. they are treated as one different application flow by the rrc and rlc layers). For that, we had to add a **ConfigureNode()** function inside ICMPv6L4Protocol and call it from inside the function which first generates an ICMPv6 message (in our case this is ForgeNS() which is called to perform the Duplicate Address Detection process).

4 Validation and Testing

To verify that the integration of the IPv6 and ICMPv6 mechanisms work fine for our version of ns-3 we had to make some tests. In this section you can see the results of this verification.

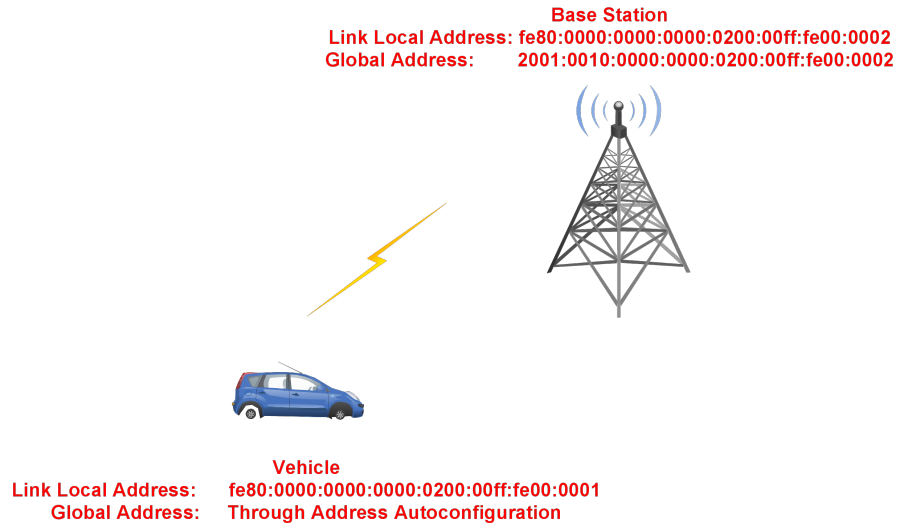

Figure 15: Scenario 1

4.1 Scenario 1: Mobile Node within the coverage of One Base Station

According to this scenario, the goal is for the Mobile Node to make a successful packet transmission to a Base Station. Before that, however, some ICMPv6 operations have to take place.

4.1.1 Duplicate Address Detection

Based on the IPv6 mechanisms, each node first acquires a link local address. Then, he has to verify that no other node in his local link is using the same address, by multicasting a Neighbor Solicitation. If there is a node with the same address he will reply with a Neighbor Advertisement containing this target address as shown in figure 16.

In figure 17 you can see the generation of the NS message from the ICMPv6 protocol of the mobile node:

And in figure 18 the reception and handling of the NS message by the Base Station:

4.1.2 Address Autoconfiguration

After having obtained a link-local address, the mobile node has to acquire a global IPv6 address through the address autoconfiguration procedure. To do this, he needs the prefix of the current network which is advertised by the Base Station through solicited or unsolicited Router Advertisements. We will show now the traces of this process, which is triggered from Router Solicitation messages (solicited case) from the MN, right after he has obtained his link local address.

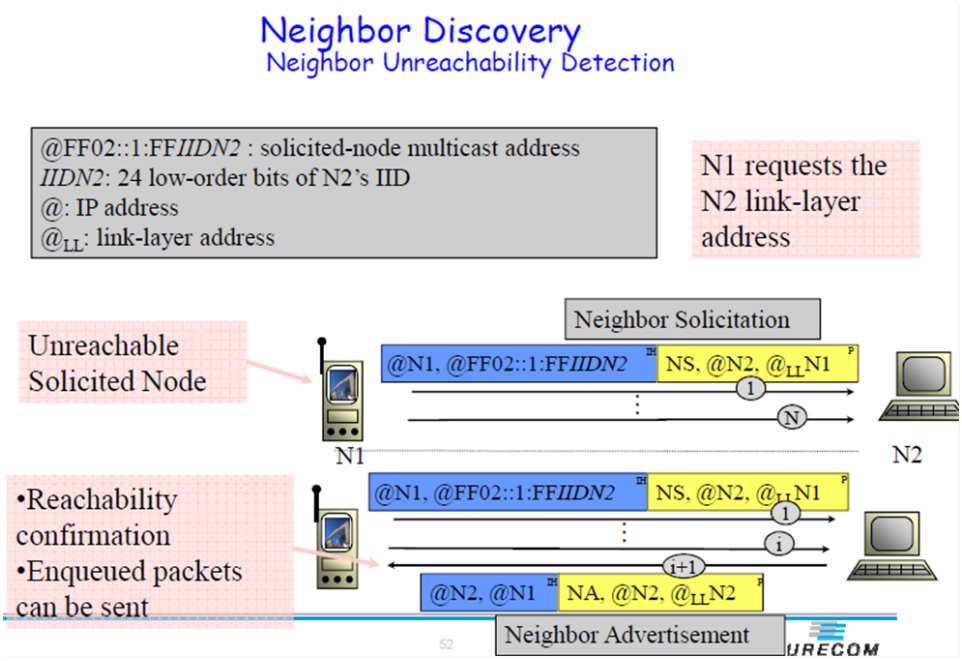

Figure 16: Duplicate Address Detection

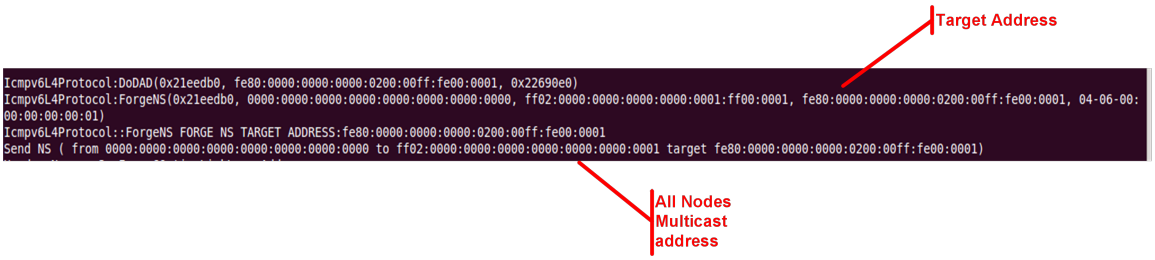

Figure 17: Generate Neighbor Solicitation

```
Packet received from ICMPv6!!!
Icmpv6L4Protocol:Receive(0x1439bb0, 0x13ba150, 0000:0000:0000:0000:0000:0000:0000:0000, ff02:0000:0000:0000:0000:0000:0000:0001, 0x16709e0)
Icmpv6L4Protocol:HandleNS(0x1439bb0, 0x13ba390, 0000:0000:0000:0000:0000:0000:0000:0000, ff02:0000:0000:0000:0000:0000:0000:0001, 0x16709e0)
Not a NS for us
```

Figure 18: Receive Neighbor Solicitation

In figure 19 you can see the generation of the Router Solicitation messages from the MN.

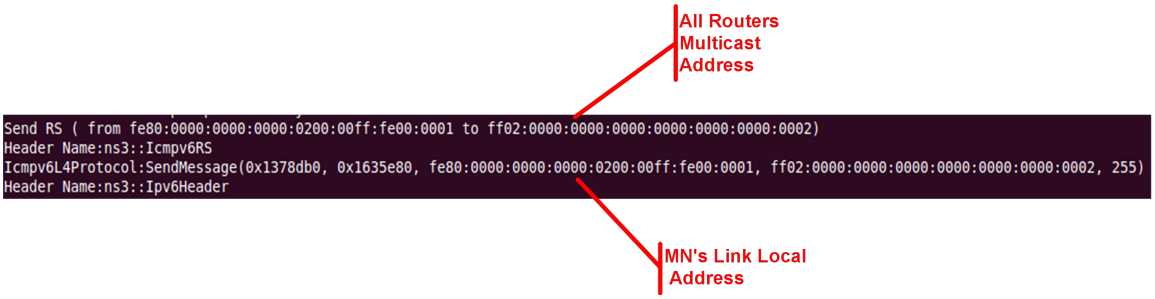

Figure 19: Generate Router Solicitation

From the next traces, you can see the reception and the handling of the Router Solicitation messages from the Base Station (router). The IPv6L3Protocol of the receiver is connected with the Radvd application through an IPv6RawSocket as shown in figure 20.

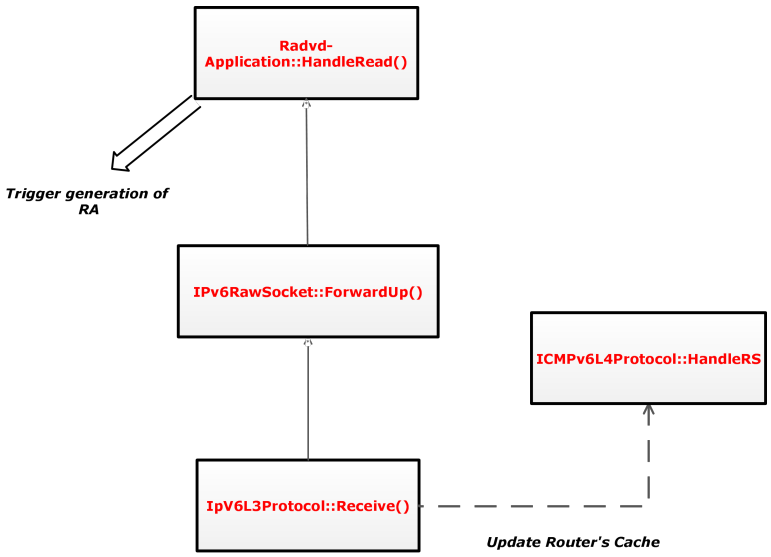

Figure 20: Router Solicitation handling

Moreover, the BS updates its cache with the link-local address of the Mobile Node as shown in 22.

Finally, the MN receives the RA and its ICMPv6 protocol triggers the stateless IPv6 address autoconfiguration. As we can see, it acquires address: **2001:0010:0000:0000 :0200:00ff:fe00:0001**.

```
Inside Socket::NotifyDataRecv
RadvdApplication:HandleRead(0x23295a0, 0x232e560)
Received ICMPv6 Router Solicitation from fe80:0000:0000:0000:0200:00ff:fe00:0001 code = 0
schedule new RA
```

Figure 21: RS reaches Radvd-application (installed in BS) which schedules the RA

```
Packet received from ICMPv6!!!
Icmpv6L4Protocol:Receive(0x20f1bb0, 0x232fb20, fe80:0000:0000:0000:0200:00ff:fe00:0001, ff02:0000:0000:0000:0000:0000:0000:0002, 0x23289e0)
Icmpv6L4Protocol:HandleRS(0x20f1bb0, 0x2332600, fe80:0000:0000:0000:0200:00ff:fe00:0001, ff02:0000:0000:0000:0000:0000:0000:0002, 0x23289e0)
Icmpv6L4Protocol:FindCache(0x20f1bb0, 0x20f2db0)
Cache updated by RS
```

Figure 22: Update Cache

4.1.3 Regular packet transmission

After having obtained its global address, the MN can now generate a regular packet Transmission, within the C2C-IP application. In trace 24 you can see the initialization of the packet transmission from the application.

And finally in trace 25 you can see the successful reception of the packet from the C2C-IP application layer of the receiver (Base Station).

4.2 Scenario 2: Mobile Node moves from the coverage of one Base Station to another

According to this scenario, the Mobile node moves to a position which is out of the coverage of the previous Base Station and within the coverage of a second Base station. As a result, a connection has to be made with the new Base station.

At this point, it would be useful to have a look at how this connection process is implemented in NS-3, in order to justify the modifications which needed to be done for the realization of this scenario. As we will show next, this is a two-step process which first identifies the connection loss with the first base station and then discovers the new Base station and triggers the association with it.

First of all we should make clear that, currently, the scanning procedure (performed by the vehicle to discover a serving BS around its position) is triggered by the **addressing support** facility which is called from the **iTETRISNS3-Facilities** to acquire the IPv6 Address of the destination node (given its id). The sequence of function calls eventually leads to calling the **Umts-User-Equipment-Manager::**

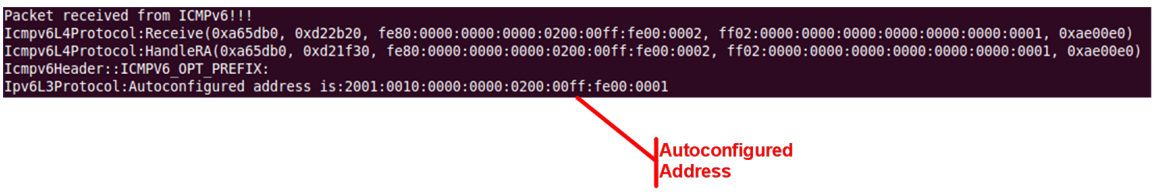

Figure 23: Reception of RA and address autoconfiguration

```
[ns3][C2CIPApp]********* START current IP periodic transmission before transmitting a new message on node  *********

[ns3][C2CIPApp] ======NODE 0 Tx to address 2001:0010:0000:0000:0200:00ff:fe00:0002 AppType UM-NON_FRAG  ===========

C2C-IP-App DESTINATION:2001:0010:0000:0000:0200:00ff:fe00:0002


[ns3][C2CIPApp] ======NODE 0: connection attempt to address 2001:0010:0000:0000:0200:00ff:fe00:0002   =====
```

Figure 24: C2C-IP application starting Txon

```
Inside Socket::NotifyDataRecv
[ns3][C2CIPApp] Start Receiving - Call SocketIP -> RecvFrom()

[ns3][C2CIPApp] SUCCESS: Receiving IP packet no. 1 from 2001:0010:0000:0000:0200:00ff:fe00:0001 at 5.6025 seconds | IP packet size  = 500 Bytes

[ns3][C2CIPApp]========= SUCCESS : IPv6 reception on node 1  ==============
```

Figure 25: C2C-IP application successful packet reception

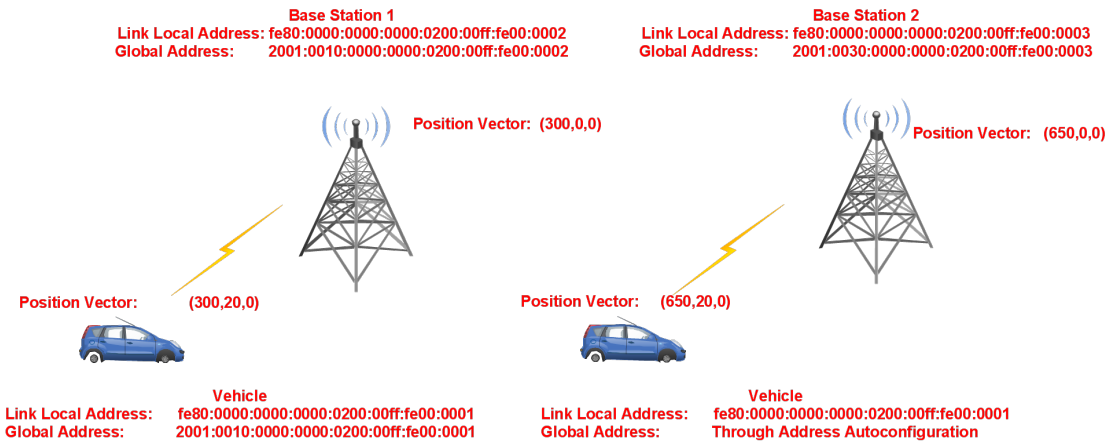

Figure 26: Scenario 2

PeriodicScanning() function of the vehicle and then the **ScanningNodeB()** function within the same class.

We will desrcibe this two-step process by the FSM that you can see in figure 27. As you can see, we have two inputs: **m_nodeBIdentifier** and **newNodeBIdentifier**. newNodeBIdentifier is the output of ScanningNodeB() function which returns the serving Base Station manager (i.e. Base station which has the vehicle in its coverage area), or -1 in case there is no Base station which covers the vehicle.

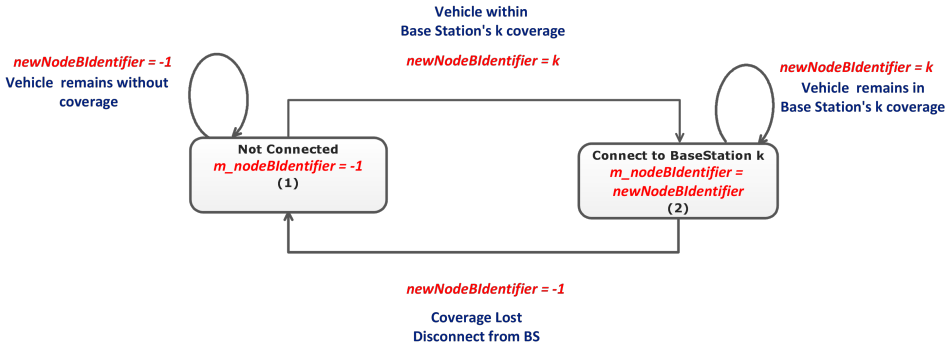

Figure 27: FSM of the connection and disconnection process from a umts BS

The important issue here is that, each time the Node B scanning is triggered by the Facilities, **only one transition is made**. This means that, when in our scenario we move the vehicle out of the coverage of Node 1 and inside the coverage of Node 2 and then we initiate an IdBasedTxon, which triggers the Facilities and eventually the scanning, the only action which will be performed is the identification of the lost coverage and the disconnection from BS 1 (transition 2 to 1). **The discovery of the new Base Station and the association with it (transition 1 to 2) will not take place until the next triggering by the Addressing Support facility**. As a result this first IdBasedTxon will not be successful.

To deal with this, if we do not want to waste an IdTxon only to make the first transition in the FSM (2 to 1), we have to make a check on the returned Base Station's address by the addressing support inside iTetris-NS3-Facilities. If this is null, then we trigger again the addressing support procedure to make the second transition (1 to 2). Otherwise, we could just accept a lost transmission and make the necessary FSM transition through the next transmission.

We should note here that this process is also necessary to be able to receive the Router Advertisements and configure the IPv6 Address according to the Network of the new Base station. In other words, the Vehicle needs to have acquired the m_nodeBIdentifier of the advertised Base Station in order be able to forward the Router Advertisements up to the Network stack.

Let's now have a look at the traces from the execution of this scenario. From traces 28 and 29, you can see a successful Id based Txon to the first base station

(node id=1). As you can see, the vector position of the vehicle at this time is (300,20,0).

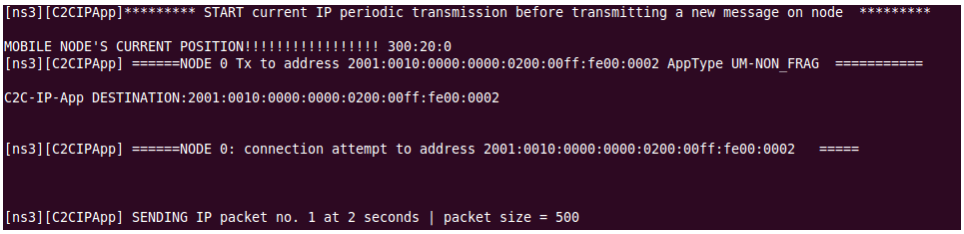

Figure 28: Id Based Txon to Base Station 1

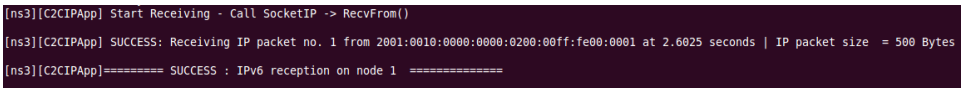

Figure 29: Packet Reception from Base Station 1

Then, the vehicle moves to position (650,20,0) and it's not in Base Station's 1 coverage anymore. As a result, it cannot complete an IdBasedTxon to Base Station 2 (node id=2), although it is in its coverage, due to the two-step association procedure that we described above. You can identify this by trace 30.

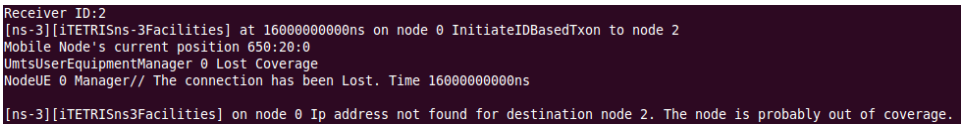

Figure 30: Vehicle loses coverage

Then, at the second triggering of the scanning procedure, the vehicle acquires the new Base station's id and connects to the new Base Station, as you can see in trace 31.

As a result, it is now able to receive Router Advertisements and perform the Address Autoconfiguration process as you can see in trace 32.

Finally, the vehicle is now able to make a successful transmission to the new Base station as we can see in traces 33 and 34.

4.3 Scenario 3: Mobile Node within the coverage of One Road Side Unit

In this scenario we tested the IPv6 integration in Wave technology. First of all, the RSU acting as a Router, advertises its network prefix and triggers the Address autoconfiguration procedure. In figure 36 you can see the handling of the Router advertisement and the acquisition of the global IPv6 address by the Vehicle node.

```
Header Name:ns3::Ipv6Header
Header Name:ns3::LlcSnapHeader
UmtsUserEquipmentManager 0 We are going to connect to NodeB 2
UmtsUserEquipmentManager 0 Setup Reply Received from Node B 2 Ip 2001:0030:0000:0000:0200:00ff:fe00:0003
UmtsUserEquipmentManager::PeriodicScanning, Connection successfully established2
```

Figure 31: Association to the 2nd Base Station

```
Packet received from ICMPv6!!!
Icmpv6L4Protocol:Receive(0x1dced90, 0x1dce410, fe80:0000:0000:0000:0200:00ff:fe00:0003, ff02:0000:0000:0000:0000:0000:0000:0001, 0x1e49a80)
Icmpv6L4Protocol:HandleRA(0x1dced90, 0x22cc0a0, fe80:0000:0000:0000:0200:00ff:fe00:0003, ff02:0000:0000:0000:0000:0000:0000:0001, 0x1e49a80)
Icmpv6Header::ICMPV6_OPT_PREFIX:
Ipv6L3Protocol:Autoconfigured address is:2001:0030:0000:0000:0200:00ff:fe00:0001 at time:20.7377
```

Figure 32: Mobile station performs the Address Autoconfiguration under the new network prefix

```
ServiceManagement::ActivateIPv6Service DESTINATION:2001:0030:0000:0000:0200:00ff:fe00:0003
[ns3][C2CIPApp]********* START current IP periodic transmission before transmitting a new message on node  *********

MOBILE NODE'S CURRENT POSITION!!!!!!!!!!!!!!!! 650:20:0
[ns3][C2CIPApp] ======NODE 0 Tx to address 2001:0030:0000:0000:0200:00ff:fe00:0003 AppType UM-NON_FRAG  ===========

C2C-IP-App DESTINATION:2001:0030:0000:0000:0200:00ff:fe00:0003


[ns3][C2CIPApp] ======NODE 0: connection attempt to address 2001:0030:0000:0000:0200:00ff:fe00:0003   =====



[ns3][C2CIPApp] SENDING IP packet no. 6 at 28 seconds | packet size = 500
```

Figure 33: Generate a regular packet destined at Base Station 2

```
[ns3][C2CIPApp] Start Receiving - Call SocketIP -> RecvFrom()

[ns3][C2CIPApp] SUCCESS: Receiving IP packet no. 2 from 2001:0030:0000:0000:0200:00ff:fe00:0001 at 28.6025 seconds | IP packet size  = 500 Bytes

[ns3][C2CIPApp]========= SUCCESS : IPv6 reception on node 2  ==============
```

Figure 34: Packet Reception from Base Station 2

Figure 35: Scenario 3

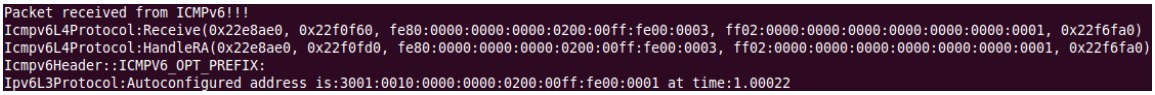

Figure 36: Address autoconfiguration

For this scenario we enabled the Beaconing protocol which triggers periodical Beacon Transmissions by each node to its neighbors [1]. In this way the Location Tables of the nodes are kept updated. Based on our description in section 3.1.5, we can now show in figure 37 the generation of the C2C-Common header, including the new IPv6 source address field (in this case RSU). The header is inserted in the Beaconing messages and C2C-L3-Protocol initiates the Broadcast transmission to RSU's neighborhood.

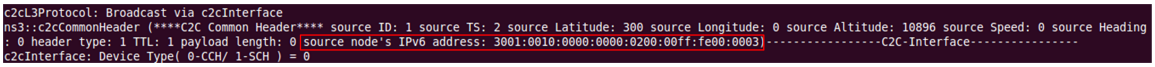

Figure 37: C2C Common header including the new IPv6 address field

The receiver of the Beaconing messages (in this case Vehicle node), decapsulates the C2C header and updates its location table, as shown in trace 4.3.

Then, when the Id Based Transmission is triggered by the vehicle node, the Ipv6 address of the RSU can be easily acquired within the Vehicle Station Management, by contacting the appropriate entry of the vehicle node's Location Table, as shown in trace 38. Finally the IP packet is successfully received in the RSU, as shown in trace 39.

4.4 Scenario 4: Mobile Node moves from the coverage of one Umts Base station to a Road Side Unit

In this scenario, the vehicle is equipped with both Umts and Wave technologies. At first its location is within Umts Base station's coverage and then it moves to

[1] Beaconing protocol is enabled in Network-Transport-Installer

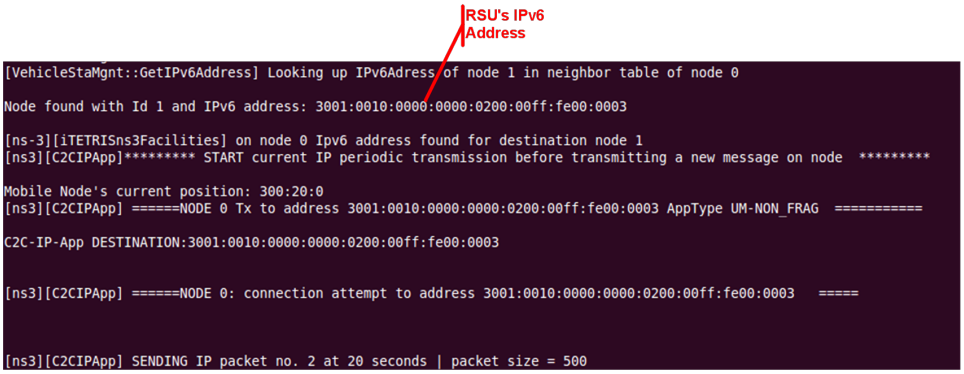

Figure 38: Retrieve RSU's IP and start the IP transmission

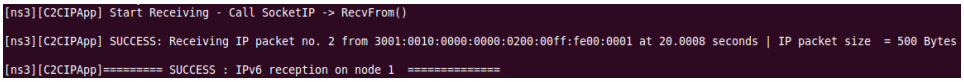

Figure 39: Receive IPv6 packet in the RSU

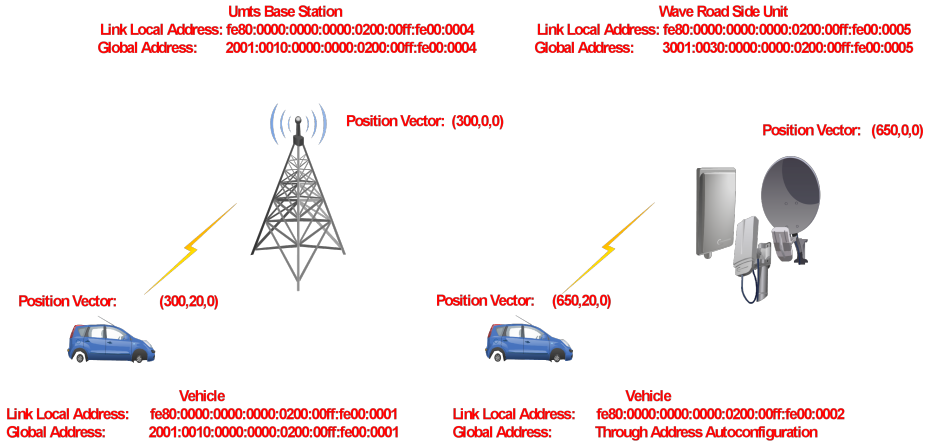

Figure 40: Scenario 4

Road Side Unit's coverage (**position change scheduled for time: 21 sec.**). Looking at the .pcap traces of this scenario we can get some interesting results.

In trace 41 we can see some Router Advertisement receptions and some packet generations from the Mobile Station to the Umts base station. In figure 42, we can see the trace of the umts base station's interface and in figures 43 and 44 the wave interface traces for the mobile station and the RSU respectively.

An important remark that we can make here, is that the 3rd regular packet which is generated from the MS (trace 41) is never received from neither the Base station nor the RSU. This is due to the fact that the Mobile station is, at this moment, out of Umts Base station's coverage and although he has tracked the RSU as the best serving infrastructure node (we can verify that by looking at the destination address of the generated packet), he still hasn't received any Router Advertisement to perform the address autoconfiguration, based on the RSU's network prefix. As a result, the generated packet tries to be transmitted through the umts interface, based on its old IPv6 address, and the transmission is not successful.

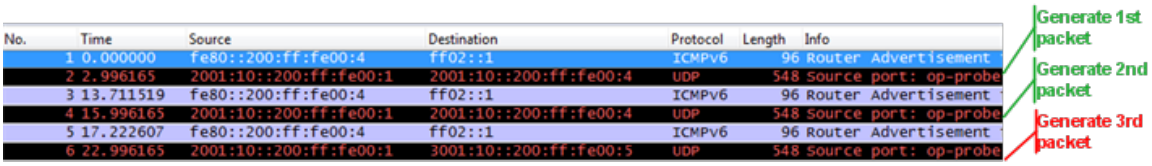

Figure 41: Trace of Mobile Station's Umts interface

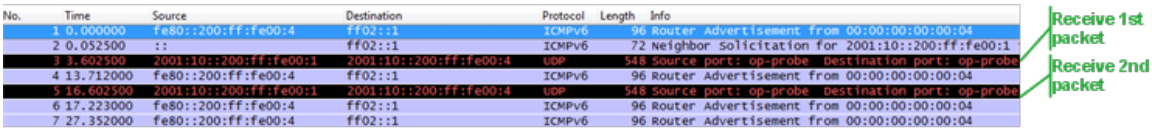

Figure 42: Trace of Umts Base Station's interface

By the timestep of the fourth packet generation, however, the Vehicle has activated its new IPv6 address in its Wave interface as you can see in trace 43. As a result, you can see the successful packet reception by the RSU in trace 44.

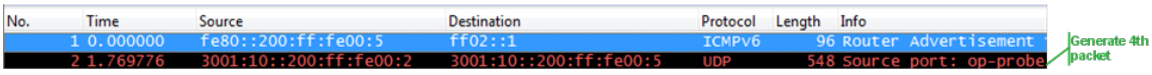

Figure 43: Trace of Mobile Station's Wave interface

4.5 Scenario 5: Infrastructure-Based Txons

In this scenario, the Traffic Management Control (TMC) is the one that initiates the transmissions. As a result, based on the target destination area, it picks the best

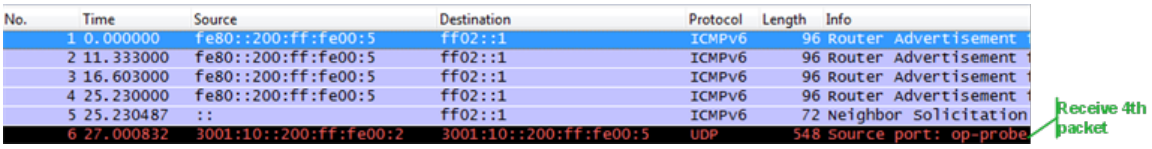

Figure 44: Trace of RSU's interface

serving infrastructure Node. In order to "monitor" this behavior, we introduce a vehicle, equipped with both technologies, which moves through the coverage of three infrastructure nodes (in particular two Wave RSUs and one Umts Base Station as you can see in figure 45). The target destination area of each transmission is always close to the position of the moving vehicle.

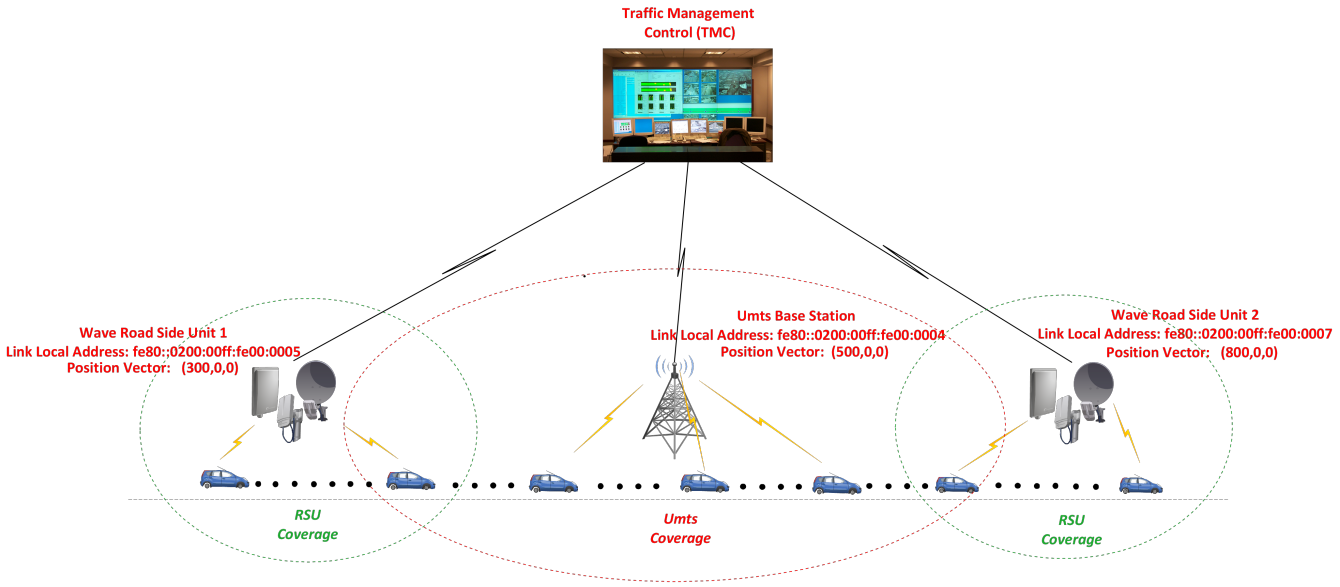

Figure 45: Scenario 5 (TMC Based Txon)

The nature of the messages which are transmitted is IPv6 multicast, destined at all vehicles belonging to an infrastructure node's local link. Thus, the destination address of the packets is always the all-nodes link local address (**ff02::1**). This scenario can actually be the case for POI transmissions, where an infrastructure node needs to forward, for example, a local advertisement to all the nodes situated in its transmission range.

At this point, we should highlight an addition which needed to be made in the code for this scenario. For the case of link-local multicast transmissions, the ipv6-static-routing component of the transmitting node demands for an association with a particular interface(device). This makes sense for this type of transmissions, since the appropriate interface to be used cannot be retrieved by the network prefix of the

destination address, as it would be done for the case of unicast transmissions for example (i.e. the all-nodes link local multicast address is not technology specific).

As a result, there has to be a binding of the C2C-IP application's IPv6 socket to the particular net device which is installed in each infrastructure node (Umts BS or Wave RSU). This is done now by calling the function Socket::BindToNetDevice(NetDevice) from the application level, **only for infrastructure nodes**, and then assign this interface as the output device of the route entry returned by RouteOutput() method of Ipv6-static-Routing.

Assigning one single NetDevice to a static Infrastructure Node does not imply any restriction for our simulations at this point, since the infrastructure nodes are expected to be using only one type of device(technology) for IPv6 transmissions.

Let's now have a look at the traces from this simulation scenario. As you can see from figure 45, the target destination area (vehicle) is closer to the 1st RSU at the beginning. As a result, the TMC should pick this RSU as the static node which will forward the multicast message, based on the minimum distance criterion that we discussed on section 3.1.2. This is verified by traces 46 and 47, where you can see that the static node which transmits the first two UDP packets is the 1st RSU and the vehicle receives them from its wave interface.

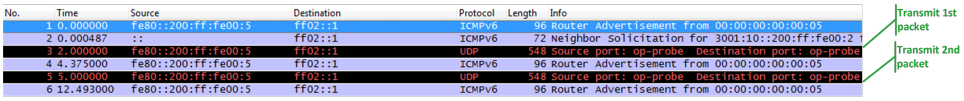

Figure 46: Trace of RSU 1 interface

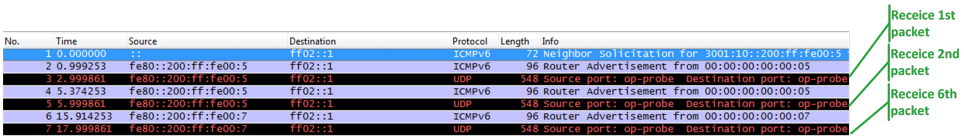

Figure 47: Trace of Vehicle's Wave interface

By the time of the 3rd packet transmission the destination area (and the position of the vehicle) have changed enough so that it is now closer to the umts base station. As a result, you can see in traces 48 and 49 that the three next multicast transmissions are generated from the Umts base station and received by the respective interface in the vehicle.

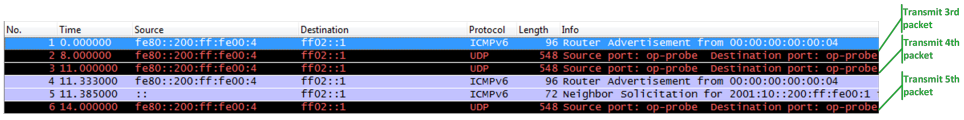

Figure 48: Trace of Umts BS interface

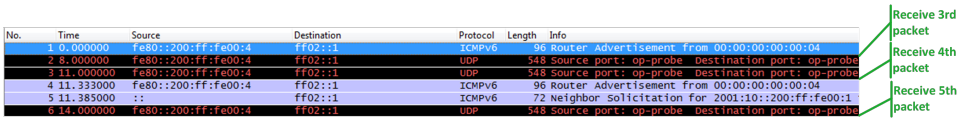

Figure 49: Trace of Vehicle's Umts interface

Finally, the target area changes enough so that it's now closer to the 2nd RSU and as a result the 6th packet is transmitted from this static node. You can verify this from traces 47 and 50.

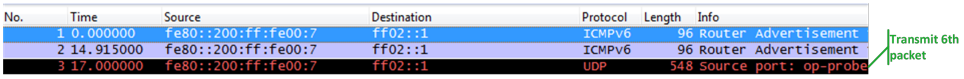

Figure 50: Trace of RSU 2 interface

References

[1] http://www.nsnam.org/docs/manual/html/index.html.

[2] http://www.ict-itetris.eu/.

[3] "iTETRIS Building, Installation and Configuration Guidelines".