Geographic addressing and routing
for vehicular communications
http://www.geonet-project.eu/

ICT-2007.6.1: ICT for intelligent
vehicles and mobility services

GeoNet STREP N°216269

D5.1 GeoNet Emulation Environment Results

DATE	04 June 2010
CONTRACTUAL DATE OF DELIVERY TO THE EC	M22 – November 2009
ACTUAL DATE OF DELIVERY TO THE EC	M25 – March 2010
EDITOR, COMPANY	Hamid Menouar, HITACHI Europe
WORKPACKAGE	WP5 Emulation Environment Development
DOCUMENT CODE	GeoNet-D.5.1-v1.2
SECURITY	Public

DOCUMENT HISTORY

Release	Date	Reason of change	Status	Distribution
0,1	30/10/09	First internal draft	Draft	Internal
0,2	11/12/09	Running C2CNet in NCTUns	Draft	Internal
0,3	14/12/09	Connecting external nodes in NCTUns	draft	internal
0,4	18/12/09	Running C2CNet in NCTUns - implementation guidelines	draft	internal
0,5	22/12/09	Available tools, Introduction to NCTUns	Draft	internal
0,6	09/01/10	Emulation results	Draft	internal
0.7	19/01/10	Completed some more sections	Draft	Internal
0.8	21/01/10	Updating Section 4.2	Draft	Internal
0.9	24/01/10	Pre-final version for internal review	Draft	Internal
0.10	08/02/10	Revision for second internal review	Draft	Internal
1.0	01/03/10	Final version	Final	EU
1.1	04/05/10	Revised	Final	Internal
1.2	17/05/10	Final revision	Final	EU

Name of the coordinating person: Arnaud de La Fortelle, INRIA
E-mail: Arnaud.de_La_Fortelle@inria.fr

Contents

1. Executive Summary

The GeoNet project aims at providing a reference specification of IPv6 GeoNetworking for vehicular communication networks. This specification is provided by Work Package 2 *Specification* (see [GeoNetD1.2] and [GeoNetD2.2]). In Work Package 3 *Implementation*, IPv6 adaptation modules and two independent prototype implementations of the C2CNet layer are implemented and integrated into a common IPv6 GeoNetworking protocol stack. For a good understanding of this document, the reader is assumed to be familiar with the IPv6 GeoNetworking architecture and terminology as defined in [GeoNetD1.2].

The purpose of developing two prototype implementations of the C2CNet layer is to validate the GeoNetworking capabilities and to ensure their compliance with the reference specification. Indoor and field test experiments using four equipped vehicles have been performed in Work Package 7 *Experimental Validation* (see [GeoNetD7.1]). The tests performed in WP7 are valuable in improving the GeoNet implementations and specification and evaluating the performance of GeoNet results in real-life scenarios. However, because of the inherent limitations of the real-life scenarios in terms of number of cars, simulation and emulation is needed to validate the GeoNet implementations and specification in large scale networks. By means of simulation and emulation the C2CNet Layer implementations can be tested extensively in order to validate their compliance with the specifications and their performance, scalability and efficiency in scenarios which are too complex to be performed in real-life scenarios.

The well-known emulation and simulation tool NCTUns [NCTUns1, NCTUns2 and NCTUns3] was selected.

In the rest of this document we first explain the requirements and motivations of this work with a brief introduction of existing emulation and simulation tools. Then we explain how the emulation environment is set-up. Finally, we explain how the emulation has been performed by showing the results and their analysis.

2. Objectives

The work reported in this document aims at developing a platform over which the GeoNet implementations of the C2CNet layer [GeoNetD3.1] can be tested extensively in order to validate their compliance with the GeoNet specification [GeoNetD2.2] and their performance, scalability and efficiency in scenarios which are too complex to be performed in real field tests [GeoNetD7.1]. Validation of the GeoNet implementations is important to validate the specification of GeoNet.

Initially the objective was to prepare an emulation environment capable of simulating a set of nodes equipped with the two implementations of the C2CNet layer, and emulating a set of real cars equipped also with the full IPv6 GeoNetworking architecture. To manage the emulation part it was planned to make the emulation environment interacting with cars equipped with the same GeoNet implementations as used [GeoNetD7.1].

These objectives as defined at the beginning of the project have been changed due to some technical limitations of NCTUns. A couple of months after the beginning of this work, it was discovered that the current version v.5 of NCTUns does not support IPv6 which made the simulation of GeoNet nodes in NCTUns impossible. However, NCTUns was selected based on the discussion GeoNet partners had with NCTUns developers who ensured that IPv6 could be supported by means of tunnelling. The impossibility to use such a method was discovered at a later stage when the adaptation of the C2CNet Layer in NCTUns was already engaged.

The version v.5 of NCTUns used to perform the emulation work was the latest available version when NCTUns was adopted. Version v.6 has been released since then but unfortunately it does not solve the limitations encountered earlier. To overcome this limitation in NCTUns, the GeoNet partners decided for an alternative solution where NCTUns is used to simulate the C2CNet Layer only, without simulating the IPv6 part. This solution is not sufficient since the purpose of this work is to deal mainly with IPv6 over C2CNet Layer. In order to keep IPv6 included in the tests performed, it was decided to make NCTUns, which simulates only the C2CNet Layer, interacting with real cars that support IPv6. The real cars would play the role of sources and destination and all intermediate nodes would be emulated. IPv6 packets would be generated at the source and then transmitted by the GeoNet OBU within the vehicle as payload to the C2CNet Layer. These C2CNet packets would be routed through intermediate nodes simulated in NCTUns, until reaching real cars serving as destinations at the IPv6 layer.

To make this interaction between the simulation environment and real cars possible, NCTUns provides a dedicated emulation feature. Unfortunately, this emulation feature works fine only when the number of simulated nodes is very small (two or three), which makes the interaction with real cars not possible anymore. This is another technical problem that obliged looking for another alternative solution.

As an alternative solution, it was decided to keep using NCTUns but to make only simulation. Although NCTUns does not support IPv6, the additional cost of IPv6 over the

C2CNet layer was considered by attaching a dummy payload (IPv6 packet) to C2CNet packets that are generated at source nodes.

After considering all the changes in the objectives as listed previously, the effort was concentrated on the evaluation of the performances of C2CNet GeoRouting protocols.

After preparing the simulation/emulation platform, we performed tests considering among the scenarios listed [GeoNetD1.2] those that are mostly used for IPv6 over the C2CNet layer communications: GeoUnicast and GeoBroadcast. These two communication schemes use respectively the two following GeoRouting schemes at the C2CNet layer:

- GeoUnicast (used to transmit IPv6 Unicast): We set up a communication between two GeoNet nodes, Source and Destination. The emulation was started making sure that the Source and the destination are sufficiently far away geographically. They shall at least be out of the communication range from one another. We launched a small application which sends packets from the Source to the Destination. Then we observed how the communication between Source and Destination is managed. We could also calculate the different performances metrics, such as packets delivery ratio, end-to-end communication delay, etc.

- GeoBroadcast (used to transmit IPv6 Multicast): We set up a source node which transmits GeoBroadcast packets targeting a set of GeoNet nodes located within a desired geographic destination are (GeoDestination). On top of the source node we launched an application which keeps transmitting GeoBroadcast packets. The GeoDestination is a circle area previously defined. In the emulation environment we keep some GeoNet nodes outside of the GeoDestination. Here we observed how GeoBroadcast packets reaches all the GeoNet nodes located within the GeoDestination. We could also evaluate different performance metrics, such as packet delivery ratio, end-to-end communication delay, etc.

We will not go deep in the tests, but we will provide some basic performances tests results. We want also to mention that there are European projects that deal mainly with emulation/simulation and field operational tests that could go deeper in testing the GeoNet specification and implementations. **iTETRIS** [iTetris] and **PRE-DRIVE C2X** [PDVC2X] projects are good examples for that. Some GeoNet partners are already involved in these projects and have managed to make both implementation and specification of GeoNet adopted.

3. Emulation Environment Tools

3.1 Motivations and Requirements

The need for an emulation environment is motivated by the nature of the communication system we are dealing with in GeoNet, i.e. vehicular wireless communication systems. Such cooperative systems are distributed and too complex to be validated by means of simple real field of tests. For example, the implemented GeoRouting algorithms use multi-hop communication where intermediate hops are selected based on their geographical information. If we want to validate their performance, with considering network scalability and over large geographical areas, the only way to do which fits with the GeoNet resources is to use simulation. By relying to simulation, the performance of GeoRouting protocols and IPv6 GeoNetworking could be tested in complex scenarios..

The emulation environment should comply as much as possible with the following requirements:

- Capability to evaluate IPv6 over C2CNet layer functionality;
- Capability to support realistic communication scenarios;
- Capability to interact with real cars;
- Capability to support realistic wireless communication behaviours;
- Capability to support a large number of nodes.

3.2 Available and Selection Tools

The tool required for performing simulation and emulation has to comply with several requirements such as:

- Support node mobility;
- Capability to feed the simulation with real network traffic (e.g. as data traffic generator);
- Multiple real applications could be operated simultaneously in one single machine;
- Information between simulation machine and real world components;
- Real protocol stacks like TCP/IP, or simulation of other protocols such as MAC/PHY layer protocols;

- Radio propagation may be simulated by an appropriate propagation model;
- Allow the modification of the protocol stack;
- Capability to provide its own location updates.

Several popular and relevant tools have been investigated and discussed with respect to the requirements listed above:

- **VMware:** A widely-used virtualization software for X86-compatible machines supporting most operating systems, such as Microsoft Windows, Linux, and Mac OS. It provides a virtualization environment which keeps the hardware and virtual machine terminology unambiguous. That means, the physical hardware computer is a host machine, and the virtual operating system running inside a virtual machine is the guest. VMware offers a complete set of virtual hardware to the virtual operating system like an emulator. According to this feature, it seems that VMware fits well to the required testbed. But for running VMware software, the consumption of resource in a computer like CPU and memory is quite big. Thus, in a normal computer only 5 to 6 nodes can run. Furthermore VMware can not provide a MAC/PHY protocols implementation neither [VMware].

- **OpenVZ:** Similar to VMware, OpenVZ is an operating-system level server virtualization solution build on Linux. Compared to VMware, OpenVZ has a limitation that it requires both host and guest Operating System (OS) to be Linux [OpenVZ]. Although this limitation doesn't influence the required testbed, OpenVZ has the same problem as VMware, i.e. no MAC/PHY implementation and supporting only few number of nodes.

- **NS2:** A very famous and popular network simulator used in many research projects. It has almost all functionalities that are required by test-beds, such as topology and environment design, MAC and PHY implementation, node mobility and integrated emulation. Moreover NS2 has an optional Graphical User Interface (GUI), and is a open source software, which is free to use and develop. Nevertheless, NS2 can experience some problems with emulation since some severe bugs have been reported and have not been corrected so far [NS2]. Furthermore, NS2 cannot use real application programs as traffic generation programs for simulation.

- **OPNET:** OPNET is a leading commercial network simulator. It has a very huge discrete event simulation model library which includes almost complete modellers. However it is a licensed commercial software which is very expensive [OPNET].

- **IMUNES:** An Integrated Multi-protocol Network Topology Emulator/Simulator (IMUNES) runs on the FreeBSD operating system. FreeBSD kernel is partitioned into multiple lightweight virtual nodes, which can be interconnected via kernel-level links to form arbitrarily complex network topologies. Therefore each node has an independent replication of the entire standard network stack, which enables not only highly realistic and detailed emulation of network routers, but also each virtual

node to run a private copy of any unmodified user-level application including routing protocol daemons, traffic generators, analysers and applications. Still it doesn't provide wireless MAC protocol implementation, mobility and floating point operation, which is a relatively big disadvantages [IMUNES].

- **VNUML:** Virtual Network User Mode Linux (VNUML) is an open source general purpose virtualization tool for defining and testing network simulation scenarios based on User Mode Linux (UML) virtualization software. It supports MANETs, but the number of links increases extremely fast as the number of nodes increases. Also the tool has no MAC protocol implementation [VNUML].

- **MobUML:** It is an extension of the original simulator developed in the project Wifi for UML. Same as VNUML, its network simulation scenario is also based on UML. It supports node mobility and ad hoc networks, but has no MAC implementation [MobUML].

- **SUMO:** Simulation of Urban MObility (SUMO) is an open source, highly portable, microscopic road traffic simulation package designed to handle large road networks. It is used for road traffic simulation, and offers road networks design and a car movement model. SUMO has no functionality to provide communication network simulations [SUMO].

- **VIMSIM:** Similar as SUMO, VIMSIM is a microscope road traffic simulator. It provides complex car movement models, road traffic control and topology design, but nothing about communication networks [VIMSIM].

- **MMTS:** Multi-agent Microscopic Traffic Simulator (MMTS) is capable of simulating traffic over real regional road maps of Switzerland with a high level of realism. From it, many vehicular traces are obtained and modified to NS2 movement format, which are used to the simulation by NS2 [MMTS].

- **NCTUns:** A novel tool that supports both simulation and emulation [NCTUns1, NCTUns2].

Among all these listed tools, GeoNet has selected NCTUns as the one to be used for performing the emulation work. NCTUns is a nice tool that has been already experienced by some GeoNet partners in some previous projects. It has the capability of easily integrating real communication implementations, which is helpful for GeoNet that aims of testing real implementations. Additionally, it is user-friendly thanks to the nice graphical interface it offers. It satisfies our criteria for tool selection best out of all presented tools.

NCTUns is detailed in the following section.

3.3 Introduction to NCTUns

NCTUns is the abbreviated name of the National Chiao Tung University network simulator. It is a high-fidelity and extensible network simulator and emulator capable of simulating various devices and protocols used in both wired and wireless networks. It is based on a novel methodology, the kernel-reentering simulation methodology [NCTUns2] [NCTUns3].

3.3.1 Kernel-Reentering Methodology

The kernel-reentering methodology is created by Prof. Shie-Yuan Wang, the leader of NCTUns developer group. With this methodology, NCTUns can overcome several problems in most traditional network simulators, e.g. simulation results are not as convincing as those produced by real hardware and software. The key facility in the kernel reentering simulation methodology is the tunnel network interface. A tunnel network interface on most UNIX systems is a pseudo network interface which is not attached to a real physical network, but it is not different from an Ethernet network interface from point of view of its functionality. An application program can send or receive packets from a tunnel network interface with the same behaviour as a normal Ethernet network interface. A simple simulation case with a single-hop TCP/IP network using a kernel network interface is depicted in Figure 1.

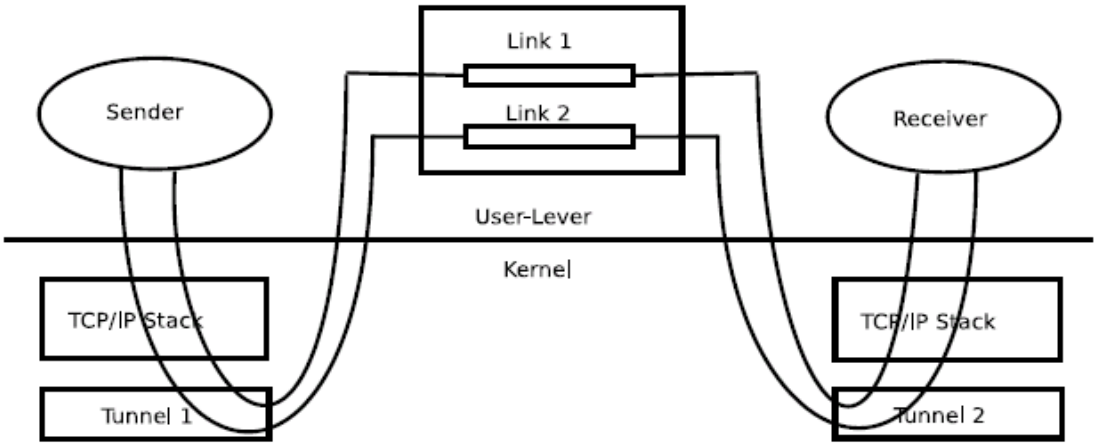

Figure 1: Simulation Structure in NCTUns

In Figure 1, a TCP sender program running on host 1 (sender) sends TCP packets to a TCP receiver program running on host 2 (receiver). In this NCTUns simulation architecture, two tunnel interfaces are used: tunnel interface 1 is used for TCP packets transmission from host 1 to host 2, and tunnel interface 2 is used for TCP packets transmission from host 2 to host 1. The characteristics of these two links are simulated by the simulation engine.

3.3.2 Protocol Stack

NCTUns offers a flexible protocol stacks under IP layer. It provides a complete solution to simulate an IEEE 802.11p based vehicular network: the IEEE 802.11p/1609 protocol stack is shown on Figure 2. The existing 802.11p solution supports transmission of Internet Protocol (IP) packets and WAVE Short Messages (WSMs). The IEEE 1609.3 specification defines the Wave Short Message Protocol (WSMP), which is a layer-3 and layer-4 protocol to regulate transmission and reception of WSMs. The data traffic applications are realized at the application layer. Through the protocol stack, packets are transmitted into receiver nodes. Moreover, this NCTUns protocol stack can be modified by a user, e.g. deleting a module, inserting other modules or changing default characteristics of a module.

NCTUns supports hybrid simulation and emulation also. Basically, in this hybrid mode, an external emulation machine running an application interacts with a simulation machine running a simulated network by NCTUns. In the simulated network a set of nodes emulate real nodes.

Thanks to the kernel-reentering methodology, NCTUns provides useful and unique advantages that make it appropriate for the emulation of GeoNet implementations. Some of these characteristics are listed in what follows:

- NCTUns supports real network application programs as traffic generators. Therefore, in this work we have integrated C2CNet daemon as application.
- It directly uses Linux TCP/IP protocol stacks which can generate more accurate results.
- It supports IEEE 802.11p.
- It supports nodes mobility, which is important when simulating VANETs.
- The default protocol modules can be modified, furthermore a new protocol module can be created by a user and can be added into the NCTUns.
- Multiple real application programs can be running simultaneously in a simulated network on one single machine.
- It offers many valuable APIs that are useful and helpful for developers.
- It provides a professional GUI environment which makes work on it easy
- NCTUns is a Linux-based and open source software, which means, a user may modify and/or improve it.

4. Emulation Environment Set-up With NCTUns

4.1 Running C2CNet Layer in NCTUns

4.1.1 Adaptation of NCTUns

In order to integrate the C2CNet layer into NCTUns, several restrictions should be taken into consideration:

- both entities have their own complete implementation and interfaces,
- both entities require configuration.

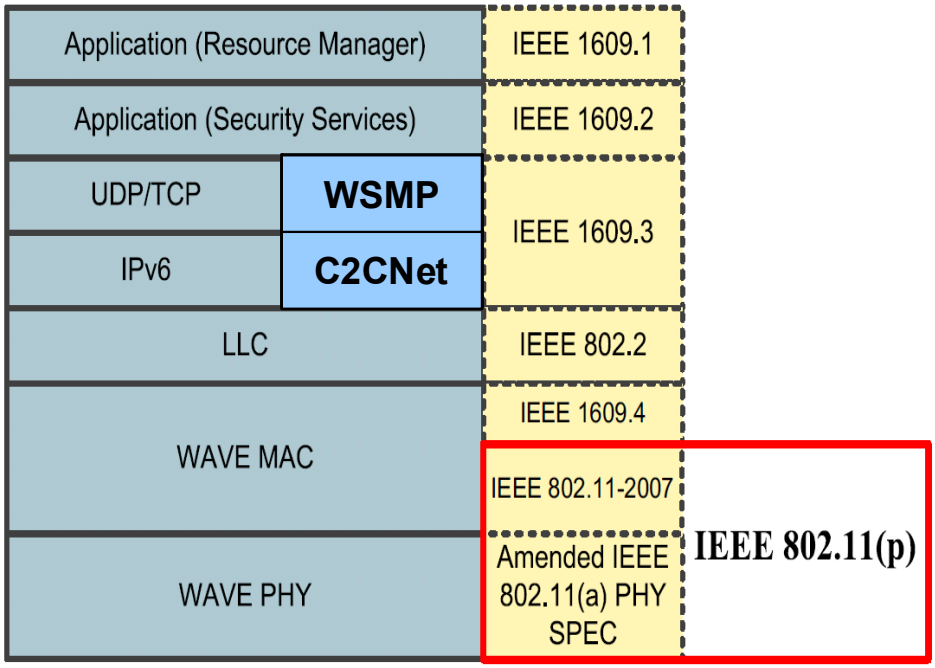

Figure 2: NCTUns Protocol Stack

In order to avoid many changes or adaptations in NCTUns, we have integrated the GeoNet implementation of C2CNet as a user-space traffic generating application. The existing protocol stack of NCTUns allows us to integrate C2CNet in parallel to the existing WSMP protocol and on top of the 802.11p (see Figure 2). By doing it in this way, NCTUns uses TCP/UDP sockets via tunnel interfaces to communicate with its traffic-generating programs. Hence we have realised C2CNet similarly to the way WSMP is implemented.

Alternatively we have also implemented C2CNet over 802.11b as the latter is better supported in NCTUns.

C2CNet runs on virtual GeoNet nodes that are above IP layer (see Figure 3). The C2CNet packets are encapsulated in NCTUns UDP packets. Upon receiving them, NCTUns will treat C2CNet as a traffic generator application and would not require major changes to do so. Furthermore, NCTUns decapsulates received UDP packets and obtains the C2CNet packet. It passes it through its lower protocol layer (802.11p or 802.11b) and finally sends it to the simulated physical interface. Similarly, the packet is received by 802.11p/802.11b at destination, which decapsulates the lower layer headers and transmits the contain to the traffic application program (i.e. C2CNet at destination).

In this way NCTUns is responsible for the lower layers while C2CNet is in charge of the upper protocol layers (Figure 3).

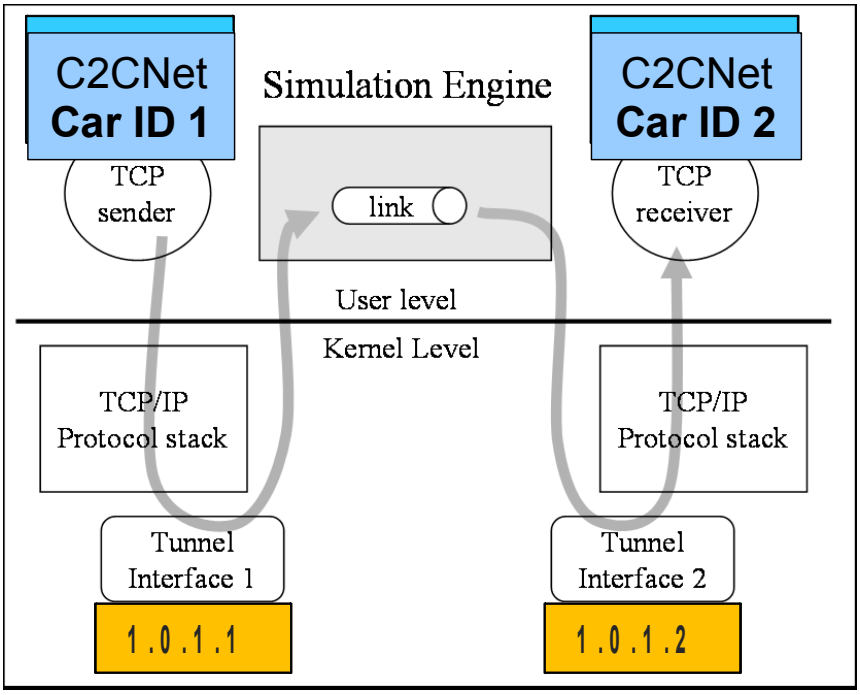

Figure 3: C2CNet on top of NCTUns

4.1.2 Position Update Implementation

In C2CNet the position update is provided by a GPS device. But for simulation we would need to provide a position update from the NCTUns to C2CNet. Additional problem is that the emulation tool uses Cartesian coordinates while C2CNet uses geodesic. Hence it is needed to implement a position update in NCTUns which sends position data in geodesic

coordinates. The emulation tool is adapted with a new position update function to be sent to C2CNet and a position conversion function.

In order to send the position update a socket is opened similarly as it is done for the Position Sensor (for simplicity the same port number should be used as well) [GeoNetD3.1, GeoNetD2.2]. In this way the emulation tool fully replaces the Position Sensor update (see Figure 4). The only difference is the position sensor socket of each node has to be bound to its IPv4 address assigned by NCTUns (instead of connecting to a localhost address as before).

Implementation guidelines for 802.11p

1. Add a position update function in class phy_80211p. First get the location by using the GetNodeLoc() function defined in /NCTUns_5.0/src/nctuns/nctuns_api.cc. Then convert the received position values, create a packet and open a socket in order to send them.

2. Add a scheduler which should call the position update function in int phy_80211p::init().

Implementation guidelines for 802.11b: Do the same but in awphy.cc and awphy.h or wphy.cc and wphy.h files in the /NCTUns_5.0/src/nctuns/module/phy/ directory.

4.1.3 802.11p Modification

NCTUns 5.0 being used for emulation has newly introduced 802.11p. We have discovered that the standard is not fully supported hence it needs some small modifications in order to run smoothly: we need to set the priority of the C2CNet packet correctly upon sending it in mac_80211p, otherwise the packet is never sent out.

Implementation guidelines:

- In /NCTUns_5.0/src/nctuns/module/80211p/mac/mac_80211p.cc modify the mac_80211p::send(ePacket_ *pkt) function.

- Add a flag for C2CNet packets and set their priority to high by modifying the value of “ac”. The priority levels are defined in /NCTUns_5.0/src/nctuns/module/80211p/mac/mac_80211p.h as follows:

```
#define AC_BE   0
#define AC_BK   1
#define AC_VI   2
#define AC_VO   3
```

No changes are required for 802.11b.

4.1.4 Simulation Speed

C2CNet currently uses the Linux clock. NCTUns also uses the Linux time and updates it during its simulation run. Nevertheless we have discovered that when running a simulation the clock is advancing too fast for C2CNet to follow. For that reason the simulation speed has to be set to real speed from the NCTUns toolbar..

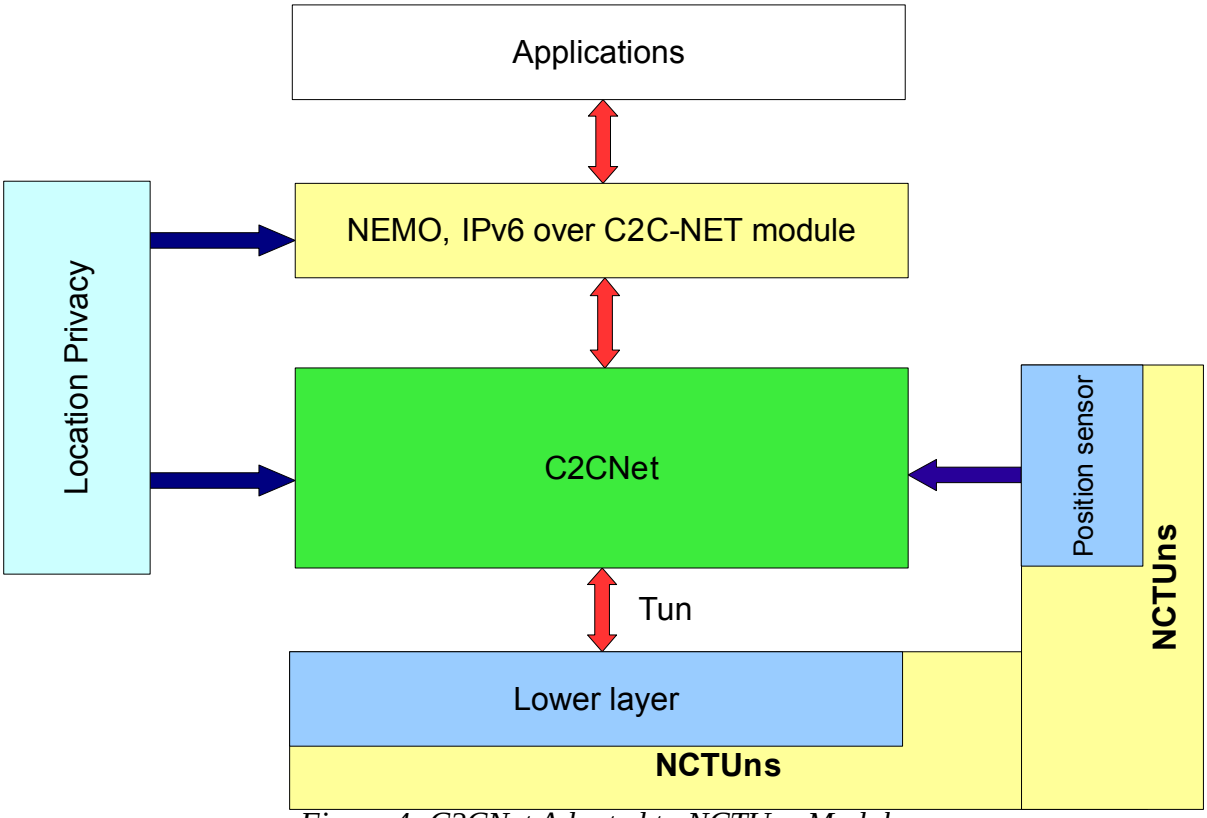

Figure 4: C2CNet Adapted to NCTUns Module

4.1.5 Adaptation and Configuration of C2CNet Layer Implementation

C2CNet needs to be adapted for usage with NCTUns. For that reason we have listed the required changes as indicated below:

Implementation of two new sockets for communication with the emulation tool: One socket is needed for sending and another for receiving packets. The tun interfaces are opened by NCTUns for each simulated node (the IPv4 addresses are already assigned when creating a scenario). The sending socket has to be bind to the node's IPv4 address and a port number while the receiving socket needs to be bound only to a port number. The usage of one socket is not possible, due to the fact that when we bind to a specific address broadcast messages fail to be delivered . The port numbers have to be specified in the emulation tool as well.

C2CNet node identifiers in the NCTUns: It is vitally important for the C2CNet to select its node identifier correctly. Each node's configuration file has to specify the correct car Id corresponding to the one given in NCTUns. For fast and easy implementation we apply the following: firstly the simulation scenarios will be created in NCTUns where the tool will automatically assign IDs and IPv4 addresses for each simulated node. Then the C2CNet configuration files will be updated correspondingly so that the NCTUns Ids and addresses are used.

Correct IPv4 address for the tunnel interface: In NCTUns each node's tunnel interface address is given as 1.0.1.nodeID (see Figure 4). By knowing the node ID we know the tunnel interface of the simulated node in C2CNet. Upon binding the sending socket for communication with the emulation tool, C2CNet should check its configuration files for the nodeID and bind to the according IPv4 address.

4.1.6 Sample Scenario Configuration

The creation and configuration of a simple ITS scenario is provided below:

1) Create a new scenario in the NCTUns GUI program.

2) Place 3 GeoNet OBUs with 802.11p (or 802.11b) interface.

3) Enter Edit property mode and right click on each GeoNet OBU icons. Then for each GeoNet OBU start the C2CNet daemon by modifying the application properties. Add a new application as follows:

Start time (s)	4 seconds (leave some warming up time for the emulator)
Stop time (s)	400 seconds (example value)
Command	./geonetd /usr/local/nctuns/tools/geonet1.conf

The command designates the name of the binary (which should be placed in /usr/local/nctuns/tools) and the C2CNet configuration file (we have created a separate configuration file for each GeoNet OBU, e.g. geonet1.conf, geonet2.conf etc.). This configuration can be seen later and checked in the scenario_name.tfc file which describes the traffic configuration.

4) Set up the correct node ID in the geonet.conf files. After creating the GeoNet OBUs, NCTUns assigns a node ID and IPv4 packet for each of them which are respectively:

Node 1 with Node1_Interface_LINK_1.ip = 1.0.1.1

Node 2 with Node2_Interface_LINK_1.ip = 1.0.1.2

Node 3 with Node3_Interface_LINK_1.ip = 1.0.1.3

Those values can be found in the scenario_name.tcl file.

5) Finally enter G_Setting/Simulation/Speed and choose „As fast as the real world clock“ (for reasons given in 4.2.1)

4.2 Connecting External Nodes in NCTUns

NCTUns provides an emulation feature which allows a simulation node to emulate an external real-world host. WP5 has investigated this emulation feature and tested it, and Figure 5 shows a basic scenario that has been used to test this NCTUns feature.

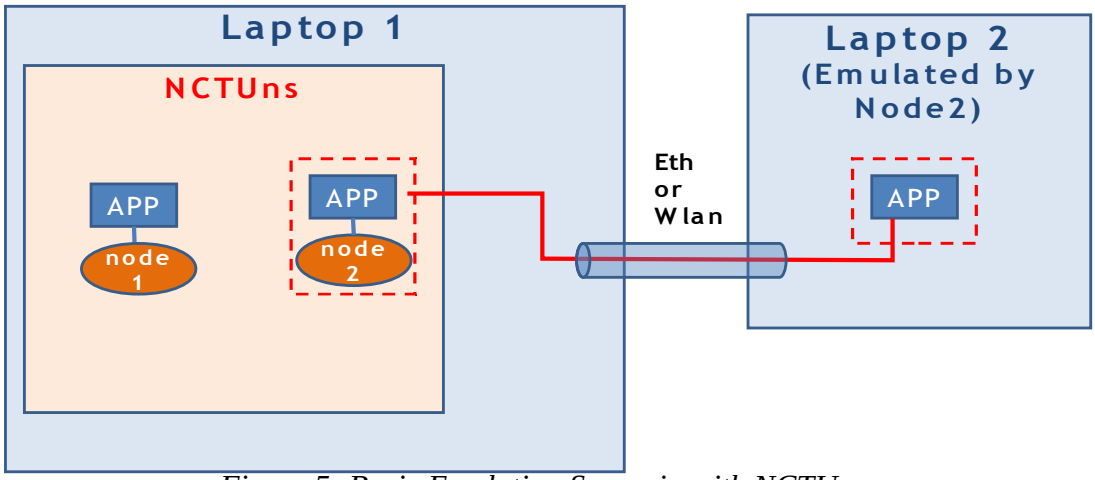

Figure 5: Basic Emulation Scenario with NCTUns.

This scenario consists of two computers (Laptop 1 and Laptop 2). Laptop 1 is used to run NCTUns and Laptop 2 hosts a GeoNet implementation and represents a real car.

In NCTUns running on Laptop 1, we simulate two nodes (node 1 and node 2). Node 2 emulates the real car (Laptop 2).

In the following we explain step by step the procedure to set up interactions between real car and emulation environment.

4.2.1 Setup Description

Creation of basic scenario on Laptop 1: As shown in Figure 6, a basic scenario can be created in NCTUns by selecting a simulation node (node 1 - IP: 1.0.1.1) and an NCTUns 'emulation capable' node (node 2 - IP: 1.0.1.2.

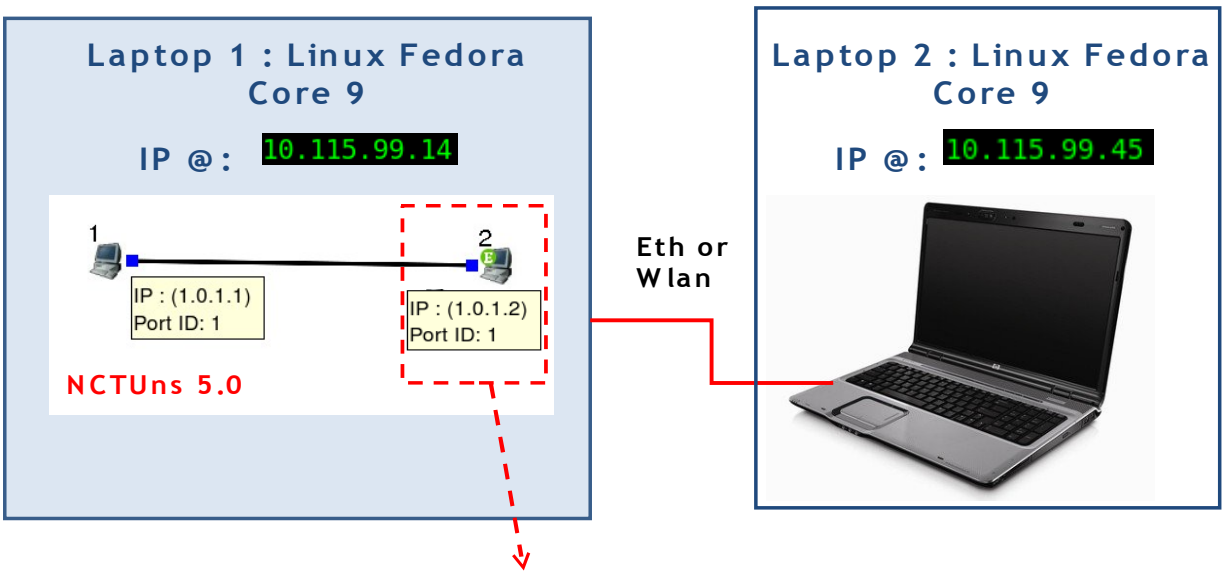

Figure 6: Basic Emulation Scenario Setup

The objective now is to configure the emulation node (node 2 on Laptop 1) to emulate the external node (Laptop 2) which is connected to Laptop 1 through WLAN or Ethernet interfaces.

4.2.2 Emulation Configuration

Emulation node configuration in NCTUns: Node 2 on Laptop 1 should be configured to emulate the external node Laptop 2, and that by setting up the IP address of the real node (here it is Laptop 2 with IP address 10.115.99.45).

Figure 7 shows how to set up and configure an emulation node in NCTUns. In NCTUns there is a module called Dispatcher which takes care of interaction between the real node (Laptop 2) and its corresponding emulation node (node 2).

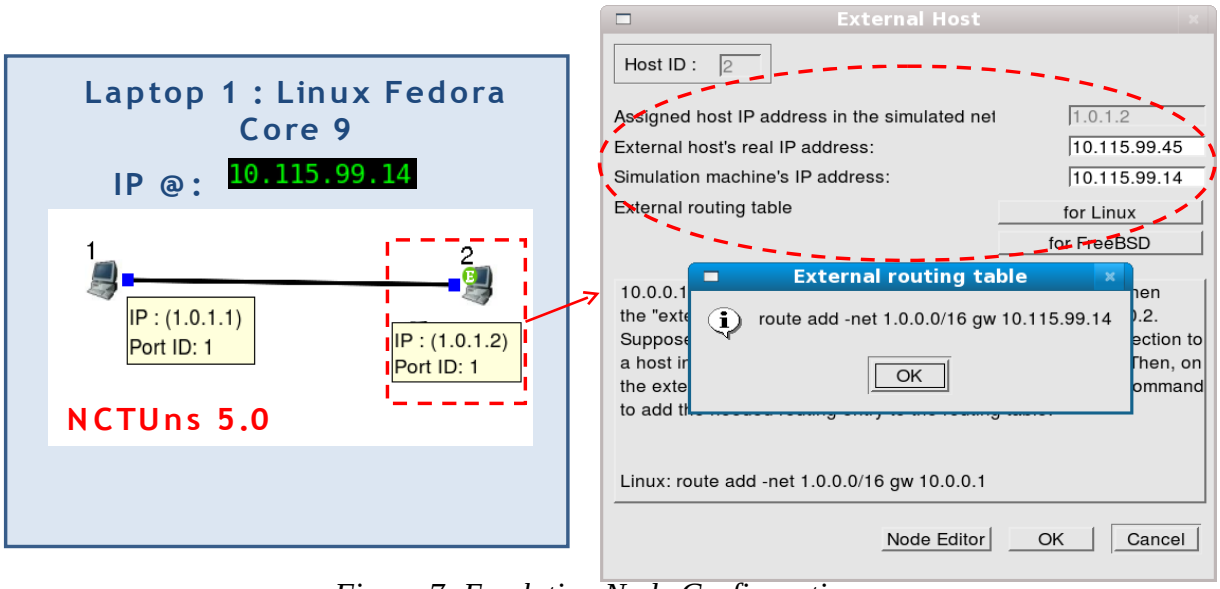

Figure 7: Emulation Node Configuration

Configuration of real node (Laptop 2): On Laptop 2, it is only IP routing table which needs to be updated as shown in Figure 8. The routing table on the real node (Laptop 2) has to be updated by adding the route to the machine which hosts the emulation environment (i.e. IP address of Laptop 2).

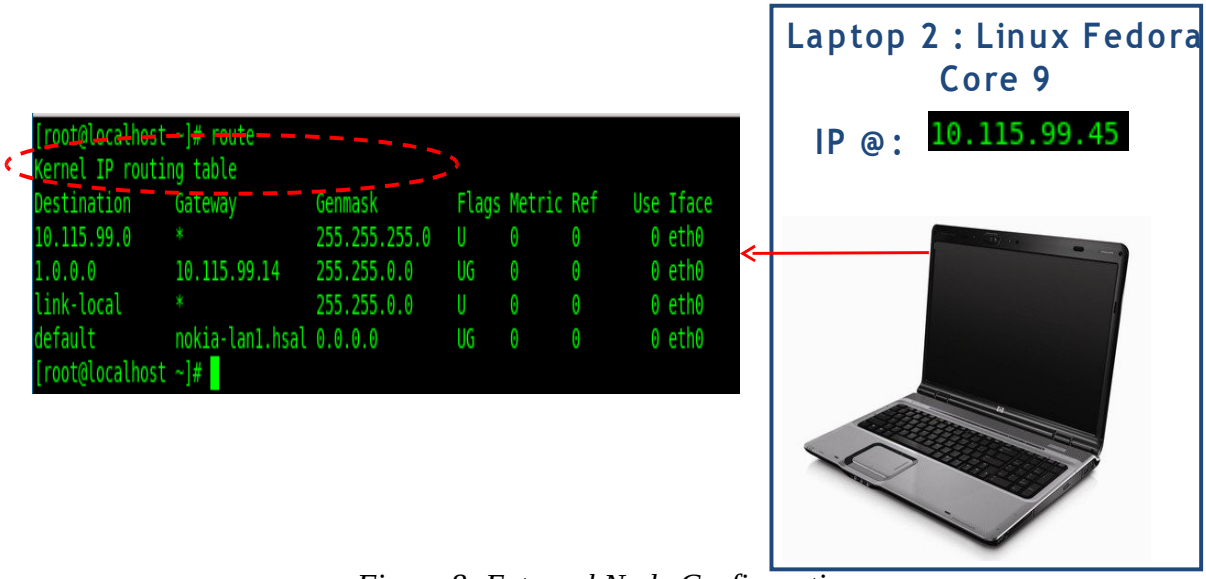

Figure 8: External Node Configuration

In Linux, when considering the current set up, the following command line needs to be executed at laptop 2: *route add -net 1.0.0.0/16 gw 10.115.99.14*

Note that “10.115.99.14” is the IP address of the machine which host NCTUns (i.e. IP address of Laptop 1).

Testing External Node Connection: As shown in Figure 9, the external node (of IP address: 10.115.99.45) is emulated by an emulation node (of IP address: 1.0.1.2). Thus, the emulation node should react as the external node. If the external node ping a node of address 1.0.1.1, then the emulation node ping the simulated node of IP address 1.0.1.1.

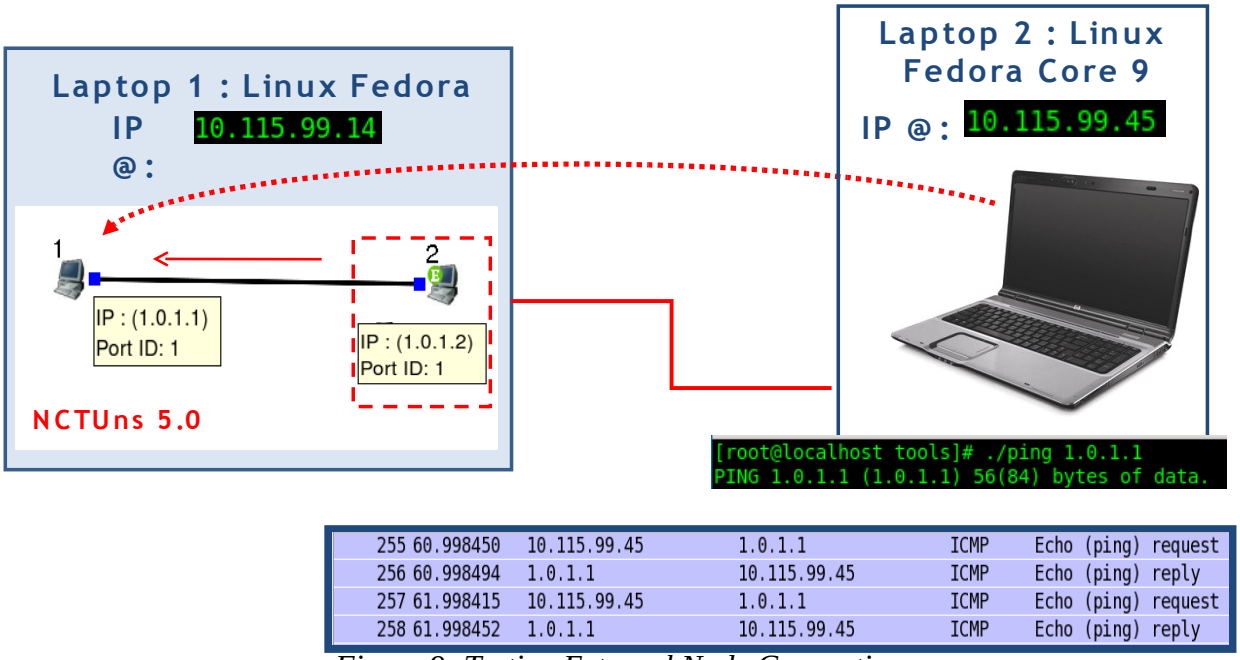

Figure 9: Testing External Node Connection

We have tested this, and when making ping 1.0.1.1 on the external node, the simulated node receives ping packet from emulation node 1.0.1.2 on behalf of the external node. Then, the node with IP address 1.0.1.1 reply to the ping by sending a reply packet to emulation node 1.0.1.2, and in the same time the external node receives the reply from 1.0.1.1, thanks to the NCTUns dispatcher.

4.2.3 Limitations and Technical Issues with Emulation in NCTUns

Initially the idea was to simulate several nodes (20 to 50 nodes) on Laptop 1, and on top of each node attaching a GeoNet C2CNet daemon as application. Then, we prepare two or three real cars (real nodes) which we emulate in NCTUns (on Laptop1) by connecting them to their corresponding emulation nodes (among the simulated nodes) .

NCTUns provides an emulation feature through a dispatcher software. This dispatcher makes a simulation node reacting in the same way as a real node (the emulated node). When the real node sends a packet X on the air, then its corresponding emulation node reacts in the same way and sends also the same packet X. And, when the emulation node receives a packet Y, then the emulated node (real node) receives also the same packet Y.

From theoretical point of view this configuration should work, thanks to NCTUns dispatcher. But unfortunately, during our tests we faced some technical problems with this

dispatcher. We discovered that it suffers when number of simulated nodes is relatively high.

We did some tests with five simulated nodes and one emulation node (emulating one real node). When running the simulation we noticed that some times the dispatcher works in one direction only (from real to emulation, or from emulation to real node) while sometimes it does not work at all (i.e. dispatcher does not guarantee the interaction between the emulation and the emulated nodes). We discovered that the Dispatcher of NCTUns sometimes changes the destination address of packets when routing them between the emulation and the emulated nodes.

This limitation made the usage of the NCTUns emulation feature not possible any more.

5. Emulation and Results Analysis

5.1 Objectives and Evaluation Metrics

With the IPv6 over C2CNet functionality, the C2CNet layer provides the IP layer with IPv6 communication links through the GeoNet domain. We have tested the quality of these IPv6 communication links by means of simulations and here in this section we present and analyse the results we have obtained.

The C2CNet layer should guarantee a certain level of quality of service to IPv6 layer, such as the *Round Trip Time* (RTT) and the *Packet Delivery Ratio* (PDR). Here in this section we aim at evaluating mainly these two metrics.

The evaluation of RTT at the IPv6 layer corresponds to the evaluation of the end-to-end communication delay at the C2CNet layer. The evaluation of the RTT that the C2CNet layer could provide to the IPv6 layer is done by the evaluation of the end-to-end delay between the time when C2CNet receives the IPv6 packet from the IP layer at the source node, and the time when the C2CNet layer delivers that IPv6 packet to the IP layer at the receiver node.

For what concerns the evaluation of PDR at the IPv6 layer, it corresponds exactly to the evaluation of PDR at the C2CNet layer. The PDR corresponds to the ratio between packets sent from the source node(s) and packets received at the destination node(s). We compare the number of IPv6 packets received from IP layer at the source node(s) with the number of IPv6 packets delivered to the IP layer at the destination node(s).

Therefore, we have evaluated both End-to-End Delay and PDR average at C2CNet layer. In the rest of this section we present the scenarios and configuration we have considered in our simulations, and then we present and analyse the results we obtained.

5.2 Scenario Configuration and Parameters

We have chosen a scenario which simulates seven GeoNet OBUs, equipped with 802.11b at the lower layer. The OBUs are distributed as shown in Figure 10.

Initially we intended to use 802.11p at the lower layer, but due to poor performances and not full support of the 802.11p standard by NCTUns, we have switched to 802.11b. When running simulations with 802.11p we observed frequent loss of connectivity and therefore unstable simulation results. We believe that this problem is due to the way NCTUns implements the 802.11p physical interface, which we were unable to investigate further due to lack of time and limited information on the matter (NCTUns forum does not supply sufficient information). For the reasons stated above we continued the simulation using 802.11b interface instead of 802.11p.

The simulated GeoNet OBUs are distributed such as each simulated GeoNet OBU has no more than two neighbours (in communication range) when testing under a GeoUnicast scenario. This layout is chosen to allow testing multi-hop with up to 6 hops. When testing with GeoBroadcast, simulated GeoNet OBUs are distributed more densely , thus having up to three or four maximum hops in total. In this case we would like to evaluate the performances of the IPv6 over the C2CNet link (see terminology in [GeoNetD1.2] while the network is flooded with GeoBroadcast packets.

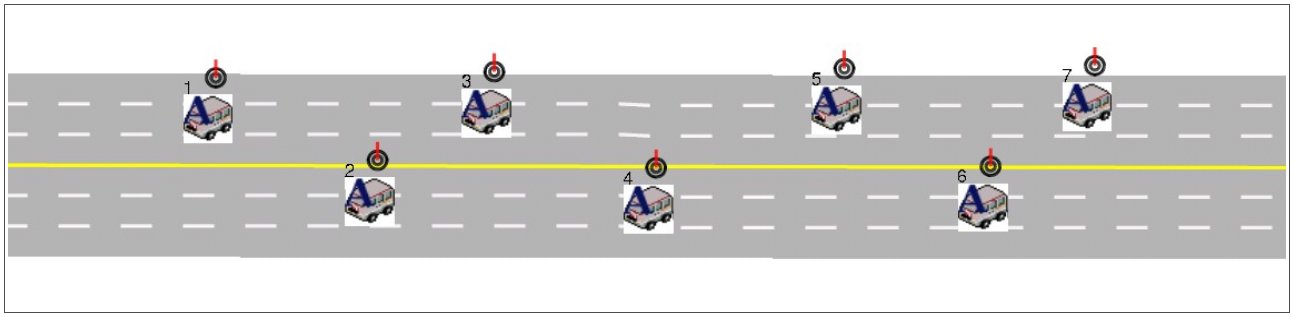

Figure 10: Scenario with seven simulated GeoNet OBUs.

The relevant configuration parameters for our simulation scenarios are given in a configuration file (Scenario.tcl) with the following settings:

- simulation speed:
  Set SimSpeed = AS_FAST_AS_REAL_CLOCK)

- NCTUns generates IPv4 addresses for each simulated GeoNet OBU as follow:

```
Create Node 1 as CAR_ADHOC with name = CAR_ADHOC1
        Module Interface : Node1_Interface_LINK_1
                Set Node1_Interface_LINK_1.guitag_ordinary = yes
                Set Node1_Interface_LINK_1.ip = 1.0.1.1
                Set Node1_Interface_LINK_1.netmask = 255.255.255.0

Create Node 2 as CAR_ADHOC with name = CAR_ADHOC2
        Module Interface : Node2_Interface_LINK_1
                Set Node2_Interface_LINK_1.guitag_ordinary = yes
                Set Node2_Interface_LINK_1.ip = 1.0.1.2
                Set Node2_Interface_LINK_1.netmask = 255.255.255.0
```

- The OSI stack and required NCTUns modules for a 802.11b GeoNet OBU are specified in the following lines:

```
Bind Node1_Interface_LINK_1 Node1_AODV_LINK_1
Bind Node1_AODV_LINK_1 Node1_ARP_LINK_1
Bind Node1_ARP_LINK_1 Node1_FIFO_LINK_1
Bind Node1_FIFO_LINK_1 Node1_MNode_LINK_1
Bind Node1_MNode_LINK_1 Node1_MAC80211_LINK_1
Bind Node1_MAC80211_LINK_1 Node1_WTCPDUMP_LINK_1
Bind Node1_WTCPDUMP_LINK_1 Node1_Wphy_LINK_1
Bind Node1_Wphy_LINK_1 Node1_CM_LINK_1
Bind Node1_CM_LINK_1 Node1_LINK_1
```

In order to start the operation of the C2CNet layer in NCTUns, we need to modify the NCTUns traffic generator file (Scenario.tfc). Thus, the following lines have to be added:

```
#nctuns traffic generator file
$node_(1)          9.800000          400.000000          ./geonetd          /usr/local/demo/11b/geonet1.conf
/usr/local/demo/11b/geonet.log
$node_(2)          9.400000          400.000000          ./geonetd          /usr/local/demo/11b/geonet2.conf
/usr/local/demo/11b/geonet2.log
```

With the above we define for GeoNet OBU 1 the related C2CNet daemon, called *geonetd,* which is started as a traffic generator application on top of the related GeoNet OBU. The C2CNet layer daemon starts at the second 9 and stops at second 400.

Other NCTUns parameters that need to be specified include: (1) position update interval should be set to 1 second, and (2) simulation time should vary between 100 and 300 seconds.

The remaining scenario parameters are given in the configuration file of the C2CNet layer daemon (*geonet.conf).*

Each simulated GeoNet OBU gets its position information from the position update function as provided in NCTUns.

The own C2CNet ID is used by the GeoNet OBU to identify whether packets are designated to it or not, as well as to identify itself when sending out packets.

```
# Own Node Identifier
ownNodeId=1
```

In order for a simulated GeoNet OBU to send GeoUnicast packets, it needs to know the C2CNet ID of the destination. In our simulation we set the destination C2CNet ID as follows:

```
# Destination ID for Unicast Message
destNodeId=2 … 6
```

C2CNet needs to periodically send out beacons in order to keep connectivity with its neighbours. The following entry allows to set up the interval time between each two C2CNet beacon transmissions:

```
# Enable/disable beacon
# valid values is true or false
sendBeacon=true

# Set beacon interval [in ns]
beaconInterval=2000000
```

GeoUnicast transmission is enabled and its time interval is set as shown below.

```
# Enable unicast

# valid value is true or false
sendUnicast=true

# Set unicast interval [in ns]
unicastInterval=1000000
```

GeoBroadcast transmission is enabled and its time interval is set as shown below:

```
# Enable GeoBroadcast circle
# valid value is true or false
sendGBCCircle=false

# Set GeoBroadcast interval [in ns]
gbcCircleInterval=1000000
```

As previously explained, we concentrate our simulation on two set of scenarios:

1. **GeoUnicast:** sending dummy data through C2CNet by using GeoUnicast from GeoNet OBU of node ID 1 to GeoNet OBUs of node IDs 2, 3, 4, 5 and 6 respectively. The beacon transmission interval is set to 2 seconds and GeoUnicast transmission interval is varied between 2 ms and 1 s. The size of the dummy payload data is also varied and evaluation is performed for 0 and 1000 bytes of payload.

2. **GeoBroadcast:** sending dummy data through C2CNe by using GeoBroadcast from GeoNet OBU of node ID 1 to all nodes belonging to a GeoDestination of several thousand meters. The beacon transmission interval is set to 2 seconds and GeoBroadcast transmission interval is varied between 1ms and 1 s. The size of the dummy payload data is also varied and evaluation is performed for 500 and 1000 bytes of payload.

5.3 Results and Analysis

This subsection presents some preliminary simulation results, which are obtained by 100 simulation runs with 90% confidence interval. The metrics studied are the following:

Packet Delivery Ratio (PRD): is defined as the number of correctly received packets at the destination over the number of packets sent by the source.

End to End Delay: is defined as the duration of time that a packet takes to travel from the source to destination.

Figure 11 shows the end-to-end delay (in ms) obtained at each intermediate node (hop) when fixing the unicast packet generation to one packet per second (1/s). In this test we set the size of the IPv6 packet to null. This does not correspond to reality but the target

from this first test is to simply show that in multi-hop wireless communications the end-to-end delay increases when increasing the distance (in terms of hops) from the source node. As shown in Figure 11, the end-to-end delay in general increases when increasing the number of intermediate hops between the source and the destination nodes. When comparing the end-to-end delay at hop 3 and hop 4, we do not see any increase, we even see some small decrease. This is mainly due to the fact that we have less samples at hop 4 when compared to hop 3, which is due to more packets loss at hop 4.

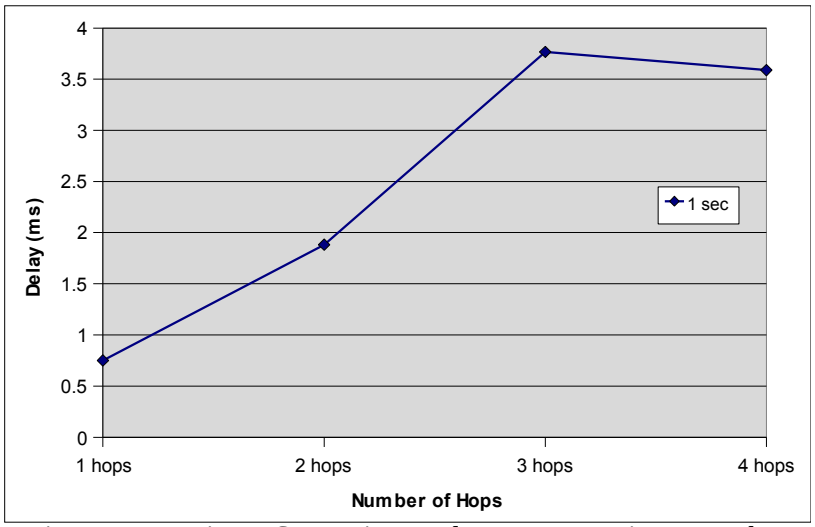

Figure 11: Unicast Scenario - Delay vs. Hop Distance when IPv6 Packet Size set to Null.

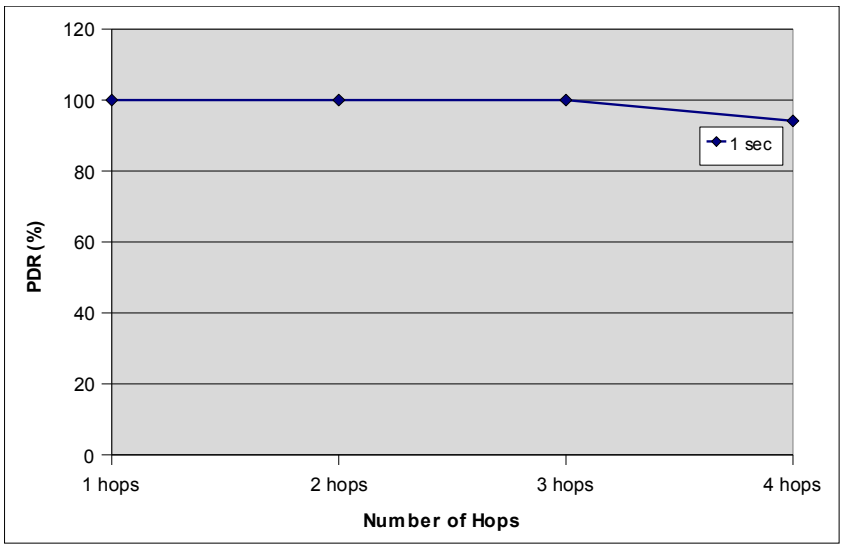

Figure 12: Unicast Scenario – PDR vs. Hop Distance when IPv6 Packet Size set to 1000 Bytes.

In the same scenario, i.e. when the size of the attached IPv6 packet is set to null, we also computed the Packet Delivery Ratio (PDR) at the destination and corresponding results

are shown in Figure 12. In this figure, the plot shows that 100% of packets sent from the source node are received at hop 1, 2 and 3. We start to loss packets from 4th hop.

In the two previous figures we considered a simple communication scenario. We considered only communication at the C2CNet layer without attaching any payload (IPv6 packet's size equal to null), and we considered only GeoUnicast scenario with transmission interval set 1 second only i.e. only one transmission every one second.

In the rest of this section, more complete scenarios are considered. We vary the size of the payload (i.e. size of attached IPv6 packets) between 0, 500 and 1000 bytes and then observe to understand the impact of IPv6 packet's size on the end-to-end delay and PDR. At the same time we vary the time interval of IPv6 packets' transmission between 1 ms, 10 ms, 100 ms, 500 ms and 1000 ms. This time we consider not only GeoUnicast scenarios, but also GeoBroadcast scenarios.

Figure 13 shows the end-to-end delay in the GeoUnicast scenario. Two plots are shown, one corresponds to tests where IPv6 packet's size is set to null, and another one corresponds to tests where IPv6 packet's size is set to 1000 bytes. As it can be seen, the two plots show that end-to-end delay increases when increasing the size of the transmitted packet (i.e. size of attached IPv6 packet, which is logical and expected in this kind of communication.

What is more interesting to read through these two plots is the fact that the end-to-end delay is almost under 10 ms when time interval between transmissions does not go bellow 10ms. Which means the end-to-end delay does not go upper than 10ms when the number of generated IPv6 packets does not exceed 100 packets per second.

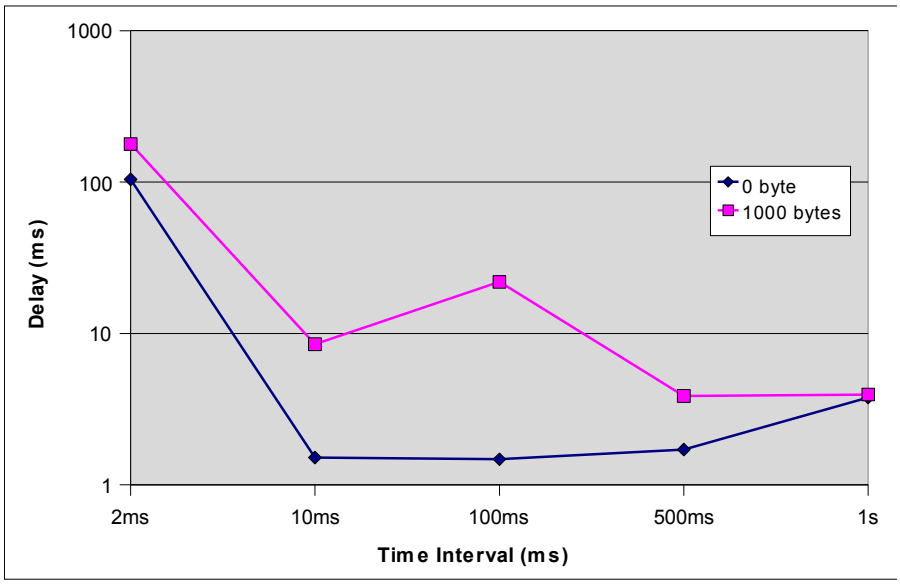

Figure 13: Unicast Scenario – Delay vs. Transmission Time Interval.

Considering an IPv6 packet size of 1000 bytes and a transmission frequency of 100 packet per second, the pinky plots in Figure 13 shows that **C2CNet can provide to IP layer a communication throughput of around 0.1 Mbps**.

Figure 14 shows results we got when running tests under the GeoBroadcast scenario, where we aimed at evaluating end-to-end delay at each intermediate node (hop) from source to destination.

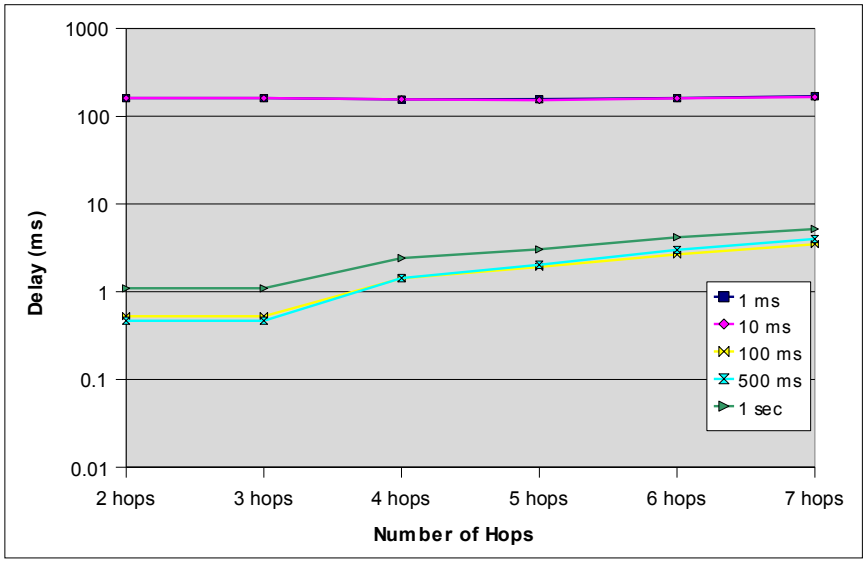

Figure 14: GeoBroadcast Scenario – Delay vs. Hop Distance when IPv6 Packet Size set to 500 Bytes.

Figure 14 presents five plots corresponding to five different inter-transmission interval time respectively. For example, the blue plot corresponds to tests where inter-transmission interval time is set to 1 ms, which means one IPv6 packet is generated and processed every 1 ms (1000 packets per second).

Considering the three plots corresponding to inter-transmission interval time of 1 sec, 500 ms and 100 ms respectively, we notice that when varying the inter-transmissions interval time, the end-to-end delay does not exceed one millisecond until third hop. And, starting from fourth hop, the end-to-end delay starts to increase slowly but without exceeding 10 ms.

Now, looking to the two last plots (corresponding to 10 ms and 1 ms respectively) we notice that end-to-end delay goes over 100 ms but in general keeps stable. Having the end-to-end delay being the same at all intermediate nodes is due to buffering delay. The traffic generated at C2CNet layer, when inter-transmissions interval time is small, increases too much and very fast. Thus, huge number of packets will be buffered to be processed after some delay (says Buffering delay). The end-to-end communication delay is negligible when compared to buffering delay.

Same behaviour is shown in Figure 15, where end-to-end delay keeps high and stable when the size of IPv6 packets is set to 1000 bytes, and inter-transmission interval time is

set 10 ms or less. In this case the end-to-end delay is slightly decreasing for a high data rate and high number of hops (e.g. 1 and 10ms over 6-7 hops). This occurs due the fact that we are experiencing more packet losses and the impact of the buffering delay becomes slightly smaller.

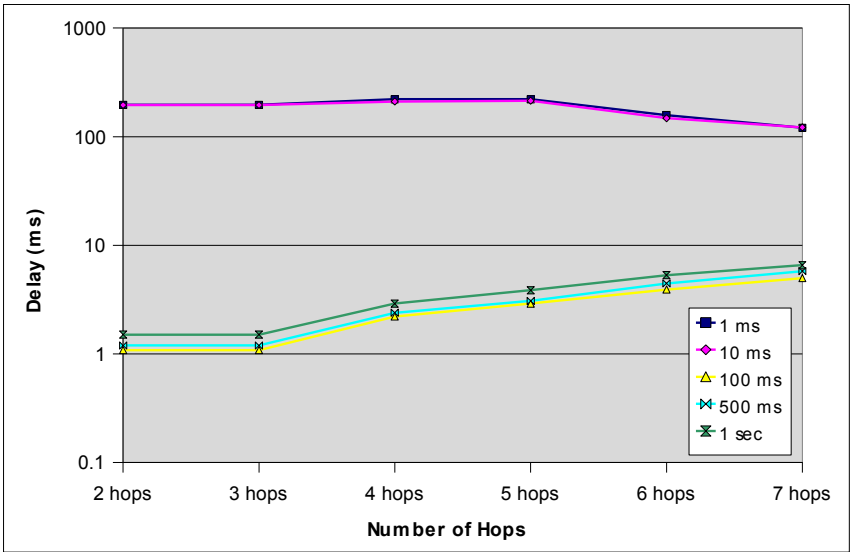

Figure 15: GeoBroadcast Scenario – Delay vs. Hop Distance when IPv6 Packet Size set to 1000 Bytes.

Both Figure 16 and Figure 17 show PDR as obtained when running tests in the same scenario GeoBroadcast. Figure 16 corresponds to tests where IPv6 packet's size is set to 500 bytes, and Figure 17 corresponds to tests where IPv6 packet's size is set to 1000 bytes. The three plots corresponding to inter-transmission interval time 1 s, 500 ms and

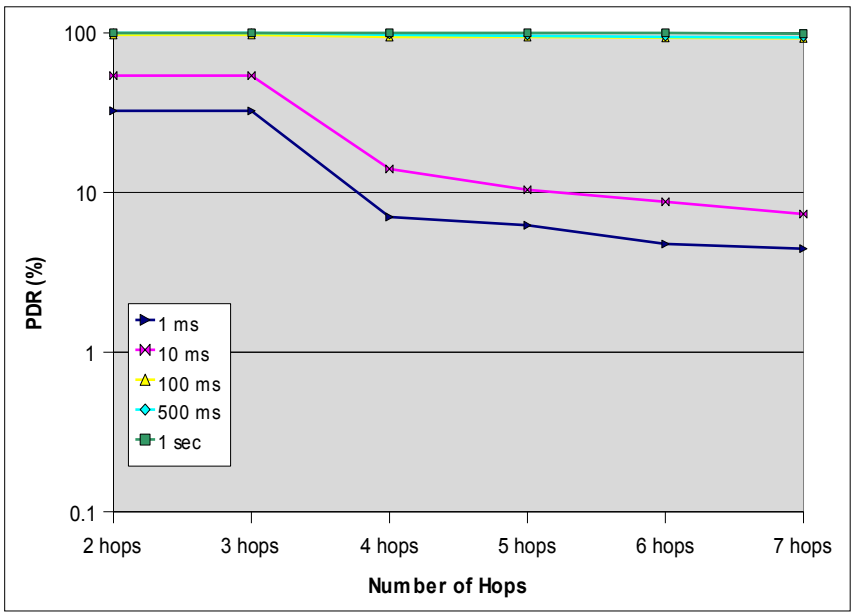

Figure 16: GeoBroadcast Scenario – PDR vs. Hop Distance when IPv6 Packet Size set to 500 Bytes.

100 ms respectively, show that almost 100% of packets are correctly delivered by C2CNet to destination nodes up to the 7th hop.

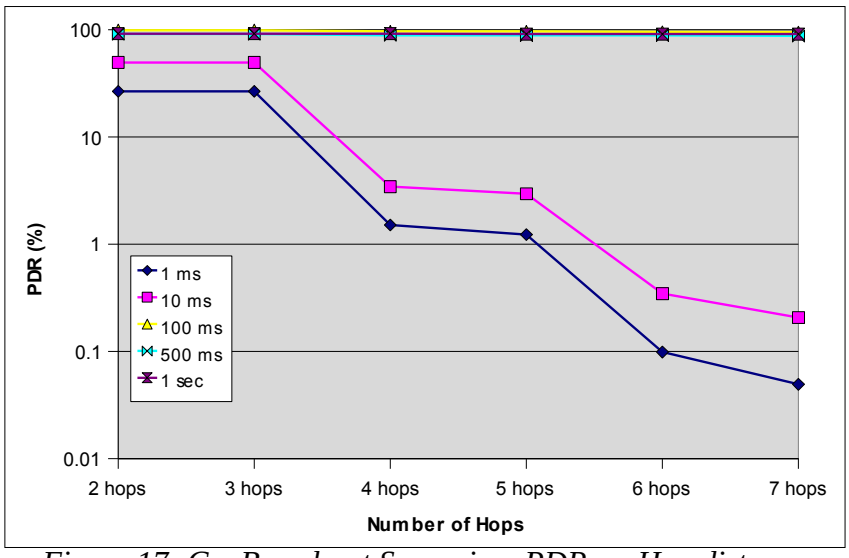

Figure 17: GeoBroadcast Scenario – PDR vs. Hop distance when IPv6 Packet Size set to 1000 Bytes.

When looking at two other plots, corresponding to 1 ms and 10 ms of inter-transmission interval time respectively, as shown in Figure 16, we see that PDR is about 50% when inter-transmission interval time is set to 10 ms and about 30% when inter-transmission interval time is set to 1 ms. Up to 3rd hop, calculated PDR is almost the same when IPv6 packets' size is set 1000 bytes as shown in Figure 17.

Starting from the 4th hop, PDR goes bellow 15% when the size of attached IPv6 packet is set to between 500 and 1000 bytes, but it is worst when IPv6 packet's size if set to 1000 bytes (i.e. we have almost 100% of losses when IPv6 packet size is 1000 bytes).

Considering PDR results as shown in Figure 16 and Figure 17, and end-to-end delay results as shown in Figure 14 and 15, we conclude that C2CNet can provide the IP layer with communication links of a throughput up to 10Kbps (1000 bytes x 10) with a end-to-end delay of less than 1 ms at second and third hop, and with a end-to-end delay of up to 10 ms at hops 4 to 7.

The same results show that C2CNet can also provide the IP layer with C2CNet links of up to 500Kbps (1000 x 1000 / 2) throughput, with around 100 ms of end-to-end delay.

The end-to-end delay and PDR results which we obtained in these emulation evaluation matches with the results we got during the experimentation evaluation as reported in [GeoNetD7.1], even if they do not look so when looking to Table 1. This table compares the end-to-end delay as calculated in the emulation evaluation (WP5) with the Round Trip Time (RTT) as calculated in the experimentation evaluation (WP7).

	Delay in WP5	**RTT in WP7**
1 hop	1 ms	2.5 ms
2 hops	2 ms	5 ms
3 hops	3.5 ms	N/A

Table 1: End-to-end Delay Difference between WP5 and WP7 Results.

The end-to-end delay is lower in the WP5 emulation results when compared to WP7 experimental results presented in [GeoNetD7.1]. This is due to the tunnel interface tun0 processing time which is not included in the emulation environment, and included in the experimental environment. Using tun0 is processing time consuming because the packet exchanges between user and kernel spaces.

PDR emulation results can not be compared to PDR experimental results because the two are based on different test scenarios. In the experimentation, the GeoUnicast scenario has been tested while the GeoBroadcast scenario has been tested in the emulation.

6. Conclusion

This deliverable presented the work performed in GeoNet Work Package 5 *Emulation Environment Development*. This work package aimed at preparing an emulation environment to be used for testing and validating the GeoNet C2CNet layer implementations, and the overall IPv6 GeoNetworking specification, in complex scenarios (e.g. high density or dynamic mobility of the vehicles) that are difficult to manage in real field tests (actually in WP7 *Experimental Validation*, see [GeoNetD7.1]).

The well known wireless emulation tool NCTUns has been chosen to perform the emulation tasks within GeoNet project. This emulation tool has shown some unexpected technical limitations, such as the lack of IPv6 support and the instability of the dispatcher function which is a key feature for emulation. This unexpected limitations made some of the initial targets of this work not being achievable, but GeoNet partners have managed to go over this and alternative solutions have been adopted; i.e. virtual emulation of IPv6 transmission over C2CNet has been adopted as the alternative solution.

The emulation environment was used to evaluate the performance of the GeoNet implementations under two metrics: *end-to-end delay* and *packet delivery ratio* that C2CNet layer can offers to the IPv6 layer. In general, the results found show that C2CNet layer can provide the IP layer with IPv6 over C2CNet communication links of up to 500Kbps (0.5Mbps) of throughput. The reader shall report to [GeoNetD7.1] for the test field results.

Annex A: Contributors

The following people have contributed to this GeoNet document, by alphabetical order:

- Thierry Ernst – INRIA
- Maria Goleva – NEC
- Brigitta Lange – NEC
- Massimiliano Lenardi – Hitachi Europe
- Hamid Menouar – Hitachi Europe
- Wenhui Zhang – NEC

Annex B: References

[GeoNetD1.2] GeoNet. “Final GeoNet Architecture Design”. GeoNet Deliverable D1.2, January 2010.

[GeoNetD2.2] GeoNet. “Final GeoNet Specification”. GeoNet Deliverable D2.2, January 2010.

[GeoNetD3.1] GeoNet. “GeoNet Development Results”. GeoNet Deliverable D3.1, January 2010.

[GeoNetD7.1] GeoNet. “GeoNet Experimentation Results” GeoNet Deliverable D7.1, January 2010.

[IMUNES] IMUNES Homepage. http://imunes.tel.fer.hr/

[iTetris] European Project iTETRIS http://*www.ict-itetris.eu*

[MMTS] MMTS. Realistic Vehicular Traces (VANET Traces). http://www.lst.inf.ethz.ch/research/ad-hoc/car-traces/

[MobUML] MobUML Homepage.
http://www.metz.supelec.fr/metz/personnel/galtier/PagesPerso/MobUML/index.html

[NCTUns1] NCTUns. http://nsl10.csie.nctu.edu.tw/

[NCTUns2] The Protocol Developer Manual for the NCTUns 5.0 Network Simulator and Emulator, Network and System Laboratory, Department of Computer Science, National Chiao Tung University, Taiwan, September 1st , 2008

[NCTUns3] The GUI User Manual for the NCTUns 5.0 Network Simulator and Emulator, Network and System Laboratory, Department of Computer Science,National Chiao Tung University, Taiwan, April 2nd, 2009.

[NS2] NS Homepages. http://www.isi.edu/nsnam/ns/.

[OpenVZ] OpenVZ Homepages. http://openvz.org

[OPNET] OPNET. OPENT Modeller Introduction Homepages.
http://www.opnet.com/solutions/network_rd/modeler.html

[PDVC2X] European Project PRE DRIVE C2X http://www.pre-drive-c2x.eu

[SUMO] SUMO Homepages. http://sumo.sourceforge.net

[VISSIM] PTV. VISSIM User Manual. Planung Transport Verkehr AG, Karlsruhe, Germany, version 4.20 edition, November 2006
[VMware] VMware. Virtualization Overview. http://www.vmware.com/index.html

[VNUML] VNUML Homepages. http://www.dit.upm.es/vnumlwiki/index.php