Integration of NS-3 with MATLAB/Simulink

Dmitry Kachan
Integration of ns-3 with MATLAB/Simulink

Dmitry Kachan

Lulea University of Technology
Dept. of Computer Science and Electrical Engineering
Communication Networks Research Group

June, 2010
Simulators are popular in nearly every field of engineering, like vehicle dynamics, network communication, and electronics, only to mention some. When it comes to simulate the behavior of an intelligent transportation system (ITS), these fields suddenly overlap which calls for a combined simulation environment, where vehicular motion together with sensing, electronics and communication need to be simulated. Due to the fact that a future ITS most likely uses wireless communication, this communication becomes the backbone of the ITS.

This work is dedicated to modifying the structure of ns-3 network simulator in order to enable interactions with simulator of dynamic models MATLAB/Simulink. We integrated two simulators using sockets, by establishing a bidirectional communications between the two. To this end, the major functionality of the integrated simulation environment is implemented. The thesis describes integration principles as well as presents the details of the implementation. The thesis also identifies the need for scalability optimization and further extensive testing of the concept.
This work has been carried out between March 2010 and June 2010 at Lulea University of Technology in Lulea (Sweden).

I would like to thank the people at Communication Networks Research Group for helping me with this master thesis. Special thanks to my supervisor, Evgeny Osipov, for helping me all the way, and for sharing his knowledge.

A special thank you goes to all my friends, from Lulea and from Russia, for cheering me up during the bad moments.

And last, but not least, an enormous thank to my family for giving me the chance to study and develop my Master Thesis in Lulea.

Dmitry Kachan
CONTENTS

1. INTRODUCTION .......................................................... 1
   1.1 Simulation environment ............................................... 1
   1.2 Problem area background ............................................. 2
       1.2.1 ITS .......................................................... 2
       1.2.2 Sensor Network ............................................... 2
       1.2.3 iRoad ....................................................... 4
   1.3 Purpose .............................................................. 4
   1.4 Review .............................................................. 4
   1.5 Delimitation ........................................................ 5
   1.6 Thesis Objectives ................................................... 5
   1.7 Technologies used .................................................. 6
       1.7.1 Operating system .............................................. 6
       1.7.2 Network simulator ns-3 ...................................... 6
       1.7.3 Eclipse ..................................................... 7
   1.8 Installation and configure ......................................... 7
       1.8.1 Eclipse installation .......................................... 7
       1.8.2 Configure ns-3 in Eclipse .................................. 8
       1.8.3 Debug settings .............................................. 8

2. METHOD ....................................................................... 9

3. THEORY ...................................................................... 11
   3.1 Socket programming in Linux ........................................ 11
       3.1.1 Socket creating ................................................. 11
       3.1.2 Addresses ...................................................... 12
       3.1.3 Connect (Server) ............................................... 12
       3.1.4 Connect (Client) ............................................... 13
       3.1.5 Data Exchange ................................................ 13
       3.1.6 Socket closing ............................................... 14

4. IMPLEMENTATION AND EVALUATION ................................. 15
   4.1 Implementation ....................................................... 15
       4.1.1 The description of secondary classes ....................... 15
       4.1.2 Simulator implementation ................................... 16
List of Figures

1.1 Overview of main sensor node hardware components .................. 3
2.1 Interaction between ns-3 and MATLAB ............................... 10
4.1 UdpEchoClienNew class ............................................. 20
4.2 UdpEchoServerNew class ............................................. 21
4.3 ExternallyDrivenSim class ............................................. 25
1. Introduction

This thesis contains the description of integration of the combined simulation environment in the context of ITS

1.1 Simulation environment

Designing and launching an intelligent transport system is a complex and difficult task, due to large amount of components with their parameters that need to be considered both during design and launch. Although, there is a clear picture of the requirements and specification of the ITS from an overall perspective down to component level, these specifications rarely capture the complexity which is added by the distributed character of the functionality.

Problems that arise due to this added complexity are usually hard to capture in analytical analysis and thus often neglected. Additionally, the outcome of the analysis is also usually affected by uncertainties, due to simplifications and approximations. In order to achieve a higher level of a-priori certainty on the validity of the functionality, virtual verification through using simulator is very helpful and probably the only feasible tool. Additionally, simulator studies can reveal problems in early stages and thereby speed-up both design and launch.

Simulators are popular in nearly every field of engineering, like vehicle dynamics, network communication, and electronics, only to mention some. Clearly, when it comes to simulate the behavior of an ITS, these fields suddenly overlap which calls for a combined simulation environment, where vehicular motion together with sensing, electronics and communication need to be simulated. Due to the fact that a future ITS most likely uses wireless communication, this communication becomes the backbone of the ITS. Proper analysis is therefore indispensable from perspectives of vulnerability, dependability and performance, especially, as road infrastructure is a harsh environment.
1.2 Problem area background

1.2.1 ITS

ITS – Intelligent Transport Systems – is the integration of information and communications technology with transport infrastructure, vehicles and users. By sharing vital information, ITS allows people to get more from transport networks, in greater safety and with less impact on the environment.

The main priorities of ITS are:

- **SafeMobility**
  Safety is a key issue for European mobility today. More than 40,000 people die on the road each year and road accidents cost the European economy around 200 billion euro every year [1]. Researches - and initial deployment - have shown the great potential for improving road safety with intelligent and advanced and driver assistance systems. These systems can detect dangers on the road ahead, inform drivers of them even before they are visible, keep vehicles at a safe distance from one another and inform drivers of the local conditions.

- **EcoMobility**
  Mobility is a vital component of modern society, but at the same time, mobility is placing increased pressure on our environment.
  ITS applications have a key part to play in making transport greener. They have the potential to reduce energy consumption, improve energy efficiency, cut greenhouse gas emissions and reduce our dependence on fossil fuels.

- **InfoMobility**
  Real time traffic and travel information is the backbone of any intelligent mobility service. Real time, reliable and personalized traffic and travel information is the key to ensure safer, smarter and more efficient transport systems. Context aware services consider the needs, preferences and accumulated travel bonuses of individuals, their environmental signature (CO2 measurements etc), their travel habits etc.
  The data used by ITS applications must be reliable, accurate and continuously available, also across borders.

1.2.2 Sensor Network

A sensor network is a set of small autonomous systems, called sensor nodes which cooperate to solve common applications. Their tasks include some kind of perception of physical parameters.
A basic sensor node comprises five main components [2] (Figure 1.1):

**Controller.** A controller to process all the relevant data, capable of executing arbitrary code.

**Memory.** Some memory to store programs and intermediate data; usually, different types of memory are used for programs and data.

**Sensors and actuators.** The actual interface to the physical world: devices that can observe or control physical parameters of the environment.

**Communication.** Turning nodes into a network requires a device for sending and receiving information over a wireless channel.

**Power supply.** As usually no tethered power supply is available, some form of batteries are necessary to provide energy. Sometimes, some form of recharging by obtaining energy from the environment is available as well (e.g. solar cells).

![Figure 1.1: Overview of main sensor node hardware components](image)

A sensor network normally constitutes a wireless ad-hoc network, meaning that each sensor supports a multi-hop routing algorithm (several nodes may forward data packets to the base station)[3].

These networks can be used in the following applications: disaster relief applications, environment control and biodiversity mapping, intelligent buildings, facility management, machine surveillance and preventive maintenance, precision agriculture, medicine and health, logistics, telematics. All these applications are performed by one of the following tasks, which sensor networks perform: event detection, periodic measurements, function approximation and edge detection, tracking.
1.2.3 iRoad

iRoad project – intelligent road. In limits of this project, with help of MATLAB/Simulink, the project of simulating road traffic was created. In this project the behavior vehicle and vehicle driver are simulated. The goal of iRoad project is to know dependency of behavior what considered above on indication of small nodes situated along the road. These nodes can detect the vehicle, which passes near, and, for example, show the following direction of movement. These nodes are communicated via wireless network. This network is simulated in ns-3. That the question how to synchronize both Simulink and ns-3? The question is how to communicate these two simulators? Because both of them have different relations with time. Also ns-3 has not build-in communication tools.

1.3 Purpose

Having in view all of the above, actual problem is to create a combined simulation environment, which can handle dynamic vehicle motion, wireless communication, sensing, the installation environment and the distributed functionality. Properties from each of these areas are parameters in the CSE and the behavior of the ITS can be simulated by setting up different scenarios.

1.4 Review

There are some works that developed integrated simulators achieving the interaction between traffic and network simulation environment. In [4] authors of TraNS project are using the previously version of ns-3 – ns-2 [5] as network simulator and they using SUMO [6] as traffic model. The communication channel between simulators is set up over a TCP/IP. The TraNS project has GUI and it open source. In [7] authors of simITS are using network simulator ns-3 and road traffic simulator SUMO. They provide an interface ensuring the interaction between the simulators and new ns-3 mobility model. Also they implement a new communication stack in ns-3 consistent with current draft of ETSI/ITS to use it instead of existing TCP/IP. The TraNS and simITS projects are used TraCI [8] interface to communicate with SUMO. TraCI uses a TCP based client/server architecture to provide access to SUMO. In [9] the authors using ns-3 as network simulator, SUMO as vehicular traffic simulator and the third and central block is the iTETRIS Control System (iCS) which apart from providing a coupling and controlling entity between SUMO and ns-3, provides a user interface to the iTETRIS platform for the application development.
1.5 Delimitation

Purpose of this work is not creation of a realistic model in the available simulator. Originally the goal consists to combine two different simulation environment and achieve the agreed work both of them. These are the limitations for this thesis:

1. There are many different simulators both dynamic models and networks.
   - Good simulators for dynamic model are: MATLAB/Simulink, Vissim [10], TransModeler [11], SUMO [6]. In these simulators mobility model was well developed. For simulating vehicle traffic and driver behavior (dynamic model simulator), in this work, using MATLAB/Simulink. In this simulator the direction of vehicles is creating during the simulation and the probabilistic component is participates in this creation.
   - Network simulators are presented by following programs: ns-3, ns-2 [5], OPNET modeler [12], cnet [13]. Now shortly about them. Opnet modeler is a commercial product for network modeling and simulation. It allows users to design and study communication networks, devices, protocols, and applications. Cnet - is a free software which allows to create different simulations. Simulation sizes may range from two to a few hundred nodes and cnet has the GUI.

In this work using network simulator - ns3. This simulator has good implementation of the most part of network protocols and this program is open source, that allows to understand the logic of the simulator work and modify the source files with necessary changes. Also ns3 allows to create big scale simulations.

2. The interaction of simulators is carried out by means of TCP protocol by socket Berkley.

3. Both simulator take turns being the server, the regime change on the client.

1.6 Thesis Objectives

The objective of this work is:

1. To modify network simulator ns-3 by opportunity of interconnection with external application (for example another simulator) within the limits of combined simulation environment creating.

2. To show the way how to softly change simulator to the specific goal.

3. To do real interaction with external simulator.

4. To consider opportunities of simulator external management.
1.7 Technologies used

1.7.1 Operating system

Ubuntu is a computer operating system based on the Debian GNU/Linux distribution.

Linux is a general name of UNIX-like computer operating system based on the Linux kernel and gathered for it libraries and system programs developed within the limits of the project GNU. Their development is one example of free and open source software collaboration. This means that all the underlying source code can be used for freely viewed, modified and redistributed, both commercially and non-commercially, by anyone under the terms of the GNU GPL and other free software licenses

Ubuntu is chosen as the operating system because:

1. Not need to acquire a license. Distributive of Ubuntu can be downloaded for free, for example, from Internet, or order a free CD with it by mail

2. It is possible to have all the latest updates to the necessary applications that the open source world has to offer

3. It is easy to find and install all necessary applications, programming environment and etc.

I used the latest version of Ubuntu, Ubuntu 9.10 with code name “Karmic koala”. It includes the latest enhancements and is maintained until the beginning of March 2010. It comes with the 2.6.31 Linux-kernel.

1.7.2 Network simulator ns-3

Computer simulating is a one of effective methods to research difficult system. Computer models easier and more convenient to investigate because it is possible to have computational experiments when real experiments are impossible due high cost of experiment or experiments have some physical difficulties, or due to experiments may give unpredictable results. The properties of computer models like the logic and formalization are revealing the main factors, which determine properties of research original object.

The creation of computer simulation is based on abstracting from the specific nature of phenomena or of the original object. The computer simulation is the number of computational experiments which conducting on the computer. Goals of these experiments are analysis, interpretation and comparison of the simulating results with real behavior of researched object and etc.

A computer network is a collection of computers and devices connected by communications channels that facilitates communications among users and allows users to share resources with other users. Networks may be classified according to a wide range of characteristics.
Ns-3 is a discrete-event network simulator in which the simulation core and models are implemented in C++. Ns-3 is built as a library which may be statically or dynamically linked to a C++ main program that defines the simulation topology and starts the simulator. ns-3 also exports nearly all of its API to Python, allowing Python programs to import an “ns-3” module in much the same way as in C++[14].

Ns-3 is free software, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use.

I used the latest version of ns-3, ns-3.7.1. Release on march, 17th 2010.

1.8 Installation and configure

1.8.1 Eclipse installation

Before eclipse installation started it’s necessary to install packet sun-java6-jre with packet’ manager help. You can use this manager, to installation Eclipse IDE, than using Eclipse interface you can install Eclipse C/C++ Development Tools (CDT) and other plugins for comfortable work. More easily is just download distributive “Eclipse IDE for C/C++ Developers” from http://eclipse.org/

ns-3 installation

Before installing ns-3 you need to prepare the system for this. The core of ns-3 requires a gcc/g++ installation of 3.4 or greater, and python 2.4 or greater.

On the page http://www.nsnam.org/wiki/index.php/Installation you can find detailed instructions for installing everything you need to work properly ns-3. Also on this page you can instructions for installation of ns-3, but it is better to download the latest version from the official site: http://www.nsnam.org/.
1.8.2 Configure ns-3 in Eclipse

When you start Eclipse, you should change workspace on the folder that contains the ns-3 (for example home/username/repos); Create the new empty project C/C++ with the name, the same as name of the folder with the project ns-3(for example “ns-3.7”). When an empty project creating is finished, it will fill with ns-3 files. Actually it is enough to comfortable work with ns-3 files.

1.8.3 Debug settings

Caution: /path to ns3/build/debug is included all executable files in ns-3.

Follow the path in Eclipse “run / debug configurations”. Double clicking on “C/C++ Application” a new configuration will create. In appeared menu necessary to enter the path to executable file which will be debugged. Press apply button to confirm changes. Now link on ns-3 library file into operation system needed to create. You can do it using the following command in the terminal:

```
$ ln -s /path to ns3/build/debug/libns3.so /usr/lib
```

Now very strong debugging tool is available.
2. Method

There is a program written in the MATLAB. In this program, a new socket is created, then initialize a new tcp connection and after that, data exchange starts.

Simulink is a discrete-time simulator. The calculations of events are occurring at the end point of each simulation step. So in the end of each simulation step, the MATLAB information about all nodes in simulation is send to ns-3.

In the ns-3, there is only two case of time relations: realtime (all simulation events occurs over time of these events in real life) defaulttime (all simulation events occurs over time as fast as it is possible for computer), and there is no possibility to stop simulator in the necessary moment in the limits of experiment. In the frames of this project, customized modules are ejected in ns-3. It allows to stop simulator every exhibited previously, time limit.

After simulation running, ns-3 waits for the MATLAB running. Then ns-3 waits instructions to action from MATLAB. At that time, the application is filling the sending structure by instructions for each node “what to do at this moment (beginning of next simulation step) send a packet, or do nothing” and sends it to socket. Than MATLAB is waiting for instructions, which contained information about all nodes (for each node within last simulation step notify about receiving a packet or no), from ns-3.

In turn, ns-3, begin to record transmitting packets in scheduler after receiving the transmitting structure. Then ns-3 runs all the scheduled events until the expiration of simulation step. In that time, simulator creates report which includes some information about received packets. When the time is up, formed receiving structure sent to an MATLAB, and ns-3 goes into freeze mode and waiting for instructions. Now MATLAB will process a report, and again performs the work transmitting structure filling. It is means on each simulation step both simulators are exchange with messages included status information for each node (MATLAB send instruction to send packet or not, and ns-3 send notify about receiving or not).

The main problem is in the MATLAB socket using. MATLAB cannot receive the structure, which was sent to it. MATLAB listen socket, and when there is something in the socket – MATLAB just receive the stream of bits, and then cast it to one type of variables.

On figure 2.1 you can see interaction model of applications.
Figure 2.1: Interaction between ns-3 and MATLAB
3. Theory

3.1 Socket programming in Linux

3.1.1 Socket creating

Sockets provide a very powerful and flexible mechanism for interprocess communication (IPC). They can be used for interaction programs on one computer on a local network or via the Internet, which allows creating distributed applications of different complexity.

Sockets supports many standard network protocols, and provide a unified interface for working with them. Sockets supports many standard network protocols, and provide a unified interface for working with them.

Socket - this is the endpoint of network terminations. The application sends the data to the socket, and their following buffering, forwarding and transportation used by the stack of protocols and network equipment.

In program socket identified by descriptor – it is just variable of integer type. The program take descriptor from operation system when the socket are creating

Each socket binded with three attributes:

1. Domain
2. Type
3. Protocol

The attributes are set when the socket is created, and remain unchanged all the socket life time. Socket is creating by function “socket”, which have next format:

```c
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

the most used domains are : AF_UNIX and AF_INET
3.1. Socket programming in Linux

- AF_UNIX – File system is used for transmission
- AF_INET – Its Internet domain. Sockets can be used for transmission in any IP networks

Socket type is defined by method of data transmission. The most used types are: SOCK_STREAM and SOCK_DGRAM

- SOCK_STREAM – The data transmit with the preliminary connection. In that case reliable channel of data transmit is provided.
- SOCK_DGRAM – The data transmit as separate messages (datagram). There is no preliminary connection. This way is faster, but less reliable. In that case it is possible to multicasting and to broadcasting

If Internet domain is selected and socket type is SOCK_STREAM – it is means the TCP protocol is used. If SOCK_DGRAM – it is means the UDP protocol.

The last attribute is defining a protocol type. Often protocol type is uniquely defined when the domain and the socket type are selected. In that case as function socket third parameter you can assign 0. Sometimes, protocol value explicitly set is necessary. You can find digital id in documentation.

3.1.2 Addresses

Socket is necessary to bind with address in selected domain before data transmission. At Internet domain address is defined by IP-address and 16-bits port number combination. IP-address is defining host in the net, and port is the specific socket on this host. To bound socket with any address is used bind function, which have next format:

```c
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *addr, int addrlen);
```

Socket descriptor is assigned as first parameter, which is bounding with that address. Pointer on the address structure is included in second parameter. The length of this structure is assigned as third.

3.1.3 Connect (Server)

Connection on the server part is included four steps.
First step is a socket creating and binding it with local address.
There is turn of requests is created on the next step and the socket is going to requests waiting mode. Function “listen” is performing all this

```c
int listen(int sockfd, int backlog);
```

The first parameter is the socket descriptor, and the size of request turn is the second one. Every time when the next client trying to connect to the server, its request is becomes in turn, because the server can be busy with other requests processing. If turn is full, all next requests will be ignored. When the server ready to reply next request, it use function “accept”

```c
#include <sys/socket.h>

int accept(int sockfd, void *addr, int *addrlen);
```

Accept function is creating a new socket and return it descriptor. Parameter sockfd is defining the listening socket. After function calling socket also staying in listening mode and socket can provide connection.

### 3.1.4 Connect (Client)

For connection on a client part “connect” function is used.

```c
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

- sockfd – It is the socket, used to data exchange with server.
- serv_addr – It is a pointer on structure, included server address.
- addrlen – Length of this structure.

### 3.1.5 Data Exchange

After connection, you can start data exchange. For this “send” and “recv” functions are used. “send” function used for data transmitting

```c
int send(int sockfd, const void *msg, int len, int flags);
```
3.1. Socket programming in Linux

- sockfd – it is a socket descriptor, used for data transmit.
- msg – it is a pointer on buffer with data.
- len – buffer length in bytes.
- flags – it is flags for function management.

The “send” function returns the number of really transmitted bytes (in error case it returns -1) “recv” function used for data receiving

```c
int recv(int sockfd, void *buf, int len, int flags);
```

- sockfd – it is a socket descriptor, used for data sent.
- msg – it is a pointer on buffer with data.
- len – buffer length in bytes.
- flags – it is flags for function management.

3.1.6 Socket closing

When the data exchange is end, it is necessary to close the socket with help of function “close”. This function leads to disconnection.

```c
#include <unistd.h>

int close(int fd);
```
4. Implementation and evaluation

4.1 Implementation

4.1.1 The description of secondary classes

Classes, used in following description, are considered in this section.

_Ptr class - smart pointer_

This smart-pointer class assumes that the underlying type provides a pair of Ref and Unref methods which are expected to increment and decrement the internal refcount of the object instance.

This implementation allows manipulating the smart pointer as if it was a typical pointer: it is possible to compare it with zero, compare it with other pointers, assign zero to it and etc.

This information is important, because usually in ns-3 used smart-pointers, but sometimes typical C++ pointers, and it is useful to know how to do conversion between them. With help of GetPointer and PeekPointer methods conversion is available.

_EventId class_

Each EventId identifies a unique event scheduled with one of the many Simulator::Schedule methods. This EventId can be used to Cancel or Remove events after it scheduled with “Cancel” and “Remove” methods of Simulator class.

Also it is the class with help of which it is possible to identify any event. There are following information are included in this class: event implementation pointer, event id, event context and also auxiliary methods.

_Application class_

This class used as base class (parent class) for the majority of ns-3 applications. This class included usual methods for applications like: StartApplication, StopApplication and etc. Applications are associated with individual nodes. Each node holds a list of smart pointers to its applications.
4.1.2 Simulator implementation

Basically in ns-3 used two models of simulator implementation – Default simulator Implementation and Real-time simulator implementation. From the names of these implementations clear that real-time implementation need to that the simulation took place in real time. Default implementation need to do all scheduled events as quickly as possible, not looking on real time.

4.1.3 Default Simulator Implementation

Consider simulator working principle, in general case, when DefaulSimulatorImpl is used, and when in user script used only 2 nodes with installed udp echo client on first of them and udp echo server on the second.

After starting the command ./waf-run SkriptName in process of program performance record in the schedule of events described in a script (Creation of nodes, installation and start of programs etc) starts. When turn of performance simulator class of the instruction “Simulator::Run();” run method of DefaultSimImpl class is called.

```cpp
void
DefaultSimulatorImpl::Run (void)
{
    m_stop = false;
    while (!m_events->IsEmpty () && !m_stop)
    {
        ProcessOneEvent ();
    }
    NS_ASSERT(!m_events->IsEmpty () || m_unscheduledEvents == 0);
}
```

This function starts the ProcessOneEvent method while in the scheduler following event is not empty. In method ProcessOneEvent the next event from the scheduler is taken and it starts to fulfill.

It is necessary to know what kind of information is registering in the scheduler. Each event in the scheduler – is a structure “Event” from aScheduler class

```cpp
struct Event
{
    EventImpl *impl;
    EventKey key;
};
```

EventImpl -- it is a pointer on event implementation.
EventKey -- it is a structure of the next type:

```c
struct EventKey {
    uint64_t m_ts;
    uint32_t m_uid;
    uint32_t m_context;
};
```

m_ts -- This time of event start beginning.
m_uid -- it is a unique event id.
m_context -- event context.

Now it is necessary to understand where packet is send event registering take a place. There is “UdpEchoClient::ScheduleTransmit” method.

```c
void UdpEchoClient::ScheduleTransmit (Time dt) {
    NS_LOG_FUNCTION_NOARGS ();
    m_sendEvent = Simulator::Schedule(dt, &UdpEchoClient::Send, this);
}
```

There are three arguments which transfer to the function: time after which event should be started, pointer on the application method asend as and this application object. If these three parameters are known it is possible to specify uniquely from what node, what operation, and during what time will represent as event.

The m_sendEvent variable it is the object of “EventId” class which the “Scheduler” function returns.

Let’s recollect what kind of parameters are setting up when the application UdpEchoClient installation take a place: maximum packets number, interval between packets, packet size etc.

In the “UdpEchoClient” class the “ScheduleTransmit” function is called from two places. The first of it for each application called once and it enabled in the “StartApplication” method with 0 as argument. The second of it is included in the “Send” method (i.e. it is called every time when a packet is sent) with the value of interpacket time interval (which can be set up by user) as argument.

Let’s see the simulator work process. When the script is started, information about simulation is collecting, necessary events are record in the scheduler, and then the simulation is beginning. In process of simulation one by one event from the scheduler is
4.1. Implementation

running. In the application start time (set by user) event of a packet sending is written at this time in scheduler.

When it is turn of packet to be sent the “UdpEchoClient” application class object method “Send” is called. While its execution, event of new packet sending by this class object with time step of interpacket interval in the scheduler is registering.

The Sent packet the new events, related with transmission to the destination node, is generates. When the packet is received by destination node the “UdpEchoServer::HandleRead” method is called. This method contains functions with help which an echo-package will transmitted back. When the packet reached a source “UdpEchoClient::HandleRead” method is starting (i.e. The packet is received). On it packet transmission cycle is finished. Soon time to transmit of the next recorded packet in scheduler will come. All packet transmission operations will repeat again. It repeats until application stop time has not come.

When the “DefaultSimulatorImpl” is used, important simulation feature is that simulator try to complete all events as fast as it possible and it not dependently on real time. In other words, if user set simulation time, for example, 100 seconds, all simulation can be finished in 2 seconds, thus that all planned events are fulfilled. Let’s see how it is work. In function “ProcessOneEvent” execution time timer update is.

\[ m_{\text{currentTs}} = \text{next.key.m_ts}; \]

\[ m_{\text{currentTs}} \quad \text{-- is current simulation time} \]

\[ \text{next.key.m_ts} \quad \text{-- is next event planned time}. \]

I.e. last executed event time is assigned to simulation time. It is does not matter how new events are planned: 50 seconds between the adjacent or only one. Simulator does it equally fast.

4.1.4 Externally Driven Simulator Implementation

The main idea of externally driven simulator implementation is to manage events with external application help. For example, to sending packets not in the pre-scheduled time, but in the time which come from external to ns-3 in process of the simulation.

That idea was realized based on default implementation. The new class, whose parent is “DefaultSimulatorImpl” was written. In differs from the parent in the new class, new functions are added and some old functions are modified to interconnection with external applications, to take needed information from simulation, and to take control of the simulation. Also was created structures to transmit and to receive information to action.

Working principle of externally driven simulator is as follows: simulation runs short time, then wait for external call to record new events in the scheduler and to
continue to perform simulations for a short time. That is short time – simulation step, it set by user.

This work will be possible, if:

1. Class ExternallyDrivenSim can use application class methods.

2. Application class can use ExternallyDrivenSim class methods.

It is necessary not to spoil existing applications. In view of this to work with ExternallyDrivenSim the new applications in ns-3 based on udp echo was created. New applications has name – UdpEchoClientNew and UdpEchoServerNew. Following methods in applications classes was modified:

**UdpEchoClientNew class**

1. UdpEchoClientNew::StartApplication. In new application it is not needed to transmit first packet at application start time. It is not needed to start method

   ```cpp
   ScheduleTransmit (Seconds(0.));
   ```

   any more. Also new application not needs to wait server reply. Because it method

   ```cpp
   m_socket->SetRecvCallback(MakeCallback
   (&UdpEchoClientNew::HandleRead, this));
   ```

   not necessary any more too.

2. UdpEchoClientNew::Send. Firstly – it is not necessary to plan new event when this method is ran. Instruction

   ```cpp
   ScheduleTransmit (m_interval);
   ```

   is not used more. Secondly – every time when this method is called the value of “SetEventId” is assign to variable “m_sendEvent”. To understand why it is necessary, you can look through ExternallyDrivenSim::SetEventId method description.

   Other changes in the UdpEchoClienNew class are not so important

**UdpEchoServerNew class**

Only HandleRead Function has been changed. The instruction “GetNotices (this)” was added; more details about it is in ExternallyDrivenSim::GetNotices method description.
4.1. Implementation

For easy installation applications on the node the new class “UdpEchoNewHelper” was created. This class parent is “UdpEchoHelper”. In new class the following functions was modified: object creating functions, application installing functions etc. Only functions contains names of applications was modified

**ExternallyDrivenSim**

To understand how it works, auxiliary functions consider at first. Then pass to the main functions The main innovation in new simulator implementation is waiting function for external information. Socket opening functions are included in the constructor of “ExternallyDrivenSim”:

- function of socket creating – `listener = socket(AF_INET, SOCK_STREAM, 0);`
- bind function – `bind(listener, (struct sockaddr *) &addr, sizeof(addr));`
- listen function – `listen(listener, 1);`
- accept function – `accept(listener, NULL, NULL);`
Also address parameters are sets in constructor:

- `addr.sin_family = AF_INET;`
- `addr.sin_port = htons(3425);`
- `addr.sin_addr.s_addr = htonl(INADDR_ANY);`

**Listen method**

This method just use socket programming function “receive” to waiting for packet with transmit structure from external. When information has come, it is assign to these class variables. Inside this method, the function `RunScheduleTransmit` is also included. It will be described later.

SetSimulationStep function. Time is an argument in it. In this method the value of simulation step is recorded on the class variable.

**GetEventId method**

The sufficient information, to create the class “EventId” object, is the argument of this method. It is several variables. This information is recorded in class “Externally-DrivenSim” static variables, to use it outside this class.

**TransmitNotices method**
This method appointment – to transmit “Receive” structure to external application.

*Stop method*

This method is modified by adding instruction to record simulation stop time in class “ExternallyDrivenSim” static variable.

*GetNotices method*

In our case we need to transmit information about received packets. On a server part in one simulation step can appear situation when several packets are received. “GetNotices” is a tool to collect information about received packets. In function “HandleRead”, after packet receiving, add instruction: “GetNotices” method starting which has application object as argument. This function collects information about the node received message and records all this information to the structure. In the end of simulation step it sends the structure with this information to external application.

*SetEventId function*

In the application class, the object of “EventId” class included information about packet sending event, assigned to variable “m_sendEvent”. When it is time to send a packet, the program simply call “Send” method of specific object. There is comparison of “EventId” recorded in “m_sendEvent” and “EventId” of next event in the scheduler takes a place in this method. However, if request to send another packet come before the first was transmitted, m_sendEvent will correspond to “EventId” of last scheduled packet, and it will be compared with “EventId” of previous one which already recorded in the scheduler. To avoid this, each time when “Send” method is called, it is necessary to assign “EventId” of the current event to variable “m_sendEvent”. For this purpose, the method “SetEventId” was created.

Summary: “SetEventId” method is used without arguments, and it return EventId class object of current event. It is called from “Send” method of client class.

There are two main functions “Run” and “RunSchedulerTransmit” in the ExternallyDrivenSim.

RunScheduleTransmit function used auxiliary method:

*GetClient*

It is very interesting place in this work. In the “GetClient” function the pointer of source node used as argument, and it returns pointer for the client application which installed on this node. It is needed to return typical C++ pointer (not smart-pointer). It is not easy because in the beginning of this method only the node pointer is available. After using “GetApplication(0)” method of “Node” class the smart-pointer on the
“Application” class object is available. This pointer – is a pointer on the “UdpEchoClientNew” object because first installed application on this node is it (It is important that the client application has been installed first on this node). It is impossible to use “UdpEchoClientNew” class methods using pointer on the application. There are no methods how to get “UdpEchoClientNew” class object pointer using smart-pointer taken above. The problem solution is in following way. Firstly should be taken typical C++ pointer on the “Application” class object. It possible by Ptr::PeekPointer method used the smart-pointer on the “Application” class object as argument. This function returns the typical pointer on the “Application” class object in this case. Now need to convert this pointer on “UdpEchoClientNew” class pointer. It is possible by typecasting operation. Typecasting refers to changing an entity of one data type into another.

The instruction

\[
\text{UdpEchoClientNew* pucl = (UdpEchoClientNew*) applic1;}
\]

returns to variable “pucl” the “UdpEchoClientNew” class object pointer from pointer on “Application” class object contained in applic1.

*RunScheduleTransmit method*

This method appointment – is to launch “ScheduleTransmit” method with necessary arguments using the pointer on the “UdpEchoClientNew” class object (pointer on the object of the precisely defined application). RunScheduleTransmit at first creates the NodeContainer class object. Then assign to it GetGlobal method of NodeContainer class. This method returns the NodeContainer class object, included all nodes which use in the current simulation. Then GetClient method with the node source pointer as argument is called.

We needed typical pointer on “UdpEchoClientNew” class object because in the simulation the smart-pointer on this class object never creating before. Only smart-pointer to “Application” was created. As known “Application” is a parent class for “UdpEchoClientNew”, and in typical case its do not needed to call methods of child class outside of it. All what needed in typical case it is just called “Start” method on the object of application, and “UdpEchoClientNew” will start to work. It is impossible to use “Send” method on “Application” class method, because it membered only in it child class. Inside “UdpEchoClientNew” class the “Simulator::Schedule” method is called using typical pointer on “UdpEchoClientNew” class object. In our case needed to start this method from outside, using typical pointer on it.

After that, calculation of event planning time is performed. More in detail about it see at below description of Run method. Now, when all necessary information is available, method “ScheduleTransmit” is called using already taken client class object.

*Run method*
Instruction to launch the method “ProcessOneEvent” is contained in this method. “ProcessOneEvent” – it is a function which perform one next event. It is possible before “ProcessOneEvent” function is ran to take event from the scheduler and to analyze it.

As told before, a simulation should pass step by step with known simulation step. In this function it is realized. The variable “m_LimitTime” is the time value till which all operations from the scheduler should be executed. Every time before “ProcessOneEvent” function is run, the next event from scheduler is analyzed (i.e. the next event which will be in executed). If time, of its execution is smaller than time in “m_LimitTime” the “OneProcessEvent” method is run. This process will be repeated while that comparison is true. If comparison is false the cycle included this instruction is breaking. At that place to “m_limitTime” increase by the value of simulation step. Now report to external application is transmitted by “TransmitNotices” method. Than Listen method is run for waiting of the next information from external application. When it is received, and all new events are written in the scheduler, the same check is starting: the next event from scheduler is analyzed, event time is comparing etc. Also in “Run” method the logic for an emergency output from simulation, and for the planned is provided.

Changes

In general case “ExternallyDrivenSim” can work with different applications installed on simulator nods, but before you need to make some changes in the application, and in the simulator implementation.

In the implementation class method “GetClient” is need to change name of application in argument field and in the code.

In the “RunSchedulerTransmit” method is need to change instruction which called method of recording in scheduler.

If structures to receive and to transmit to external application are changed, you also need to change structures filling methods

In applications class is need to prepare: in client class – planning and sanding methods, and in server class – receiving method.

Interaction with external application

The interaction performed by using tcp connection. For this goal port number 3525 is used. It is very simple connection, and you can see example included in appendix to understand how is it works. The socket and address contained structures are created in the beginning of it. Then connection with server is performing, and information structures are sending. After that client wait server reply. All it repeats until simulation stop.

Usage

The usage of the externally driven simulator is straightforward, from a user
4.1. Implementation

ExponentiallyDrivenSim

**Parent:** DefaultSimulatorImpl

**Attributes**

- `int listener;`
- `struct sockaddr_in addr;`
- `char buf[15], message[1024];`
- `int bytes_read;`
- `Ptr<Application> m_applic;`
- `Ptr<Node> m_node;`
- `Ptr<UdpEchoClientNew> m_pud;`
- `double m_timeOfStep;`
- `bool m_quit;`
- `int m_sock;`
- `int m_nodeID;`
- `int m_nodeIDTo;`
- `std::string m_Payload;`
- `Time m_TimeOfEnd;`
- `uint64_t m_TimeLimit;`

**Methods**

- `ExternallyDrivenSim();`
- `static EventId GetEventId(void);`
- `static void GetNotices(UdpEchoServerNew * p);`
- `static TypeId GetTypeId(void);`
- `static void SetSimulationStep(Time);`
- `virtual void RunScheduleTransmit(void);`
- `virtual void RunScheduleTransmitTo(void);`
- `virtual void GetEventId(ExternallyDrivenSim * m_eventimpl, uint64_t m_ts, uint32_t m_context, uint32_t m_uid);`
- `virtual void TransmitNotices(void);`
- `virtual void Stop(Time const &time);`
- `virtual void Run(void);`

**Private**

- `virtual void DoDispose(void);`
- `virtual ~ExternallyDrivenSim();`
- `void ProcessOneEvent(void);`
- `void Listen(void);`
- `UdpEchoClientNew * GetClient(Ptr<Node> );`
- `Ipv4Address GetDestIp(Ptr<Node> );`

*Figure 4.3: ExternallyDrivenSim class*
code. Users just need to set the attribute SimulatorImplementationType to the externally driven simulator implementation, such as follows:

```cpp
GlobalValue::Bind ("SimulatorImplementationType",
StringValue ("ns3::ExternallyDrivenSim"));
```

Simulation Step setting by next function:

```cpp
ExternallyDrivenSim::SetSimulatorStep(Time);
```

Default value is – 200 milliseconds.
If using UdpEchoClient(Server)New you need use in user script commands to install this applications on nodes. You can see these commands in the description of example script (ExternalTest.cc) in appendix.

**Implementation**

The implementation of “ExternallyDrivenSim” is contained in the following files:

`src/simulator/externally-driven-sim.{cc,h}`

when externally driven simulator was created the following file have been modified:

`src/simulator/default-simulator-impl.h`

this file was modified by adding instruction

```cpp
friend class ExternallyDrivenSim;
```

It is necessary because “ExternallyDrivenSim” privat variables and methods of “DefaultSimulatorImpl” is necessary to use.

The implementation of “UdpEchoClientNew” is contained in the following files:

`src/applications/udp-echo/udp-echo-client.{cc,h}`

The implementation of “UdpEchoServerNew” is contained in the following files:

`src/applications/udp-echo/udp-echo-server.{cc,h}`

The implementation of “UdpEchoServerNew” is contained in the following files:

`src/applications/udp-echo/udp-echo-server.{cc,h}`

The implementation of “UdpEchoNewHelper” is contained in the following files:
4.2 Evaluation

It was held the joint start of simulators with following parameters.
There are only two nodes in this simulation with installed “UdpEchoClientNew” and "UdpEchoServerNew“ on both nodes.

To have a connection it is necessary to simplify receive/send structures in ns-3 so much as it is possible. For this interaction structure contained two integer variables. This structure used for receiving and for transmission. In case when ns-3 receive structure from MATLAB, first of variable is instruction to transmission or to not for first node and second variable is the same information for second node. In opposite case, first variable is information about first node status (received or not) and the second one for second node.

Simulation time is 10 seconds, and simulation was planned by next scenario: during all simulation time, at the end of each simulation step (default 200 milliseconds) data exchange take a place. Previously agreed information is sent by MATLAB to ns-3. At time from 5 second to 6 (5 times) MATLAB send instruction like ”send packet from first node“. At other time this instruction was like ”nothing to do“.

To make interaction possible I have changed following methods in simulator (changes for a current case):

```c
src/helpers/udp-echo-new-helper.cc,h
```

In parent classes of all this classes is needed to add instruction to take child class as a friend. Like in case with “ExternallyDrivenSim”.

**How to create new class in ns-3**

Usually each ns-3 class consists of two files: header file and code file. These files have identical names, but different extensions. File name – is class name and each word in it separate with “-” symbol.

All code and header files included in src folder. It is better to creating files of new class in the folder, which corresponds to this class. After creating, it is necessary to add names of this files in the script “wscript”, included in that folder. Each code file need to have a instruction:

```c
NS_LOG_COMPONENT_DEFINE ("ClassName");
```

beof namespace is declarated, and

```c
NS_OBJECT_ENSURE_REGISTERED (ClassName);
```

after.
1. ExternallyDrivenSim::Listen;
2. ExternallyDrivenSim::GetNotify;
3. ExternallyDrivenSim::TransmitNotify;
4. ExternallyDrivenSim::RunScheduleTransmit;

The first time of MATLAB and ns-3 communication finished well: all needed packets were sent, and all information from ns-3 received by MATLAB.

This is just first step on the way to connect network simulator ns-3 to MATLAB environment.
5. Discussion

This work is aimed to creation of a combined simulation environment, which allows to spend the different scenarios of vehicle traffic on the roads with sensor nodes with good approximation to reality. As outcome of simulation are: firstly – result of vehicle traffic, rather, reaction of driver on the work of applications which installed on the nodes; secondly – result of network communications between nodes. Combined simulation will allow us:

- Develop optimal applications for installing on the sensors, and test of them efficiency in the limits of simulation.
- The design of sensor networks on the real parts of a road.
- The detecting of possible problems in communication between nodes in depended of environment.

All lines above will help for developers, until the smooth functioning of scenario. This allows starting discussion about implantation of real sensor nodes into the road’s parts.

The stated goals were performed in this work. Network simulator ns-3 was supplemented by module for communication with external application via a Socket. All changes were described. This allows adapting this modification of the simulator for specific, high level goals in the future. The trial joint launch of both ns-3 and MATLAB also have a place in this work. The simple interaction between simulators was configured and result of this interaction is success.

However, usage the combined simulation environment in the goals of ITS depends on additional efforts that necessary.

- It is important to inject the third side in the combined simulation environment. Besides of network simulator and dynamic model simulator for a combined simulation environment the operating system for nodes in the simulation is also needed.
- The sensor nodes are work under management of TinyOS – operating system for tiny nodes. It written in NesC – is a special programming language. It is necessary to write library, which allows using TinyOS as module of ns-3 to install it on the nodes in the limits of simulation.
5.1. Subject conclusion

The applications for wireless sensor network, which will work on the TinyOS platform is necessary to develop and inject them into the simulation.

The developing tools, which make scalable simulations to rapidly expand and create new scenarios is an important task in this area.

Remains an open problem to find a way to synchronize events in both simulators.

5.1 Subject conclusion

The one of the problem during this work was to understand the working principle of program ns-3 and understand how the scheduler is filling by events, and how the simulation performed. However, when the solutions to these problems were found, the difficulties were not over. The new principal task is – to clarify where all these functions are realized in the source code and find appropriate modules. All difficulties during the work have been overcome. As result of this work is patch for ns-3, which included all modifications, and instructions for changing ns-3 are also in this thesis.
6. Extra features

Earlier described functions are contain all necessary instruction to successful interaction between ns-3 and MATLAB. This part of report included describing of function which is not used in interaction with MATLAB, but which can be useful in future work with ExternallyDrivenSim.

I want describe about option, which allow to external application to select destination node for each packet. The information about destination node should contain in instruction, which ns-3 receive from external application. Before describing of working principle of this option will described, let’s consider a functions which allow it.

6.1 UdpEchoClientNew class

6.1.1 ScheduleTransmit method

First change is in “UdpEchoClientNew“ class. The new method “ScheduleTransmit” was added. This method is modified method “ScheduleTransmit” which already consisted in this class, but new method have two arguments. The first is – time, after which, the packet will be sent, and second is – packet destination node Ip address. Each function call, fill new array cell by pair of elements. The first one is event UID and another one is destination Ip address. Then, using this data, we can determine Ip destination address by its UID, when packet sending event take a place. Also in this method the Simulator::Schedule function call instruction is changed to:

\[
m_{\text{sendEvent}} = \text{Simulator::Schedule}(dt, \\
&\text{UdpEchoClientNew::SendTo, this});
\]

Now the pointer on “SendTo” method using as function argument, (on this place was UdpEchoClient::Send method). This modification allows transmitting packets not only to in advance designated server, and to any node on which the server-based application is installed.

6.1.2 SendTo method

This new method of “UdpEchoClientNew“ class has the same instructions with “Send” method. The deferent are in the following:
It is necessary to know destination node Ip address. To achieve this goal the array, which contained UIDs and Ip addresses, in the method “SheduleTransmit” is filling. In method “SendTo” the destination Ip address is taken with help of this array by the compare of current event uid and even uid from array. When destination Ip address is available – need to add the class Socket method “connect” to have the connection with needed node:

```cpp
m_socket->Connect(InetSocketAddress(dest_ip, dest_Port));
```

### 6.2 ExternallyDrivenSim

#### 6.2.1 GetDestIp method

Following changes is in the “ExternallyDrivenSim” class. New method “GetDestIp” was added. In this method the node pointer is used as argument. It returns node IP address. The way how to get Ip-address is in following operations: Firstly, “Ip4” class object is necessary to get. This is achieved by using “Object::GetObject” method on the smart-pointer on the “Node” class object. The second step is getting pointer on “NetDevice” class object by using “NodeGetDevice(0)” method on the pointer on “Node” class object. This method returns pointer on first installed device on this node. To take interface number for this device it is needed to run the ipv4::GetInterfaceForDevice method using pointer on the “Ipv4” class object taken above with “NetDevice” pointer as argument. The “Ipv4InterfaceAddress” class object is available after using Ipv4::GetAddress method with the number of interface and address index as arguments. To Get Ip Address if “Ipv4InterfaceAddress” is available, the Ipv4InterfaceAddress::GetLocal method on the “Ipv4InterfaceAddress” class object is need to use (it is important, that each node should to have only one interface and only one net device).

#### 6.2.2 RunScheduleTransmitTo method

Now it is necessary to change the method, which called “ScheduleTransmit” method. New method “RunScheduleTransmitTo” was created. The differences of this method and “RunScheduleTransmit” function is in following instruction:

After time is calculated it is necessary to take destination Ip address by pointer on destination node (in this case destination node id is receive with instructions from external application and with help of node id we can take pointer on this node). “GetDestIp” function with the destination node pointer as argument is called.

Now, when all necessary information is available, “ScheduleTransmit” method is called (with two arguments: time and Ip address) using taken before client class object.
6.2.3 Usage

All described functions already included in the patch, but to use this feature needed to change instruction

\[ \text{RunScheduleTransmit();} \]

to

\[ \text{RunScheduleTransmitTo();} \]

in the ExternallyDrivenSim::Listen() method.
A. Appendix – External client application example

/*
 * File: CLIENT.cpp
 * Version: 1.0
 * Description: Client’s application for Externally driven by
tcp socket simulator implementation.
*
 * Author: Dmitry Kachan
 * E-mail: dskachan@gmil.com
 * Date: April 10, 2010
 *
 * This software was developed in the scope of iRoad project
*iRoad: Intelligent Road
 * http://www.iroad.se
 *
 *
 * Copyright notice
 *
 * Copyright (c) iRoad consortium 2010.
 * All rights reserved.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <resolv.h>
```cpp
#include <iostream>
#include <cstdlib>
#include <string>

int main()
{
    int sock;
    struct sockaddr_in addr;
    struct Recive rec;
    for (;;)
    {
        sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock < 0)
        {
            perror("socket");
            exit(1);
        }
        addr.sin_family = AF_INET;
        addr.sin_port = htons(3425);
        addr.sin_addr.s_addr = htonl(0x7f000001);
        if (connect(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0)
        {
            perror("connect");
            exit(2);
        }

        // instruction to filling atransa structure.
    }
    send(sock, &trans, sizeof(Transmit), 0);
    recv(sock, &rec, sizeof(Recive), 0);
    {
        // instructions to analyzing
    }

    close(sock);
}
```
B. Appendix – ExternallyDrivenSim

/*
 * File: externally-driven-sim.cc
 * Version: 1.0
 * Description: Externally driven by tcp socket
 * simulator implementation. Source file.
 *
 * Author: Dmitry Kachan
 * E-mail: dskachan@gmail.com
 * Date: April 10, 2010
 *
 * This software was developed in the scope of iRoad project
 * iRoad: Intelligent Road
 * http://www.iroad.se
 *
 *
 * Copyright notice
 *
 * Copyright (c) iRoad consortium 2010.
 * All rights reserved.
 */
#include "simulator.h"
#include "default-simulator-impl.h"
#include "scheduler.h"
#include "event-impl.h"
#include "ns3/global-value.h"
#include "ns3/string.h"
#include "ns3/boolean.h"
#include "ns3/nstime.h"
#include "ns3/ipv4.h"
#include "ns3/externally-driven-sim.h"
#include "ns3/udp-echo-client-new.h"
#include "ns3/node-container.h"
#include "ns3/ptr.h"
#include "ns3pointer.h"
#include "ns3/assert.h"
#include "ns3/log.h"
#include <cstdlib>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <stddef.h>
#include <math.h>

NS_LOG_COMPONENT_DEFINE ("ExternallyDrivenSim");

namespace ns3
{
    NS_OBJECT_ENSURE_REGISTERED (ExternallyDrivenSim);

    static int g_EventCount = 0;
    static uint64_t g_currentTs;
    static Ptr<EventImpl> g_EventImpl;
    static uint64_t g_ts;
    static uint32_t g_context;
    static uint32_t g_uid;
    static struct Data trans;
    // static struct Data rec;
    // static struct Recive g_rec;
    static Time g_SimulationStep = MilliSeconds(200);

    TypeId ExternallyDrivenSim::GetTypeId(void)
    {
        static TypeId tid =
            TypeId("ns3::ExternallyDrivenSim") .SetParent<
                DefaultSimulatorImpl> () .AddConstructor<
                ExternallyDrivenSim> ()
            return tid;
    }

    /*
     * there is the socket opening in constructor.
     */
    ExternallyDrivenSim::ExternallyDrivenSim()
m_TimeLimit = 0;
listener = socket(AF_INET, SOCK_STREAM, 0);
if (listener < 0)
{
    perror("socket");
    exit(1);
}
addr.sin_family = AF_INET;
addr.sin_port = htons(3425);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(listener, (struct sockaddr *) &addr, sizeof(addr)) < 0)
{
    perror("bind");
    exit(2);
}
printf("server loaded\n");
listen(listener, 1);
m_sock = accept(listener, NULL, NULL);
if (m_sock < 0)
{
    perror("accept");
    exit(3);
}
for (int i = 0; i < 2; i++)
{
    trans.sig[i] = 0;
}

void ExternallyDrivenSim::DoDispose(void)
{
    close(m_sock);
}

ExternallyDrivenSim::~ExternallyDrivenSim()
{
}

/*
* function to receive packets from external application.
*/

void ExternallyDrivenSim::Listen()
{
    printf("Waiting for client..\n");
    printf("Client:");

    struct Data rec;
    bytes_read = recv(m_sock, &rec, sizeof(Data), 0);

    //information analyse functions
    std::cout << " first " << rec.sig[0] << " second "
    << rec.sig[1] << " \n";
    if ((rec.sig[0] == 0) && (rec.sig[1] == 0))
    {
        std::cout << "within 200 milliseconds till "
        << (double) m_TimeLimit / 1000000000
        << "s. new events are not planned\n";
    }
    else
    {
        for (int i = 0; i < 2; i++)
        {
            if (rec.sig[i] == 1)
            {
                std::cout << "Nod ">#"<<i" Sending \n";
                m_nodId = i;
                m_timeOfStep = 0;
                RunScheduleTransmit();
                /*
                * if you need transmit packets to specific node,
                * and you already formed structures of exchange,
                * included necessary information,you just need
                * to comment " RunScheduleTransmit(); " string,
                * and uncomment following string.
                */
            }
        }
    }

    // RunScheduleTransmitTo();
}
std::cout << "\n";

void ExternallyDrivenSim::RunScheduleTransmit(void)
{
    NodeContainer m_nodes = NodeContainer::GetGlobal();
    m_pucl = GetClient(m_nodes.Get(m_nodId));
    Time JumpTime = NanoSeconds(m_TimeLimit) - g_SimulationStep -
void ExternallyDrivenSim::RunScheduleTransmitTo(void)
{
NodeContainer m_nodes = NodeContainer::GetGlobal();
m_pucl = GetClient(m_nodes.Get(m_nodId));
Time JumpTime = NanoSeconds(m_TimeLimit) - g_SimulationStep -
TimeStep(m_currentTs) + Seconds(m_timeOfStep);
Ipv4Address IpTo = GetDestIp(m_nodes.Get(m_nodeIdTo));
m_pucl->ScheduleTransmit(JumpTime, IpTo);
}

void ExternallyDrivenSim::Run(void)
{
  m_stop = false;
m_TimeLimit = (uint64_t) g_SimulationStep.GetNanoSeconds();
while (!m_events->IsEmpty() && !m_stop && (TimeStep(m_TimeLimit) < m_TimeOfEnd + g_SimulationStep))
{
  Scheduler::Event f_next = m_events->PeekNext();

  //test for end of simulation
  if ((m_TimeLimit != 0) && (NanoSeconds(f_next.key.m_ts) >= m_TimeOfEnd) &&
      (m_TimeLimit >= (uint64_t) m_TimeOfEnd.GetNanoSeconds()))
  {
    TransmitNotices();
    Stop(Seconds(0.));
    m_quit = true;
  }

  if (f_next.key.m_ts > m_TimeLimit)
  {
    std::cout << "Nothing to do now.\n" << "\n";
m_TimeLimit +=(uint64_t) g_SimulationStep.GetNanoSeconds();
    if (!m_quit)
    {
      
    }
void ExternallyDrivenSim::ProcessOneEvent(void)
{
    TransmitNotices();
    Listen();
}

else
{
    TransmitNotices();
}

else
{
    while (f_next.key.m_ts < m_TimeLimit)
    {
        f_next = m_events->PeekNext();

        if (f_next.key.m_ts >= m_TimeLimit)
        {
            break;
        }
        ProcessOneEvent();
    }
    std::cout << "\n" << "Time’s up. \n" << "\n";
    m_TimeLimit += (uint64_t) g_SimulationStep.GetNanoSeconds();

    if (!m_quit)
    {
        TransmitNotices();
        Listen();
    }
    else
    {
        TransmitNotices();
    }
}

// If the simulator stopped naturally by lack of events, make a
// consistency test to check that we didn’t lose
// any events along the way.
NS_ASSERT(!m_events->IsEmpty() || m_unscheduledEvents == 0);

void ExternallyDrivenSim::ProcessOneEvent(void)
{Scheduler::Event next;

next = m_events->RemoveNext();

g_EventCount++; sprintf(message, "%d", g_EventCount);

NS_ASSERT(next.key.m_ts >= m_currentTs);
m_unscheduledEvents--;
NS_LOG_LOGIC("handle " << next.key.m_ts);

//double m_prevTs = m_currentTs;
m_currentTs = next.key.m_ts;
m_currentContext = next.key.m_context;
m_currentUid = next.key.m_uid;

double m_prevTs = m_currentTs;

m_currentTs = next.key.m_ts;
m_currentContext = next.key.m_context;
m_currentUid = next.key.m_uid;

GetEventId(next.impl, next.key.m_ts, next.key.m_context, next.key.m_uid);

next.impl->Invoke();
next.impl->Unref();
}

/*
 * this function forming the notices to external application.
 */

void ExternallyDrivenSim::GetNotices(UdpEchoServerNew* p)
{
Ptr<Application> Applic = (Application*) p;
Ptr<Node> node = Applic->GetNode();
trans.sig[node->GetId()] = 1;
void ExternallyDrivenSim::TransmitNotices(void)
{
    std::cout << "sending " << trans.sig[0] << " and "
             << trans.sig[1] << "\n";
    send(m_sock, &trans, sizeof(Data), 0);
    for (int i = 0; i < 2; i++)
    {
        trans.sig[i] = 0;
    }
}

void ExternallyDrivenSim::GetEventId(Ptr<EventImpl> m_eventImpl, uint64_t m_ts, uint32_t m_context, uint32_t m_uid)
{
    g_EventImpl = m_eventImpl;
    g_ts = m_ts;
    g_context = m_context;
    g_uid = m_uid;
}

EventId ExternallyDrivenSim::SetEventId(void)
{
    return EventId(g_EventImpl, g_ts, g_context, g_uid);
}

}
void ExternallyDrivenSim::SetSimulationStep(Time time)
{
    g_SimulationStep = time;
}

void ExternallyDrivenSim::Stop(Time const &time)
{
    m_TimeOfEnd = time;
    Simulator::Schedule(time, &Simulator::Stop);
}
C. Appendix – Example of ns-3 applications

C.1 UdpEchoClienNew

/*
 * File: udp-echo-client-new.cc
 * Version: 1.0
 * Description: client application for testing
 * Externally driven simulator implementation
 *
 * Author: Dmitry Kachan
 * E-mail: dskachan@gmail.com
 * Date: April 20, 2010
 *
 * This software was developed in the scope of iRoad project
 * iRoad: Intelligent Road
 * http://www.iroad.se
 *
 * Copyright notice
 *
 * Copyright (c) iRoad consortium 2010.
 * All rights reserved.
 */
#include "ns3/log.h"
#include "ns3/ipv4-address.h"
#include "ns3/nstime.h"
#include "ns3/inet-socket-address.h"
#include "ns3/socket.h"
#include "ns3/simulator.h"
#include "ns3/packet.h"
#include "ns3/uinteger.h"
#include "ns3/trace-source-accessor.h"
#include "udp-echo-client.h"
#include "ns3/externally-driven-sim.h"
#include "udp-echo-client-new.h"

namespace ns3
{
    NS_LOG_COMPONENT_DEFINE ("UdpEchoClientNewApplication");
    NS_OBJECT_ENSURE_REGISTERED ( UdpEchoClientNew);

    static struct IpId ipid[50];
    static bool test[50];

    TypeId UdpEchoClientNew::GetTypeId(void)
    {
        static TypeId tid = TypeId("ns3::UdpEchoClientNew") .SetParent<
        UdpEchoClient> () .AddConstructor<UdpEchoClientNew> ();
        return tid;
    }

    void UdpEchoClientNew::StartApplication(void)
    {
        NS_LOG_FUNCTION_NOARGS();

        if (m_socket == 0)
        {
            TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
            m_socket = Socket::CreateSocket(GetNode(), tid);
            m_socket->Bind();
            m_socket->Connect(InetSocketAddress(m_peerAddress, m_peerPort));
        }

        // m_socket->SetRecvCallback(MakeCallback
        // (&UdpEchoClientNew::HandleRead, this));
    }

    void UdpEchoClientNew::ScheduleTransmit(Time dt)
    {
        NS_LOG_FUNCTION_NOARGS();
    }
m_sendEvent = Simulator::Schedule(dt, &UdpEchoClientNew::Send, this);
}

void UdpEchoClientNew::ScheduleTransmit(Time dt, Ipv4Address IpTo)
{
NS_LOG_FUNCTION_NOARGS();
m_sendEvent = Simulator::Schedule(dt, &UdpEchoClientNew::SendTo, this);
uint32_t n;
for (int i = 0; i < 50; i++)
{
  if (test[i])
  {
    n = i;
    break;
  }
}
upid[n].m_ipTo = IpTo;
upid[n].m_uid = m_sendEvent.GetUid();
test[n] = false;
}

void UdpEchoClientNew::Send(void)
{
NS_LOG_FUNCTION_NOARGS();
m_sendEvent = ExternallyDrivenSim::SetEventId();

NS_ASSERT(m_sendEvent.IsExpired());
Ptr<Packet> p;
if (m_dataSize)
{
  // If m_dataSize is non-zero, we have a data buffer of the same size that we
  // are expected to copy and send. This state of affairs is created if one of
  // the Fill functions is called. In this case, m_size must have been set
  // to agree with m_dataSize
  //
  NS_ASSERT_MSG(m_dataSize == m_size,
    "UdpEchoClientNew::Send(): m_size and m_dataSize inconsistent");
  NS_ASSERT_MSG(m_data,
    "UdpEchoClientNew::Send(): m_dataSize but no m_data");
  p = Create<Packet> (m_data, m_dataSize);
else
{

// If m_dataSize is zero, the client has indicated that she doesn't care
// about the data itself either by specifying the data size by setting
// the corresponding attribute or by not calling a SetFill function. In
// this case, we don't worry about it either. But we do allow m_size
// to have a value different from the (zero) m_dataSize.
//
    p = Create<Packet>(m_size);
}

// call to the trace sinks before the packet is actually sent,
// so that tags added to the packet can be sent as well
m_txTrace(p);
m_socket->Send(p);

++m_sent;
Ptr<Application> Applic = (Application*) this;
Ptr<Node> node = Applic->GetNode();

NS_LOG_INFO("Node # " << node->GetId() << " Packet # " << m_sent << ": sent " << m_size << " bytes");
}

void UdpEchoClientNew::SendTo(void)
{
    NS_LOG_FUNCTION_NOARGS();
m_sendEvent = ExternallyDrivenSim::SetEventId();
uint32_t n;
for (int i = 0; i < 50; i++)
{
    uint32_t ui = m_sendEvent.GetUid();
    if (ipid[i].m_uid == ui)
    {
        n = i;
brea
    }
}
test[n] = true;
m_socket->Connect(InetSocketAddress(ipid[n].m_ipTo, m_peerPort));
NS_ASSERT(m_sendEvent.IsExpired());
Ptr<Packet> p;
if (m_dataSize)
{
//
// If m_dataSize is non-zero, we have a data buffer of the same size that we
// are expected to copy and send. This state of affairs is created if one of
// the Fill functions is called. In this case, m_size must have been set
// to agree with m_dataSize
//
NS_ASSERT_MSG(m_dataSize == m_size,
"UdpEchoClientNew::Send(): m_size and m_dataSize inconsistent");
NS_ASSERT_MSG(m_data,
"UdpEchoClientNew::Send(): m_dataSize but no m_data");
p = Create<Packet> (m_data, m_dataSize);
}
else
{
//
// If m_dataSize is zero, the client has indicated that she doesn’t care
// about the data itself either by specifying the data size by setting
// the corresponding attribute or by not calling a SetFill function. In
// this case, we don’t worry about it either. But we do allow m_size
// to have a value different from the (zero) m_dataSize.
//
// call to the trace sinks before the packet is actually sent,
// so that tags added to the packet can be sent as well
m_txTrace(p);
m_socket->Send(p);

++m_sent;
Ptr<Application> Applic = (Application*) this;
Ptr<Node> node = Applic->GetNode();

NS_LOG_INFO("Node # " << node->GetId() << " Packet # " << m_sent
<< ": sent " << m_size << " bytes to " << ipid[n].m_ipTo);
}

UdpEchoClientNew::UdpEchoClientNew()
for (int i = 0; i < 50; i++)
{
    test[i] = true;
}

UdpEchoClientNew::~UdpEchoClientNew()
{
}

void UdpEchoClientNew::HandleRead(Ptr<Socket> socket)
{
    NS_LOG_FUNCTION(this << socket);
    Ptr<Packet> packet;
    Address from;
    while (packet = socket->RecvFrom(from))
    {
        if (InetSocketAddress::IsMatchingType(from))
        {
            InetSocketAddress address =
            InetSocketAddress::ConvertFrom(from);
            NS_LOG_INFO("Received " << packet->GetSize() << " bytes from "
                "bytes from "
            << address.GetIpv4());
        }
    }
}

C.2 UdpEchoServerNew

//*
* File: udp-echo-client-new.cc
* Version: 1.0
* Description: server application for testing
* Externally driven simulator implementation
* 
* Author: Dmitry Kachan
* E-mail: dskachan@gmail.com
#include "ns3/application.h"
#include "ns3/event-id.h"
#include "ns3/ptr.h"
#include "ns3/address.h"
#include "udp-echo-server-new.h"
#include "ns3/externally-driven-sim.h"

namespace ns3
{
    NS_LOG_COMPONENT_DEFINE ("UdpEchoServerNewApplication");
    NS_OBJECT_ENSURE_REGISTERED (UdpEchoServerNew);
   TypeId UdpEchoServerNew::GetTypeId(void)
    {
        static TypeId tid = TypeId("ns3::UdpEchoServerNew") .SetParent<UdpEchoServer>() .AddConstructor<UdpEchoServerNew>();
        return tid;
    }

    void UdpEchoServerNew::HandleRead(Ptr<Socket> socket)
    {
        Ptr<Packet> packet;
        Address from;
        while (packet = socket->RecvFrom(from))
        {
            if (InetSocketAddress::IsMatchingType(from))
            {
                Ptr<Application> Applic=(Application*) this;
                Ptr<Node> node=Applic->GetNode();
            }
        }
    }
}
InetSocketAddress address=InetSocketAddress::ConvertFrom(from);
NS_LOG_INFO("node # " << node->GetId() <<" Received "
<< packet->GetSize() << " bytes from " << address.GetIpv4());

packet->RemoveAllPacketTags();
packet->RemoveAllByteTags();
ExternallyDrivenSim::GetNotices(this);
   // NS_LOG_LOGIC ("Echoing packet");
   // socket->SendTo (packet, 0, from);
}
}

void UdpEchoServerNew::StartApplication(void)
{
NS_LOG_FUNCTION_NOARGS();
if (m_socket == 0)
{
    TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
    m_socket = Socket::CreateSocket(GetNode(), tid);
    InetSocketAddress local = InetSocketAddress(Ipv4Address::GetAny(),
        m_port);
    m_socket->Bind(local);
    if (addressUtils::IsMulticast(m_local))
    {
        Ptr<UdpSocket> udpSocket = DynamicCast<UdpSocket> (m_socket);
        if (udpSocket)
        {
            // equivalent to setsockopt (MCAST_JOIN_GROUP)
            udpSocket->MulticastJoinGroup(0, m_local);
        } else
        {
            NS_FATAL_ERROR(
                "Error: joining multicast on a non-UDP socket");
        }
    }
m_socket->SetRecvCallback(MakeCallback(&UdpEchoServerNew::HandleRead,this));
}
C.3 Helper for new applications

#include "ns3/externally-driven-sim.h"
#include "udp-echo-helper.h"
#include "ns3/udp-echo-server.h"
#include "ns3/udp-echo-client.h"
#include "ns3/uinteger.h"
#include "ns3/names.h"
#include "ns3/udp-echo-new-helper.h"
#include "ns3/udp-echo-server-new.h"
#include "ns3/udp-echo-client-new.h"
namespace ns3
{

UdpEchoServerNewHelper::UdpEchoServerNewHelper(uint16_t port)
{
    m_factory.SetTypeId (UdpEchoServerNew::GetTypeId ());
    SetAttribute("Port", UintegerValue(port));
}

UdpEchoServerNewHelper::~UdpEchoServerNewHelper()
{}

Ptr<Application> UdpEchoServerNewHelper::InstallPriv (Ptr<Node> node) const
{
    Ptr<Application> app = m_factory.Create<UdpEchoServerNew> ();
    node->AddApplication (app);

    return app;
}

UdpEchoClientNewHelper::UdpEchoClientNewHelper (Ipv4Address address, uint16_t port)
{
    m_factory.SetTypeId (UdpEchoClientNew::GetTypeId ());
    SetAttribute("RemoteAddress", Ipv4AddressValue(address));
    SetAttribute("RemotePort", UintegerValue(port));
}

void UdpEchoClientNewHelper::SetFill (Ptr<Application> app, std::string fill)
{
    app->GetObject<UdpEchoClientNew>()->SetFill (fill);
}

void UdpEchoClientNewHelper::SetFill (Ptr<Application> app,
    uint8_t fill, uint32_t dataLength)
{
    app->GetObject<UdpEchoClientNew>()->SetFill (fill, dataLength);
}
void
UdpEchoClientNewHelper::SetFill (Ptr<Application> app, uint8_t *fill,
   uint32_t fillLength, uint32_t dataLength)
{
    app->GetObject<UdpEchoClientNew>()->SetFill (fill, fillLength, dataLength);
}

Ptr<Application>
UdpEchoClientNewHelper::InstallPriv (Ptr<Node> node) const
{
    Ptr<Application> app = m_factory.Create<UdpEchoClientNew> ();
    node->AddApplication (app);

    return app;
}

UdpEchoClientNewHelper::~UdpEchoClientNewHelper()
{
}
D. Appendix – Example of user script

/*
 * File: ExternallyTest.cc
 * Version: 1.0
 * Description: user script for testing
 * Externally driven simulator implementation
 *
 * Author: Dmitry Kachan
 * E-mail: dskachan@gmil.com
 * Date: March 26, 2010
 *
 * This software was developed in the scope of iRoad project
 * iRoad: Intelligent Road
 * http://www.iroad.se
 *
 * Copyright notice
 *
 * Copyright (c) iRoad consortium 2010.
 * All rights reserved.
 */
#include "ns3/externally-driven-sim.h"
#include "ns3/simulator.h"
#include "ns3/core-module.h"
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
#include "ns3/helper-module.h"
#include "ns3/global-routing-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include <iostream>
#include <sstream>
#include <fstream>
```cpp
#include <string>
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ExternallyTestScript");

class ExternallyTest
{
public:
    /// Init test
    ExternallyTest();
    /// Configure test from command line arguments
    void Configure(int argc, char ** argv);
    /// Run test
    int Run();

private:
    int m_nodes;
    double m_totalTime;
    bool m_pcap;

    int m_server;
    uint16_t m_servPort;
    /// List of network nodes
    NodeContainer nodes;
    /// List of all devices
    NetDeviceContainer devices;
    // Addresses of interfaces:
    Ipv4InterfaceContainer interfaces;
    // wifiHelper
    WifiHelper wifi;

private:
    /// Create nodes and setup their mobility
    void CreateNodes();
    /// Install internet m_stack on nodes
    void InstallInternetStack();
    /// Install applications
    void InstallApplication();
    ListPositionAllocator* TakePositionFromFile(char file[]);
};
ExternallyTest::ExternallyTest();
```
m_totalTime(10), m_pcap(false), m_servPort(9)
{
    GlobalValue::Bind("SimulatorImplementationType", StringValue("ns3::ExternallyDrivenSim");
    ExternallyDrivenSim::SetSimulationStep(MilliSeconds(200));
}

void ExternallyTest::Configure(int argc, char *argv[])
{
    CommandLine cmd;

    cmd.AddValue("pcap", "Enable PCAP traces on interfaces. [Default 0]", m_pcap);
    cmd.AddValue("totalTime", "tie of the simulation[Default 20 sek]", m_totalTime);

    cmd.Parse(argc, argv);
}

ListPositionAllocator* ExternallyTest::TakePositionFromFile(char file[])
{
    NS_LOG_INFO("Building chain topology.");
    //taking topology from file.
    const char u =
    {
        
    };
    std::string line;
    std::string ll;
    // char file[SizeOfName]=&pFileName;
    std::ifstream myfile(file);

    std::getline(myfile, line);
    //first line in the file included number of nodes.
    for (int c = 0; (line[c] != u); c++)
    {
        ll = ll + line[c];
    }
    const char* tt = ll.c_str();
    m_nodes = atol(tt);
    m_server = m_nodes - 1;
    //nodes creating.
nodes.Create(m_nodes);

wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default();
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
wifiPhy.SetPcapFormat(YansWifiPhyHelper::PCAP_FORMAT_80211_RADIO_TAP);

YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default();
wifiPhy.SetChannel(wifiChannel.Create());

Ssid ssid = Ssid("wifi-default");
wifi.SetRemoteStationManager("ns3::AarfWifiManager");

wifiMac.SetType("ns3::AdhocWifiMac");
// devices installing
devices = wifi.Install(wifiPhy, wifiMac, nodes);
if (m_pcap)
wifiPhy.EnablePcapAll(std::string("gg-"));

// Creating topology from file topol.txt
double g[3] =
{
  0
};

ns3::ListPositionAllocator *listPosNod = new
ns3::ListPositionAllocator();
if (myfile.is_open())
{
  while (!myfile.eof())
  {
    std::getline(myfile, line);
    int count = 0;
    int k = 0;
    while (k != 3)
    {
      ll = "";
      for (; (line[count] != u); count++)
      {
        ll = ll + line[count];
      }
      devices.AddDevice(line[count]);
    }
  }
}
const char* tt = ll.c_str();
g[k] = atof(tt);
if (k == 2)
{
    listPosNod->Add(*new ns3::Vector3D::Vector3D(g[0], g[1],
            g[2]));
    k++;
    count++;
}
myfile.close();
else
{
    std::cout << "Unable to open " << file << "\n";
    return (listPosNod);
}
void ExternallyTest::CreateNodes()
{
    char file[] = "/home/dmitrik/repos/ns-3.7/scratch/top.txt";
    ListPositionAllocator* listPosNod = TakePositionFromFile(file);
    MobilityHelper mobility;
    mobility.SetPositionAllocator(listPosNod);
    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(nodes);
}
void ExternallyTest::InstallInternetStack()
{
    NS_LOG_INFO(
        " Installing Internet stack on all nodes and assigning IP Addresses."");
    InternetStackHelper internetStack;
    internetStack.Install(nodes);
    Ipv4AddressHelper address;
    address.SetBase("6.6.0", "255.255.0");
interfaces = address.Assign(devices);

Ipv4StaticRoutingHelper ipv4RoutingHelper;
Ptr<Ipv4> ipv4Node;
Ptr<Ipv4StaticRouting> staticRoutingNode;

int nod = 0;

while (nod < m_nodes)
{
    ipv4Node = nodes.Get(nod)->GetObject<Ipv4> ();
    staticRoutingNode = ipv4RoutingHelper.GetStaticRouting(ipv4Node);

    int leftNodes = nod - 1;
    int leftExit = leftNodes;

    while (leftNodes >= 0)
    {
        staticRoutingNode->AddHostRouteTo(interfaces.GetAddress(leftNodes),
                                           interfaces.GetAddress(leftExit), 1);
        leftNodes--;
    }

    int rightNodes = nod + 1;
    int rightExit = rightNodes;

    while (rightNodes <= m_nodes - 1)
    {
        staticRoutingNode->AddHostRouteTo(
                                           interfaces.GetAddress(rightNodes),
                                           interfaces.GetAddress(rightExit), 1);
        rightNodes++;
    }

    nod++;
}

void ExternallyTest::InstallApplication()
{
    NS_LOG_INFO("Create applications.");
    GlobalValue::Bind("ChecksumEnabled", BooleanValue(true));
}
UdpEchoClientNewHelper echoClient(interfaces.GetAddress(m_server), 9);
echoClient.SetAttribute("PacketSize", UintegerValue(1024));

ApplicationContainer clientApps;
for (int i = 0; i < m_nodes; i++)
{
    clientApps = echoClient.Install(nodes.Get(i));
}
clientApps.Start(Seconds(0.));
clientApps.Stop(Seconds(m_totalTime));
UdpEchoServerNewHelper echoServer(9);

ApplicationContainer serverApps;
for (int i = 0; i < m_nodes; i++)
{
    serverApps = echoServer.Install(nodes.Get(i));
}
serverApps.Start(Seconds(0.));
serverApps.Stop(Seconds(m_totalTime));

int ExternallyTest::Run()
{
    CreateNodes();
    InstallInternetStack();
    InstallApplication();

    LogComponentEnable("UdpEchoClientNewApplication", LOG_LEVEL_INFO);
    LogComponentEnable("UdpEchoServerNewApplication", LOG_LEVEL_INFO);
    Simulator::Stop(Seconds(m_totalTime));
    Simulator::Run();
    Simulator::Destroy();

    return 0;
}

int main(int argc, char *argv[])
{
    ExternallyTest Et;
Et.Configure(argc, argv);
return Et.Run();
}
E. Appendix – Example of structures when the extra features are using

E.1 Structure for transmit to ns-3

struct Data
{
    uint32_t node;
    char payload;
    double time;
    uint32_t nodeTo;
};

struct Transmit
{
    int HowMuch;
    bool quit;
    struct Data vect[10];
};

here:
uint32_t node -- id of the node which will send a packet.
char payload -- information for ns-3
double time -- sending time of the packet
uint32_t nodeTo -- id of the node which receive a packet
int HowMuch -- the total number of packets
bool quit -- signal the end of the simulation in ns-3

E.2 Structure for receive from ns-3

struct Recive
E.2. Structure for receive from ns-3

```c
{
    int HowMuch;
    uint32_t nodes[20];
    uint64_t time[20];
};
```

Here:

- `HowMuch` -- Total number of received packets in ns-3 in one simulation step.
- `uint32_t nodes[20]` -- Node which receive a packet
- `uint64_t time[20]` -- Time in which the node received a packet
REFERENCES


John Wiley & Sons Ltd, 2005.


Hubaux, “trans: Realistic joint traffic and network simulator for vanets .”

http://www.isi.edu/nsnam/ns/.


vehicular networks .”


- a large scale integrated simulation platform for v2x communications: Application to real-time traffic management .”
“a graphical language for simulation and model-based embedded development .”

“about transmodeler .”

“opnet .”

“cnet .”

“the ns-3 network simulator .”

“eclipse.org home .”

“stroustrup: C++ .”


“ubuntu home page .”

“wireshark go deep .”

“linux network programming, part 1 .”