



317959

Mobile Opportunistic Traffic Offloading

***D5.1.2 – Description and development of MOTO
simulation tool environment – Release b
(public)***



Grant Agreement No.	317959
Project acronym	<i>MOTO</i>
Project title	Mobile Opportunistic Traffic Offloading
Advantage	
Deliverable number	D5.1.2
Deliverable name	Description and development of MOTO simulation tool environment (release b)
Version	V 1.0
Work package	WP5 – Experimental validation
Lead beneficiary	INTECS
Authors	Daniele Azzarelli (INT), Eva Pierattelli (INT), Alessandro Marchetto (CRF), Leandro D’Orazio (CRF), Filippo Rebecchi (TCS), Andrea Passarella, Raffaele Bruno, Giovanni Mainetto
Nature	O – Other
Dissemination level	PU – Public
Delivery date	28/10/2014 (M24)

Table of Contents

ACRONYMS.....	7
EXECUTIVE SUMMARY	8
1 OVERVIEW	8
1.1 INTRODUCTION	8
1.2 SCOPE OF THIS DOCUMENT	8
1.3 RELATED DOCUMENTS	9
2 NS-3 NETWORK SIMULATOR	10
2.1 WHAT IS NS-3.....	10
2.2 CONCEPTUAL OVERVIEW	10
2.3 NS-3 FEATURES	10
3 MOTO SIMULATION PLATFORM REQUIREMENTS	12
4 MOTO SIMULATION PLATFORM DESIGN.....	13
4.1 PUSH&TRACK MODEL.....	13
4.2 SIMULATION SCENARIO	14
4.2.1 Simulation Parameters.....	14
4.2.2 Simulation Steps.....	14
4.3 MODULES.....	15
4.3.1 CDM-Node.....	15
4.3.1.1 Description	15
4.3.1.2 Attributes	15
4.3.1.3 Public Method.....	16
4.3.1.4 Private Method	16
4.3.2 UE-Node	16
4.3.2.1 Description	16
4.3.2.2 Attributes	16
4.3.2.3 Public Methods	17
4.3.2.4 Private Methods.....	17
4.3.3 CDM-Receiver	17
4.3.3.1 Description	17
4.3.4 UE-Receiver.....	18
4.3.4.1 Description	18
4.3.5 Seeders-Calc	18
4.3.5.1 Description	18
4.3.5.2 Attributes:	18
4.3.5.3 Public Methods	18
4.3.5.4 Protected Methods	18
4.3.6 Epidemic-Routing	18
4.3.6.1 Description	18
4.3.6.2 Public Methods	18
4.4 MOTO SIMULATION PLATFORM METRICS	19
5 MOTO SIMULATION PLATFORM USER MANUAL.....	20
5.1 SOFTWARE INVENTORY	20
5.2 SOFTWARE ENVIRONMENT	21
5.3 HOW TO START SIMULATION	21
6 ITETRIS PLATFORM.....	22
6.1 WHAT IS iTETRIS.....	22
6.2 CONCEPTUAL OVERVIEW	22
6.2.1 ETSI ITS communication architecture	22
6.2.1 iTetris architecture and features.....	23
7 ITETRIS PORTING ACTIVITY	27

7.1	WHY?	27
7.2	ARCHITECTURE.....	28
7.2.1	Changes from ns3 version 3.7 to version 3.18.....	28
7.2.2	Modules added to ns3 from original iTetris.....	30
7.2.3	Sources files changed in ns3 from iTetris	32
7.2.4	The ns3 (integrated in iTetris) final architecture.....	34
7.3	RESULTS	35
8	LTE INTEGRATION ACTIVITY	37
8.1	WHY?	37
8.2	ARCHITECTURE.....	37
8.2.1	Changes into iCS	37
8.2.2	Changes into ns3 (v3.18)	40
8.2.3	Files and modules changed and added to iTetris.....	41
8.2.4	The final iTetris architecture	43
8.3	RESULTS	43
9	ITETRIS EXAMPLE APPLICATION.....	44
9.1	FEATURES	44
9.2	ARCHITECTURE.....	46
9.3	RESULTS	48
10	CONCLUSION & OUTLOOK.....	49
11	REFERENCES.....	50

List of Figures

Figure 1: Push&Track software architecture model	13
Figure 2: Project Tree	20
Figure 3: src folder.....	20
Figure 4: epidemic-routing-app folder.....	20
Figure 5: scratch folder.....	20
Figure 6: cdm-node folder	20
Figure 7: ue-node folder	20
Figure 8: seeders-calc folder.....	21
Figure 9: ETSI ITS Architecture for Intelligent Transport Systems Communications (ITSC).....	23
Figure 10: iTetris high-level architecture	24
Figure 11: iTetris architecture.....	25
Figure 12: iTetris configuration files hierarchy (a) and run-time loop iteration (b)	26
Figure 13: ns3 3.7 source organization	28
Figure 14: ns-3.18 source organization.....	29
Figure 15: ETSI's ITSC architecture	30
Figure 16: Final Ns3/iTetris src folder.....	35
Figure 17: Sumo interface showing vehicles mobility during simulation.....	36
Figure 18: A first example of code change (in the highlighted box the added code) in an iCS files	38
Figure 19: A second example of code change (in the highlighted box the added code) in an iCS file	39
Figure 20: An example of code change (in the highlighted box the added code) in an iCS file.....	39
Figure 21: An example of code change (in the highlighted box the added code) in the code of ns3	40
Figure 22: Overall architecture of iTetris after the LTE integration. Even if the name of files and modules (i.e. directories) is not expected to be visible in the figure, it shows the set of modules modified during the LTE integration. Such modifications are highlighted with black triangles in the module icons	43
Figure 23: Sumo interface showing the vehicle mobility during the running of the example	45
Figure 24: Detail of the Sumo interface of the example	45

List of Tables

Table 1: MOTO simulation platform Requirements	12
Table 2: Files modified	33
Table 3: Restored core functionalities	34
Table 4: Files added to original modules	34
Table 5: List of main files that have been changed and/or added for the LTE integration activity	43
Table 6: List of relevant files implementing the LTE application example	48

Acronyms

Acronym	Meaning
AED	Average End-to-End Delay
CDM	Content Diffusion Manager
C2c	Car-toCar
DTN	Delay Tolerant network
eNB	evolved NodeB
EPC	Enhanced Packet Core
ETSI	European Telecommunications Standard Institute
e-UTRA	evolved UMTS Terrestrial Radio Access
iAPP	iTETRIS implementation of a cooperative ITS application
iCS	iTETRIS Control System
IP	Internet Protocol
iTetris	An Integrated Wireless and Traffic Platform for Real-Time Road Traffic Management Solutions
ITS	Intelligent Transportation Systems
ITSC	Architecture for Intelligent Transport Systems
ITS G5	European profile standard for the PHY and MAC layer of ITS operating in the 5 GHz frequency band
LTE	Long Term Evolution
MAC	Media Access Control
NIC	Network Interface Card
ns-3	Network Simulator
PDR	Packet Delivery Ratio
PGW	Packet Gateway
QoS	Quality of Service
RSU	Roadside Unit
UDP	User Datagram Protocol
Ue	User equipment
V2V	Vehicle-to-Vehicle

Executive Summary

MOTO investigates the limits of LTE technologies in congested conditions and how opportunistic routing protocols can guarantee the offloading and diffusion of data. With respect of these specifications the MOTO project develops and maintains a common simulation platform to be used for the performance evaluation and experimental validation in Task T5.2 of the algorithms and protocols proposed in the WP3 and WP4.

The simulation platform is based on the open source simulator ns-3, which provides a basic environment for running event-driven packet-level simulations, also including packet tracing and collection of statistics. iTETRIS platform has also been enhanced in order to fully cover the MOTO requirements for vehicular scenarios.

This document further details the MOTO simulation tool environment, the specifications and development status of the ns-3 implemented modules, the software architecture, the definition of a common set of metrics and finally all functionalities enhanced and developed in iTETRIS platform.

1 OVERVIEW

1.1 Introduction

To evaluate the performance of the different offloading strategies and protocols implemented in MOTO, an appropriate simulation environment is needed. The MOTO simulation platform accomplishes this goal and provides a common simulation platform to be used for the performance evaluation and experimental validation of the proposed algorithms.

1.2 Scope of this Document

The scope of this document is to describe the MOTO simulation platform based on ns-3, which is a widely used network simulator for research and education on Internet systems. This document also describes the enhancements on the iTetris platform in order to provide to all partners two powerful tools able to simulate a sufficient sample of the defined use cases.

The document is organized as follows:

- Section 2 introduces the ns-3 simulator and describes its main features and capabilities.
- Section 3 defines the requirements of the MOTO simulation platform as a result of previous discussions with project partners.
- Section 4 reviews the MOTO simulation platform architecture and proposes a set of metrics that will be used to analyse the performance of the proposed offloading solutions.
- Section 5 provides a user manual for the MOTO Simulation tool.
- Section 6 introduces the iTetris platform and describes its main features and capabilities.
- Section 7 describes the porting executed on the original iTetris platform to adapt it to the MOTO simulation requirements, and describes the added features.
- Section 8 describes the features added in order to integrate LTE support in iTetris.
- Section 9 proposes a description of a simple application showing how the enhanced iTetris platform works.

1.3 Related Documents

- [A.1] D2.1 – Use Cases and Requirements
- [A.2] D2.2.1 – General Architecture of the Mobile Offloading System
- [A.3] D3.1 – Initial results on offloading foundations and enablers

2 ns-3 network simulator

2.1 What is ns-3

ns-3 is a discrete-event driven network simulator. *ns-3* aims to develop a complete simulation environment for networking researches. It starts on a solid and well documented simulation core which is in charge of the entire simulation logical flow, from simulation configuration to trace collection and analysis.

From a high level perspective, *ns-3* has a modular architecture to facilitate the addition of new simulation models and modules (new application, new wireless technologies, etc...).

ns-3 core modules are designed with the objective of being aligned with real world systems as much as possible, and they provide basic components that are common across all protocol, hardware, and environmental models.

ns-3 is not only an environment in which to run simulations but is also a collection of simulation models, which are abstract representations of real-world objects, protocols, devices, etc.

2.2 Conceptual overview

The first thing we need to do before actually starting to look at *ns-3* is to explain a few core concepts and abstractions that are used for describing the simulated network system.

The basic computing device abstraction is called `Node` (a *host system* or an *end system*). You should think of a `Node` as a device to which you can add a set of functionalities. More precisely, applications, protocol stacks and peripheral cards with their associated drivers can be attached to a `Node` in order to enable the device to perform specific functions.

In the real world, one can connect a computer to a network. Often the media over which data flows in these networks are called `channels`. When you connect your Ethernet cable to the plug in the wall, you are connecting your computer to an Ethernet communication channel. In the simulated world of *ns-3*, one connects a `Node` to an object representing a communication channel, called `channel`.

If you wanted to connect a computer to a network, you had to buy a specific kind of network cable and a hardware device called a *peripheral card* that needs to be installed in your device. If the peripheral card implements some networking function, they are called Network Interface Cards, or *NICs*. A *NIC* will not work without a software driver to control the hardware. In *ns-3* the `net device` abstraction covers both the software driver and the simulated hardware. Therefore, a `net device` is installed in a `Node` in order to enable the `Node` to communicate with other `Nodes` in the simulation via `channels`.

Whenever one is using a simulation, it is important to understand exactly what is being modelled and what is not. A model is, by definition, an abstraction of reality. *ns-3* has several `models` which are an abstract representation of real world objects, protocols, devices, etc. *ns-3* generally favours flexibility, and many models allow freely setting `Attributes` without trying to enforce any arbitrary consistency or particular underlying specification.

To summarize, a simulation scenario will be composed by `Nodes` communicating with other `Nodes`, each one with different functionalities implemented by different simulation models that characterize them.

2.3 ns-3 features

If a simulator does not describe the main features of a real system model with sufficient accuracy, it becomes difficult to compare the results and validate the simulated models. For these reasons, *ns-3* tries to reduce model approximations, so as to have modules that can be efficiently reused.

Although it is a very powerful tool and provides several features and models, only a sub-set of such modules will be considered in MOTO. This sub-set will encompass those sufficient to define a simulation scenario that fits the purposes of the MOTO project.

The following actions have to be performed in order to use ns-3 simulation tool:

- Define the scenario to simulate:
 - define the topology
 - create nodes, channel, network interfaces
 - configure Internet stack and IP protocols
 - configure models to be added in nodes
 - configure application to be started
 - set attributes
- build the simulation script using a text editor
- execute the .cc program
- analyse the ns-3 output

The *ns-3* simulator features an integrated attribute-based system to manage default and per-instance values for simulation parameters. All of the configurable default values for parameters are managed by this system, integrated with command-line argument processing. The setting of attributes is a very important operation and it has to be done carefully since a not reasonable attribute value can result in an inconsistent or unexpected behaviour.

Once the simulation is over, performance results can be obtained by processing trace files. Trace files can be filtered via a script and the filtered results can be processed via a plotting tool. *ns-3* can also create .pcap files readable with tcdump or Wireshark (which are two widely used network packet analysers).

3 MOTO simulation platform requirements

This section identifies the functional requirements that the MOTO simulation platform has to satisfy. These requirements have been collected based on the proposal, ideas and experience provided by all partners.

It is important to know what is expected from the simulator, in both qualitative and quantitative terms. In MOTO, it is preferred to provide a limited set of clear and defined requirements in terms of functionality and behaviour. Based on the list of features, technologies, protocols, interfaces and the description of test bed, simulation, testing scenarios proposed by the partners, and the capabilities offered by ns-3, the identified MOTO functional requirements are listed below:

Requirement ID	Description
R-1	The MOTO simulation platform MUST allow the creation of ns-3 nodes with a Wi-Fi interface
R-2	The MOTO simulation platform MUST allow the creation of ns-3 nodes with an LTE interface
R-3	The MOTO simulation platform MUST allow the creation of ns-3 nodes with Wi-Fi interface in ad hoc mode
R-4	The MOTO simulation platform MUST allow the creation of ns-3 hybrid nodes, equipped with both Wi-Fi and LTE interfaces
R-5	The MOTO simulation platform MUST permit the simultaneous use of both the Wi-Fi and LTE interfaces, in the ns-3 hybrid node
R-6	The MOTO simulation platform MUST guarantee that S1-U interface (reference point between E-UTRAN and Serving GW for the per bearer user plane tunnelling and inter eNB path switching during handover) is modelled
R-7	The MOTO simulation platform MUST guarantee that X2 interface (reference point between two eNBs within E-UTRAN architecture) is modelled
R-8	The MOTO simulation platform MUST allow the configuration of flexible scenarios with a number of eNBs that can range from a few ones to a few hundreds, and a number of Ues that can range from a few tens to a few thousands
R-9	The MOTO simulation platform MUST accurately implement the handover procedures within LTE infrastructure
R-10	The MOTO simulation platform MUST allow the loading and configuration of mobility traces
R-11	The MOTO simulation platform MUST allow the loading and configuration of propagation channel models
R-12	The MOTO simulation platform SHOULD allow the integration with road traffic generator to simulate vehicular mobility
R-13	The MOTO simulation platform MUST permit the calculation of performance metrics such as throughput, end-to-end delay, packet loss
R-14	The MOTO simulation platform MUST permit to implement different offloading solutions

Table 1: MOTO simulation platform Requirements

4 MOTO simulation platform design

This section describes the architecture defined for the MOTO simulation platform in order to realize an efficient and effective simulation tool that can be used to validate the offloading solutions proposed by MOTO project.

Figure 1 shows the building-block models of MOTO simulation platform.

The subsequent sections describe the role of each model, going deeper into the description of the new ns-3 models.

4.1 Push&Track model



Figure 1: Push&Track software architecture model

As has been discussed in Deliverable D3.1, Push&Track is a practical solution for offloading in cellular network, which has been defined by one of the partners of the MOTO project and that has been adopted as reference solution for offloading in the overall project. Therefore, the development of the MOTO

simulation platform has started from the implementation in ns-3 of the new functionalities that are needed to support Push&Track operations. Furthermore, the overall design of the MOTO simulation platform complies with the first release of the MOTO system architecture as described in Deliverable D2.2.1 (ref. [A.2]).

4.2 Simulation Scenario

In the following paragraph the simulation parameters and the main steps in scenario configuration are summarized.

4.2.1 Simulation Parameters

Configurable parameters used to design the simulation scenarios are listed below:

- traceFile: mobility traceFile
- n-ues: number of Ues
- n-enbs: number of eNBs
- messageTimeLife: message lifetime (sec)
- staticNInjects: number of periodic injects
- initialPush: time of first inject (msec)
- enableSendPos: enable Ues forwarding of position message
- frequencySendPos: frequency of Position forwarding
- deltaT: diffusion State evaluation frequency (msec)
- sendPanicZone: time required for sending messages in Panic Zone (msec)
- dimPacket: bundle dimension (bytes)
- enableEpidemic: enable epidemic routing
- enableTrace: enable trace source connect
- helloIntv: hello messages frequency (msec)
- when-strategy: name of the strategy (initial, linear, slow-linear, fast-linear, square-root)
- who-strategy: name of the strategy (random)

4.2.2 Simulation Steps

A brief description of the main steps to start a simulation run follows:

1. Configure simulation scenario:
 - a. Configuration of simulation parameters using the function GeneralConfig()
 - b. Creation of EPC network
 - c. Instantiation and configuration of Cdm_Node
 - d. Creation of n-enbs eNBs
 - e. Instantiation and configuration of n-ues Ue_Node

- f. Mobility configuration for eNBs Ues (ConfigureMobility())
 - g. Wi-Fi configuration (ConfigureWifi())
 - h. Installation of epidemic routing on Ues
 - i. Handover activation
2. Set the Stop time for simulation basing on messageTimeLife, *initialPush* and sendPanicZone.
3. Start Simulation
4. Make available collected metrics (PrintUesMetrics())

4.3 Modules

In this section it is provided a description of all the new modules added to the ns-3 official release. All added modules extend ns-3 existing ones (Parent Class) and for each module we are going to report the following features:

- C++ Class name
- C++ Parent class name
- Functionalities description
- Class Attributes
- Class methods

4.3.1 CDM-Node

Class: Cdm_Node

Parent Class: Node

4.3.1.1 Description

The goal of the CDM-Node is to pilot the dissemination procedure. To this end, it periodically evaluates the status of the dissemination process. If the data dissemination level does not meet an expected value the CDM-Node eventually decides to execute a number of additional data injections.

4.3.1.2 Attributes

m_Ack: map used to indicate if a feedback for a certain content has been received; the key is the Ue ID

m_Position: map used to indicate the current position of Ues; the key is the Ue ID

m_dTimeDelay: bundle's lifetime (sec)

m_bFirstInject: bool to identify the first inject

m_dTimeFirstInject: time of first inject execution (sec)

m_dDeltaT: frequency for diffusion status evaluation (msec)

m_uiPacketDim: bundle dimension (bytes)(max-val=10024)

m_uiNueByCdm: number of Ues which have received bundle from CDM through LTE

m_uiNueByEpidemic: number of Ues which have received bundle from CDM through Wi-Fi

m_UesAddress: map of ue-node and LTE IP address

m_seederCalcHelper: helper used to install seeder-calc module

m_StartPacket: empty bundle which has to be sent from CDM to Ues

4.3.1.3 Public Method

SetTimeDelay: sets m_dTimeDelay

GetTimeDelay: gets m_dTimeDelay

SetDeltaT: sets m_dDeltaT

SetPacketDim: sets m_uiPacketDim

GetNUesReached: gets m_uiNuesByCdm and m_uiNueByEpidemic

AddUeCdm: adds to m_UesAddress the related data

SetScheduling: schedules the diffusion status evaluation function

SetRecApp: installs PacketSink and connects SinkRx to its relative source trace

InstallCdmApplication: installs the module seeders-calc specifying the required When-Strategy and Who-Strategy

4.3.1.4 Private Method

SinkRx: callback connected to ns3::PacketSink application, in order to receive bundle from CDM

GetNAllUesNotServed: returns the number of Ues currently not reached from the bundle

SendBundleToUe: sends bundle to Ue

CreateInitialPacket: creates an empty message of the required length in bytes

PushValuation: function periodically scheduled to evaluate the bundle diffusion

- verifies if messageTimeLife seconds are elapsed from first Pushing and eventually if this time is elapsed (Panic zone management) gets the list of all not reached Ues
- if messageTimeLife seconds are not elapsed gets a smart pointer to the seeders-calc module (BLACK BOX) in order to calculate the number and the ID of Ue designed to receive the content through LTE interface
- sends bundle to designed Ues using **UDP** protocol.

4.3.2 UE-Node

Class: ns3::Ue_Node

Parent Class: ns3::Node

4.3.2.1 Description

The UE-Node registers a call back in order to receive from the CDM the content through LTE interface and **UDP** protocol. It schedules a periodic macro used to send Ue's coordinates to CDM by **UDP** protocol, gets a smart pointer at the epidemic-routing-app module and enables the epidemic routing for the content type data. When a Ue receives some content, it sends a tracking feedback to CDM through the LTE interface by using **UDP** and forwards the content through Wi-Fi interface using epidemic routing algorithm.

4.3.2.2 Attributes

m_ns3UdpSocket: socket for ack/position message sending

m_cdmAddress: CDM node LTE interface address

m_dPosFreq: frequency of position messages sending (ms)

m_MyLteAddress: owns LTE interface address

m_bSendPos: bool to indicate if the UE has to execute the position sending

m_bInject: bool to indicate if the UE has received the bundle by direct inject or epidemic routing (default false - epidemic)

m_delayJitterTrace: stores the delay information for each Ue

m_bytesTotal: structures to collect bytes on different interfaces

4.3.2.3 Public Methods

SetSchedulingPos: schedules SendPosition private method

SetFreqPosSend: sets m_dPosFreq

SetCdmAddress: sets m_MyLteAddress

SetMyAddress: sets m_MyLteAddress

SetPosSendEnable: sets m_bSendPos

SetDelay: sets m_delayJitterTrace

GetDelay: gets m_delayJitterTrace

GetNBytesForInterface: gets m_bytesTotal

GetInjectMode: gets m_bInject

SendAck: sends one ack message to CDM for each received bundle

SetRecApp: installs PacketSink and connects RxTrace, TxTrace and SinkRx to their relative source trace

4.3.2.4 Private Methods

SinkRx: callback connected to ns3::PacketSink application, in order to receive ack/position from Ues

RxTracer: updates metrics whenever a packet is received at Node level

TxTracer: updates metrics whenever a packet is transmitted at Node level

SendPosition: sends Ue position to CDM

4.3.3 CDM-Receiver

Class: ns3::PacketSink

Parent Class: ns3::Application

4.3.3.1 Description

Installed on the CDM-Node, it receives, via **UDP**, tracking feedbacks or position data sent from UE and, for each feedback received, updates the map Cdm_Node::m_Ack. For each position data received, it updates the map Cdm_Node::m_Position.

4.3.4 UE-Receiver

Class: ns3::PacketSink

Parent Class: ns3::Application

4.3.4.1 Description

Installed on each UE, it receives bundle from CDM, notifies it to CDM-Node and forwards the bundle to its neighbours.

4.3.5 Seeders-Calculators

Class: ns3::SeedersCalc

ns3::WhoCalc

Parent Class: ns3::Object

4.3.5.1 Description

Installed on the CDM, it offers a customizable interface required to give back the list of Ues designed for the inject procedure.

4.3.5.2 Attributes:

ns3::SeedersCalc::m_uiStaticNInject: number of inject statically configurable by ns3::Config class

4.3.5.3 Public Methods

ns3::SeedersCalc::GetNPKetsToDelivery: gets the number of Ues which have to receive the content through LTE interface in order to respect the reference function

ns3::WhoCalc::GetUeToDelivery: gets all Ues designed for the pushing procedure

ns3::WhoCalc::GetUesForPanicZone: gets all Ues not reached before Panic Zone

4.3.5.4 Protected Methods

ns3::WhoCalc::GetFirstUeld: gets the first ID of Ues

4.3.6 Epidemic-Routing

Class: ns3::DtnApp

Parent Class: ns3::Application

4.3.6.1 Description

Epidemic Routing [R2] spreads contents over the network until the bundle's lifetime is expired by establishing a wireless connection between Ues that come into contact (neighbours).

The model supports the forwarding, in broadcast mode, of hello message containing bundles table to detect neighbours' node and to avoid the reinjection of the content to just infected neighbours.

The model supports the storing of the message in a UE local buffer in order to forward it all times that UE enters in contact with a new UE.

4.3.6.2 Public Methods

SetTimeLive: sets bundle lifetime

ReceiveHello: receives hello message using a socket on port 80 of Wi-Fi interface and updates neighbours' information. e.g. what bundles from they currently have

SendHello: creates and sends in broadcast mode the hello message which contains the identifiers of those bundle that are stored in this node. Hello message has priority over all other messages

ReceiveBundle: receives bundle using a socket on port 5000 of Wi-Fi interface

SendBundle: creates a bundle and appends it to output bundle queue

CheckBuffer: reorders bundles, removes expired bundle, checks if MAC queue is below a given threshold. In this case selects the next packet to be transmitted.

4.4 MOTO simulation platform metrics

If the scenario is executed by the CDMSimulate script, the following files are generated at the end of simulation:

1. **CDMLogs.txt:** reports each data printed on standard output, in detail for each Ues:
 - a. the modality of reception of the content (LTE injection or Epidemic routing)
 - b. number of bytes sent for LTE each interface
 - c. number of bytes received for LTE interface
 - d. number of bytes sent for WIFI interface
 - e. number of bytes received for WIFI interface
 - f. throughput for each interface
 - g. reception time
 - h. delay from first injection
 - i. LTE-data acquired through RadioBearerStatsCalculator for each UE (disabled commenting the line `lteHelper->EnableTraces`)
2. **PieDiffusion.dat:** number of Ues reached by content split up for interface type divided by the total number of reached Ues
3. **NBForInterfaces.dat:** total number of bytes split up for interface type divided by the total number of bytes (sum for all the Ues)
4. **ThrForInterfaces.dat:** total throughput split up for interface type divided by the total throughput (sum for all the Ues)
5. **DelayForInterfaces.dat:** total delay in msec from first inject time split up for interface type
6. **BytesHello.dat:** number of bytes received by WIFI split up for message type (control or data)
7. **GraphResults.ps:** postscript file reporting graphs generated by GnuPlot for point 2, 3, 4, 5

5 MOTO simulation platform user manual

5.1 Software inventory

The following pictures represent the MOTO project tree, with details of its main folders and subfolders listed in order to help MOTO's users to identify modules added to official ns-3 release.

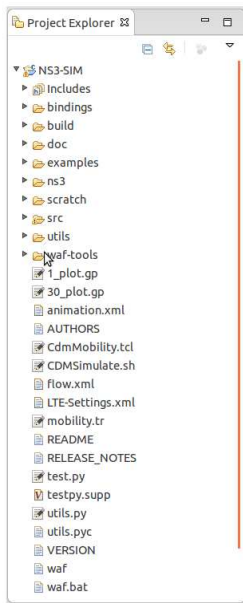


Figure 2: Project Tree

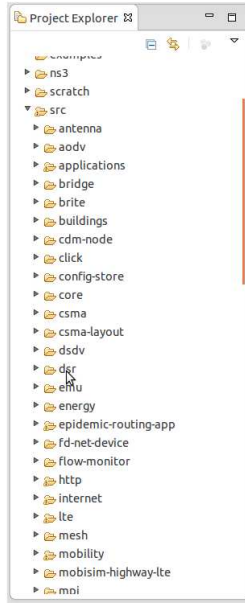


Figure 3: src folder

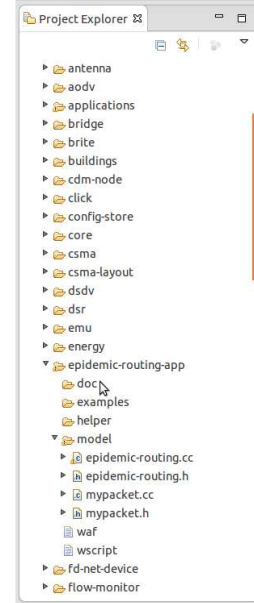


Figure 4: epidemic-routing-app folder

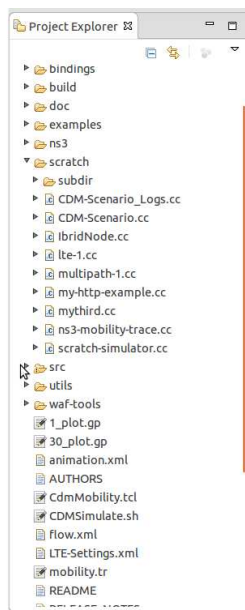


Figure 5: scratch folder

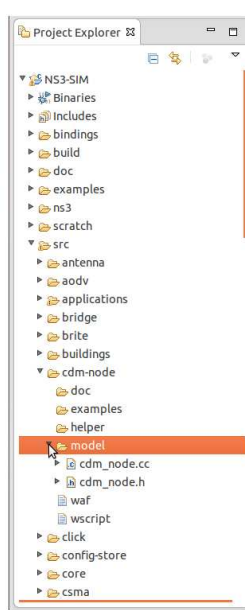


Figure 6: cdm-node folder

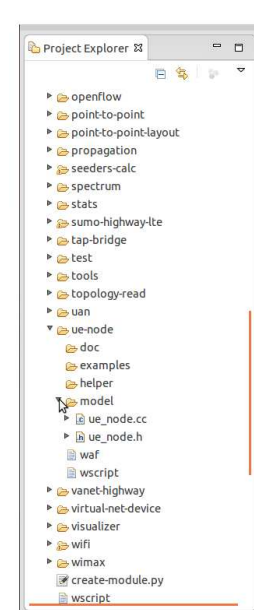


Figure 7: ue-node folder

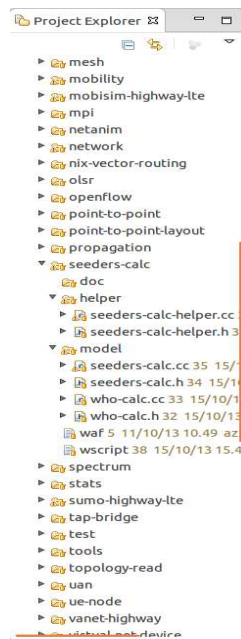


Figure 8: seeders-calc folder

5.2 Software environment

This section exposes the required hardware and software for running simulations and describes how to configure the environment for developing new features and testing the simulation scenario.

The PC has to be equipped with Linux O.S. (e.g. Ubuntu distribution v 12.04 LTS), with a development environment for C++ (e.g. Eclipse). The ns-3 used release starts from the LENA module LTE-EPC network simulator aligned with v 3.17 (<http://lena.cttc.es/hg/lena>).

The user has to download ns-3 trunk by SVN and copy it into an ns3folder following the tree structure in Figure 2 and then to compile environment using the following command:

1. `./waf distclean`
2. `./waf -d debug -o build/debug --enable-examples --enable-test configure`
3. `./waf`

5.3 How to start simulation

Users have to configure the scenario before starting the simulation, as follows:

1. Set Wi-Fi and LTE parameters in GeneralConfig and ConfigureWifi function in file CDM-Scenario.cc
2. Execute the script called CDMSimulate.sh to start simulation setting the simulation parameters summarized in 4.2.1 as file parameters input.

6 iTetris platform

The iTetris platform is described in the following sections.

6.1 What is iTetris

iTetris [R4] is a unique open source simulation platform characterized by a modular architecture that allows integrating two widely adopted traffic and wireless simulators, while supporting the implementation of cooperative ITS applications in a language-agnostic fashion.

iTetris was developed under the European FP7 Program (iTetris: an Integrated Wireless and Traffic Platform for Real-Time Road Traffic Management Solutions, <http://ict-itetris.eu/>) for investigating cooperative ITS systems and services.

iTetris integrates and extends SUMO and ns-3, two widely reference open source platforms for vehicular mobility and wireless communications simulations, and allows the implementation of cooperative ITS applications in various programming languages. Its open source nature and modular architecture facilitates the future expansion of the platform.

The platform is also capable of simulating large scale scenarios, which represents a very appealing feature for the investigation of cooperative ITS systems. In fact, the large scale evaluation of cooperative ITS strategies (25.000 vehicles) poses important unprecedented challenges regarding simulation complexity and precision in the wireless communication modelling

6.2 Conceptual overview

6.2.1 ETSI ITS communication architecture

The ETSI's ITS architecture for Intelligent Transport Systems Communications (ITSC) defines four main communication sub-systems for the execution of cooperative ITS applications. A Personal ITS sub-system is a handheld communications device (e.g. a smartphone). A Central ITS sub-system is a Traffic Management Centre (TMC) responsible for the centralized control of the road traffic. In order to disseminate road traffic information to vehicles or collect floating car data, a Central ITS sub-system can be connected to an 802.11p or ITS G5-based Roadside ITS sub-system (also referred to as Roadside Unit or RSU) or other infrastructure nodes (e.g. cellular, WiMAX or DVB base stations). Finally, a Vehicle ITS sub-system is a connected vehicle capable to communicate with other vehicles and the infrastructure nodes, and execute cooperative ITS applications. iTetris is aligned with the communication architecture defined by the ETSI for the ITS.

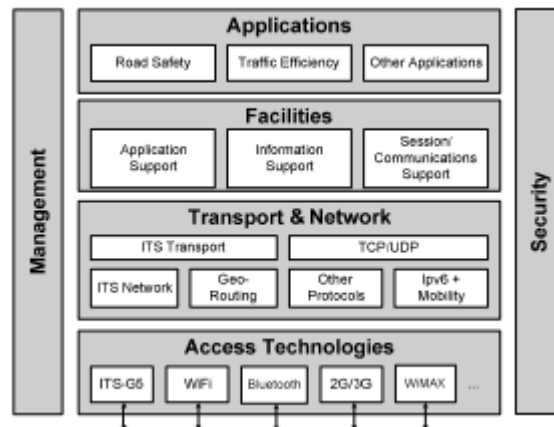


Figure 9: ETSI ITS Architecture for Intelligent Transport Systems Communications (ITSC)

The ITSC communication sub-systems implemented in the architecture illustrated in Figure 9 .Cooperative ITS applications can be supported by various radio access technologies. The IEEE 802.11p standard was specifically developed to enable reliable communications between vehicles and with RSUs, and has been adapted at European level through the ETSI ITS G5 standard.

The Access Technologies layer includes a variety of communications technologies enabling short range, broadcast and cellular-type communications. The Transport & Network layer includes two different protocol stacks:

- The GeoNetworking or Car-to-Car (C2C) Stack implements specific addressing schemes, georouting and transport protocols based on ITS G5.
- The IP Stack contains pre-existing TCP/UDP transport and IP networking protocols.

The Facilities layer includes a set of common functionalities and data structures to support cooperative ITS applications and communications. They can be classified into Application Support, Information Support and Communication Support facilities. An important Facility is the Local Dynamic Map (LDM), which stores and manages the dynamic information characterizing the local neighbourhood of a vehicle or RSU. This information can be collected through received cooperative messages or on-board sensors. The information stored in the LDM can be used by the cooperative ITS Applications that exploit the underlying functionalities to provide road safety, traffic management and infotainment services.

The vertical Management layer is responsible for monitoring and management functionalities.

Finally, the Security layer provides security services for the complete communications stack in order to prevent external attacks, guaranteeing the user’s privacy, and ensuring secure and trustworthy exchange of information.

6.2.1 iTetris architecture and features

Vehicles and RSUs wirelessly exchange information with other nodes, and therefore need to be represented in the network simulator (ns-3); vehicles are also represented in SUMO to simulate their mobility.

To handle the representation of a node over different platforms, iTetris implements a new central block referred to as iCS (iTetris Control System). The iCS handles SUMO and ns-3 interaction, in addition to preparing, triggering, coordinating and controlling the execution of iTetris simulations. The resulting iTetris architecture is represented in Figure 10 (high level) and Figure 11 (detailed) that also maps the real world aspects to be modelled and simulated over the distinct blocks of the platform.

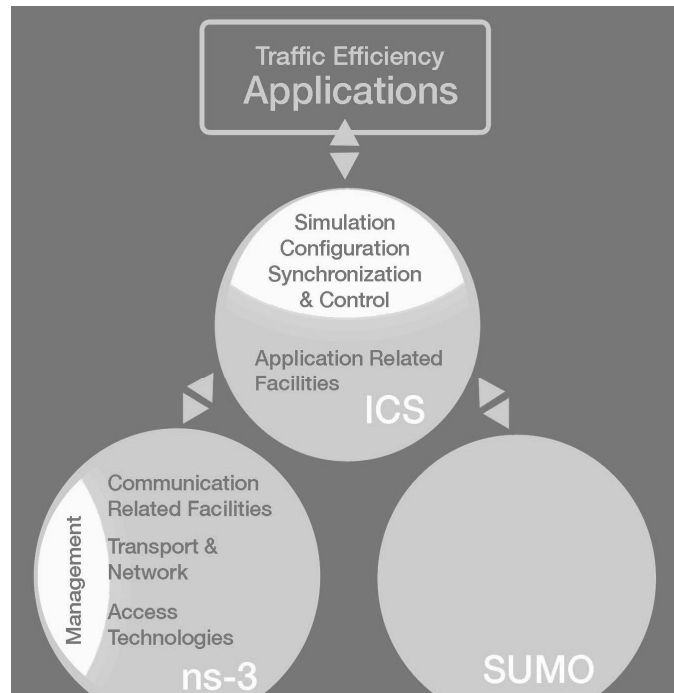


Figure 10: iTetris high-level architecture

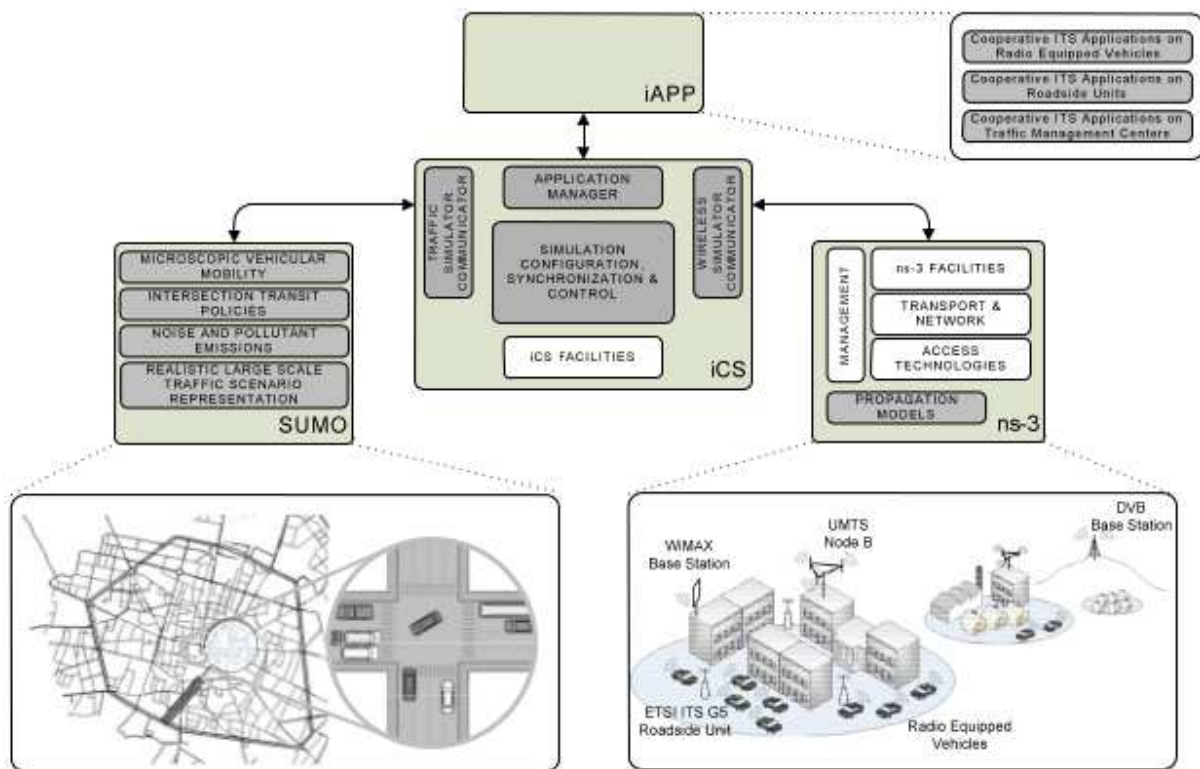


Figure 11: iTetris architecture

Cooperative ITS applications running on vehicles, RSUs or TMCs are implemented in external blocks referred to as iTetris Applications (iAPPs).

SUMO models and simulates transportation aspects like vehicles' mobility at microscopic level, road intersections transit policies, pollutant and noise emissions produced by the vehicles, as well as their fuel consumption. It also supports detailed representations of real world large scale traffic scenarios in terms of road networks, traffic demands and traffic lights management.

ns-3 supports accurate and realistic modelling and simulation of wireless transmissions for cooperative ITS systems in heterogeneous communications scenarios. As discussed previously in the current document, ns-3 includes suitable models to emulate the radio propagation effects and to reproduce functionalities and protocols for every layer of the communications protocol stack.

An important feature is the fact that iTetris is aligned with the ETSI ITSC architecture (white blocks in Figure 11), as already addressed before.

The iCS provides some supporting functionalities for the cooperative ITS applications implemented on the iAPP. Consequently, the implementation of the ITSC Facilities has been split between ns-3 and iCS. In particular, the Facilities more closely related to cooperative ITS applications (and thereby requiring a higher interaction with the iAPP) are implemented on the iCS (iCS Facilities in Figure 11), while those needed to support communication sessions have been implemented in ns-3 (ns-3 Facilities in Figure 11).

The modular architecture depicted in Figure 11 permits that cooperative ITS applications can be defined and implemented in a language-agnostic fashion, while ns-3 and SUMO can be separately and independently extended with new features. Therefore, iTetris allows the integration of a great variety of traffic simulators and networks by means of open APIs. All modules interact through the iCS central unit through a set of implemented open interfaces linking the iCS with the other iTetris building blocks. This

approach increases modularity, and allows easily updating or replacing any of the iTetris modules without interfering with the others (in case of replacement, only the open interfaces will need to be re-programmed).

In this context, the iCS can communicate with the other blocks by just specifying on which IP address and port the SUMO, ns-3 and iAPP blocks have to listen to. A “client/server” association is adopted with the iCS always acting as controller. For this purpose, the iCS implements three internal components that handle the communications with the rest of the iTetris blocks: the Traffic Simulator Communicator, the Wireless Simulator Communicator, and the Application Manager (Figure 11). Through dedicated client entities implemented in these components, the iCS triggers actions to be executed in the other blocks, and actively requests the resulting outcomes. On the other hand, server entities implemented in the other iTetris blocks reactively accomplish the tasks requested by the iCS.

The execution of iTetris simulations is controlled by the iCS. First, the iCS sets up the simulation environment by initialising the various iTetris configurable objects. A hierarchical XML configuration file structure is adopted (Figure 12a) to improve the readability of the simulation configuration, and facilitate the customization of the various iTetris blocks. The master configuration file also includes the path to the files used to configure ns3, SUMO and the iAPP.

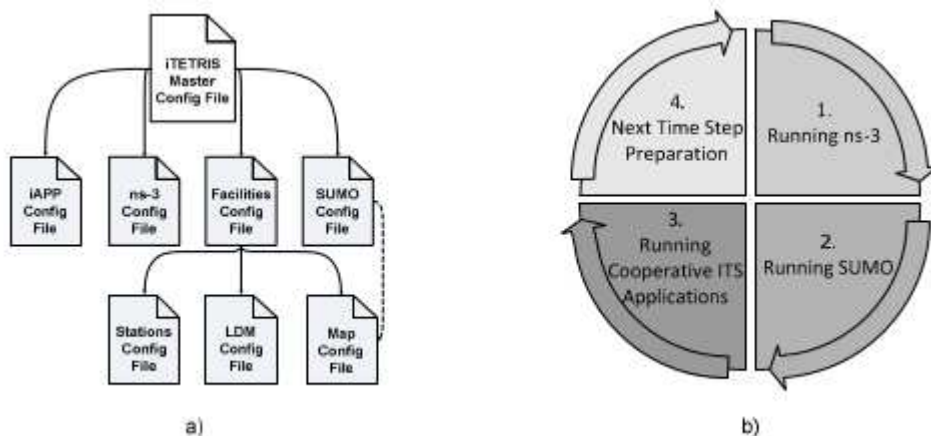


Figure 12: iTetris configuration files hierarchy (a) and run-time loop iteration (b)

When iTetris is started, SUMO and ns-3 are launched by the iCS with their executables registered in the system in separate threads so that, from that point on, they can receive commands. At the same time, the iCS allocates dedicated execution threads for each of the simulated cooperative ITS applications (iAPPs), and reads the configuration file needed to create the iCS Facilities.

SUMO and ns-3 configuration files are then read to prepare the traffic and wireless environments (e.g. road map, vehicular traffic flows, type and communications parameters of the simulated wireless technologies, etc.).

iTetris simulations consist of subsequent iterations of a loop (Figure 12.b) in which ns-3, SUMO and the iAPPs are sequentially triggered by the iCS to execute their tasks. The simulated time period specified in the master configuration file is divided into simulation time steps of one second². For each simulation time step, the different iTetris blocks simulate all the application, traffic or wireless communications events scheduled for the corresponding time step.

The entry point in the run-time loop is the simulation of the transmission of wireless messages in ns-3. The application’s payload for these messages is created and stored in the iCS. When the iCS schedules message transmissions in ns-3, a reference to these payloads is passed to ns-3. Once ns-3 has simulated all the events scheduled for the current time step, the iCS retrieves the simulation results as lists of successful

wireless transmissions (i.e. messages that have been correctly received by their recipient nodes). The iCS can then match the received messages with the previously stored payloads, and update specific communications-related iCS Facilities (e.g. LDM database) which can then be accessed and used by the applications implemented in the iAPPs.

In the following stage of the run-time loop, SUMO is triggered to simulate all the traffic mobility events corresponding to the established simulation time step. As required by the iCS' Traffic Simulator Communicator, SUMO provides as outcome the updated position and speed of active vehicles, along with the position and speed of vehicles entering the simulated scenario in the current time step.

Upon retrieving the SUMO simulation outcomes, the iCS updates the mobility-related iCS Facilities to store the information related to active or new vehicles that could be used by the applications implemented in the iAPPs. The iCS passes then the simulation token to the iAPP block. iAPP is asked by the iCS' Application Manager to "subscribe" to the SUMO and ns-3 simulation results that are needed for the execution of the application implemented in it. Based on these subscriptions, the iCS forwards the needed information to the iAPP. After executing the applications implemented in the iAPPs during the corresponding simulation time step, the iCS retrieves the iAPPs' results that in turn may generate new actions to be executed over SUMO or ns-3 (e.g. the transmission of a new wireless message or the traffic rerouting of a vehicle).

The last stage of the run-time loop is devoted to prepare the execution of the next time step in ns-3. In particular, the iCS' Wireless Simulator Communicator schedules the transmission of new messages, commands ns-3 to update the position of nodes based on SUMO's outcomes, and instructs ns-3 to create the new connected vehicles that have just entered the simulation scenario. Then, the iCS updates the time step counter, and checks whether its value is equal to the previously configured simulated time period's duration. If this is the case, the simulation is ended; otherwise a new iteration loop is performed.

7 iTetris Porting activity

7.1 Why?

The original iTetris platform supports the ns-3 platform in its 3.7 version for wireless communications' simulation; in order to use iTetris in MOTO project it is required to have at least the ns-3 platform in its 3.18 version.

This requirement is necessary because in MOTO a complete support for LTE and WiFi modelling is needed; LTE features were not supported in 3.7 version of ns-3 and only in the 3.18 version it has been reached an adequate modelling detail able to simulate handover mechanism manually and automatically.

The goal of this activity was to update the ns3 version 3.7 integrated in iTetris with the more recent version 3.18; this activity has required the following steps:

- Identify changes from version 3.7 to version 3.18.
- Identify all modules developed and added in original ns3 version during the iTetris project.
- Add all individuated modules in ns3 3.18.
- Identify all changes executed during the iTetris project on ns3's source files.
- Repeat all individuated changes in ns3 v.3.18.
- Get a fully working compilation of the ported ns3 code.
- Update or remove all ns3 parameters and attributes that iTetris used in its original version but are not more supported in ns3 v.3.18.

- Execute a fully working simulation session of iTetris using the demo app delivered with iTetris.
- Verify that the behaviour of iTetris simulation has not changed once completed the porting.

7.2 Architecture

7.2.1 Changes from ns3 version 3.7 to version 3.18

There are many changes between these two ns3 version also due to a lot of new modules added during the time, however the most important difference are in the way modules are organized.

In ns3 v.3.7 the organization of modules is much less structured than in v.3.18, there are a lot of folders which contains sets of modules but for each module the source and header files are not organized in sub folders. The helper folder is common to all modules and contains helper's source and header files for each module (Figure 13). This makes less modular the environment and makes more difficult the upgrading of modules; to avoid this problematic, in ns3 v.3.18 a different approach has been followed.

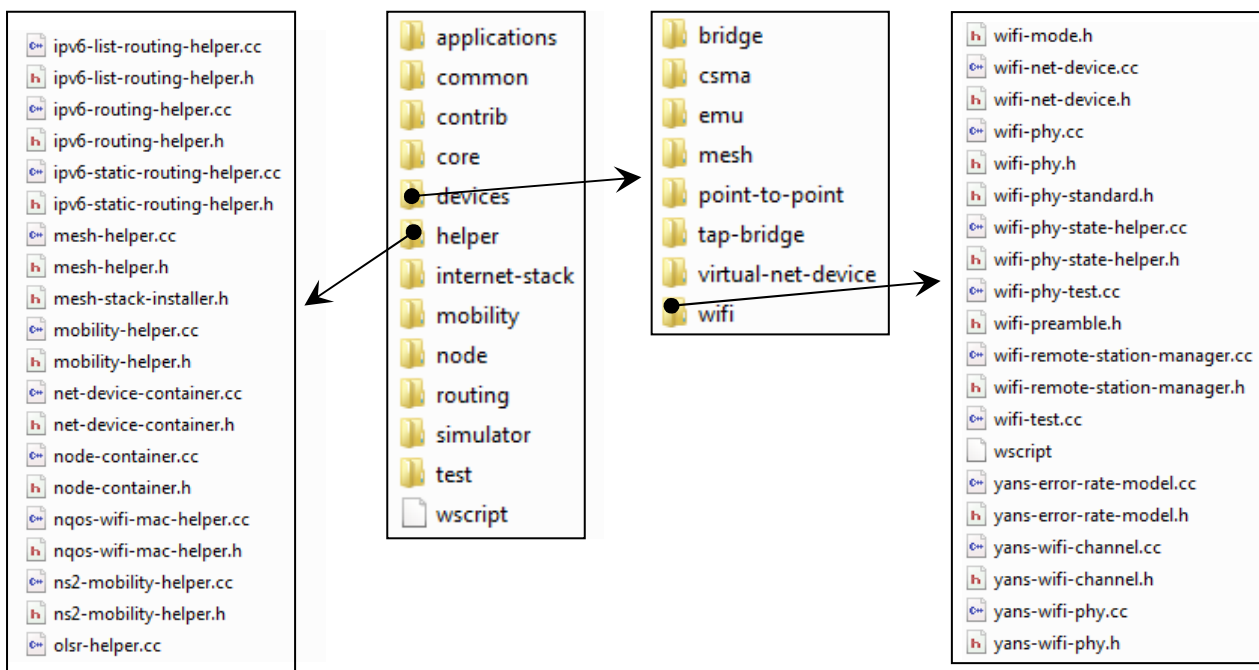


Figure 13: ns3 3.7 source organization

In v.3.18 the focus is on the modular organization of the code, each module is structured independently; this makes much simpler to update/upgrade a module without impacting other ns3's modules, everything about a module is included in its own folder. The overall architecture is reported in Figure 14.

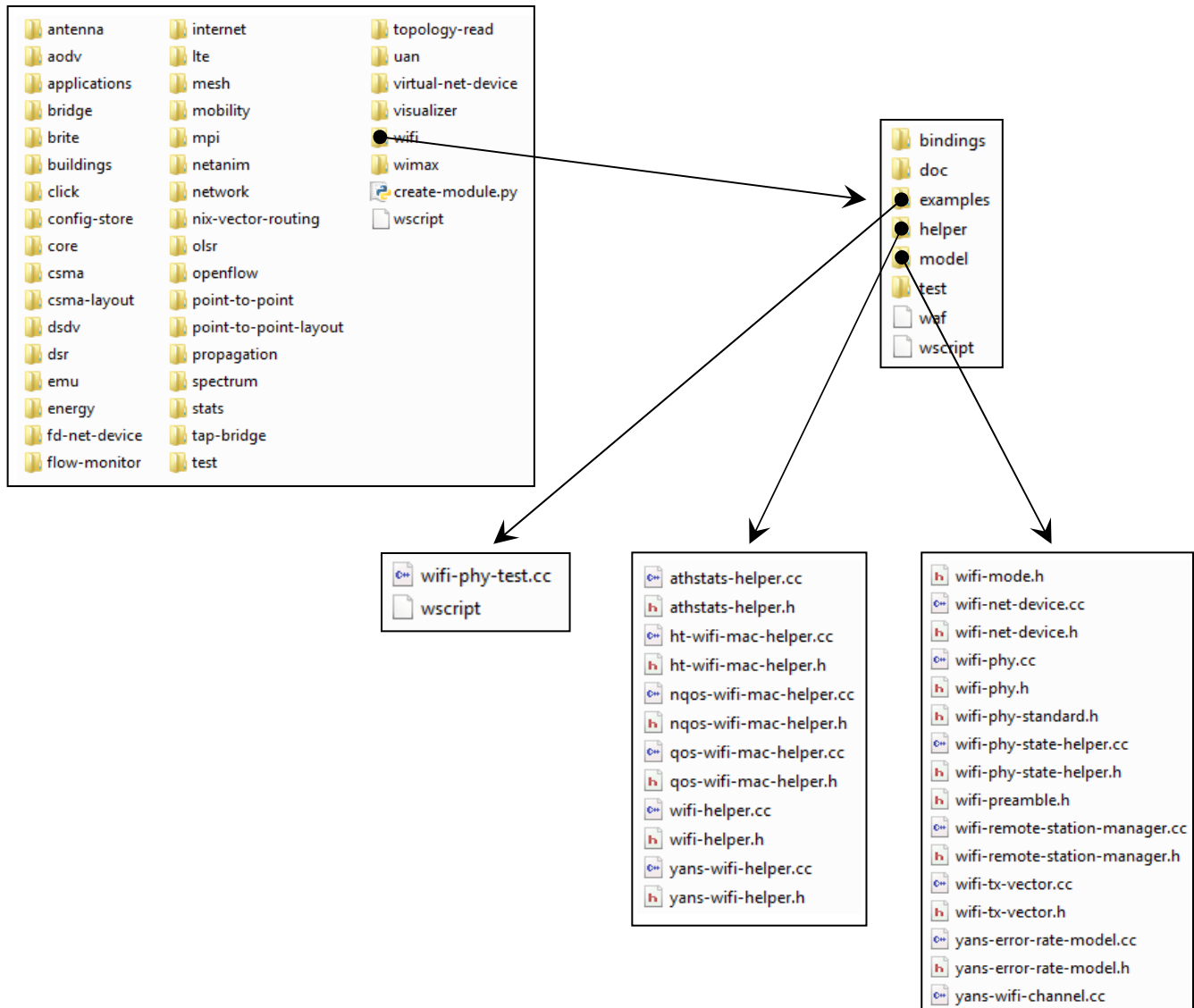


Figure 14: ns-3.18 source organization

7.2.2 Modules added to ns3 from original iTetris

The transmission of wireless messages between iTetris nodes has been implemented in ns-3 following ETSI's ITSC architecture. The Figure 15 shows the implemented ns-3 modules, and the interaction between ns-3 and the iCS using the iTetris iNCI interface.

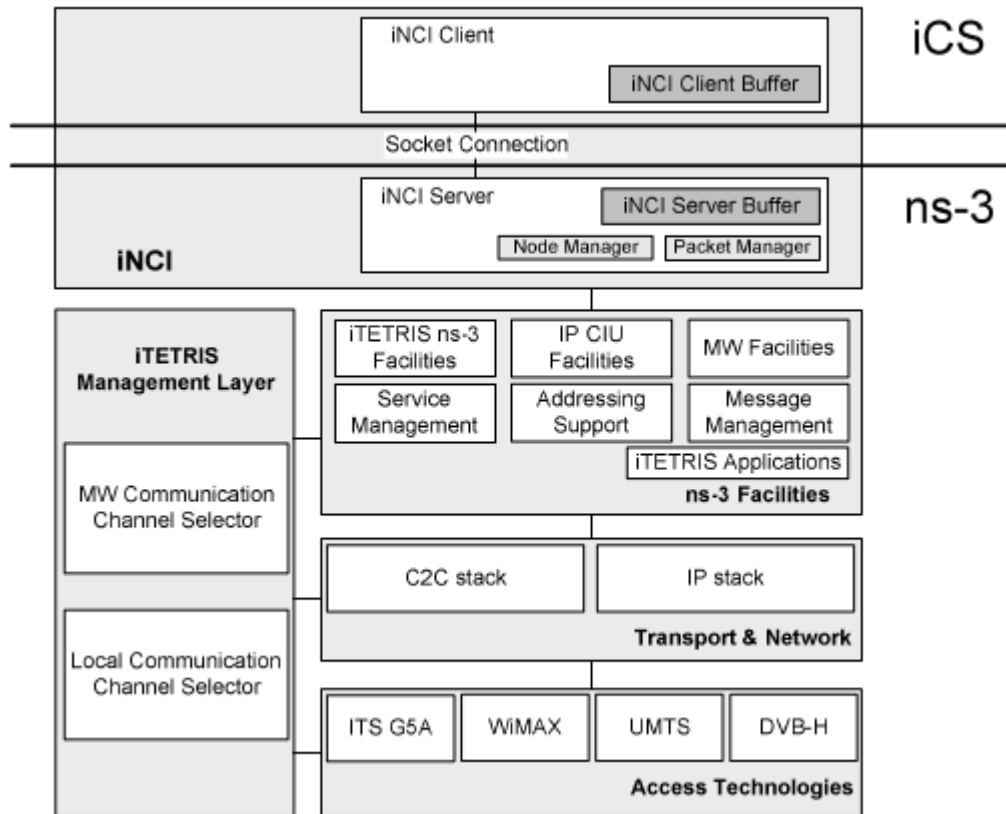


Figure 15: ETSI's ITSC architecture

- iNCI Server: implements specific primitives that allow the iCS to dynamically create new ns-3 nodes and update their position and speed at each simulation time step (iNCI Node Manager) and offers primitives for the iCS to activate and deactivate in ns-3 the simulation of message transmissions.
- ns-3 Facilities: On each ns-3 node, its Facilities generate the messages to be transmitted, and implement the functions required to forward these messages to the lower layers of the communications protocol stack. The ns-3 Facilities are also responsible to forward received simulated messages to the iNCI Packet Manager, so that the iCS can retrieve the required information about these messages
- Transport & Network
 - C2C stack: based on ETSI ITSC's GeoNetworking stack offers the transport and networking capabilities needed in vehicular environments using ETSI's ITS G5 radio access technology
- Access technologies:
 - ITS G5A: evolution of the IEEE 802.11a standard including communication functions required to operate in rapidly varying vehicular environments and exchange messages with short connection establishment delays
 - UMTS
 - DVB-H

- Management layer: a vertical cross-layer component of an ITS station that can coordinate the operation of the Access Technologies, Transport & Network, Facilities and Application layers

Layers reported in Figure 15 have been developed in ns3 v.3.7 adding a lot of modules. During the porting all modules added to ns3 from iTetris project have been added in ns3 v.3.18; each module has the structure reported in **Erreur ! Source du renvoi introuvable.** in order to satisfy the new modular architecture foreseen in ns3 v.3.18. The following are the added modules:

- Inci: Contains classes related to the communication between the iCS and ns3.
- Inci-utils: Contains installers for each technology and the iTetris -nodeManager definition.
- Facilities: Contains files mostly related to transmission and message delivery (services) processes.
- iTetrisNode: Contains files which implements some classes related to the C2C stack.
- iTetrisApplications: Contains application layers related to each transmission protocol (DVB-H, UMTS, WIMAX, etc). All of them inherit from iTetris-Application which marks the necessary functionalities that one new application must support in order to be employed by an ns-3 node in iTetris.
- C2c-stack: Contains all the necessary files to implement the C2C protocol in a node.
- visibilityMap: Contains files required by the winner-models/VisibilityMapModel to obtain the visibility conditions between a given pair of locations in a road network .
- winner-models: Contains all files required to implement a VisibilityModel that provides the visibility conditions between transmitter and receiver based on the position of a set of obstacles.
- Geo-routing: Contains all the necessary files to implement the Geo Anycast, Geo Broadcast, Geo Unicast and Topo Broadcast protocols.
- Geolib: Contains some useful libraries files.
- Dvb-h: Contains all the necessary files to implement the protocol Dvb-h.
- Umts: Contains all the necessary files to implement the protocol Umts.
- itetris-station-mgmt: Contains management classes for each technology to provide vital functionalities like getting the ip /c2c address of a node or obtaining the closest RSU or Base Station for a particular vehicle.

7.2.3 Sources files changed in ns3 from iTetris

iTetris project changed many files of standard ns3 release in order to fully support new added functionalities and protocols; during the porting, and when possible, all changes have been repeated in ns3 v.3.18 otherwise, if not possible, these changes have been adapted to the new architecture of ns3 v.3.18 classes. The following are the most important files that have required significant effort:

Internet	Internet-stack-helper.h	Wimax	Bs-net-device.h
	Internet-stack-helper.cc		Bs-net-device.cc
	Internet-trace-helper.h		Bs-net-device.h
	Internet-trace-helper.cc		Bs-scheduler-simple.cc
Network	Node.h		Connection-manager.h
	Node.cc		Connection-manager.cc
	Net-device.h		Ipcs-classifier-record.cc
Mobility	Mobility-helper.h		Mac-messages.h
	Mobility-helper.cc		Service-flow-manager.h
	Mobility-model.h		Service-flow-manager.cc
	Mobility-model.cc		Service-flow-record.h
Wifi	Wifi-helper.h		Simple-ofdm-wimax-channel.h
	Wifi-helper.cc		Simple-ofdm-wimax-channel.cc
	Yans-wifi-helper.h		Simple-ofdm-wimax-phy.h
	Yans-wifi-helper.cc		Simple-ofdm-wimax-phy.cc
	Edca-tcop-n.h		Ss-manager.h
	Edca-tcop-n.cc		Ss-manager.cc
	Wifi-mac-queue.h		Ss-net-device.h
	Wifi-mac-queue.cc		Ss-net-device.cc
	Wifi-mac.h		Ss-record.h
	Wifi-mac.cc		Ss-record.cc
	Wifi-net-device.h		Ss-scheduler.h
	Wifi-net-device.cc		Ss-scheduler.cc
	Wifi-phy.h		Wimax-channel.h
	Wifi-phy.cc		Wimax-channel.cc
	Wifi-phy-standard.h		Wimax-mac-queue.h
	Yans-wifi-channel.h		Wimax-mac-queue.cc
	Yans-wifi-channel.cc		Wimax-net-device.h
	Yans-wifi-phy.h		Wimax-net-device.cc
	Yans-wifi-phy.cc		Wimax-phy.h

			Wimax-phy.cc
			Wimax-helper.h
			Wimax-helper.cc

Table 2: Files modified

7.2.4 The ns3 (integrated in iTetris) final architecture

Some files have required extensions due to the fact that some functionality foreseen in ns3 v.3.7 and useful for iTetris are not anymore supported in ns3 v.3.18. In order to make fully compatible the ported code it has been necessary to add to ns3 v3.18 some removed functionalities. In particular “RunOneEvent” and “Next” functions has been added in the followings files in order to give the possibility to run one single event and to move to the next one step by step; these modalities are essentials for the run-time loop of iTetris.

Core	Simulator.h	Core	Realtime-simulator-impl.h
	Simulator.cc		Realtime-simulator-impl.cc
	Default-simulator-impl.h	Mpi	Distributed-simulator-impl.h
	Default-simulator-impl.cc		Distributed-simulator-impl.cc

Table 3: Restored core functionalities

Some files have been added to existing modules of ns3 3.18 in order to fully implement iTetris functionalities; the following are the more relevant:

Mobility	Geo-utils.h	Internet	C2c-routing-helper.h
	Geo-utils.cc		C2c-routing-helper.cc
	Itetris-mobility-model.h		C2c-list-routing-helper.h
	Itetris-mobility-model.cc		C2c-list-routing-helper.cc
	Itetris-pos-helper.h		C2c-list-routing.h
	Itetris-pos-helper.cc		C2c-list-routing.cc
Wifi	Mcs-tag-wifi-manager.h	Wimax	Bs-command-manager-container.h
	Mcs-tag-wifi-manager.cc		Bs-command-manager-container.cc
	Switching-manager.h		Bs-command-manager.h
	Switching-manager.cc		Bs-command-manager.cc
	Switching-manager-helper.h		ss-command-manager.h
	Switching-manager-helper.cc		ss-command-manager.cc
	Switching-manager-container.h		ss-command-manager-container.h
	Switching-manager-container.cc		ss-command-manager-container.cc
			Wimax-version-type.h
			Wimax-command-manager-helper.h
			Wimax-command-manager-helper.cc

Table 4: Files added to original modules

The Figure 16 reports all modules supported from NS3 v.3.18 once completed the porting of iTetris modules.

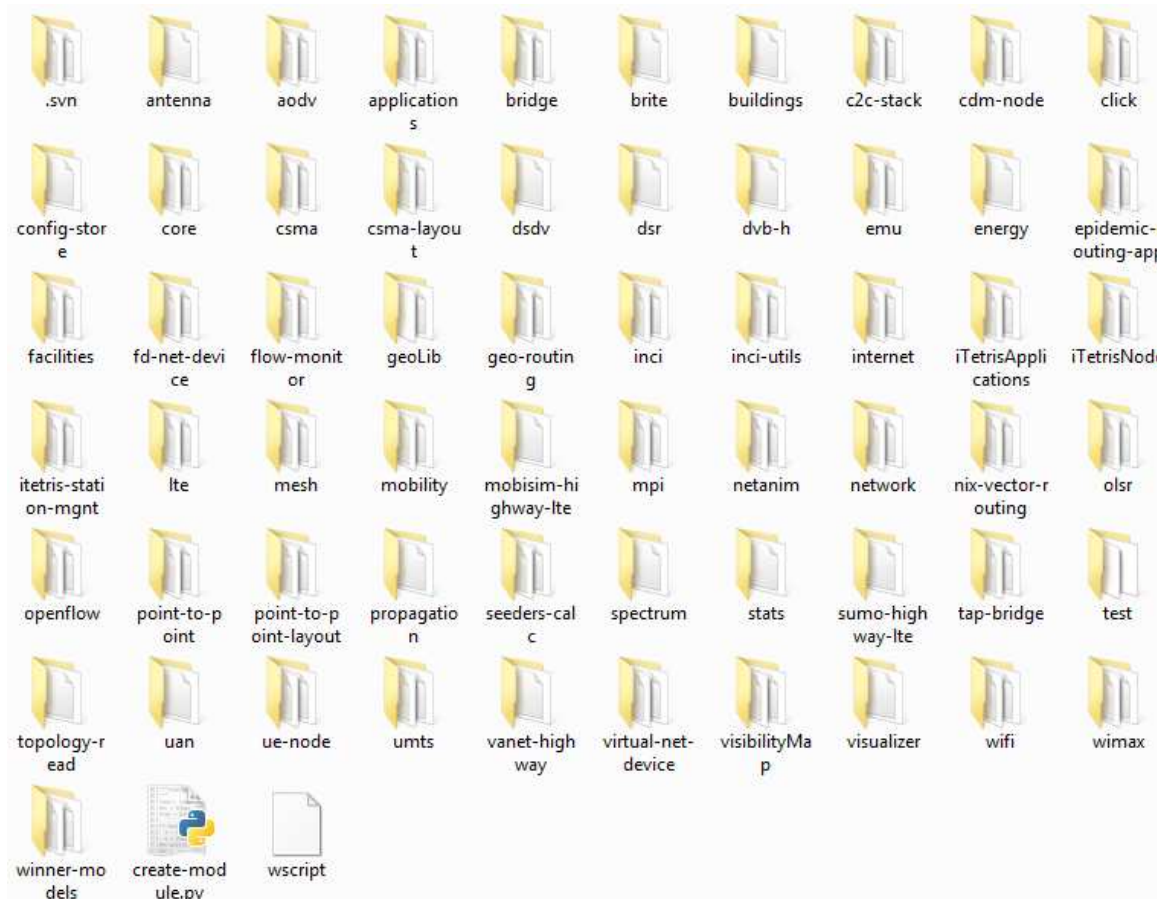


Figure 16: Final Ns3/iTetris src folder

7.3 Results

The porting has been fully completed and a new version of iTetris environment has been released with NS3 in its 3.18 version.

The application “community-demo-app” delivered with the official release of iTetris has been used in order to test the functionalities of the ported code. At the end of the execution of the simulations the ported code has the same behaviour of the original version of iTetris.

The Figure 17 is a snapshot showing the SUMO interface aspect once executed the command `iCS -c itetris-config-file.xml`. The original version of iTetris code proves the same behaviour of the new iTetris release which support ns3 3.18.

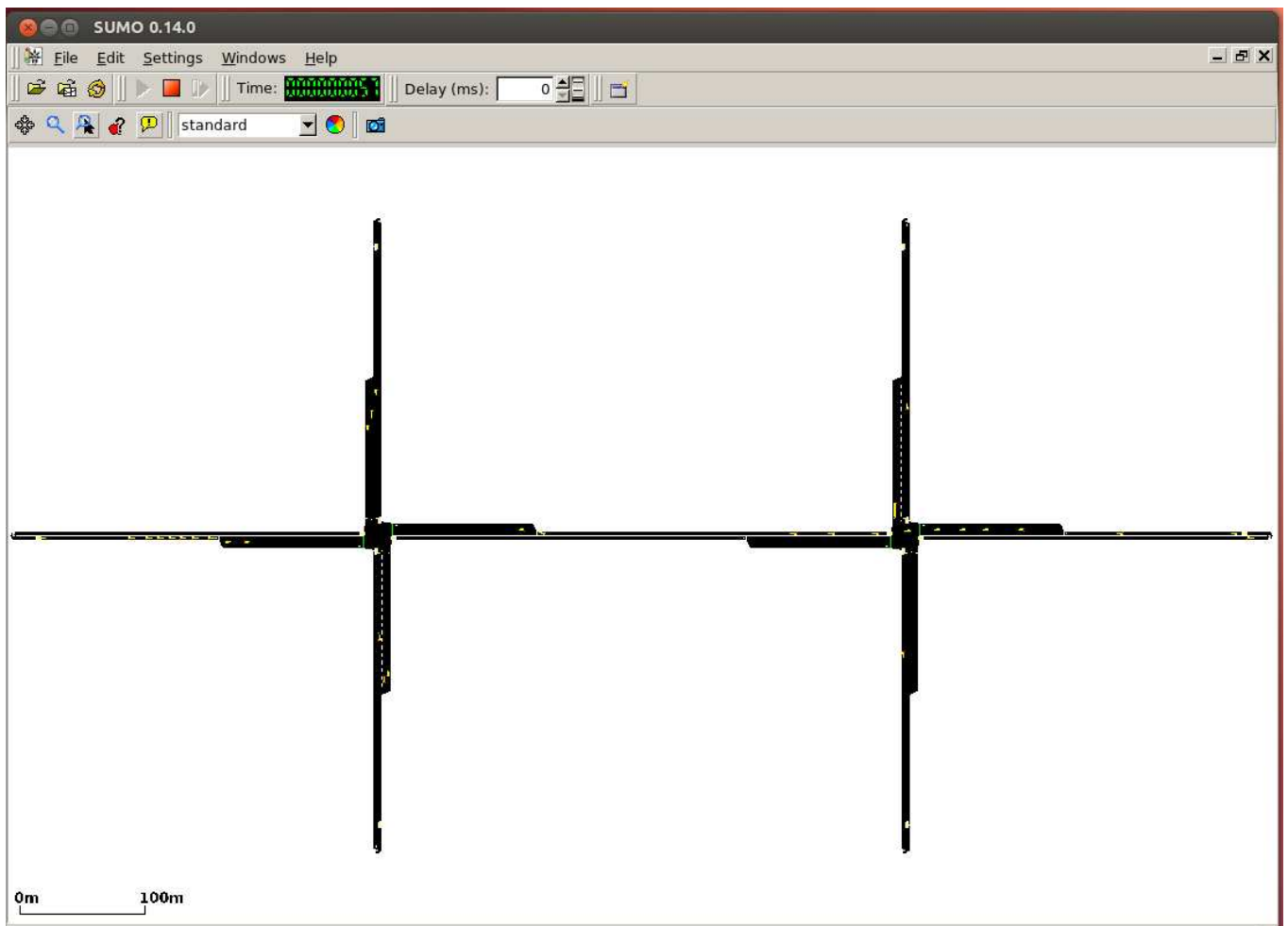


Figure 17: Sumo interface showing vehicles mobility during simulation

8 LTE integration activity

8.1 Why?

The porting activity described in the previous section allowed the iTetris platform to work using Ns3 v.3.18. However, for the complete support of the LTE communication technology provided by Ns3 v.3.18, additional effort has been required to extend again iTetris with the aim of making it able to use, and provide to the user, LTE features. This activities is named “LTE integration”, that is the integration of the LTE features capability in the iTetris platform we obtained after the porting activity.

In other terms, the goal of this LTE integration activity, hence, was to extend and refine iTetris to let it use LTE features provided by Ns3 v.3.18.

To this aim the following steps have been conducted:

- 1) Identification of changes to realize into the (ported) iTetris and, in particular, into the code of both iCS and Ns3 (v.3.18) modules of iTetris.
- 2) Execution of the identified changes by modifying and adding needed files.
- 3) Creation and execution of a simple application that uses LTE features, for testing the LTE integration.
- 4) Verification of the iTetris platform integrity with respect to the changes performed during the LTE integration, that is checking that the (original) behaviour of iTetris has not been compromised by the LTE integration.

This section describes both the process and the activities conducted to realize the LTE integration as well as the obtained iTetris architecture.

8.2 Architecture

With the aim of understanding how to integrate the LTE features in the iTetris platform, a preliminary analysis of the communication technologies (e.g., UMTS, Wave and WiMax) already integrated into iTetris has been conducted. In other terms, these technologies (and in particular UMTS), have been adopted as reference points to identify files and modules that had to be added or modified for enabling the LTE integration.

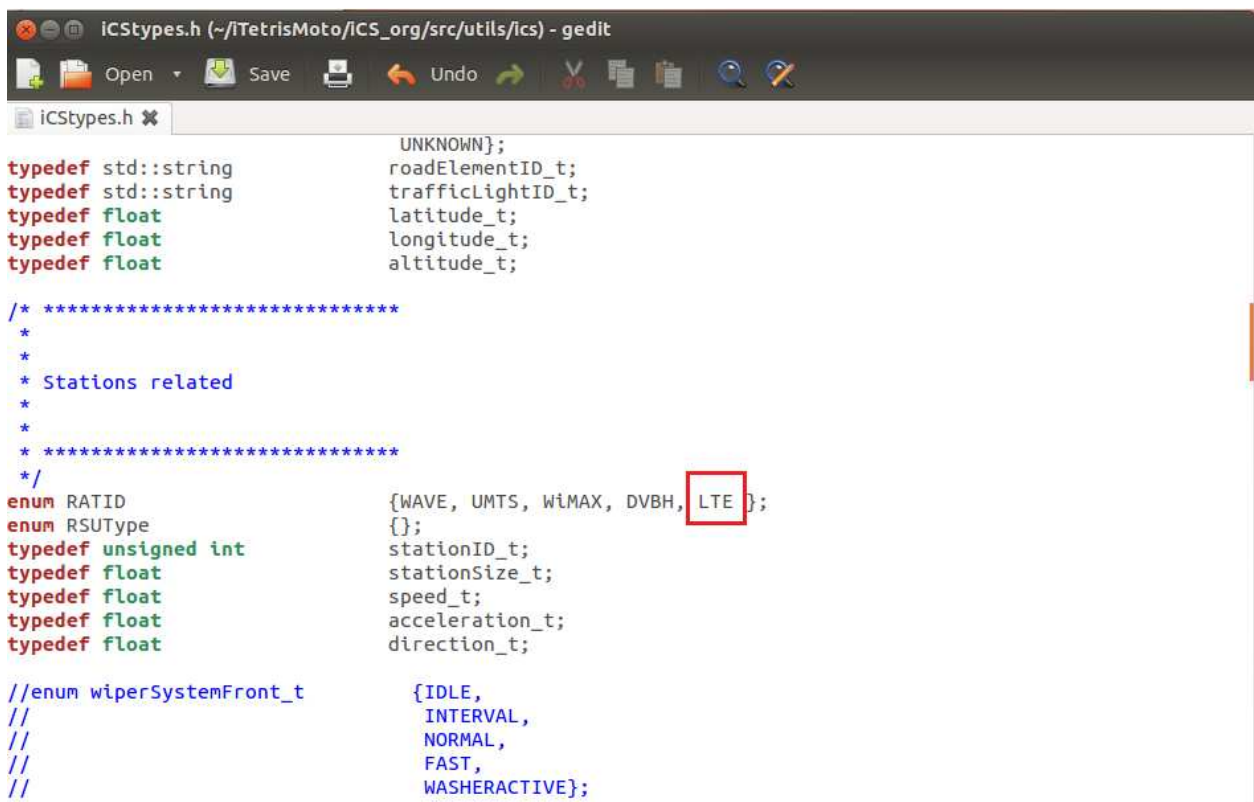
To analyse the iTetris implementation of such communication technologies, source code analysis such as grep-based textual searches of relevant keywords (e.g., “UMTS”), dependency analysis of the code modules and functions, and manual inspection of the iTetris code (in both iCS and Ns3) have been performed. Hence, a list of candidate files and modules of iTetris have been identified and then manually inspected to detect and identify those changes (e.g., files modification and addition) that were performed to the original iTetris code for adding a (new) communication mean, such as UMTS. This list of changes has been adopted as starting point to define the changes required to add LTE in the set of communication technologies provided by the iTetris platform.

8.2.1 Changes into iCS

The changes required in the iCS code are mainly aimed at:

- (i) Setting up constant values (representing command identification codes) for the selection of command to be executed in the iCS, Ns3 and Sumo modules;
- (ii) Allowing both selection and use of the LTE communication technology for wireless communications;
- (iii) Refactoring operations in the original iTetris implementation to facilitate the reuse of the original code written to use communication technology like WAVE, UMTS.

Figure 18 shows an example of code change required for extending the value of a command variable. In particular, it is related to the file ICSTypes.h of the iCS module. This file lists the communication means allowed by iCS: hence, LTE has to be added to this list as shown in the figure.



```

ICSTypes.h (~/ITetrisMoto/iCS_org/src/utils/ics) - gedit
Open Save Undo
icTypes.h x
typedef std::string UNKNOWN;
typedef std::string roadElementID_t;
typedef float trafficLightID_t;
typedef float latitude_t;
typedef float longitude_t;
typedef float altitude_t;

/* *****
 *
 *
 * Stations related
 *
 * *****
 */
enum RATID {WAVE, UMTS, WiMAX, DVBH, LTE};
enum RSUType {};
typedef unsigned int stationID_t;
typedef float stationSize_t;
typedef float speed_t;
typedef float acceleration_t;
typedef float direction_t;

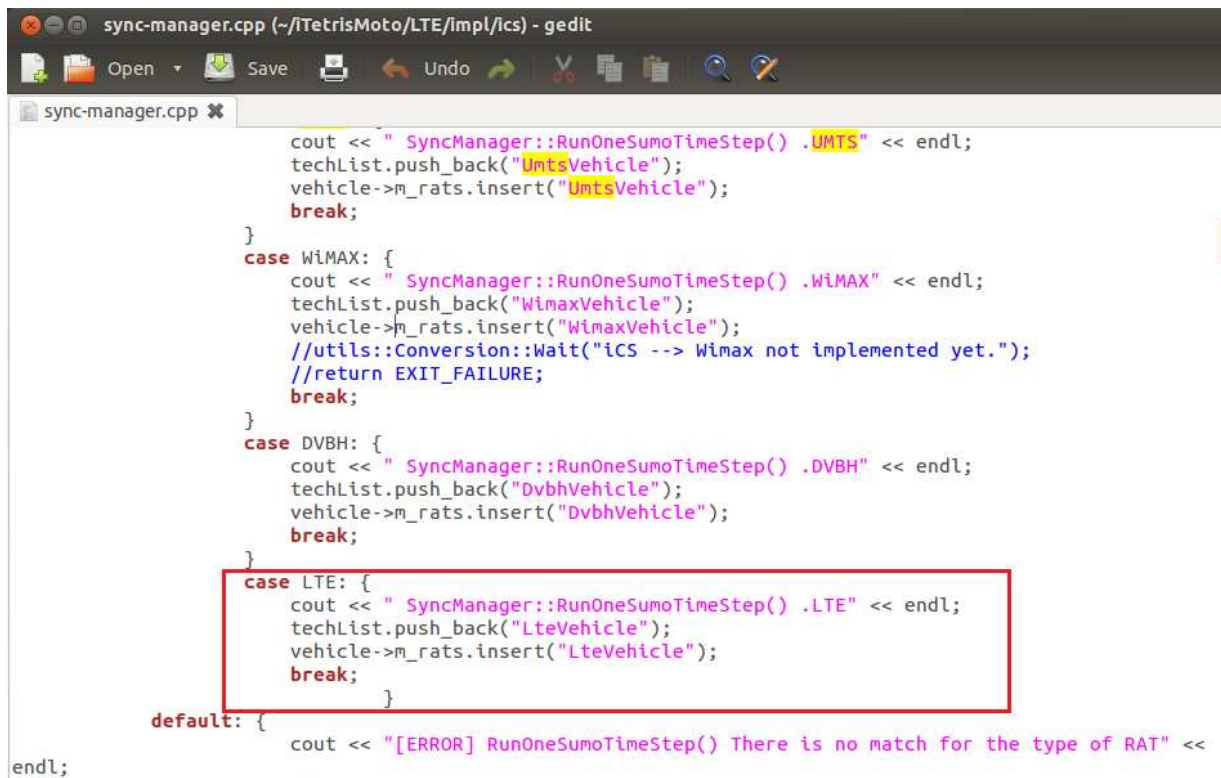
//enum wiperSystemFront_t {IDLE,
//                          INTERVAL,
//                          NORMAL,
//                          FAST,
//                          WASHERACTIVE};

```

Figure 18: A first example of code change (in the highlighted box the added code) in an iCS files

Figure 19 shows an example of code change required to allow the selection of the LTE technology for communication purposes. In detail, the figure shows a fragment of code of the file sync-manager.cpp, in the iCS module. This file manages the runtime-loop iteration cycle used by iTetris in the simulations: execution of Ns3, SUMO, and ITS application, and preparation to the execution of the next simulation step. Similar changes have been performed in other files of iCS (see next sections for additional details).

Figure 20 shows a third example concerning a refactoring operation of the original iCS code aiming at allowing the selection of communication technologies different from Wave and UMTS.



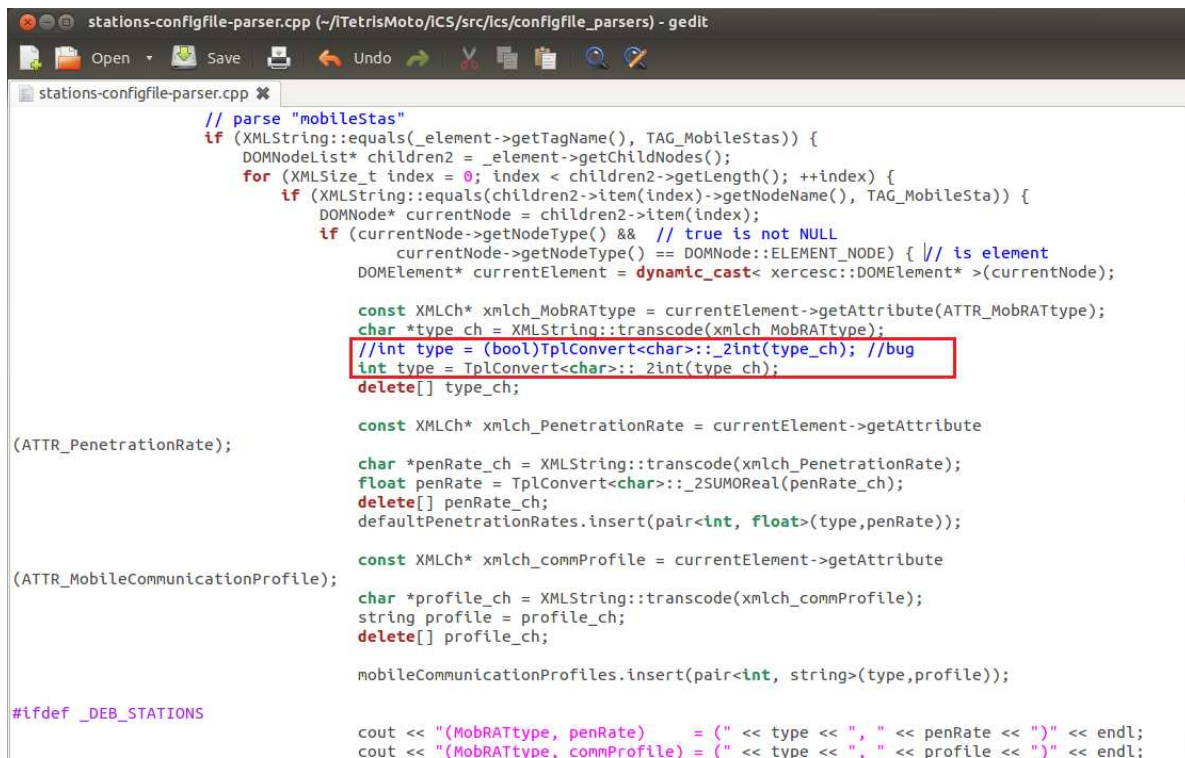
```

sync-manager.cpp (~/ITetrisMoto/LTE/impl/ics) - gedit

cout << " SyncManager::RunOneSumoTimeStep() .UMTS" << endl;
techList.push_back("UmtsVehicle");
vehicle->m_rats.insert("UmtsVehicle");
break;
}
case WiMAX: {
cout << " SyncManager::RunOneSumoTimeStep() .WiMAX" << endl;
techList.push_back("WimaxVehicle");
vehicle->m_rats.insert("WimaxVehicle");
//utils::Conversion::Wait("ics --> Wimax not implemented yet.");
//return EXIT_FAILURE;
break;
}
case DVBH: {
cout << " SyncManager::RunOneSumoTimeStep() .DVBH" << endl;
techList.push_back("DvbhVehicle");
vehicle->m_rats.insert("DvbhVehicle");
break;
}
case LTE: {
cout << " SyncManager::RunOneSumoTimeStep() .LTE" << endl;
techList.push_back("LteVehicle");
vehicle->m_rats.insert("LteVehicle");
break;
}
default: {
cout << "[ERROR] RunOneSumoTimeStep() There is no match for the type of RAT" <<
endl;
}

```

Figure 19: A second example of code change (in the highlighted box the added code) in an iCS file



```

stations-configfile-parser.cpp (~/ITetrisMoto/ICS/src/ics/configfile_parsers) - gedit

// parse "mobileStas"
if (XMLString::equals(_element->getTagName(), TAG_MobileStas)) {
DOMNodeList* children2 = _element->getChildNodes();
for (XMLSize_t index = 0; index < children2->getLength(); ++index) {
if (XMLString::equals(children2->item(index)->getNodeName(), TAG_MobileSta)) {
DOMNode* currentNode = children2->item(index);
if (currentNode->getNodeType() && // true is not NULL
currentNode->getNodeType() == DOMNode::ELEMENT_NODE) { // is element
DOMELEMENT* currentElement = dynamic_cast<xercesc::DOMELEMENT*>(currentNode);

const XMLCh* xmlch_MobRATtype = currentElement->getAttribute(ATTR_MobRATtype);
char *type_ch = XMLString::transcode(xmlch_MobRATtype);
//int type = (bool)TplConvert<char>::2int(type_ch); //bug
int type = TplConvert<char>::2int(type_ch);
delete[] type_ch;

const XMLCh* xmlch_PenetrationRate = currentElement->getAttribute
(ATTR_PenetrationRate);
char *penRate_ch = XMLString::transcode(xmlch_PenetrationRate);
float penRate = TplConvert<char>::2SUMOReal(penRate_ch);
delete[] penRate_ch;
defaultPenetrationRates.insert(pair<int, float>(type, penRate));

const XMLCh* xmlch_commProfile = currentElement->getAttribute
(ATTR_MobileCommunicationProfile);
char *profile_ch = XMLString::transcode(xmlch_commProfile);
string profile = profile_ch;
delete[] profile_ch;

mobileCommunicationProfiles.insert(pair<int, string>(type, profile));

#ifdef _DEB_STATIONS
cout << "(MobRATtype, penRate) = (" << type << ", " << penRate << ")" << endl;
cout << "(MobRATtype, commProfile) = (" << type << ", " << profile << ")" << endl;
#endif
}
}
}
}

```

Figure 20: An example of code change (in the highlighted box the added code) in an iCS file

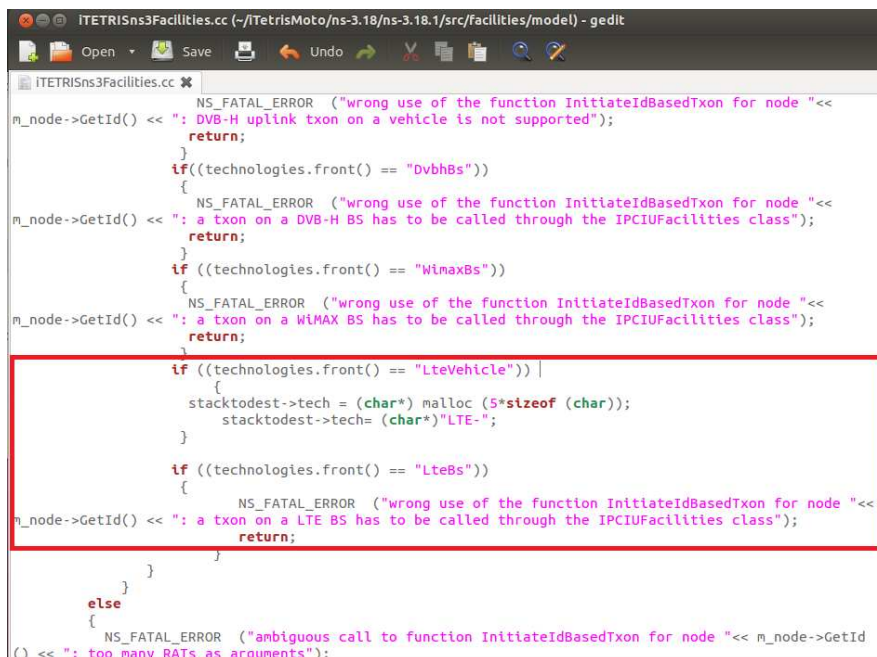
8.2.2 Changes into ns3 (v3.18)

The changes required in the Ns3 code are mainly aimed at:

- 1) allowing Ns3 to provide features that can be called by iTetris (e.g., by the iCS module) to install the communication technology in each node involved in the simulation and manage step-by-step the Ns3-based communication;
- 2) extending the LTE implementation (i.e., the LENA module integrated into Ns3 v.3.18) to be adopted by iTetris as a communication technology for wireless communications.

The former type of changes mainly concern four modules of the Ns3 adopted by (the ported) iTetris:

- iTetrisApplications. This module is devoted to create and associate LTE devices to nodes (e.g., a fixed RSU, a mobile vehicle) involved in a simulation scenario. An LTE device, for example, can initialize a net device, add a MAC address, and so on.
- Facilities. This module is devoted to allow the selection of the adequate communication type (e.g., Id-based, unicast, geobroadcast, IP-based, broadcast) for the selected communication technology (e.g., LTE). Moreover, it also controls the distribution of messages of type CAM and DEN among the nodes (fixed and mobile) involved in a simulation. This module, for example, allows the unicast transmissions from a user equipment node (e.g., vehicle) toward a road side unit or another user equipment node (e.g., vehicle).
- Itetris-station-management. This module is devoted to provide scanning functionalities to let a node (either mobile or fixed node) use LTE technology to analyse and scan its context during a simulation to look for other nodes and connect to them for communication purposes.
- Inci-utils. This module is devoted to: provide functionalities for loading and parsing configuration files containing LTE configuration information; and install the LTE protocol on each node involved in the simulation scenario.



```

ITETRISns3Facilities.cc (~/ITetrisMoto/ns-3.18/ns-3.18.1/src/facilities/model) - gedit
NS_FATAL_ERROR ("wrong use of the function InitiateIdBasedTxon for node "<<
m_node->GetId() << ": DVB-H uplink txon on a vehicle is not supported");
return;
}
if((technologies.front() == "DvbHs"))
{
NS_FATAL_ERROR ("wrong use of the function InitiateIdBasedTxon for node "<<
m_node->GetId() << ": a txon on a DVB-H BS has to be called through the IPCIUFacilities class");
return;
}
if ((technologies.front() == "WimaxBs"))
{
NS_FATAL_ERROR ("wrong use of the function InitiateIdBasedTxon for node "<<
m_node->GetId() << ": a txon on a WiMAX BS has to be called through the IPCIUFacilities class");
return;
}
if ((technologies.front() == "LteVehicle")) {
{
stacktodestd->tech = (char*) malloc (5*sizeof (char));
stacktodestd->tech= (char*)"LTE-";
}
}
if ((technologies.front() == "LteBs"))
{
NS_FATAL_ERROR ("wrong use of the function InitiateIdBasedTxon for node "<<
m_node->GetId() << ": a txon on a LTE BS has to be called through the IPCIUFacilities class");
return;
}
}
else
{
NS_FATAL_ERROR ("ambiguous call to function InitiateIdBasedTxon for node "<< m_node->GetId
() << ": too many RATs as arguments");
}

```

Figure 21: An example of code change (in the highlighted box the added code) in the code of ns3

Figure 21 shows an example of change done in the code of the file `InitiateIdBasedTxon.cpp` of the Ns3 version used by iTetris to enable the unicast transmission from an LTE user equipment node.

The latter type of changes concern the implementation of the LTE module integrated in the Ns3 version used by iTetris in order to allow dynamic creation and deletion of simulation nodes. The original LTE implementation does not allow the dynamic creation/deletion of nodes in a simulation scenario. iTetris, in fact, relies on the capability of dynamically create and remove nodes during a simulation. However, this feature is not natively supported by Ns3 and in particular by the LTE implementation integrated in the considered Ns3 version.

8.2.3 Files and modules changed and added to iTetris

After the porting of iTetris to let it use Ns3 v.3.18, several files have been modified and added in order to realize the LTE integration as described in the previous subsections, thus allowing iTetris to provide the LTE technology for simulation purposes.

Table 5 lists the main files that have been added or modified/changed with the aim of integrating the LTE module into the iTetris platform. The table contains the following information:

- 1) The iTetris module related to the file of interest.
- 2) The name and complete path of the file in the module.
- 3) The activity that has been conducted on the file. Indeed, listed files have been modified/changed, in case they already exist in the initial iTetris implementation, or added, in case they did not exist in the initial iTetris, iCS and Ns3 modules.

Module	File	Activity
iCS	<code>src/utils/ics/iCSTypes.h</code>	Changed
iCS	<code>src/ics/wirelesscom_sim_communicator/ns3-client.h</code>	Changed
iCS	<code>src/ics/wirelesscom_sim_communicator/ns3-comm-constants.h</code>	Changed
iCS	<code>src/ics/wirelesscom_sim_communicator/wireless-communication-simulator-communicator.h</code>	Changed
iCS	<code>src/ics/itetris-simulation-config.h</code>	Changed
iCS	<code>src/ics/sync-manager.cpp</code>	Changed
iCS	<code>src/ics/configfile_parsers/stations-configfileparser.cpp</code>	Changed
iCS	<code>src/ics/applications_manager/AppMessageManager.cpp</code>	Changed
iCS	<code>src/ics/application_manager/AppMessageManager.h</code>	Changed
iCS	<code>src/ics/wirelesscom_sim_communicator/ns3-client.cpp</code>	Changed
iCS	<code>src/ics/wirelesscom_sim_communicator/ns3-client.h</code>	Changed
Ns3	<code>scratch/confLteVehicle.xml</code>	Added
Ns3	<code>scratch/confLteBs.xml</code>	Added
Ns3	<code>scratch/configTechnologies-ics.xml</code>	Added

Ns3	scratch/stations-config-file.xml	Added
Ns3	scratch/main-inci5.cc	Changed
Ns3	src/applications/helper/Lte-App-helper.cc	Added
Ns3	src/applications/helper/Lte-App-helper.h	Added
Ns3	src/applications/model/Lte-app.cc	Added
Ns3	src/applications/model/Lte-app.h	Added
Ns3	src/inci-utils/model/iTETRISNodeManager.cc	Changed
Ns3	src/inci-utils/model/lte-bs-installer.cc	Added
Ns3	src/inci-utils/model/lte-bs-installer.h	Added
Ns3	src/inci-utils/model/lte-installer.cc	Added
Ns3	src/inci-utils/model/lte-installer.h	Added
Ns3	src/inci-utils/model/lte-vehicle-installer.cc	Added
Ns3	src/inci-utils/model/lte-vehicle-installer.h	Added
Ns3	src/inci-utils/wscript	Changed
Ns3	src/inci/ns3-server.cc	Changed
Ns3	src/inci/tcpip/config.h	Changed
Ns3	src/internet/helper/internet-stack-helper.cc	Changed
Ns3	src/itetris-station-mgnt/lte-bs-mgnt.cc	Added
Ns3	src/itetris-station-mgnt/model/lte-bs-mgnt.h	Added
Ns3	src/itetris-station-mgnt/model/lte-vehicle-scan-mngr.cc	Added
Ns3	src/itetris-station-mgnt/model/lte-vehicle-scan-mngr.h	Added
Ns3	src/itetris-station-mgnt/model/vehicle-sta-mgnt.h	Changed
Ns3	src/itetris-station-mgnt/wscript	Changed
Ns3	src/lte/model/epc-ue-nas.cc	Changed
Ns3	src/lte/model/lte-net-device.cc	Changed
Ns3	src/lte/model/lte-ue-net-device.cc	Changed
Ns3	src/lte/model/lte-ue-phy.cc	Changed
Ns3	src/lte/model/lte-bs-manger.cpp	Added
Ns3	src/lte/model/lte-bs-manger.h	Added
Ns3	src/lte/model/lte-vehicle -manger.cpp	Added
Ns3	src/lte/model/lte-vehicle-manger.h	Added
Ns3	src/mobility/helper/geo-utils.cc	Changed
Ns3	src/mobility/helper/geo-utils.h	Changed

Table 5: List of main files that have been changed and/or added for the LTE integration activity

8.2.4 The final iTetris architecture

Figure 22 shows the resulting modules of the iTetris platform after the LTE integration. The figure shows all the modules of iCS, Ns3 and Sumo that implement iTetris. Moreover, the figure highlights modules (see the black triangle in the module icons) that have been modified/added during the LTE integration activities: that is those modules in which at least one file has been changed or added during the LTE integration. Overall, one can notice that no modifications have been needed only for Sumo, while a non-trivial amount of modules have been modified for both iCS and Ns3.

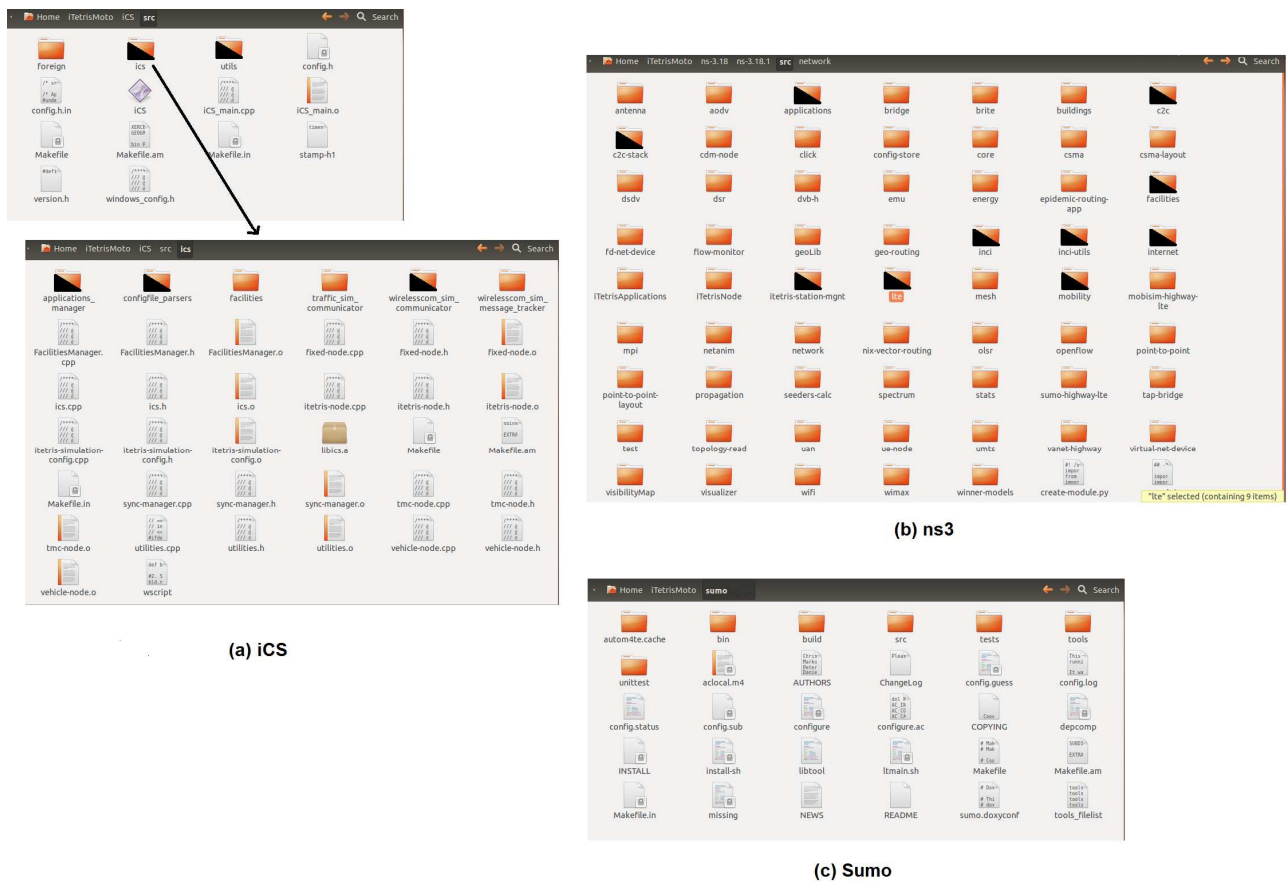


Figure 22: Overall architecture of iTetris after the LTE integration. Even if the name of files and modules (i.e. directories) is not expected to be visible in the figure, it shows the set of modules modified during the LTE integration. Such modifications are highlighted with black triangles in the module icons

8.3 Results

The LTE integration activity has been successfully completed and the new version of iTetris that provides the LTE capability has been released.

The application “community-demo-app” delivered with the original version of iTetris has been used to check that the integration of the LTE capability did not compromise the functionality provided by the original iTetris platform. Furthermore, a new application, named “community-lteDemo”, has been developed for specifically checking the LTE capability integrated to the new version of the iTetris platform. The next section will give details about this new application and about the results.

9 iTetris example Application

This section briefly presents the new application named “community-lteDemo” implemented in order to check the LTE integration.

9.1 Features

The developed example is a variant of the community demo application in which a mobile station (vehicle) and a fixed station (road side unit) are equipped with LTE technology and use it for communication purposes.

Figure 23 shows a snapshot of the Sumo interface of the example. The figure shows the network used by the application. The adopted network is composed of some streets linked by three road intersections, two of them controlled by traffic lights. The left road intersection is also equipped with a road side unit (RSU) working with LTE technology, thus it is enriched with communication capability. Indeed, by using the LTE technology, the RSU can send speed advices to the vehicles approaching and crossing the monitored road intersection.

In the example application, a vehicle also equipped with the LTE capability is crossing the network and, in particular, the left intersection. When the vehicle approaches the left intersection it stops, that is the vehicle changes its speed to 0 since it receives a speed advice (suggesting a speed value corresponding to 0) from the RSU.

Figure 24 highlights a detail of the Sumo interface of the example by showing an instant of the simulation in which the vehicle, after being entered in the CAM area (see below), received the speed advice from the RSU and stops.

The RSU can send speed advices to all vehicles approaching the monitored intersection in a given area, named “CAM area”. Indeed CAM messages are exchanged between the RSU and each vehicle entering in this CAM area and thus that is closing to the RSU, in case this is allowed by the adopted communication technology. The size of a CAM area is a parameter of the application and it can be customized by the user. Having the possibility of limiting the CAM area is relevant for the scalability of simulations; it enables large-scale simulations in iTetris. By limiting the area of interest (i.e., the CAM area), the validity of the communications between nodes involved in the simulation is also limited only to the area of interest. This allows limiting the resources allocated to a simulation.

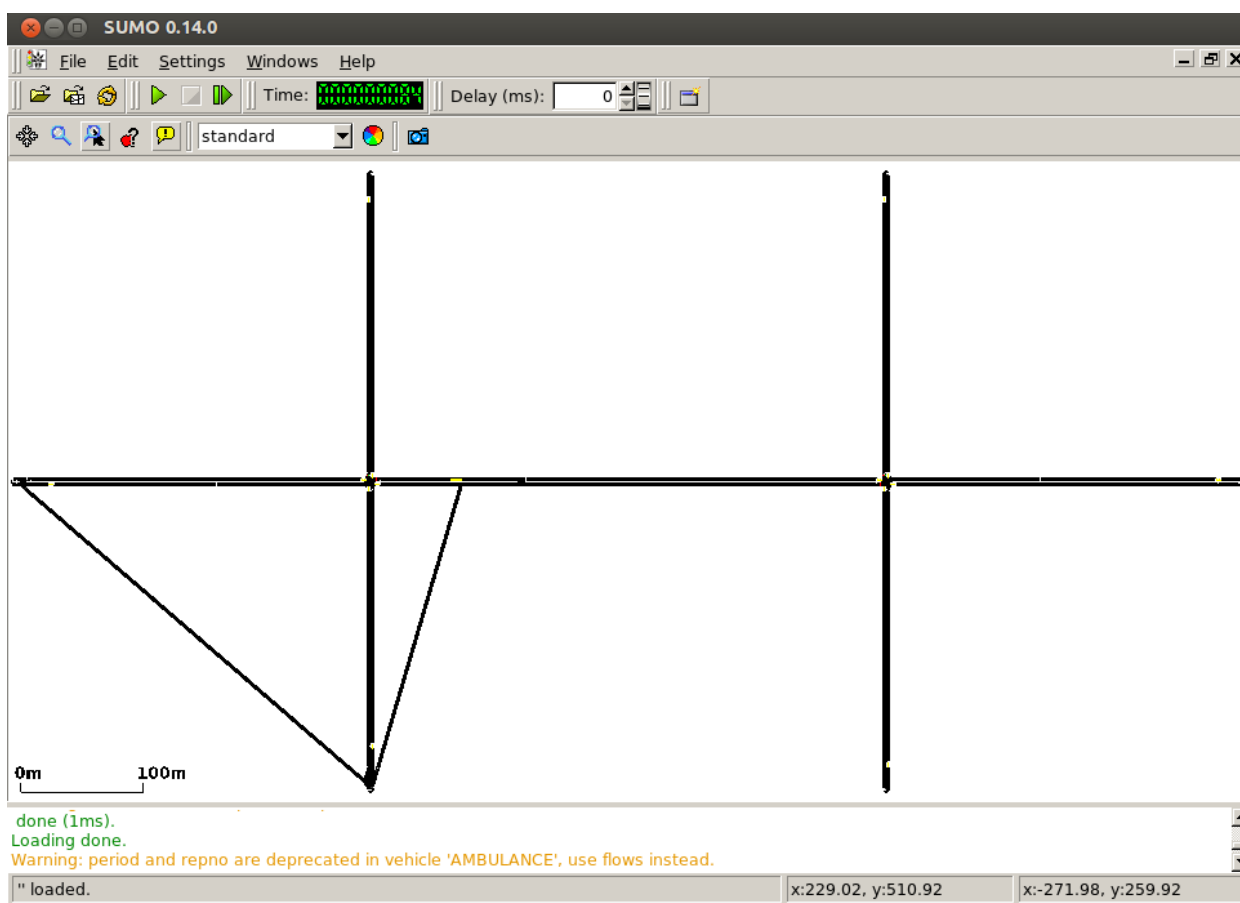


Figure 23: Sumo interface showing the vehicle mobility during the running of the example

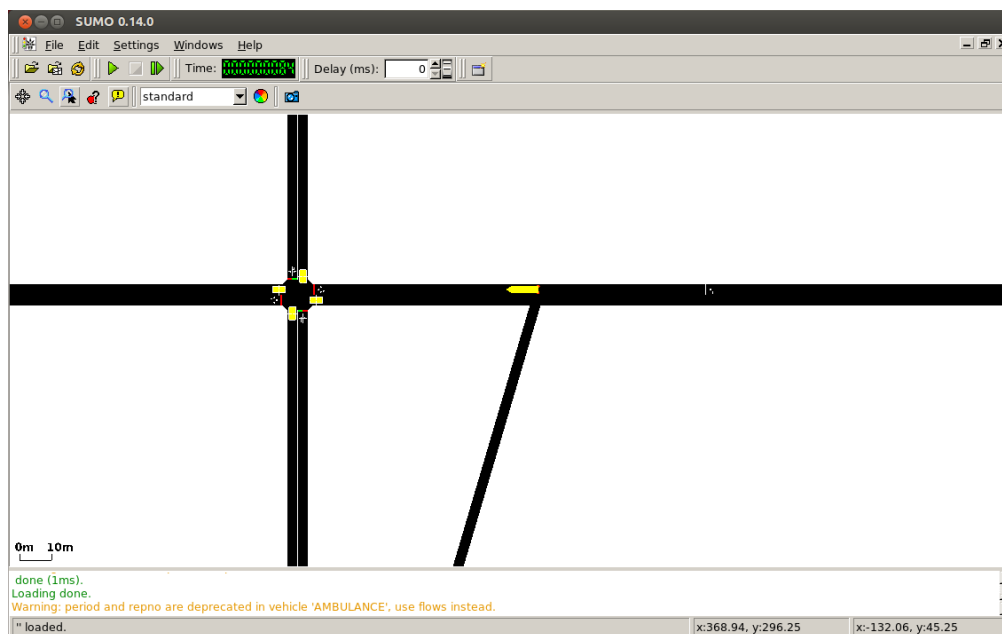


Figure 24: Detail of the Sumo interface of the example

9.2 Architecture

Table 6 lists the relevant files that implement the example application adopting the LTE technology for communication purposes.

Module	Directory	File	Intended purpose of the file
Source code	src/	app-commands-subscriptions-constants.h	It lists a set of command/id pairs used to agree between application and iCS, Ns3, and Sumo.
	src/	application-logic.cpp	It mainly implements the application logic.
	src/	application-logic.h	It mainly defines methods and variables used by the application logic implementation.
	src/	its_demo_app_main.cpp	It is the starting point of the application.
	src/	server.cpp	It handles the communications between iCS and the rest of iTetris (e.g., application, and Ns3).
	src/	server.h	It mainly defines methods and variables used by the communication handle implementation.
Simulation setting	example/	application-config-file.xml	It contains information that customizes the application under simulation. E.g., the name of the application; both IP address and port of the application; the actions to perform when iCS receives the result of a command execution from the application; definition of id of the allowed transmission type (such as unicast, broadcast).
	example/	facilities-config-file.xml	It contains information to configure the iCS facilities. E.g., name of the file containing the configuration of network and of mobile and fixed stations involved in the simulation.
	example/	itetris-config-file.xml	It contains information to run the program executing the simulation (e.g., to run Ns3 and Sumo), and for the configuration of log files.
	example/	itscoopdemoapp-config-file.xml	It contains the parameters of the application that can be customized by

			the user. E.g., the size of the CAM area.
	example/	LDMrules-config-file.xml	It contains information for the LDM configurations. E.g., default message life interval.
	example/	stations-config-file.xml	It contains information to configure both fixed and mobile stations involved in the simulation. E.g., the configuration of the communication technology to be used in the stations; the penetration rate of each involved technology; the position in the network of the fixed stations.
Ns3 setting	example/ns-3_files	configGeneral.txt	It contains information to configure all the parameters of Ns3.
	example/ns-3_files	configTechnologies-ics.xml	It contains information (e.g., name and path) about the configuration files for each communication technology used in the simulation.
	example/ns-3_files	confLteBs.xml	It contains information to configure the LTE communication technology for fixed stations (RSUs). It has to specify parameters to characterize the LTE communication channel (e.g., path loss and fading model), the physical layer (e.g., scheduler, Tx power, noise), the application layer as well.
	example/ns-3_files	confLteVehicle.xml	It contains information to configure the LTE communication technology for mobile stations (vehicles). It has to specify parameters to characterize the LTE communication channel (e.g., path loss and fading model), the physical layer (e.g., scheduler, Tx power, noise), the application layer as well.
Sumo setting	example/TestNetwork	droomdorp.net.xml	It describes the network (i.e., map) underlying the simulation.
	example/TestNetwork	droomdorp.trip.sumo.xml	It defines initial and final position on the network of each mobile node, (i.e., vehicle) involved in the simulation.
	example/TestNetwork	droomdorp.rou.xml	It describes the initial route in the network (i.e., path of the network) for each mobile node, (i.e., vehicle) involved in the simulation.
	example/TestNetwork	droomdorp.traci.sumo.xml	It represents the main file of the network configuration. It links to the Sumo configuration files related to the simulation (e.g., network and node route

			files).
	example/TestNetwork	droomdorp.add.xml	It contains additional information (e.g., presence and location of detectors in the network, visualization parameters for the Sumo interface) related to the network underlying the simulation.
Log	example/logs	*.txt	It represents the directory that contains the log files created when the simulation is executed.

Table 6: List of relevant files implementing the LTE application example

Table 6 reports the iTetris module, the directory and the name of each relevant file that has been developed to realize the example application. Moreover, the table (last column) reports the intended purpose of the file as well. It can be seen that five main logical modules have been involved in the implementation of the example:

- Source code: it is composed of files realizing the logic of the example application and controlling the communication between iCS and the other modules, such as the application itself, Ns3, and Sumo.
- Simulation settings: it is composed of files configuring the simulation scenario.
- Ns3 setting: it is composed of files configuring Ns3 and in particular the communication technologies adopted in the simulation (e.g., LTE).
- Sumo setting: it is composed of files configuring the network of the simulation, location and position where stations are and move on, and the initial (static and predefined) route of each mobile node involved in the simulation.
- Log: it represents the directory where the application can store log files when the simulation is executed.

9.3 Results

The example application is executed, as usual in iTetris, by executing the command “iCS -c iteris-config-file.xml”.

The simulation of the example application worked as expected. When the vehicle approaches the (left) road intersection monitored by the RSU, they exchange CAM messages. Thanks to these messages, the RSU knows that it has to send speed advices to the approaching vehicle. The vehicle hence receives such a speed advice and changes its speed to 0, thus stopping its run as suggested by the RSU.

This behaviour can be observed by means of the Sumo interface, as shown by the snapshot of Figure 23. Furthermore it can be also observed by inspecting the log files recorded by the application and tracing information about the behaviour of the application and of the (mobile and fixed) stations involved in the simulation, such as the messages exchanged between vehicle and RSU.

10 CONCLUSION & OUTLOOK

The MOTO simulation tool implements LTE infrastructure with S1-U/X2 interfaces and ad-hoc interfaces for Wi-Fi direct connections.

The software allows the creation of flexible simulation scenarios involving ns-3 hybrid nodes equipped with LTE and Wi-Fi interfaces, allows the loading and configuration of mobility traces, permits the calculation of performance metrics such as throughput, end-to-end delay, packet loss, and it is able to validate the different offloading solutions.

In order to fully support the MOTO simulation requirements, a modular and flexible architecture has been implemented which provides the creation of new modules to add to ns-3 official release.

In the current release of the MOTO simulation tool, the followings modules have been added:

- CDM-Node: it pilots the dissemination procedure; with a periodic evaluation decides if the dissemination status respect what expected and eventually executes the required number of injections.
- UE-Node: it receives from CDM-Node the content through LTE interface and UDP protocol, sends a tracking feedback to CDM-Node by UDP and forwards the content through Wi-Fi interface using epidemic routing algorithm.
- Seeders-Calc: installed on the CDM-Node, offers a customizable interface required to give back the list of Ues designed for the injection procedure.
- Epidemic-Routing: Epidemic Routing spreads contents over the network until the bundle's lifetime is expired by establishing a wireless connection between the Ues that come into contact (neighbours).

In first instance, the MOTO simulation architecture implementation has been used to verify the PUSH&TRACK [R1] dissemination framework using an epidemic routing algorithm [R3].

The release b has to ensure support for vehicular scenarios, in particular the iTetris platform has been chosen in order to give a powerful tool able to combine the SUMO mobility simulator with the NS3 networking simulator.

First, iTetris has been changed in order to support NS3 v. 3.18 so that all features used in the MOTO simulation tool were granted; afterwards, in a second step, iTetris has been enhanced integrating the LTE support in order to get a tool able to cover vehicular-related scenarios.

In a final step an iTetris sample application has been developed in order to validate the LTE integration.

11 REFERENCES

- [R1] *Push and track: Saving Infrastructure Bandwidth through Opportunistic Forwarding* – J. Withbeck Y. Lopez, J. Leguay, V. Conan, M. Dias de Amorin - February 2012
- [R2] *ns-3 Module for Routing and Congestion Control Studies in Mobile Opportunistic DTNs* – J. Lakkakorpi, P. Ginzboorg
- [R3] *Epidemic Routing for Partially-Connected Ad Hoc Networks* – A. Vahdat, D. Becker
- [R4] *iTETRIS: a modular simulation platform for the large scale evaluation of cooperative ITS applications* - Michele Rondinon*, Julien Maneros, Daniel Krajzewicz, Ramon Bauza, Pasquale Cataldi, Fatma Hrizi, Javier Gozalvez, Vineet Kumar, Matthias Röckl, Lan Lin, Oscar Lazaro, Jérémie Leguay, Jérôme Haerri, Sendoa Vaz, Yoann Lopez, Miguel Sepulcre, Michelle Wetterwald, Robbin Blokpoel, Fabio Cartolano
- [R5] *Flooding data in a Cell: Is Cellular Multicast Better than Device-to-Device Communications?* – F. Rebecchi, V. Conan, M. Dias de Amorin – ACM CHANTS - September 2014

DISCLAIMER

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

Copyright 2014 by Thales Communications & Security SA, Consiglio Nazionale delle Ricerche, Asociación de Empresas Tecnológicas Innovalia, Universite Pierre et Marie Curie - Paris 6, FON wireless ltd, Avea Iletisim Hizmetleri As, Centro Ricerche Fiat Scpa, Intecs Informatica e Tecnologia del Software s.p.a. All rights reserved.