Small or medium-scale focused research project (STREP)

ICT Call 8

FP7-ICT-2011-8

Cooperative Self-Organizing System for low Carbon Mobility at low Penetration Rates

COLOMBO: Deliverable 6.4

Educational Kit

Document Information	
Title	Deliverable 6.4 –Educational Kit
Dissemination Level	PU (Public)
Version	1.0
Date	16.11.2015
Status	Final version
Authors	Robbin Blokpoel (Imtech), Federico Caselli (UniBo), Jérôme Härri (EURECOM), Wolfgang Niebel (DLR), Andreas Leich (DLR)

Table of contents

1 Introduction

1.1 COLOMBO Project

Traffic light control (TLC) as part of active traffic management aims at enabling safe and efficient passing of traffic flows at intersections. It requires determining the situation on the roads. These both actions make use of technologies which are becoming part of intelligent transportation systems (ITS).

Emerging cooperative techniques (C-ITS) like vehicle-to-infrastructure (V2I) communication via the IEEE 802.11-2012[1] standard may increase the knowledge about the traffic situation and open new channels for delivering information. However, most cooperative applications require large penetration rates in order to assure their functionality[2], making the first steps towards their deployment unattractive.

COLOMBO worked on overcoming this hurdle by delivering a set of modern, self-organizing traffic management components being applicable even at low On-Board-Unit (OBU) penetration rates below 10%, ensuring their usability from the very beginning of deployment on.

COLOMBO focused on two traffic management topics: traffic surveillance and advanced traffic light control algorithms. It delivers the methodologies and software for the following urban traffic management components, which are designed to operate locally on Road side Units (RSUs):

- A cooperative traffic state monitoring system, including
  - Traffic state estimation (TSE);
  - Automatic incident detection (AID);
  - Emissions monitoring system (EMS);
- A cost-effective, adaptive, and self-organizing traffic light (SOTL) control without stationary detection;
- A tool for automatic TLC algorithm configuration and tuning.

COLOMBO also:

- Extended and implemented available, well-established models for simulating vehicular pollutant emissions (PHEM), traffic (SUMO), and vehicular communication (iTETRIS and ns-3).
- Developed and implemented a simulation-based optimization architecture, incorporating these tools.
- Extracted guidelines on
  - Emission-optimal driver behaviour adapted to the new traffic light control system,
  - Developing eco-friendly TLC, dedicated for traffic engineers.
- Assessed the environmental and traffic impacts of the traffic light control.
- Disseminated and exploited the project results including a fully functional educational kit.

[1] The former IEEE 802.11p extension for automotive application was merged into the basic standard in 2012.

[2] Exemplary, a Green Light Optimised Speed Advisory (GLOSA) can yield in waiting time reductions of 1.9% already at 10% penetration rate and in reductions of ~15% at 20% OBUs. Fuel and emission savings start at higher penetration rates between 30% and 50%.

The project software is structured as in Figure 1:

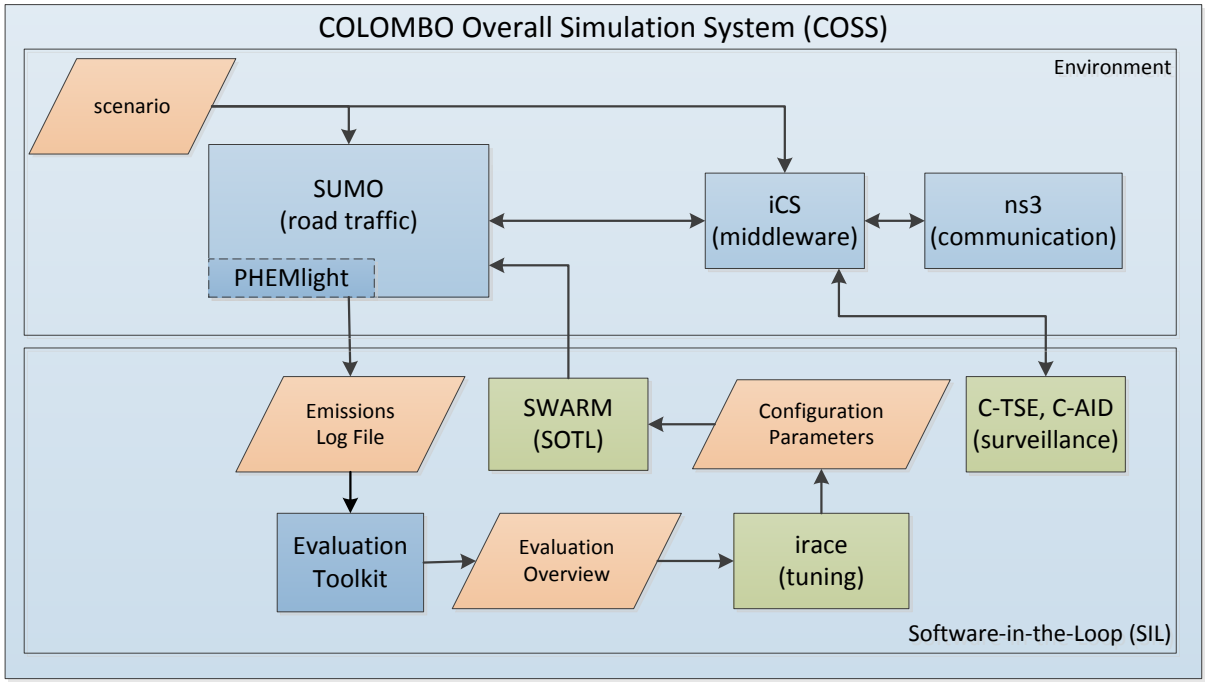

Figure 1: Software architecture

The complete online software architecture is structured around the iTetris Control Server (iCS). This component is responsible for synchronization between the applications, communication simulation of ns-3 and traffic simulation of SUMO. The apps are COLOMBO specific and with this educational toolkit there is an app for cooperative traffic detection and the SWARM traffic control algorithm. For real-time emission computation, SUMO incorporates PHEMlight. This emission model comes with vehicle datafiles which are accessed by SUMO. Therefore, while running a SUMO simulation, no separate PHEMlight process is online. PHEMlight will not be discussed in detail in this document. From SUMO release 0.20.0 onwards 2 example vehicle emission classes are included. The whole data set can be licensed from TU Graz, research group on emissions. You can contact Martin Dippold for more information on the license.

Once a simulation is done, SUMO will have finalized an emission log file, which can be accessed by the evaluation toolkit to create an evaluation overview file. irace only picks up one value from this to determine the overall performance of the simulation run. Based on that the configuration parameters for the SWARM traffic control algorithm are changed and a new simulation run can start. Depending on the tuning budget of irace, this process repeats many times.

For more information on the irace system, it is available online as open source package at:
http://iridia.ulb.ac.be/irace/

For more instructions on how to use and get started, there is also a generic tutorial available at:
http://iridia.ulb.ac.be/irace/files/irace-comex-tutorial.pdf

1.2 Objective of this document

The aim of this document is to provide instruction to those interested in the publicly available software components of COLOMBO. The document is written in the first place for the academic community, basic knowledge of linux, windows and XML file editing is therefore assumed. Road authorities, consultants, traffic planners and managers, and OEMs of relevant ITS products are also welcome to use this public deliverable to access the COLOMBO results. The document tries to explain the purpose of the components and give a manual-style overview on the usage.

1.3 Document structure

The document will first discuss the COLOMBO Overall Simulation System (COSS) in Chapter 2, which should enable the user to setup the online simulations from Figure 1. Chapter 3 will treat the Swarm traffic control algorithm in more detail. This control algorithm is compiled together with the SUMO simulator and therefore doesn't appear as a separate component in the overall architecture. The next component is the evaluation toolkit discussed in Chapter 4. Chapter 5 deals with the specifics for COLOMBO of the iRace software. Lastly, Appendix A explains the iTetris software in more detail, which is responsible for the iCS middleware.

2 COLOMBO Overall Simulation System (COSS)

2.1 Getting started

In order to ease the installation and usage of the iTETRIS platform and the COLOMBO COSS, the complete system is provided as ready-to-use VMWare virtual machine, containing one the one hand compiled version of the educational toolkit, and on the other hand, source files of the COSS development toolkit, as well as instructions to re-compile the required components.

In order to reduce the size of the VM, a lightweight version of Ubuntu has been used: **LUbuntu 14.04.01 LTS 64bit**. Reduced packages and Ubuntu applications are available, but may be later installed manually using apt-get.

2.1.1 Retrieving the COLOMBO COSS VM

In order to use the COLOMBO COSS, please follow the following steps:

1: download VMWare workstation player:
https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0

2: download Colombo virtual machine: The complete COLOMBO Overall Simulation System (COSS) is now available as a virtual machine via ftp.dlr.de under the name **LUbuntu-64-bit-COLOMBO-1.0.zip**. Please use your preferred ftp client (e.g., WinSCP, or any other from the Wikipedia list) to access the download folder with the following credentials: **COLOMBO_guest**, **HY88llSj**.

3: When loading the virtual machine the first time it will ask you whether you moved or copied the machine. Answer that you have copied the machine here.

4: While loading the vmware workstation, the client may prompt whether you want to install linux specific tools for vmware. It is recommended to do so as the virtual machine of Colombo uses linux.

2.1.2 Running the COLOMBO COSS VM

The educational toolkil (edu_tk) contains two applications:

- traffic-monitor: correspond to the COLOMBO TSE
- traffic-control: correspond to the integration between SWARM and TSE (more on this in chapter 3).

For educational purpose, we use a simple intersection scenario called RiSLA. This allows fast simulation and results computation. For more realistic scenario, the COLOMBO acosta and pasubio are available in the devel_tk.

It can be observed that there is no SUMO source files on the COLOMBO folders. This is because SUMO has been installed directly using the apt-get command. It can be started either by sumo or sumo-gui, if a GUI is required.

A development toolkit has also been included in the COLOMBO COSS VM for interested users. It contains:

- ns-3.20 – source files of the network simulator 3, including the original iTETRIS ETSI-ITS extensions, as well as the original COLOMBO ETSI-ITS extensions.

    - Due to a new way the iCS and the ns3 interact, the COLOMBO ns-3.20 is not compatible with the baseline iTETRIS distribution.
- iCS 2.0 – the iTETRIS controlling system. During the COLOMBO project, the iCS has been partially rewritten and improved in order to increase its scalability and improve the control of SUMO and ns3. Accordingly, the iCS 2.0 is not compatible with the baseline iTETRIS distribution.
- iTETRIS Apps – source files of the two COLOMBO traffic-monitor and traffic-control applications.

The COLOMBO COSS - LUbuntu 14.04.1 VM has the following structure:

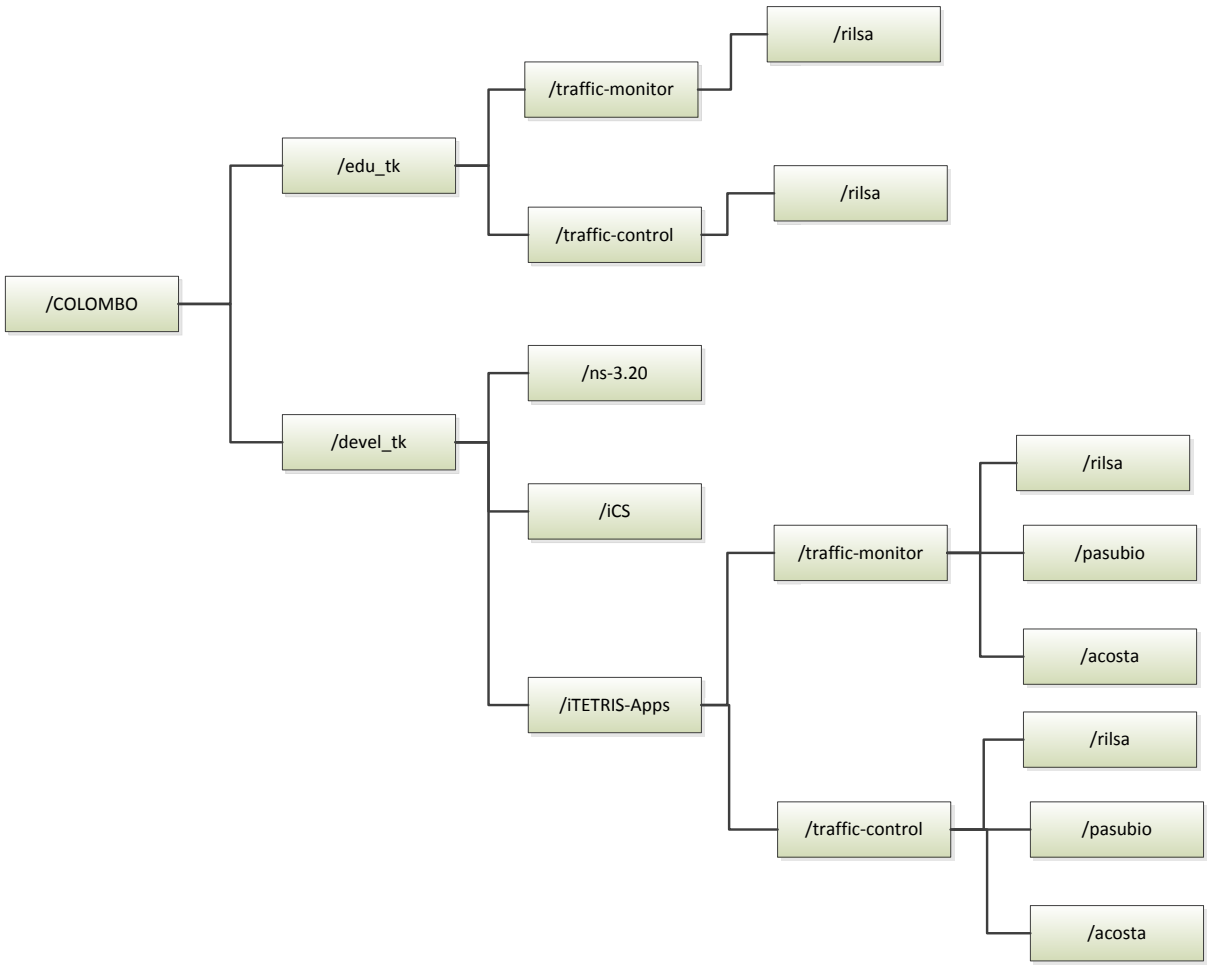

Figure 2: Folder structure on COSS virtual machine

In order to start the COLOMBO COSS VM educational toolkit:

1: At the lower left of the screen there is a start button, click it, then go to accessories and open LXTerminal. Reach the educational toolkit folder by typing in the terminal: “cd COLOMBO/edu-tk/rilsa”

2: Launch the COLOMBO COSS by typing the following command in the terminal:

2.1 with SUMO-gui: “iCS –c itetris-config-file-gui-ns3.20.xml

2.2 longer simulation, no gui: “itetris-config-file-ns3.20.xml"

2.1.3 Reading results from the COLOMBO COSS VM

While iTETRIS is running, if you have to interrupt the simulation, please follow the exact procedure below:

- Click on the terminal and press CTRL+C ONLY Once.
  - It will freeze the simulation
- Press ENTER ONLY Once.
  - It will make iTETRIS close all processes for you and terminate

Do not try to press CTRL+C more than once or press ENTER more than once. This will only result in creating orphan processes (iCS, ns-3, colombo applications) that you will need to destroy yourself manually.

Once the simulation completes, the COLOMBO educational toolkit has been configured to save specific data that you can parse (see chapter below). To do so,

1: extract results data with the following command in the terminal: "`python3 comm-protocol-parser.py prot-data.txt`"

2: A file named prot-data.csv is created, type "`gnumeric prot-data.csv.csv`" to view this file.

Chapter 3 will describe in more details the structure of the parameters in this file.

2.1.4 Simulation output parser

This script parses the data file from the communication protocol and creates a csv with the results. For every second it will output the information collected by protocol and the real data for the speed and the number of cars for each direction monitored by the protocol. A detailed description of the csv file is in Table 1.

To run the script the user can use the command:

```
python3 comm-protocol-parser.py protocol-data-file-name
```

Optionally with the flag –m the script can parse multiple simulations data file and then write the average values in the csv. To use this mode use the command

```
python3 comm-protocol-parser.py first-data-file-name -m
```

Table 1: Names of the columns in the csv file obtained as output of the parser script.

Keyword	**Meaning**
time	Current simulation time
protocol_density	Total number of vehicles counted by the protocol
protocol_max_distance_dir_X	Furthest vehicle in the direction X
protocol_min_distance_dir_X	Nearest vehicle in the direction X
protocol_quantity_dir_X	Number of vehicles counted by the protocol in the direction X
protocol_speed_dir_X	Average speed of the vehicles detected by the protocol in the direction X
real_density	Real total number of vehicles
real_quantity_dir_X	Real number of vehicles in direction X
real_speed_dir_X	Real average speed of vehicles in direction X

2.2 Setting up a new simulation

Inside the simulation folder, there are the configuration files for iTETRIS composed of:

- iCS configuration files;
- Sumo configuration file;
- Ns3 configuration file;
- Traffic monitoring application configuration file.

These files can be found in /home/colombo/COLOMBO/edu_tk/traffic-monitor/rilsa/

2.2.1 iCS configuration files

The main iCS configuration file, called in the toolkit `itetris-config-file-ns3.20.xml`, contains the main parameters and the command line to start sumo and ns3. This file also allows the user to set the duration of a simulation by changing the values of the `end` tag. To change the penetration rate use the parameter in the monitoring application configuration file as explained in section 2.2.4. Do not change the `penetration-rate` parameter in this file.

The application configuration file `application-config-file.xml` lets the user select which iTETRIS application to use in the simulation. To simulate the monitoring protocol the application has to be listed here. In the simulation of the complete system, which also uses the traffic control application, both application are listed inside this file.

The `stations-config-file.xml` allows the user to add roadside units, by specifying their id, their position and communication technology they use. The id selected here (open the file to see the example RSU has id "5000") has to share both on the application configuration file to let iTETRIS know that an application runs on the selected RSU, and on the configuration files of the applications themselves, since they need to know the id of the the RSU.

Note: the role of the `stations-config-file`.xml is also to configure the penetration rate of various technologies (ITS-G5, UMTS, etc..). COLOMBO opted to emulate the low penetration of ITS-G5 and higher penetration of alternate technologies (WLAN, Bluetooth) from an application level. Accordingly, COLOMBO kept a 100% penetration of ITS-G5, but reduced it in the COLOMBO applications. Please refer to the configuration of the COLOMBO application for more details.

Please refer to Appendix A.1 below or to the iTETRIS guidelines for a more in depth explanation.

2.2.2 Sumo configuration file

The configuration files of sumo mainly specifies the road network and the vehicular flow trace for use in the simulation. The file is contained in the sumo subfolder and can be selected editing the main iCS configuration file. Please refer to the SUMO wiki page for a complete overview of this file. http://sumo.dlr.de/wiki/SUMO

2.2.3 Ns3 configuration file

Ns3 uses a standard ns3 configuration file and a specific file used by the code that lets ns3 interact with iTETRIS. All ns-3 configuration file are found on a subfolder: ns3/.

As indicated in the <communicationsim> tag on the iCS configuration file, ns3 has two master configuration files.

The `configGeneral.ns3.20.txt` is a text file providing the defaults values for ns-3 functions. Please refer to the ns3 documentation for an explanation of this file format https://www.nsnam.org/docs/manual/html/attributes.html#configstore .

The `configTechnologies-ics.xml` is a xml file providing basic ns-3 configurations (seed, etc..) as well as linking to the specific configuration of the different ns-3 technologies.

As can be seen from the `stations-config-file.xml`, COLOMBO used the ITS-G5 technology. Accordingly, the `configTechnologies-ics.xml` file uses the files `confWaveRsu.xml` and `confWaveVehicle2.xml` located inside the ns3 subfolder to configure respectively a RSU and a mobile node. These files allow the user the selection of the transmission power (expressed in `dBm`) used by the communication equipment by changing the attributes `TxPowerStart` and `TxPowerEnd`.

Please refer to Appedix A.1 below or to the iTETRIS guidelines for a more in depth explanation.

2.2.4 The COLOMBO application configuration file

The protocol configuration file allows the user to configure the application for the specific scenario used. In this toolkit, the file name is `protocol-config-file.xml`. Following is the description of the main configuration parameters (they are xml tags):

`start`: when the protocol starts collecting data.

`data-file:` specifies the file name in which the protocol saves the data. It is required.

`log-file` and `ns-log-file` specify the names of the log files created by the application. The first is for the part that communicates with iCS, the second is the actual protocol log. Commenting these tags disables the logging.

`rsu:` specifies the road side unit node. The `id` attribute is the node id used by iTETRIS for this node. Currently the protocol supports a single rsu.

Sample monitoring direction:

```
<direction value="180" approaching="true" leaving="true" approaching-time="200" leaving-time="100">
```

- `value`: value in degree as explained in Figure 3.
- `approaching`: whether the lane approaching the crossing has to be monitored.
- `leaving`: whether the lane leaving the crossing has to be monitored.
- `approaching-time`: time waited by the RSU for the message response from the nodes on the approaching edge (ms). A larger value lowers the probability of transmission collisions, but slows the monitoring process since only one edge at a time is polled.
- `leaving-time`: as the previous point but for the leaving edge (ms).

`node`: used to specify the general communication properties:

- `probability-full`: probability of a node to have the full communication capability. Float in range `0 <= X <=1.`
- `probability-medium`: probability of a node to have a degraded communication capability. Float in range `0 <= X <=1.`

- `propagation-radius-rsu`: range of an RSU message. (meters). Please note that the transmission power is set in a different file, in the configuration of ns3, so the overall max range depends on the power level used.
- `propagation-radius-full`: range of a full node message. (meters)
- `propagation-radius-medium`: range of a degraded node message. (meters)

Note: The sum of `probability-full` and `probability-medium` has to be <= 1. This sum can be < 1, meaning that a portion of the nodes will not have communication capability and will not take part in the protocol, lowering the penetration in the simulation.

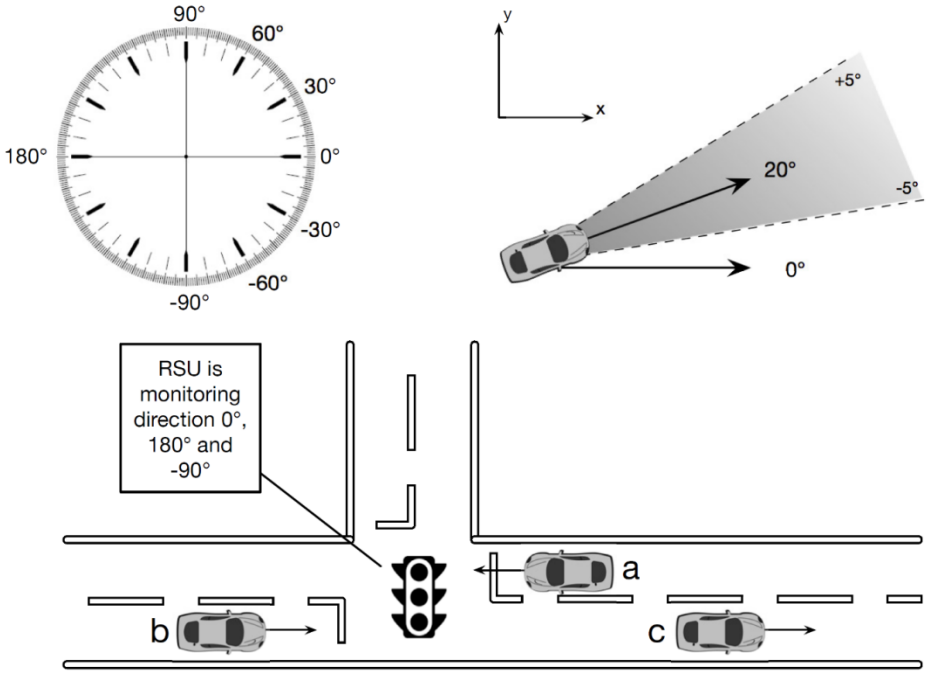

Figure 3: Directions

2.3 Access source code and building

The monitoring application is composed of two levels, as shown in the Figure 4.

The lower level, called `iCS integration`, manages all the interaction with the iTETRIS framework and presents to the protocol application an interface constituted by the `Node` class. Since this level is not tied to the specific monitoring protocol used in can function as a starting point to develop new monitoring protocol application. It is located in the subfolders `server` and `application` inside the source directory of the communication application.

The upper level, called `Protocol implementation`, is the actual implementation of the communication protocol used by COLOMBO. The class `iCSInterface` (called in the figure `Its Controller`) realizes the primitives to send and receive messages and connects the protocol level with the lower level. This part is located inside the subfolder `application/model` inside the source directory of the communication application

Note: we focus here on the description of the COLOMBO application structure and build. A major work during the COLOMBO project has been to allow almost full control of ns3 and SUMO from the application, therefore allowing a wide range of new C-ITS applications with minor changes in ns-3, iCS or SUMO. We accordingly, do not assume in this document that the user might need to extend the iCS, ns-3 or SUMO. We refer users to the iTETRIS documentation in such case.

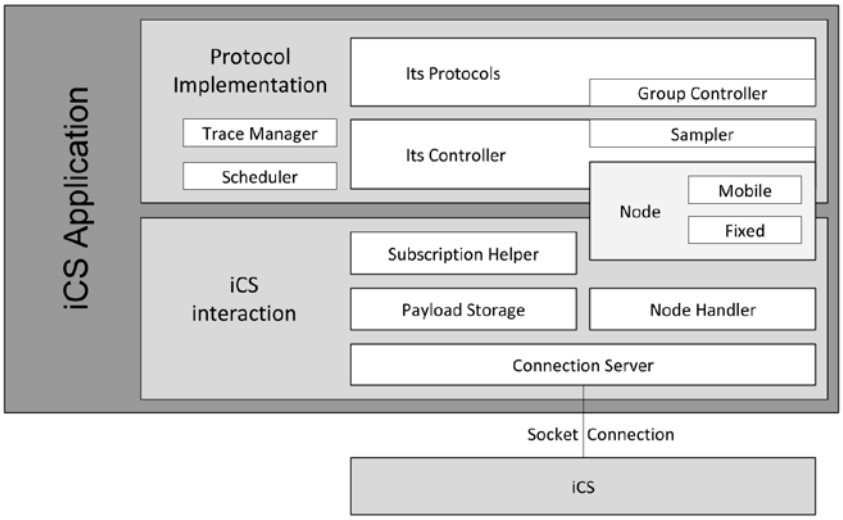

Figure 4: Overview of the monitoring protocol application.

2.3.1 Building

To build the application the following commands can be type on a terminal inside the application folder /home/Colombo/COLOMBO/devel_tk/iTETRIS-Apps/comm-protocol:

```
1$ Autoreconf –vif
2$ ./configure
3$ make
4$ sudo make install
```

After the first build only the last two steps are needed to compile changes to the existing files. The last command, the $4, will overwrite the previously installed version of the application.

Note: would the user need to recompile the iCS, ns-3 or SUMO, we briefly provide here the required commands. We yet refer users to the iTETRIS documents for more details:

Building ns-3:

```
1$ ./waf configure
2$ ./waf
3$ sudo ./waf install
```

Building iCS:

```
1$ autoreconf –vif
2$ ./configure --enable-sumo --enable-ns3 --enable-applications
        –-enable-log
3$ make
4$ sudo make install
```

note that would you not need log files, you may remove –-enable-log

3 Swarm traffic control system

3.1 Getting started

The steps 1-4 are identical to Section 2.1 for the COSS. These can be skipped if a user has already followed these.

1: download VMWare workstation player:
https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0

2: download Colombo virtual machine: The complete COLOMBO Overall Simulation System (COSS) is now available as a virtual machine via ftp.dlr.de. Please use your preferred ftp client (e.g., WinSCP, or any other from the Wikipedia list) to access the download folder with the following credentials: COLOMBO_guest, HY88llSj.

3: When loading the vitual machine the first time it will ask you whether you moved or copied the machine. Answer that you have copied the machine here.

4: While loading the vmware workstation, the client prompts whether you want to install linux specific tools for vmware. It is recommended to do so as the virtual machine of Colombo uses linux.

5: At the lower left of the screen there is a start button, click it, then go to accessories and open LXTerminal.

6: Type “cd COLOMBO/edu_tk/traffic-monitor/rilsa/

7: Type “iCS –c itetris-config-file-gui-ns3.20.xml”

8: A SUMO GUI window appears press the green arrow (“Play”) to start the simulation

9: The simulation ends after 250 seconds, close SUMO and return to the terminal to type

3.1.1 Evaluation of the performance of the traffic light controller

Sumo can output a series of simulation results that allows the user to evaluate the execution. The traffic lights controller can be evaluated with these values, by using the average waiting of the vehicles. Please refer to http://www.sumo.dlr.de/wiki/Simulation/Output/TripInfo for a detailed description.

The toolkit provides a script to parse the tripinfo file from sumo and output a set of relevant data for the evaluation of a simulation. It writes to stdout the average waiting time, the average time loss, the average duration and the total car number. The script can be launched as

```
python3 TripInfoEvaluator.py trip-info-file
```

3.2 Setting up a new simulation

The complete simulation uses both the traffic monitoring application and the traffic controller application. For the configuration of iCS, sumo, ns3 and the monitoring application refer to in section 2.2. Following is an overview of the traffic controller application configuration file.

3.2.1 Traffic controller application configuration file

The protocol configuration file allows the user to configure the controller for the specific scenario used. In this toolkit, its file name is `traffic-light-control-config-file.xml`. Following is the description of the main configuration parameters (they are xml tags):

`log-file` and `ns-log-file` specify the names of the log files created by the application. The first is for the part that communicates with iCS, the second is the actual controller log. Commenting these tags disables the logging.

`rsu:` specifies the road side unit node. The `id` attribute is the node id used by iTETRIS for this node. The `traffic-light-id` is the id of the traffic light used in the sumo road network. This is the traffic light controlled by the application.

Sample monitoring direction:

```
<direction value="180" leaving-edge="mw" arriving-edge="em">
```

- `value`: value in degree as used in the monitoring application.
- `leaving-edge`: the id of the edge leaving the crossing in the specified direction (from the sumo road network).
- `arriving-edge`: the id of the edge arriving at the crossing in the specified direction (from the sumo road network).

`additional:` lets the user specify the parameters to be used in the swarm controller and the phases for the traffic light. The overall syntax is similar to the traffic light definition used in sumo, found here http://sumo.dlr.de/wiki/Simulation/Traffic_Lights. The difference are listed below:

The `phase` tag has been extended to include

- the phase `type` to tell the controller how should treat the current phase (please refer to section 8.3.1 of deliverable 2.2 available at http://colombo-fp7.eu/results_deliverables.php);
- For a `decisional` phase is also added the attribute `targetLanes` which indicates which lanes have green in the current phase.

The traffic light definition contains a list of param tags to set the different configuration parameters of the swarm algorithm. The meaning of these parameter can be found in the tables 8.1 and 8.4 of deliverable 2.2. These values are usually obtained from an automatic parameter tuning program.

3.3 Access to source code and building

The source code can be found in /home/Colombo/COLOMBO/devel_tk/iTETRIS-Apps/traffic-control/src. The traffic control application uses an architecture similar to the monitoring application: a lower level that manages all the interaction with iCS, and a higher lever that realizes the traffic control algorithm.

The lower lever is similar between the two applications, and is located in the subfolder `ics-interface` inside the source directory of the application.

The upper level is composed of the `iCSInterface`, which handles the interactions between the two sections of the application, and the controller algorithm. This part is located in the `application` subfolder. Please refer to the chapter 8 of the deliverable 2.2 for an explanation on how the algorithm works.

The build procedure is the same for both applications, the only difference is folder to use for the traffic control application: /home/Colombo/COLOMBO/devel_tk/iTETRIS-Apps/traffic-control/. Please refer to section 2.3.1 for the instructions.

4 Evaluation Toolkit

4.1 Getting started

It is assumed the user has Java JRE 7 or higher installed. When this is not the case, step 2 will fail and the user has to install java first. The installed java version can easily be checked in a command prompt by typing “java –version”. When no java is installed, the previous command will result in a message saying that java is not a recognized internal or external command.

1: Download the evaluation toolkit from the COLOMBO website

2: Run the batch file located at <extract location>\example_evaluation.bat

3: Observe the file
“<extract location>\results 1_0_120 Imp. = 52 emission = 1202328.778321862.txt” has appeared

4: Start Excel or openOffice calc and read the results file (when Excel appears in your open with list, this option is more convenient)

5: The result file should look like this (the table has been transposed for increased readability in a document):

Table 2: Example results, transposed and only showing 5 signal heads

SignalHead	SUMO ID	1010	1000	1005	1005	1008
class		truck	truck	truck	car	truck
throughput	vehicles	5	2	3	13	6
grade	lower is better	991	1362	1195	968	1152
impact	delay + 8*stops	119895	78068	93780	300924	195234
delay_avg	milliseconds	20779	31034	23260	17612	21875
delay_std	milliseconds	18189	11078	19365	18791	12603
delay_max	milliseconds	43069	42112	49866	51511	38833
stops_avg	millistops	400	1000	1000	692	1333
stops_std	millistops	489	0	816	821	942
stops_max	millistops	1000	1000	2000	2000	2000
queue_avg	decimeter	52	130	10	13	32
queue_std	decimeter	89	97	14	25	32
queue_max	decimeter	229	227	31	97	97
dem_wait_avg	milliseconds	8850	33100	13233	10891	4183
dem_wait_std	milliseconds	14116	0	17724	16038	5784
dem_wait_max	milliseconds	33300	33100	38300	42300	16900
CO2_avg	mg/km	2926999	3178447	2564032	339478	2695273
pw_avg	perceived milliseconds	22338	26262	21154	20916	18477
pw_std	perceived milliseconds	9820	8296	7399	9922	4165
pw_max	perceived milliseconds	34645	34559	31434	41835	25167
ua_avg	1000 = 100% acceptance	908	892	917	913	931
ua_std	1000 = 100% acceptance	45	41	32	50	14
ua_min	1000 = 100% acceptance	851	851	872	793	906

4.2 Calculation method

The table acquired during the last step of the getting started section shows many evaluation results. This section will describe what these fields represent exactly and how they were calculated.

4.2.1 Signal head

The signal head is determined by the SUMO net file and used to group the evaluation results. The figure below illustrates how the signal head number can be mapped to the actual network with sumo-gui:

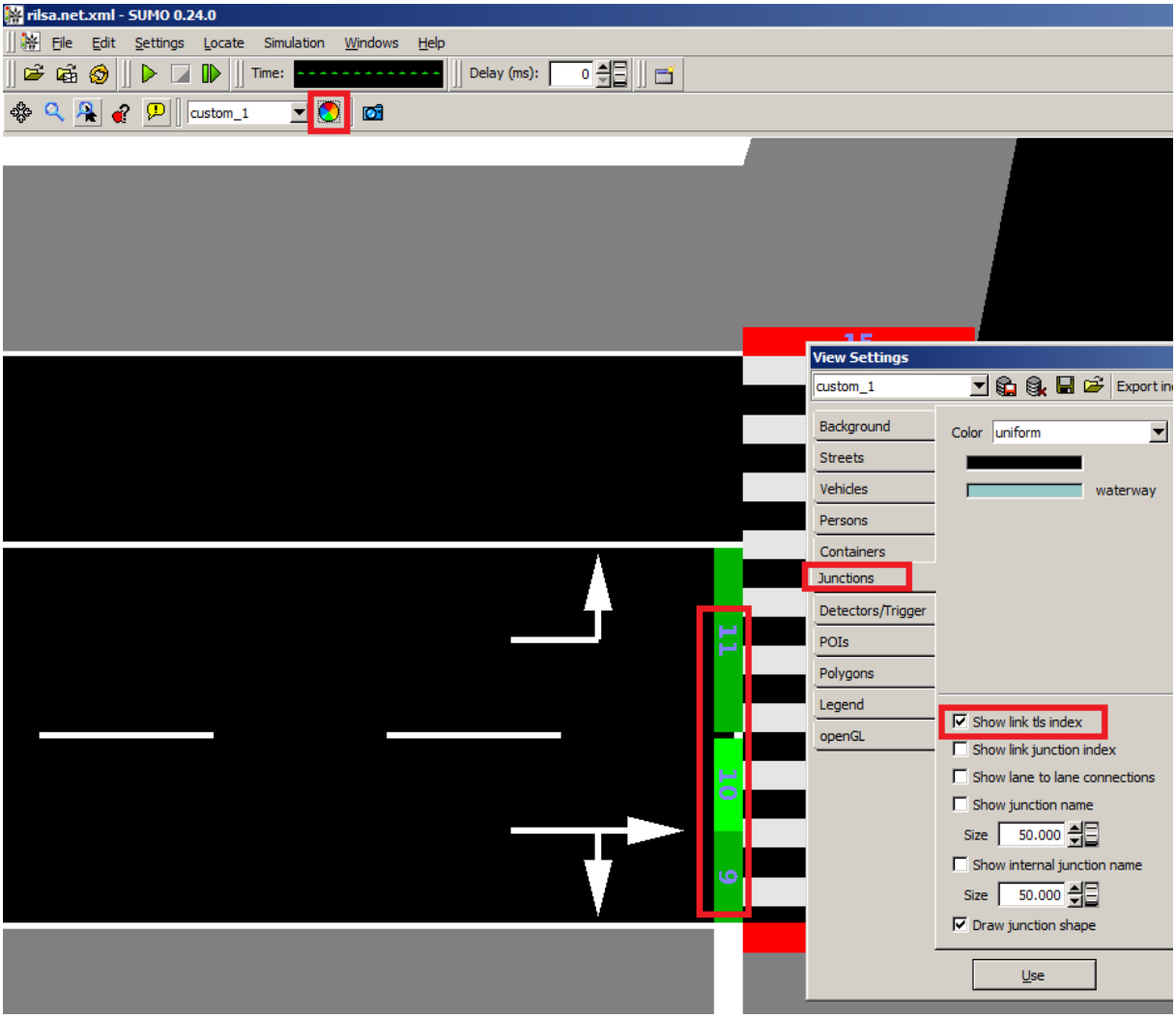

Figure 5: Obtaining signal head numbers from sumo-gui

Important items in the figure are marked with red boxes. As a first step the user should zoom in to the signal head it wants to know the number of. Then click the pie chart icon on top of the screen to access the view settings. In the view settings, the horizontal tab called “junctions” should be selected and in this tab the option “show link tls index” should be checked. As can be seen each lane has at least one signal head, but when multiple turn directions are allowed from that lane, there will also be multiple signal head numbers.

In the evaluation output file, the number of the intersection is multiplied by 1000 and the signal head index is added. Therefore, in table 2 we see signal heads 10, 0, 5 and 8 of intersection 1. Evaluation results are grouped by the signal head index, so all other rows of data represent the values for traffic

passing that signal head. The trip of each vehicle is split into sub-trips according to the signal heads. For instance a vehicle passing signal head 5 of intersection 1 and signal head 9 of intersection 2 before ending the trip, will contribute to the evaluation results as follows: the first part of the trip from entering the network up to leaving intersection 1 is attributed to 1005, the second part from leaving intersection 1 to leaving intersection 2 is attributed to 2009. The last part of the trip from intersection 2 up to leaving the network cannot be attributed to any signal head. Therefore this is put in the unknown category with signal head identifier 0. Vehicles that did not complete their sub-trip at the end of the simulation will have the last part of their delays added to the unknown category as well. If for example the simulation ends before the vehicle reaches intersection 2, but after it left intersection 1 and it was delayed by 20 seconds before leaving intersection 1 and by 45 seconds at the end of the simulation, then the contributions to the delays will be as follows: 20 seconds to signal head 1005 and 45 seconds to unknown signal head 0.

It is important to consider when a vehicle is considered to have left the intersection. In this evaluation software it is the moment it leaves the junction area. So the last time step it was on the internal edge of the junction still counts for that signal head, but any delay, stop, CO_2, etc. afterwards will be attributed to the next signal head. The reason for also taking the internal edge following the stop line into account is that many vehicles are still accelerating on the intersection area after a stop for the traffic light. The only exception for this are the queue length and the demand wait time, these are calculated up to the stop line only.

4.2.2 Class

Vehicles are subdivided in classes. This is to allow splitting the results for different classes like pedestrians, bicycles, cars, trucks, etc. The example results table shows two columns for signal head 1005, one for cars and one for trucks. The software obtains these classes from the route file, which has to be in the same directory as the net file and have the same name (e.g. rilsa.net.xml and rilsa.rou.xml). From this route file, the “vType” is obtained:
<vType id="bicycle" length="1.5" maxSpeed="5" emissionClass="none" guiShape="bicycle">

Each different vType will get separate results. The only exception are the subtypes indicated by an underscore. For example “car_s” and “car_m” are both considered “car” as the software simply removes the part after the underscore. However, if a new vehicle type “cooperativecar” would be defined, then this will be seen as a new class. The feature to disregard subtypes can be turned off in the source code by setting the boolean *splitSubType*s to true.

4.2.3 Throughput

The throughput represents the amount of vehicles of the specified vehicle class that passed the signal head in the simulation period.

4.2.4 Grade

The grade is computed according to the following formula:
*grade = (5*delayGrade + delayStdGrade + userGrade + 2*stops_avg)/9;*

With:
delayGrade = delay_avg / 20
delayStdGrade = delay_std / 12
*if (ua_avg>950): userGrade = 1000 - (ua_avg-950)*20*
*if (ua_avg > 850): userGrade = 2000 - (ua_avg-850)*10*
*if (ua_avg > 700): userGrade = 3000 - ((ua_avg-700)*20)/3*
*if (ua_avg > 500): userGrade = 4000 - (ua_avg-500)*5;*
*if (ua_avg < 500): userGrade = 5000 - (ua_avg)*2;*

So the grade is composed of a weighted average of the grade for stops, delay average and standard deviation of the delay. The grade for delay is simply increasing by one point for every 20 seconds of delay. For the standard deviation this is 12 seconds. Stops are graded in a manner that each average stop adds 2 points to the delay. The user acceptance (ua) is graded non-linearly according to the formulas mentioned above. The weighting factors are based on the research done in COLOMBO D5.3, but CO_2 has been replaced with stops as later project experience revealed that it was a difficult measure to use for comparison between different networks.

The grade is aimed to be between 0 and 5, but can go over 5 for heavily congested networks as there is no theoretical maximum to the formula. This was done to avoid non-linear grading. Also note that the values are multiplied by 1000 for accuracy and computational speed (using doubles or floats would slow down the software). A value of 1362, means a grade of 1.362.

4.2.5 Impact

The impact is a simplified version of the grade which is simply computed as delay + stops * 8. It can be used to give a quick overview of the results by summing up the impact values of each signal head and vehicle class and dividing by the total throughput. Again the numbers are multiplied by 1000 for accuracy.

4.2.6 Delay

Like many measures, the delay is split in an average, standard deviation and maximum value. Delay time is defined as the difference between the free flow travel time and the actual travel time for the sub-trip of the vehicle. Since this also includes time lost due to deceleration, it is a more complete measure than waiting time. In literature it is also sometimes called the “loss time”. Delay times for all vehicles during the simulation period are recoded and the average, standard deviation and maximum of all those individual trips are calculated. The units are milliseconds, again for accuracy and computational speed.

4.2.7 Stops

As soon as a vehicle drops below a speed of 5 km/h or $1/10^{th}$ of its desired speed, it will count as a stop. 5 km/h is generally used in the traffic engineering world as a rule of thumb, but the $1/10^{th}$ of the speed is taken to prevent pedestrians walking around 5 km/h be considered stopping every time they slow down a bit. Stops are again represented with an average, standard deviation and maximum value and multiplied by 1000 for accuracy and computational speed.

4.2.8 Queue

The queue length is determined per vehicle. This means a queue of 5 vehicles which stop at an average distance of 5m will result in average queue length of 10m: the first car experiences a queue of 0m, the 2^{nd} 5m, 3^{rd} 10m, 4^{th} 15m and 5^{th} 20m. Again this is calculated per signal head and only the

first stop is counted for the queue length. So a queue that is longer than the amount of vehicles that can leave in the next green phase will only result in one queue length measure per vehicle. The calculation method per signal head results in a risk when vehicles have to stop for other reasons than the traffic light. For instance an uncontrolled intersection or a merge point between two intersections can result in a large overestimation of the queue length. Queue length is represented with an average, standard deviation and maximum value, with a unit of decimetres (10dm = 1m) for accuracy and computational speed.

4.2.9 Demand wait time

The demand wait time represents the time the first vehicle in the queue had to wait before the light turns green. The reason for measuring this is to verify whether a control strategy is acceptable for road users or not as too long demand wait times may result in red light violations. Traditionally the cycle time is used for this as the demand wait time can never exceed the cycle time. However, modern adaptive control strategies can use detectors to see whether someone is waiting and only start counting the demand wait time once someone arrives. Demand wait time is not affected by multiple stops, so a vehicle that stops at position 15 in the queue, but cannot pass in the next green phase and then stops at position 1 in the queue, will only have the wait time during the 2nd red phase counted for the demand wait time. The measurement is represented with an average, standard deviation and maximum value. The units are milliseconds, again for accuracy and computational speed.

4.2.10 CO_2 emissions

CO_2 emissions are directly acquired from SUMO and summed over the sub-trip belonging to the signal head. The length of the sub-trip is used to determine the emissions per kilometre. It should be noted that both chosen emission model and the driver behaviour model have a high impact on the total emissions. The first is intuitive and involves choosing the right model for the simulation. The latter, however, may result in some unexpected effects. For example the driver imperfection parameter can result in the driver constantly accelerating and decelerating while being in free-flow state. This increases the emissions significantly. Another factor to consider is the length of the sub-trip, while the emissions are normalized by the length, the ratio of between time spent at free flow travel and waiting for the traffic light is still influenced by the length of the sub-trip. Therefore, the impact of the traffic control strategy on the CO_2 emissions is very large for short sub-trips, while it can be almost insignificant for very long trips. This is also the reason why the CO_2 measure was excluded from the grade during the project. For this measure only the average is reported, the unit is in mg/km for accuracy and computational efficiency reasons.

4.2.11 Perceived waiting time

The perceived waiting time is based on a user study by B. van der Bijl and the calculation method is further explained in COLOMBO D5.3. The measure represents how the user perceives the waiting time. Factors like stopping, stopping at 2 intersections in a row and general perception on how fast time passes by are taken into account. For this measure the average, standard deviation and maximum are reported. The units are milliseconds, for accuracy and computational speed.

4.2.12 User acceptance

The user acceptance is a derivative of the perceived waiting time, it represents how many users would find the control strategy acceptable and more information can again be found in COLOMBO D5.3. Acceptance can range between 0 and 100%, but is represented as 0-1000 for accuracy and

computational speed. Since the minimum is the worst for the user, the average, standard deviation and minimum value are reported.

4.3 Setting up a new simulation

For setting up a new simulation it is assumed the user knows how to create a SUMO network and the basics of running SUMO. When looking at the files downloaded, it can be seen there is also another batch script at <extract location>\runSUMO.bat. This file will probably not work as the path of the SUMO executable is likely not the same, it contains the following:

D:\Sumo\sumo-0.24.0\bin\sumo-gui.exe --configuration-file rilsa.sumo.cfg --emission-output emission.out.xml --phemlight-path D:\Sumo\sumo-0.24.0\data\emissions]PHEMlight\ --start

Important to observe from this file is the –emission-output emission.out.xml. This emission output has to be generated as it serves as input for the evaluation toolkit. It should be noted this file can become very large (10GB or more) for networks with around 45000 vehicles passing during the simulation run. Since the file is run from the directory of the extract location, the pull path for the rilsa.sumo.cfg is not required and the emission output file will be saved to the same directory. All other paths should be adapted to the location where SUMO is installed.

If a new SUMO configuration is made and the runSUMO.bat has been adapted to the new network and SUMO installation paths, the next step will be to adapt the example_evaluation.bat script to the new situation. The script contains the following:

java -jar evaluation.jar %~dp0 emission.out.xml rilsa.net.xml

Since evaluation.jar was included with the download, a full path location did not have to be specified in the batch file. However, the other paths have to be specified. The “%~dp0” is a special batch file code that refers all future paths to the location where the batch file is located. This is easier to create a demo that works straight after extracting, but may not be preferred for creating your own scenario. Therefore, the following example better explains what should be contained in this batch script:

java -jar C:\SUMO\evaluation.jar c:\SUMO\scenario emission.out.xml scenario.net.xml

Firstly, the location of the evaluation.jar file has to be specified, the first parameter is the working directory. This is where the results file will be written, but also where it will search for the other files of the network. The second parameter is the name of the emission output file that was previously created with the runSUMO.bat. The third parameter is the network file. Note that the route file will also be read by the software and has to have the same naming as the net file. So when “scenario.net.xml” is specified, there should also be a “scenario.rou.xml”.

After running this script it can be noticed that the “emission.out.xml” is renamed to “emission.out.xml #1”. This is to allow easy scripting for running multiple simulation runs in a row. After finishing the evaluation, an external script can start another runSUMO.bat and a new emission.out.xml will be created. When running the evaluation.jar afterwards, the results file will start with “results 2…” instead of “results 1…”. The emission.out.xml will then have a #2 appended to the file name. This way the user can leave many runs running in the background and still find all data on the harddrive in an ordered manner.

4.4 Access to source code and building

The source code comes with the download of the getting started section in the subfolder source. To build the code it is best to start a new java project in your favourite code editor, for example eclipse. Add the three source files to the new project and select Evaluation.java as class that has the main class entry. The code has no external dependencies other than a java JRE system library of at least 1.7. With these settings an initial build should be easy and quick to achieve.

When viewing the source code there are 3 class files. The main class is Evaluation.java. EvaluationGroup.java and VehicleProfile.java are classes to store data during the execution of the evaluation. The execution flow of the main Evaluation.java class is as follows:

1. (line 67-81) Load the execution parameters given at startup to be able to find the files and folders for the program.
2. (line 83) Determine the run number, this is done according to the following hierarchy: If the emission output file contains a number, this number will be used as run number. If this is not the case, then it will search if there is an emission file from a previous run in the same directory and it will use that number +1. If emission files get removed instead of renamed (for example to save disk space) then it will search for the highest first number in a result file. Lastly, for the very first run when there are no old emission or results files, the number 1 will be used.
3. (line 86) Process the network file, in this step the speed limit on the edges is acquired (required to determine delay as the speed limit is important for the free flow speed) and the locations of the signal heads are also determined.
4. (line 87) Read vehicle classes, this is taken from the rou.xml file (line 712 manipulates the string to get the route file, so if you want more flexible paths, that should be adjusted to use another command line parameter). This is done to build a list of vehicle types and their maximum desired speed. Together with the speed limit this data is used to determine the free flow travel time for each sub-trip.
5. (line 89) Read the vehicle profiles. This refers to a large piece of code that reads the entire emissions file.
6. (line 586-593) Optionally, skip the first x minutes of data, for which x is taken from a class variable “skipStartDataMinutes”. This option is to discard data from the time the network was still filling or to create subintervals (which will be discussed later)
7. (line 594-649) Reads the file until the timestep exceeds “maxMinutes” which is a parameter given to the method and meant to create subintervals for evaluation. By default this is set to the absolute maximum integer value allowed for the SUMO timestep. This allows for 5965 hours of simulation (248 days). The code basically build vehicle profiles from the evaluation output. These profiles contain a list of speed, lane, position on the lane and CO2 emission for each time step.
8. (line 329-474) As building these profiles can occupy a large amount of memory, they are processed every 100 seconds and once at the end of reading the emission output file. Only inactive profiles (so profiles of vehicles that finished their trip) will be used. The method sums delay time, stops, CO_2, etc. on a step-by-step basis until a signal head is passed. When a signal head is passed, the data will be added to the evaluation group belonging to the vehicle class and signal head.
9. (line 91) Write the evaluation groups to the disk in a “results…txt” file. This basically consists of creating the legend of the tab delimited file and writing data in the following lines from each evaluation group. Note that there is no ordering of the evaluation groups, so the signal heads will appear in the order at which they were created during the processing of the vehicle profiles. The calculation of average,

standard deviation, maximum and minimum values is done in the EvaluationGroup class. Lastly, the file is renamed to have a quick summary of the results in the file name as well. This has the total average impact divided by 100 (i.e. in deciseconds)

10. (line 95-105) Optionally (default turned off), the calculation is repeated for subintervals. Step 6-9 are then repeated for interval lengths defined by the variable “minuteInterval”.
11. Lastly, the emission file is renamed by using the run number and appending it with a “#” in the file name.

5 Automatic Configuration with iRace

The iRace system is available online as open source package at http://iridia.ulb.ac.be/irace/. There is also a generic tutorial available at http://iridia.ulb.ac.be/irace/files/irace-comex-tutorial.pdf

5.1 Purpose

The goal of automatic configuration techniques is to allow users to configure their own software, that is, to set the values of their parameters such that a given performance measure is optimized. Automatic configuration techniques have been shown to be extremely helpful to reach high performance in many different fields of algorithm design, and to free the human designer from the most tedious tasks required in all empirical aspects of algorithm conception. In COLOMBO, such techniques have been shown to be extremely valuable to configure traffic light controllers. More specifically, within COLOMBO, we used irace to configure the SWARM traffic light control algorithm, relying on SUMO for the simulations used to evaluate the different candidate configurations.

In the educational toolkit we provide a “ready-to-launch” example that uses irace, a software package for automatic configuration. For a detailed presentation of automatic configuration techniques and the state-of-the-art of the field, we refer to COLOMBO deliverable D3.1.

5.2 Getting started

In order to make use of the irace software for automatic configuration, the user must make sure to have the following:

- A machine running Linux/Mac/Windows

  Note that many scripts are provided with irace that make its usage easier. Some of them are written in bash and thus might not run out-of-the-loop on Windows platforms. We recommend to use Linux, especially for beginners who would prefer to have an almost ready-to-launch setup.

- A recent (>= 2.14) installation of R (www.r-project.org/)

  The official sources and binaries of R can be downloaded at https://cran.r-project.org/bin/, alternatively, on Debian-based systems, it can be installed with:

  $ apt-get install r-base-core

- A recent (>= 1.07) installation of irace
- Irace can be installed from within the R environment::

  > install.packages("irace")

  Alternatively, one can download directly the archive and follow the instructions at http://iridia.ulb.ac.be/irace/ to install it.

The next step is to set-up the specific configuration task to by writing few files that tell irace how to interact with the software to be configured, how many experiments to use, etc. In the educational toolkit we provide a ready-to-launch setup, which can be ran by simply going to the specific directory and run irace:

$ cd /home/Colombo/COLOMBO/irace/example

$ irace

To fully make use of multi-core CPUs, one can add a parameter "--parallel X", where X is the number of parallel threads that will be used to evaluate candidates.

To obtain a full list of irace parameters, one can simply type:

$ irace –help

For an extensive presentation of irace and its usage, the reader can refer to
http://iridia.ulb.ac.be/IridiaTrSeries/link/IridiaTr2011-004.pdf

5.3 COLOMBO example

In the educational toolkit virtual machine, we provide a ready-to-launch example that is launching a configuration of the SWARM traffic light control algorithm on 10 instances that represent different flows of vehicles of the Rilsa road network.

List of files

The example, which can be found at */home/Colombo/COLOMBO/irace/example*, is made of the following files:

- tune-conf

This files contains the parameterization of irace itself: how many experiments to use, where to find the instances, which statistical test to use, etc.

- arena (directory)

This (originally empty) directory is the location where the experiments will be run. Temporary files, if any, are written there so as not to flood the main directory with files.

- flows_rilsa_24h (directory)

This directory contains the "instances" to run candidate configurations. In the case of COLOMBO, those are vehicle flows.

Note that for a real tuning, more flows should be used. (Only 10 are provided in this example).

- hook-run

This file is the executable that is called by irace to evaluate candidates. It is responsible to call the underlying software to be tuned. In the context of COLOMBO, it will create SUMO input files that instantiate each specific candidate, then run SUMO, then the evaluator script that counts the waiting steps.

- instances_list

This file contains a list of flows used for the tuning. They are to be placed in flows_rilsa_24h.

- MakeAdditionalFromTuning.py

Python 3 script. This creates the XML input file for SUMO based on a list of parameters given as command line (as passed by irace).

- parameters.txt

This file contains a formal description of the parameters to be tuned, (their type, possible ranges, etc.)

- rilsa1both.net.xml

The net file that describes the road topology to be used for the SUMO simulations.

1. template_rilsa.xml

A template of the XML file used to instantiate the parameters. It is specific to the net file.

- TripInfoEvaluator.py

A python 3 script that evaluates the output of SUMO and counts the waiting steps of all vehicles.

- vtypes.add.xml

Definition of the types of vehicles to take part in the simulation

5.4 Output

The output of irace is shown on the standard output as the configuration is running. At the end of the configuration (once the budget is used), irace will stop and display the best configurations that were found, ordered by ranks. Irace will also save the entire set of experimental results, along all the candidates that were evaluated, in an R data file

(irace.Rdata). For more information on which information is stored in this file and how this is internally structured, we refer to http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf.

A. Appendix

A.1 iTETRIS CONFIGURE SIMULATION

The following covers the configuration of iTETRIS, which has been extracted from the iTETRIS Configuration Document [iTETRIS Building, Installation and Configuration Guidelines, iTETRIS project 2010].

The configuration is based in XML formatted files. The set and organization of those files is presented in Figure 6.

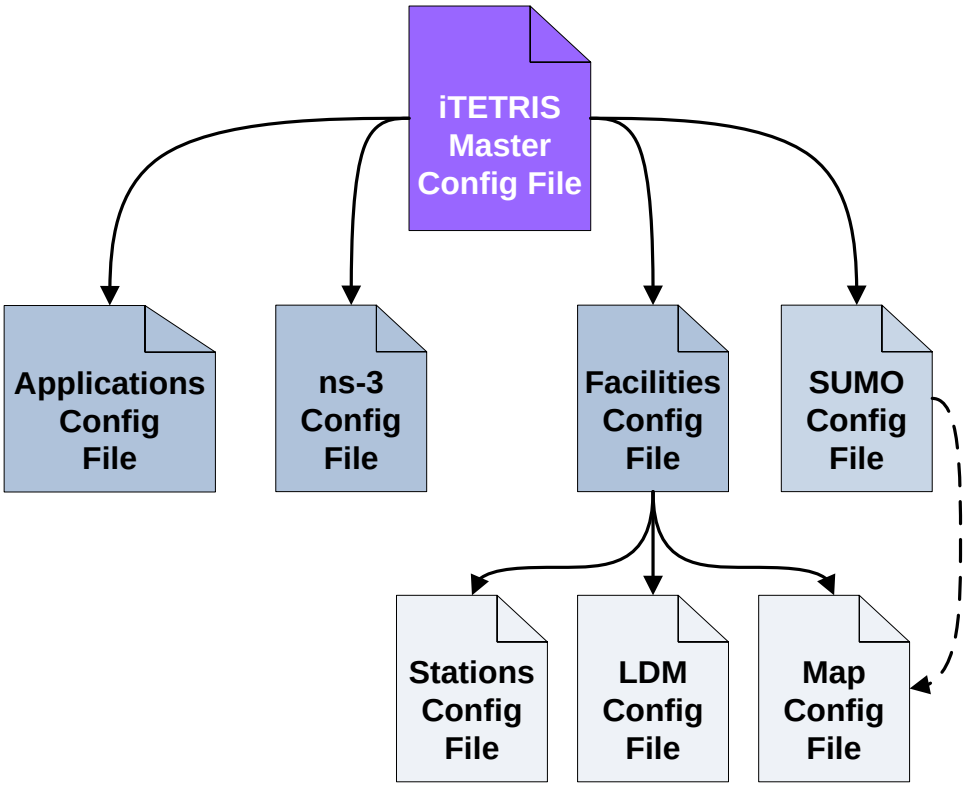

Figure 6. iTETRIS configuration files schema.

SUMO configuration file is out of the scope of this document and its format can be found in the SUMO project official site [http://sumo.dlr.de/wiki/Main_Page].

A.1.1 iTETRIS Master Configuration File

Hierarchically organized, the iTETRIS configuration file is the entry point to the customization of the simulation and it is divided in five blocks:

- Simulation basic information set.
- Traffic simulator configuration, SUMO in iTETRIS.

- Wireless communication simulator configuration, ns-3 in iTETRIS.
- Traffic management applications configuration file path.
- Logging configuration.

The first block (Table 3) is related with the main parameters of the scenario, here the most important values is the second in which the simulation ends.

TAG NAME	DESCRIPTION
begin	Time step in which the Applications will go active.
end	Time step in which the Applications will deactivate.
resolution	Time span used for every step of simulation, in ms.
penetration-rate	Percentage of the total amount of vehicles in the simulation that are going to have the application installed. To lower the percentage of communication capable device use the method explained in 2.2.4
facilities-config-file	Path of the file in which the configuration of the ETSI stack "Facility" is defined.
message-reception-window	Defines how many time steps the iCS will storage the messages before considering the messages that did not reach destination is going to be discarded.
Interactive	Interactive tag allows running the simulation stopping the simulation in each time step. To make the simulation continue press the ENTER key

Table 3. Simulation basic information set tag definition.

```
<scenario>
        <begin value="0"/>
        <end value="3600"/>
        <resolution  value="50" />
        <penetration-rate value="100"/>
        <facilities-config-file value="facilities-config-file.xml"/>
        <message-reception-window value="5"/>
        <interactive value="false"/>
</scenario>
```

The next section in the iTETRIS main configuration file references the necessary data to be provided to the iCS so it can setup and establish the connection with SUMO (Table 4).

TAG NAME	DESCRIPTION
traffic-executable	Traffic simulator executable name, including parameters.
traffic-file	Traffic simulator configuration file.
traffic-host	IP address where SUMO is listening for commands.
traffic-port	Port number where SUMO is listening for commands.

Table 4. Traffic simulator configuration tag set.

```
<trafficsim>
        <traffic-executable value="sumo -c"/>
        <traffic-file value="bolonia.sumo.cfg"/>
        <traffic-host value="localhost"/>
```

```
        <traffic-port value="1984"/>
</trafficsim>
```

Following the same approach, the file contains another block to hold the values so the iCS connects with ns-3 simulator (Table 5).

TAG NAME	DESCRIPTION
communication-executable	Name of the wireless communications simulator executable.
communication-host	IP address where ns-3 is listening for commands.
communication-port	Port number where ns-3 is listening for commands.
communication-general-params-file	Ns-3 file (.txt) where general simulation attributes are provided. This file can be created through the ns-3 functionality ConfigStore.
communication-config-technologies-file	Ns-3 file (.xml) where the communication module installers to be used in the simulation are specified. An installer allows installing a given communication module (i.e. WAVE, WiMAX, etc.) in a node in ns-3 based on iCS requests.

Table 5. Wireless communications simulator configuration tag set.

```
<communicationsim>
        <communication-executable value="main-inci5"/>
        <communication-host value="localhost"/>
        <communication-port value="1982"/>
        <communication-general-params-file value="configGeneral.xml"/>
        <communication-config-technologies-file
value="configTechnologies-ics.xml"/>
</communicationsim>
```

The traffic management applications are present in the main file with a dedicate tag addressing the path of XML configuration file devoted to store the necessary values to locate the applications in the operating system, launch and connect the iCS to them.

TAG NAME	DESCRIPTION
app-config-file	Path of the file devoted to store the necessary values to locate the Application.

Table 6. Application configuration file address tag.

```
<apps>application-config-file.xml</apps>
```

To close the iTETRIS master configuration file one more section is defined to hold the path of the files in which the iCS writes logging values. This section, unlike the ones described so far, is optional and can be omitted (Table 7).

TAG NAME	DESCRIPTION
ics-log-path	The iCS logs file location and name.
ics-log-level	The iCS logs file message. Options: INFO, WARNING, ERROR.
ics-log-time-size	To control the size of the logging files. Defines the interval of time of the logging messages generated per file.
ns3-log-path	The ns-3 logs file location and name. Is delivered as an argument to ns-3.

Table 7. Logging configuration tag set.

Example iTETRIS master configuration XML file:

```
<configuration>
	<scenario>
		<begin value="0"/>
		<end value="3000"/>
		<penetration-rate value="100"/>
		<facilities-config-file value="facilities-config-file.xml"/>
		<message-reception-window value="5"/>
		<interactive value="false"/>
	</scenario>
	<trafficsim>
		<traffic-executable value="sumo -c"/>
		<traffic-file value="bolonia.traci.sumo.cfg"/>
		<traffic-host value="localhost"/>
		<traffic-port value="5500"/>
	</trafficsim>
	<communicationsim>
		<communication-executable value="main-inci5"/>
		<communication-host value="localhost"/>
		<communication-port value="1982"/>
		<communication-general-params-file value="configGeneral.xml"/>
		<communication-config-technologies-file value="configTechnologies-
ics.xml"/>
	</communicationsim>
	<applications>
		<app-config-file value="application-config-file.xml"/>
	</applications>
	<logs>
		<ics-log-path value="ics-log.txt"/>
		<ics-log-level value="INFO"/>
		<ics-log-time-size value="100"/>
		<ns3-log-path value="ns3-log.txt"/>
	</logs>
</configuration>
```

Besides, the iTETRIS platform offers the possibility to the user to create the iTETRIS XML master file from the command line following the format of the example below:

```
iCS --end 10 –penetration-rate 100 \
--facilities-config-file facilities-config-file.xml \
--traffic-host localhost \
--traffic-port 1984 \
--traffic-file bolonia.sumo.cfg \
--communication-file ns3-config.txt \
--communication-host localhost \
--communication-port 1982 \
--ics-log-path ics-log.txt \
--ns3-log-path ns3-log.txt \
```

```
--apps /home/user/application-config-file.xml \
--ics-log-path ics-log.txt \
--ns3-log-path ns3-log.txt \
--save-configuration itetris-config-file.xml
```

A.1.2 iCS Facilities Configuration

iTETRIS is aligned with the communication architecture defined by ETSI for Intelligent Transport Systems (ITS). Part of the architecture is the Facilities layer. The components of the layers are divided between ns-3 and iCS in order to improve the performance of the platform. This section describes how the Facilities in the iCS are configured.

The iCS Facilities configuration file complements the data of the iTETRIS master file described in the previous chapter. The file contains the paths and the name of the files for the configuration of the iCS Facilities block and the coordinates of the origin point of the map.

A.1.2.1 Facilities Configuration File

```
<facilities>
  <localCoordinates latitude="44.494218" longitude="11.346486"
altitude="0.0"/>
  <mapConfig mapConFilename="map.net.xml" />
  <stationsConfig stationsConFilename="stations-config-file.xml" />
  <LDMrulesConfig LDMrulesConFilename="LDMrules-config-file.xml" />
</facilities>
```

A.1.2.3 Local Coordinates

iTETRIS, as SUMO have an internal carthesian coordinate representation (X,Y) according to a lower left corner being the origin. In order to be able import and export these coordinate to absolute coordinates, the iTETRIS lower-left corner origin must have a GPS coordinate.

A.1.2.4 Map Configuration File

The map configuration file is an XML file that contains the description of the topology map. In the case where the traffic simulator is SUMO, the map file is the net.xml file that describes the road topology for the SUMO simulation.

A.1.2.5 Stations Configuration File

In the following example the stations configuration file is presented.

```
<stations>
    <default>
        <RATseed value="12345" />
        <mobileStas>
            <mobileSta RAT-type="0" penetration-rate="100"
communication-profile="WaveVehicle"/>  <!-- .11p -->
            <mobileSta RAT-type="1" penetration-rate="80"
communication-profile="UmtsVehicle"/>  <!-- UMTS -->
            <mobileSta RAT-type="2" penetration-rate="20"
```

```
communication-profile="WimaxVehicle"/> <!-- WiMAX -->
             <mobileSta RAT-type="3" penetration-rate="10"
communication-profile="DvbVehicle"/>   <!-- DVB-H -->
        </mobileStas>

        <fixedStas> <!-- Each Fixed Station has only one RAT available
-->
            <fixedSta id="1" x="0" y="0" RAT-type="0" enabledRAT="0"
communication-profile="WaveRsu"/>
        </fixedStas>
    </default>
</stations>
```

In the example given above we can see that the file is divided in two parts. In the first part, information about the radio access technologies is given, while in the second part the fixed stations are declared together with their ID, position (either as local x-y coordinates or as latitude and longitude) and radio access technology used (and whether this is enabled at the beginning of the simulation).

More specifically, the part about the mobile station contains the seed value of the random generator used for defining which technologies every mobile station entering the simulation area is equipped with. The default penetration rate of each radio access technology is defined below the random seed. Note that also a communication-profile field is given. In this field it is specified, if necessary, the specific profile that the communication simulator (e.g., ns-3) will use for the mobile stations. In addition, if more than one profile is defined for a certain technology, this has to be considered as different technology for the iCS. That more specifically means that the enum variable RATID in the iCStypes.h file must contains additional entries than the basic {WAVE, UMTS, WiMAX, DVBH}.

A.1.2.6 LDM Configuration File

Finally, the LDM-rules-configfile.xml contains necessary parameters for the configuration of the LDM logic implemented inside the iCS Facilities. The skeleton of the XML file is provided.

```
<LDMrules>

    <defaultMessageLifeInterval value="" />    <!-- Value expressed in
simulation steps -->
    <relevantStationTypes fixed="" mobile="" /> <!-- 1=TRUE, 0=FALSE -
->
    <relevantMessages cam="" denm=""/> <!-- 1=TRUE, 0=FALSE -->

    <relevantArea>
        <circle centerX="" centerY="" [centerLat="" centerLon=""]
radius=""/>
        <ellipse focus1X="" focus1Y="" [focus1Lat="" focus1Lon=""]
focus2X="" focus2Y="" [focus2Lat="" focus2Lon=""] eccentricity=""/>
        <ellipse centerX="" centerY="" [centerLat="" centerLon=""]
majorAxis="" minorAxis="" rotationAngleRadians="" />

        <rectangle vertexAX="" vertexAY="" [vertexALat=""
vertexALon=""] vertexBX="" vertexBY="" [vertexBLat="" vertexBLon=""]
vertexCX="" vertexCY="" [vertexCLat="" vertexCLon=""] vertexDX=""
```

```
vertexDY="" [vertexDLat="" vertexDLon=""] />

        <rectangle centerX="" centerY="" [centerLat="" centerLon=""]
pointAX="" pointAY="" [pointALat="" pointALon=""] pointBX=""
pointBY="" [pointBLat="" pointBLon=""] />

        <rectangle pointAX="" pointAY="" [pointALat="" pointALon=""]
pointBX="" pointBY="" [pointBLat="" pointBLon=""] height="" />

        <convexPolygon>
            <vertex X="" Y="" [Lat="" Lon=""] />
            <vertex X="" Y="" [Lat="" Lon=""] />
            <vertex X="" Y="" [Lat="" Lon=""] />
            <vertex X="" Y="" [Lat="" Lon=""] />
            <vertex X="" Y="" [Lat="" Lon=""] />
        </convexPolygon>

         <lanes>
             <lane ID="">
         </lanes>

         <edges>
             <edge ID="">
         </edges>

         <junctions>
             <junction ID="">
         </junctions>

    </relevantArea>

</LDMrules>
```

In the XML configuration file the only mandatory field is the defaultMessageLifeInterval. This value defines the number of simulation steps before deleting a message from the facilities tables when the expiry time is not explicitly defined (e.g. in CAM messages it is not defined, while in DENM messages it is).

The other fields in the XML are optional and define the rules for the relevance of the received messages. If a message is not relevant, it is not stored in the facilities tables. The rules accounts the receiver station type (a receiver is considered relevant only if it is of a certain type), message type (fixed stations or mobile stations or both are considered as relevant receivers) and relevance area (only receivers within a defined area are considered as relevant).

The relevance area, if set can be specified in different and various ways, as indicated in the example above. Note that the overall relevance area is the union of the areas indicated in the file. These areas can be either geometrically defined (please note that the points can be defined as x-y coordinates or as latitude and longitude) or road elements such as road lanes, road edges (i.e. streets segments) or junctions.

A.1.3 Traffic Management Applications Configuration File

The last of the set of configuration files that feed with necessary values to the iCS regards the Applications. In this file the controller discovers how many Applications are going to be evaluated in the simulation and their specific details.

Likewise the iTETRIS master and iCS configuration files this file it is also formatted using the XML mark-up language. It consists in one section (Table 8) that is repeated in the file as many times as Applications are going to be running in the simulation.

TAG NAME	DESCRIPTION
Name	The name of Application given by the traffic experts.
Executable	The command line string necessary to create the execution thread of the Application in the operating system.
Ip	The IP of the machine running the Application.
Port	Socket port number where the Application is expecting commands
Seed	Value to generate random numbers for the Application assignment to a station.
Rate	The user is able to assign a degree of the penetration of the Applications in the given scenario by defining a rate in terms of percentage.
result-container	To let the iCS know how to behave when a result of the Application is received a set of values that correspond to the actions are defined.
serviceId	If the Application has to use one or more of the transmission types available in ns-3 the traffic developer has to correctly add the string in each of the attributes: unicast, multicast, broadcast, geobroadcast, topobroadcast.
Id	Although the criterion to install the Applications in the vehicles is random, it is possible to define a set of stations that definitely will run one particular application. This only applies to fixed stations already defined in the configuration files part of the "ETSI Facilities" described in the stations configuration file section (4.2.3).

Table 8. Applications configuration file tag summary.

The value 0 is reserved as the ID value for the Traffic Management Control (TMC) node. This station represents a headquarters station that could also run Applications and send back results. The communications of this station are not modelled.

The following is an example of the XML configuration file based on the ITS Cooperative Demo Application included with the iTETRIS source code:

```
<Applications>
      <Application>
            <name>DEMO APP</name>
            <executable>demoapp</executable>
            <ip>localhost</ip>
            <port>1986</port>
            <seed>123456</seed>
            <rate>0</rate>
            <result-container>OUTPUT_SET_SPEED_ADVICE_DEMO</result-
container>
```

```
            <serviceId unicast="serviceIdUnicast"
                  multicast="serviceIdMulticast"
                  broadcast="serviceIdBroadcast"
                  geobroadcast="serviceIdGeobroadcast"
                  topobroadcast="serviceIdTopobroadcast"/>
            <stations>
            <id>5000</id>
            <id>6000</id>
            </stations>
      </Application>
      <!--
      <Application>
            Describe your other Application here
      </Application>
      -->
</Applications>
```

A.1.4 ns-3 Configuration Files

In order to employ any of the implemented technologies within iTETRIS it is necessary to configure ns-3 first. The main configuration file is the one indicated in the iTETRIS Master Configuration File, within the “communication-config-technologies-file” tag. The content of the file is presented in the following example.

```
<ns3Configuration>
  <randomGenerators>
      <randomGenerator seed="1" runNumber="1" />
  </randomGenerators>
  <installers>
      <installer type="ns3::WaveVehicleInstaller" name="WaveVehicle"
file="confWaveVehicle2.xml" default="false" />
      <installer type="ns3::WaveRsuInstaller" name="WaveRsu"
file="confWaveRsu.xml" relatedInstaller="WaveVehicle" default="false"
/>
      <installer type="ns3::WifiVehicleInstaller" name="WifiVehicle"
file="confWifi.xml" default="false" />
      <installer type="ns3::UmtsBsInstaller" name="UmtsBs"
file="confUmtsBs.xml" default="false" />
      <installer type="ns3::DvbhVehicleInstaller" name="DvbhVehicle"
file="confDvbhVehicle.xml" default="false" />
      <installer type="ns3::DvbhBsInstaller" name="DvbhBs"
file="confDvbhBs.xml" default="false" />
      <installer type="ns3::UmtsVehicleInstaller" name="UmtsVehicle"
file="confUmtsVehicle.xml" default="false" />
      <installer type="ns3::WifiBsInstaller" name="WifiBs"
default="false" />
      <installer type="ns3::ItetrisNetworkTransportInstaller"
name="NetTrans" default="true" />
      <installer type="ns3::CamMngtInstaller" name="CamMngt"
default="true" />
      <installer type="ns3::MobilityModelInstaller" name="Mobility"
default="true" />
  </installers>
```

```
</ns3Configuration>
```

This file is used to set the installers for the implemented technologies, providing the configuration file of each technology in the value "file". These configuration files need to be located in the same folder as the main configuration file. iTETRIS provides support for ITS-G5, UMTS, DVB-H and LTE, but COLOMBO being focused on ITS-G5, we only describe its configuration file.

A.1.4.1 WAVE Configuration Files

WAVE distinguishes two different types of nodes, "WaveVehicle" and "WaveRsu". While the "WaveVehicle" has mobility capabilities, "WaveRsu" is fixed, presents higher antennas heights and is able to directly connect to a traffic management centre. Two examples of configuration files for both types of nodes are "confWaveVehicle-pasubio-joined-buildings.xml" and "confWaveRsu.xml". The next tables show the content of those files.

The first one corresponds to the "WaveVehicle" properties. As it can be observed, the configuration file is divided into several groups of parameters that are delimited by tags, "yansWifiChannel", "yansWifiPhy", "qosWifiMac", "mobilityModel" and applications. The group "yansWifiChannel" allows configuring the parameters related to the radio propagation channel. The main parameters in "yansWifiChannel" are:

1. propagationLoss: configures the propagation loss model to be used. The name of the propagation loss model and the set of attributes (e.g. Frequency [Hz], "EffEnvironmentHeight" [m], etc.) have to be provided.
2. visibilityModel: configures the visibility model to be used. The visibility model provides the location of the obstacles (e.g. buildings) present in the scenario. The visibility model is used to determine the LOS/NLOS conditions between transmitter and receiver. In the example, the visibility model employed provides the locations of the buildings present in the "Pasubio/Acosta" scenario.
3. shadowingModel: configures the shadowing model to be used. Correlated and uncorrelated shadowing can be simulated.

The group "yansWifiPhy" allows configuring the parameters related to the PHY layer. Different parameters can be selected for the CCH and the SCH "NetDevices". The main parameters in "yansWifiPhy" are:

1. "TxGain" and "RxGain" (dB).
2. TxPowerLevels (integer number).
3. TxPowerEnd (dBm): maximum transmission power.
4. TxPowerStart (dBm): minimum transmission power. The minimum transmission power is employed by default. If other power level wants to be used, it has to be specified in a per-packet basis though the transmission-power packet tag.

The group "qosWifiMac" allows configuring the parameters related to the MAC layer, such as "RtsCtsThreshold" and "FragmentationThreshold". The group "mobilityModel" allows configuring the antenna height ("AntennaHeight" [m]) of the vehicle. As in other radio access technologies, the parameters of the applications are grouped within the tag applications.

```
<waveVehicle>

 <yansWifiChannel>
  <propagationLoss name="ns3::WinnerB1LossModel" />
  <propagationLoss attribute="Frequency" value="5.9e9" />
  <propagationLoss attribute="EffEnvironmentHeight" value="1" />
  <visibilityModel name="ns3::BuildingMapModel" />
  <visibilityModel attribute="VisibilityMapFile"
value="/home/ramon/iTETRIS_feb/trunk/ns-
3/scratch/joined_Buildings_v2.txt" />
  <shadowingModel name="ns3::ShadowingModel" />
  <shadowingModel attribute="CorrelatedShadowing" value="false" />
 </yansWifiChannel>

 <yansWifiPhy>
  <wavePhyCCH attribute="TxGain" value="0.0" />
  <wavePhyCCH attribute="RxGain" value="0.0" />
  <wavePhyCCH attribute="TxPowerLevels" value="60" />
  <wavePhyCCH attribute="TxPowerEnd" value="33" />
  <wavePhyCCH attribute="TxPowerStart" value="20" />
  <wavePhySCH attribute="TxGain" value="0.0" />
  <wavePhySCH attribute="RxGain" value="0.0" />
  <wavePhySCH attribute="TxPowerLevels" value="60" />
  <wavePhySCH attribute="TxPowerEnd" value="33" />
  <wavePhySCH attribute="TxPowerStart" value="20" />
 </yansWifiPhy>

 <qosWifiMac>
  <wifiRemoteStationManager attribute="RtsCtsThreshold" value="2000"/>
  <wifiRemoteStationManager attribute="FragmentationThreshold"
value="2300" />
 </qosWifiMac>

 <mobilityModel>
  <antenna attribute="AntennaHeight" value="1.5" />
 </mobilityModel>

 <applications>
  <application>
   <C2CIP itetrisName="serviceIdUnicast"/>
   <C2CIP attribute="PortC2C" value="7090"/>
   <C2CIP attribute="PortIP" value="7091"/>
   <C2CIP attribute="Frequency" value="2"/>
   <C2CIP attribute="PacketSize" value="300"/>
  </application>
  <application>
   <C2CIP itetrisName="TopoBroadcast" />
   <C2CIP attribute="PortC2C" value="8083" />
   <C2CIP attribute="PortIP" value="8082" />
   <C2CIP attribute="Frequency" value="2" />
   <C2CIP attribute="PacketSize" value="300" />
  </application>
 </applications>
```

```
</waveVehicle>
```

The configuration of the “WaveRsu” is similar to the configuration of the “WaveVehicle”, but in this case there is no need to define the parameters to configure the radio propagation channel, since “WaveRsu” and “WaveVehicle” share the same channel (which is defined in the configuration file “confWaveVehicle-pasubio-joined-buildings.xml”

```
<waveVehicle>

 <yansWifiPhy>
  <wavePhyCCH attribute="TxGain" value="0.0" />
  <wavePhyCCH attribute="RxGain" value="0.0" />
  <wavePhyCCH attribute="TxPowerLevels" value="60" />
  <wavePhyCCH attribute="TxPowerEnd" value="33" />
  <wavePhyCCH attribute="TxPowerStart" value="20" />
  <wavePhySCH attribute="TxGain" value="0.0" />
  <wavePhySCH attribute="RxGain" value="0.0" />
  <wavePhySCH attribute="TxPowerLevels" value="60" />
  <wavePhySCH attribute="TxPowerEnd" value="33" />
  <wavePhySCH attribute="TxPowerStart" value="20" />
 </yansWifiPhy>

 <mobilityModel>
  <antenna attribute="AntennaHeight" value="6" />
 </mobilityModel>

 <applications>
  <application>
   <C2CIP itetrisName="serviceIdUnicast" />
   <C2CIP attribute="PortC2C" value="7090" />
   <C2CIP attribute="PortIP" value="7091" />
   <C2CIP attribute="Frequency" value="2" />
   <C2CIP attribute="PacketSize" value="300" />
  </application>
  <application>
   <C2CIP itetrisName="TopoBroadcast" />
   <C2CIP attribute="PortC2C" value="8083" />
   <C2CIP attribute="PortIP" value="8082" />
   <C2CIP attribute="Frequency" value="2" />
   <C2CIP attribute="PacketSize" value="300" />
  </application>
 </applications>

</waveVehicle>
```