Komondor User's Guide

Sergio Barrachina-Muñoz and Francesc Wilhelmi

November 1, 2018

Contents

List of Figures

List of Tables

1 Introduction

Komondor [1] is an event-based simulator based on COST [2], and which is mostly intended to reproduce the novel techniques included in the IEEE 802.11ax-2019 amendment [3].

In this document, we aim to illustrate the installation and execution procedures that must be carried out to successfully perform Komondor simulations. The main goal is therefore to encourage researchers and other practitioners to use the Komondor simulator for their own experiments, and even to participate in the project.

For more detailed information about the IEEE 802.11-based functionalities that are implemented in Komondor, please refer to this document.

Disclaimer: Komondor is a project that is constantly evolving, and the contents provided in this document are subject to changes.

1.1 Overview of Komondor

Komondor is composed by several modules that allow performing simulations with a high degree of freedom. Here we provide a general overview on the most important module, which are illustrated in Figure 1.

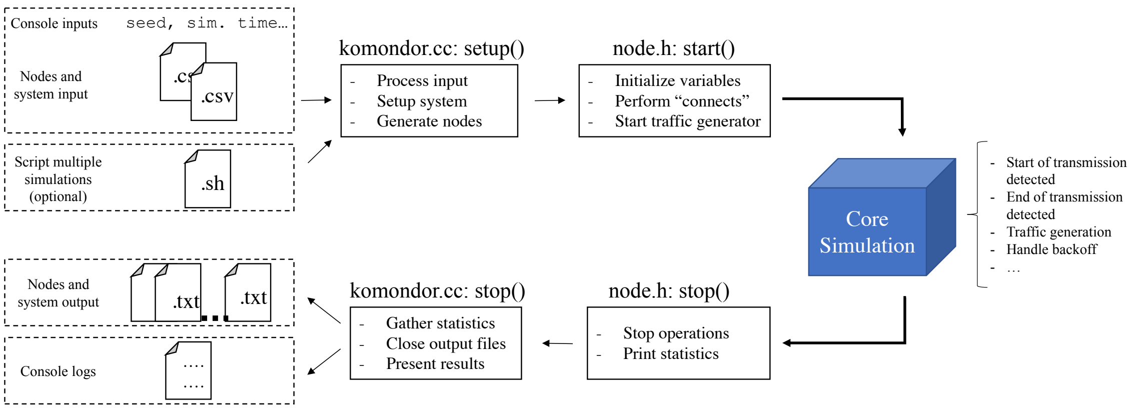

Figure 1: Komondor flowchart

Komondor receives a set of inputs (information about the nodes, system configuration, simulation time, etc.) and initializes the main module (i.e., *komondor.cc*), which is in charge of generating the network and collecting the information generated within the simulation time. In particular, *komondor.cc* initializes each node in the network (*node.h* components), which act as independent entities and generate data on their own. As a result, during the core simulation, nodes interact among each other by sending packets, so that the DCF operation is implemented for accessing to the channel. Finally, when the simulation runs out (the simulation time is over), a set of outputs are generated with regards to network performance.

The input and output layers are described with more detail later in Sections 3.2 and 3.3.

1.2 Files Organization

To properly understand the Komondor's operation, it is important to know how the project allocated in Github is organized. In particular, the project is divided into three main folders: *Apps*, *Code* and *Documentation*. *Apps* and *Documentation* folders are intended to contain additional material that supplements the core Komondor's infrastructure (this document, for instance, is inside the *Documentation* folder). Here we focus on the code part, which is organized as follows (refer to Figure 2 as well):

- **COST**: constitute the Komondor's primitive operation. Here we find the CompC++ library that allows generating discrete event simulations. For further information about COST, please refer to its main website.

- **main**: contains the core files (`komondor.cc`, `node.h`, `agent.h` and `central_controller.h`) that are in charge of orchestrating all the simulation. `komondor.cc` is the main component, which initializes all the other components of *Type II*. All these modules are aware of the existence of the simulation time. In addition to the core components, here we find `build_local`, a bash script that compiles the libraries for executing the code. Note that file `compcxx_komondor_main.h` is also required to carry out such a compilation.
- **methods**: by following clean architecture guidelines, independent methods used by core files are contained in the methods folder. Several libraries are provided according to the nature of their functions. For instance, `backoff_methods.h` contains methods to handle the backoff operation in DCF.
- **structures**: the Komondor simulator considers several header files to carry out its operation. Among them, we find `wlan.h`, which defines the main characteristics of a WLAN (WLAN id, list of associated STAs, etc.). In addition, the `notification.h` structure allows to define the information to be exchanged between devices.
- **learning_modules**: here we find the implementation of Machine Learning (ML) methods that receive feedback about the networks performance in simulation time. The utilization of learning mechanisms is further described later in Section 3.2.3.
- **list_of_macros.h**: all the static parameters (e.g., constants) are contained in this file.
- **input**: contains the input files that allow building the simulation environment.
- **output**: contains the data generated by Komondor as a result of a given simulation.
- **scripts_multiple_executions**: contains bash scripts to perform multiple simulations.

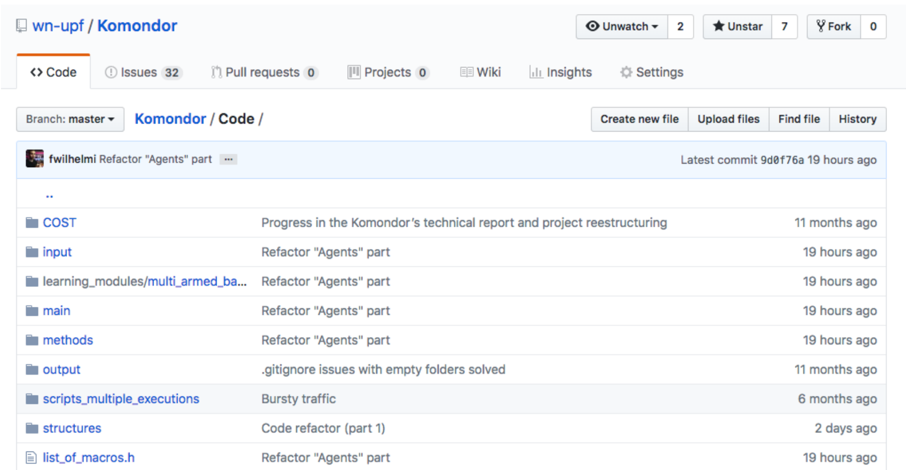

Figure 2: Komondor files organization

2 Installation

3 Compilation and Execution

Here we provide the guidelines to properly compile and execute the Komondor simulator. First, we show the execution modes available, and then we describe both the input and the output of the system.

3.1 Execution Modes

To compile and execute Komondor, the following instructions must be followed[1]:

1. Set the .csv input files (further defined in Section 3.2)
2. Access to the "main" directory
3. Execute "./build_local". This file contains the instructions for compiling the Komondor code. It has been updated to enable debugging with Valgrind[2].
4. Execute *./KomondorSimulator arg_1 arg_2 ... arg_n*, where *arg_i* is the i_{th} input argument.
5. Collect the results either in the output files or in the console.

NOTE: in case that the user does not have permissions to execute some of the files, grant them by introducing the following command in the target folder: `$ chmod -R 777 *`.

The execution mode of Komondor depends on the provided input files and the arguments introduced by console. The full list of parameters is described in Table 1, and the currently implemented execution modes are summarized below:

- **Basic execution:** simulates the operation of basic WLANs, according to the parameters indicated in the system and nodes input files. Arguments [1, 2, 4, 5, 6, 7, 9, 10, 12, 13] are required for the basic execution.
- **Quick execution:** allows to perform the basic execution by introducing only few parameters. Only the mandatory arguments are required for this kind of execution.
- **Script-based execution:** in this case, the execution is ordered by a bash script, which uses arguments [1, 2, 5, 12, 13].
- **Execution with agents:** in case of including agents, additional parameters must be added to inform about the necessary information that enables the creation of agents. In this case, all the arguments are required to be specified at the console.

3.2 Input

The input files allow Komondor to generate the simulation environment. In particular, we find three main types of input files, which are described in the following subsections.

3.2.1 System input

The system input is introduced through a .csv file, and defines global input parameters such as the number of basic channels considered or the propagation models. System input parameters are defined in Table 2.

An example of system input file can be found in the Github repository: *Komondor/Code/input/input_example/input_system_conf.csv*.

3.2.2 Nodes input

Each WLAN is built according to the nodes input, which can define both Access Points (APs) and stations (STAs). In particular, the nodes input introduces characteristics such as type of device, location, or implementing features (e.g., DCB policy). There are two ways of generating nodes, which is explicitly indicated to Komondor through the file name:

- In case of including the keyword *nodes*, all the devices (both APs and STAs) must be introduced and described.

[1] A GNU-based OS is assumed to be used for simulations, including basic compilation programs such as *gcc*.

[2] Valgrind is a programming tool for memory debugging, memory leak detection, and profiling. Valgrind main website: http://valgrind.org/

ID	Type	Name	Description
1	Mandatory	system_input_filename	File containing system information (e.g., number of channels available, traffic model, etc.). The file must be a .csv with semicolons as separators.
2	Mandatory	nodes_input_filename	File containing nodes information (e.g., position, channels allowed, etc.). The file must be a .csv with semicolons as separators.
3	Optional	agents_input_filename	File containing agents information (e.g., WLAN id, method to execute, possible actions, etc.). The file must be a .csv with semicolons as separators.
4	Optional	script_output_filename	Path to the output file to which write results at the end of the execution (if the file does not exist, the system will create it).
5	Optional	simulation_code	Code provided to a specific simulation.
6	Optional	save_system_logs	Flag to indicate whether to save the system logs into a file (1) or not (0).
7	Optional	save_node_logs	Flag to indicate whether to save the nodes logs into separate files (1) or not (0). If this flag is activated, one file per node will be created.
8	Optional	save_agent_logs	Flag to indicate whether to save the agents logs into separate files (1) or not (0). If this flag is activated, one file per agent will be created.
9	Optional	print_system_logs	Flag to indicate whether to print the system logs (1) or not (0).
10	Optional	print_node_logs	Flag to indicate whether to print the nodes logs (1) or not (0).
11	Optional	print_agent_logs	Flag to indicate whether to print the agents logs (1) or not (0).
12	Mandatory	sim_time	Simulation time in seconds.
13	Mandatory	seed	Random seed for the experiments.

Table 1: List of arguments to be passed to Komondor by console.

Parameter	Type	Description
num_channels	int	Maximum number of frequency channels in the system
basic_channel_bandwidth	int	Bandwidth for each channel [MHz]
pdf_backoff	int	PDF to compute the backoff ()
pdf_tx_time	int	PDF to compute the tx time ()
packet_length	int	Length of data packets [bits]
ack_length	int	Length of ACK packets [bits]
num_packets_aggregated	int	Number of packets aggregated per transmission
path_loss_model	int	Path-loss model (0: FSPL, 1: Hata, 2: Indoor 1, 3: Indoor 2, 4: TGax scenario 1)
capture_effect	int	Capture Effect Threshold [dB]
noise_level	int	Floor noise level [dBm]
adjacent_channel_model	int	Co-channel interference model (0: without adjacent interference, 1: contiguous adjacent interference, 2: complete adjacent interference)
collisions_model	int	Collisions model (reserved)
SIFS	int	SIFS period [μs]
constant_PER	int	Defines a constant Packet Error Rate
traffic_model	int	Traffic model (0: full buffer, 1: Poisson distr., 2: deterministic distr.)
backoff_type	int	Type of backoff (discrete: 0, continuous: 1)
rts_length	int	Length of RTS packets [bits]
cts_length	int	Length of CTS packets [bits]
cw_adaptation	bool	For activating CW adaptation
pifs_activated	bool	For activating PIFS
capture_effect_model	int	Capture effect model

Table 2: System input parameters description.

- Otherwise, if including the keyword *aps*, only the APs are defined, so that a set of STAs is randomly generated under certain introduced parameters (e.g., minimum/maximum number of STAs, maximum distance between APs and STA).[3]

Table 3 describes the Nodes input parameters for both *nodes* and *aps* files.

Parameter	Type	Nodes or APs	Description
node_code	String	nodes	Code assigned to the node
node_type	int	nodes	Type of node (0: AP, 1: STA)
wlan_code	String	both	Code assigned to the WLAN
destination_id	int	nodes	To specify the ID of the destination (packets would be only sent to that devices). Setting it to -1 indicates random destination.
min_sta_number	int	aps	Minimum number of associated STAs
max_sta_number	int	aps	Maximum number of associated STAs
max_distance_sta	int	aps	Maximum distance of associated STAs
x	int	both	X location [m]
y	int	both	Y location [m]
z	int	both	Z location [m]
primary_channel	int	both	Primary channel
min_channel_allowed	int	both	Left channel in range
max_channel_allowed	int	both	Right channel in range
cw	int	both	Fixed CW
cw_stage	int	both	Initial CW stage (for CW adaptation)
tpc_min	int	both	Minimum transmit power allowed [dBm]
tpc_default	int	both	Default transmit power allowed [dBm]
tpc_max	int	both	Maximum transmit power allowed [dBm]
cca_min	double	both	Minimum CCA allowed [dBm]
cca_default	double	both	Default CCA allowed [dBm]
cca_max	double	both	Maximum CCA allowed [dBm]
tx_antenna_gain	int	both	Gain of the tx antenna [dB]
rx_antenna_gain	int	both	Gain of the rx antenna [dB]
channel_bonding_model	int	both	Channel bonding model (0: only primary, 1: SCB, 2: SCB log2, 3: always max, 4: always max log2, 5: always max log2 MCS, 6: uniform probability log2)
modulation_default	int	both	Modulation set by default (0 to use dynamic MCS)
central_freq	int	both	Frequency band used (2,4 or 5 GHz)
lambda	float	both	Packets transmission rate [packets/s]
ieee_protocol	int	both	IEEE protocol used
traffic_load	double	both	Traffic load (pkts/s)

Table 3: Nodes input parameters description.

As a final remark, in order to ensure a proper execution, it is mandatory to introduce an input file with a list of nodes ordered by *node_id* and starting with $node\_id = 0$ (it is a requirement for the array responsible of storing the power perceived by each node). An example of nodes input file can be found in the Github repository: *Komondor/Code/input/input_example/input_nodes.csv*.

3.2.3 Agents input

Apart from the system and nodes definition, a given user may make use of the learning modules included in Komondor. To that end, the agents input file must be defined, so that specific agents' characteristics (e.g., associated WLAN, learning mechanism or modifiable parameters) are introduced. Table 4 describes the agents input parameters.

In particular, an agent may act on its own or be part of a centralized system. At the current development stage, the central entity is not fully programmed, and would require from an additional

[3]The usage of APs input files is discouraged to the lack of maintenance.

Parameter	Type	Description
wlan_code	String	Code corresponding to the WLAN to which the agent is embedded to. Note that the same code must be defined in the nodes input file.
centralized_flag	int	Flag to indicate whether the node is attached to a centralized system (1) or not (0).
time_between_requests	double	Time in seconds between requests performed by the agent to the AP
actions_channels	int*	Array of channels available to be selected by the agent.
actions_cca	double*	Array of possible CCA values (in dBm) to be selected by the agent.
actions_tx_power	double*	Array of possible transmission power values (in dBm) to be selected by the agent.
actions_dcb_policy	int*	Array of possible DCB policies to be selected by the agent.
type_of_reward	int	Type of reward to be used by the agent.
learning_mechanism	int	Type of learning mechanism to be used by the agent.

Table 4: Agents input parameters description.

dedicated input file to indicate its operational details. When it comes to the learning mechanisms, so far the `multi_armed_bandits` module is available, which has id 1. Future developments are expected to contemplate the implementation of other Reinforcement Learning (RL) techniques such as Q-learning or Neural Networks. To that end, interactions with other software systems are being contemplated.

An example of agents input file can be found in the Github repository: *Komondor/Code/input/input_example/agents.csv*.

3.2.4 Input scripts

In order to facilitate users work, we provide a set of scripts that allow performing several simulations at once, which is useful to avoid processing different output files. Such sample scripts can be found in the "scripts multiple executions" folder, which perform the following operations:

- *multiple_inputs_script.sh*: processes all the input files contained in ./input/script_input_files and generates a simulation for each one.
- *multiple_inputs_script_several_seeds.sh*: in addition to process multiple inputs, generates different seeds for each simulation.

Similar procedures can be implemented to extend the current provided functionalities, such as reading multiple system inputs or generating specific output reports.

3.3 Output

The results collected by Komondor can be analyzed after its execution through the generated output. Two information sources are provided: *i)* console logs, and *ii)* logs saved in separated files. The logs generation can be enabled/disabled through the arguments passed to Komondor by console (refer to Section 3.1). Note, as well, that major increases in the execution time may occur if logging is activated. For example, for a basic simulation of 4 nodes, simulating 1,000 seconds takes 1.12 s and 15.67 s when not logging and when doing so, respectively.

3.3.1 Output logs

Komondor, by default, generates a set of console logs regarding the current simulation. For completeness, additional logs can be displayed in relation to node statistics. Such statistics include throughput experienced, collisions, nodes sent, RTS/CTS sent, etc. An example of nodes and system statistics is shown in Figures 3 and 4.

3.3.2 Output files

Each node can generate logs about its activity, thus describing its IEEE 802.11 operation and the interaction with the other nodes. Such an information is copied into files, which are saved into the *output* folder. In addition agent output files can be generated, thus providing information about the learning procedures.

Finally, in order to make the output files more understandable and classifiable, logs are categorized according to the event that generates it. With that, further filtering processes can be carried out by developers. A potential application may be a network animator. Table 5 describes the codes used for each type of event.

3.4 Execution examples

Here we provide few execution examples with the aim of clarifying the information provided along this section.

3.4.1 Example 1: Basic execution

For this simulation, we use the system and nodes input files that can be found at the "input_example" folder.

The first step is to compile the code through the `build_local` file. For that, we access to the "main" folder and execute:

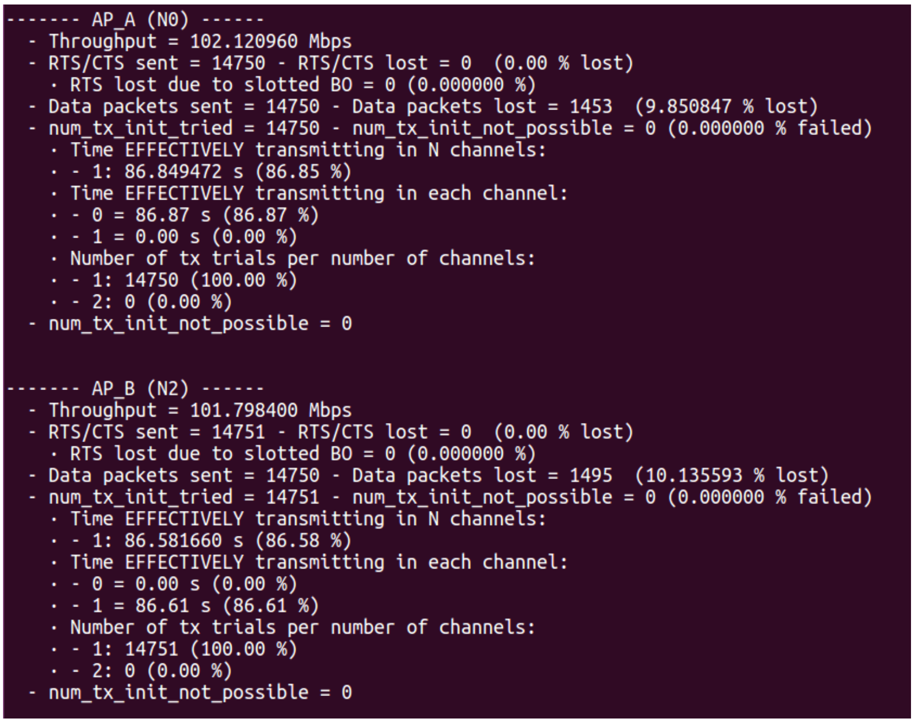

Figure 3: Example of nodes statistics in Komondor

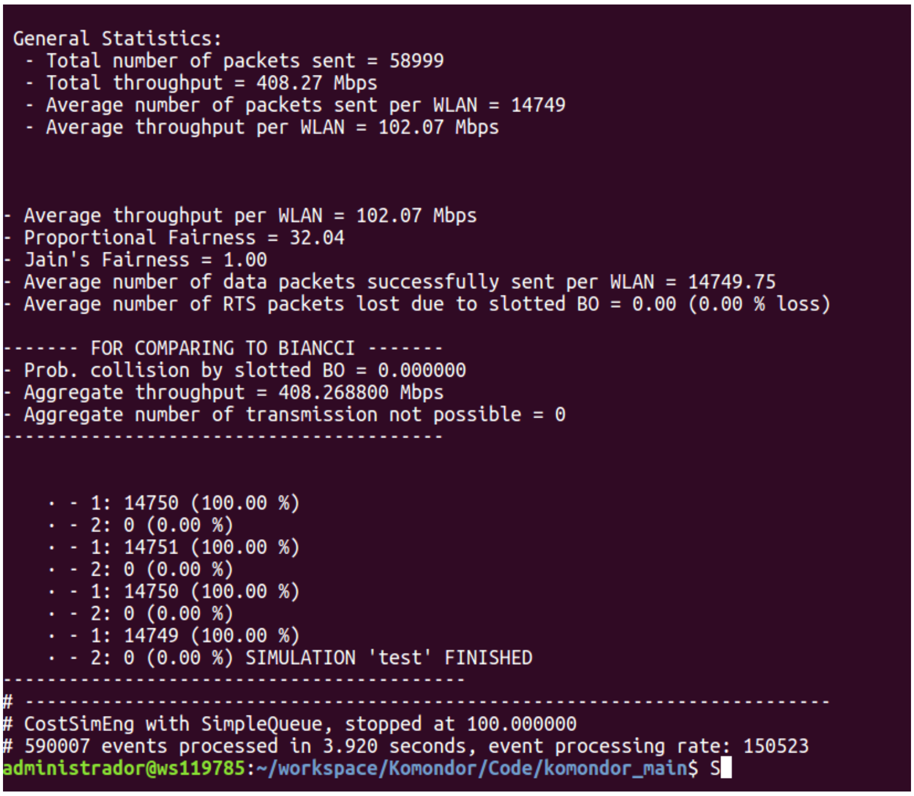

Figure 4: Example of system statistics in Komondor

Listing 1: Compling the code

```
$ cd Komondor/Code/main
$ ./build_local
```

Once the code is compiled without errors, execute the simulation as follows:

Method	Type	Sub-type	Event description
Setup()	A	-	-
Start()	B	B00	Start()
		B01	Start() end
		B02	Node's info (one line)
Stop()	C	C00	Stop()
		C01	Stop() end
		C02	Time transmitting in number of channels
		C03	Time transmitting in each channel
		C04	Packets sent
		C05	Throughput
inportSomeNodeStartTX()	D	D00	inportSomeNodeStartTX()
		D01	inportSomeNodeStartTX() end
		D02	Node N has started a TX in channels: c_left - c_right
		D03	Pre update channel state
		D04	Distance to transmitting node
		D05	Power received from transmitting node
		D06	Post update channel state
		D07	I am (or not) the TX destination
		D08	Current SINR
		D09	Capacitiy
		D10	Primary channel affected (or not)
		D11	Power sensed in primary channel
		D12	CCA exceeded (or not)
		D13	Backoof active (or not)
inportSomeNodeFinishTX()	E	E00	inportSomeNodeFinishTX()
		E01	inportSomeNodeFinishTX() end
		E02	N%d has finished a TX in channel range: %d - %d
		E03	Initial power of transmitter
		E04	Pre update channel state
		E05	Post update channel state
		E06	Primary channel affected (or not)
		E07	Power sensed in primary channel
		E08	CCA exceeded (or not)
		E09	I am transmitting (or not)
endBackoff()	F	F00	endBackoff()
		F01	endBackoff() end
		F02	Channels for transmitting
		F03	Transmission is possible (or not)
		F04	Selected transmission range
		F05	New backoff generated
myTXFinished()	G	G00	myTXFinished()
		G01	myTXFinished() end
		G02	New backoff generated

Table 5: Node's event logs encoding

Listing 2: Executing the code

```
$ komondor_main ../input/input_example/system_input.csv ../input/input_example/
    nodes_input.csv 0 0 1 1 60 789
```

In this case, the execution time has been set to 60 seconds and the random seed to 789. File logs have been disabled (the two zeros), but console logs were activated. The results of this simulation can be observed from the console (Figure 5).

Figure 5: Output logs from example 1.

3.4.2 Example 2: Execution with agents

Now, we are showing an execution where intelligent agents are used. For that, we use the same system and input files, and add the agents input file that is located at the same folder. In this case, we use Multi-Armed Bandits (MABs) under an ε-greedy action-selection strategy to allow the WLANs to modify the transmit power and the CCA. For additional details on this implementation, please refer to [?].

Again, we compile the code as before:

Listing 3: Compling the code

```
$ cd Komondor/Code/main
$ ./build_local
```

To execute the simulation, we now introduce more arguments, as described in Section 3.2.

Listing 4: Executing the code

```
$ komondor_main ../input/input_example/system_input.csv ../input/input_example/
    nodes_input.csv ../input/input_example/agents.csv 1 1 1 1 1 1 120 432
```

In this occasion, we activated all the types of logs, so the output folder will contain files generated by the system, the nodes and the agents (Figure **??**). Moreover, logs are written at the console, as shown in Figure **??**.

Figure 6: Output files generated in Example 2.

Figure 7: Output logs from Example 2.

4 Development Notes

Here we provide some clarifications regarding code implementation, wit the aim to facilitate the Komondor's usage and manipulation to developers that may be interested.

4.1 Main considerations

Some technical information regarding code development is worth to be mentioned to properly understand how to use and modify the Komondor simulator. So far, the main considerations to be taken into account are:

- **Power and CCA**: power variables are stored in pW (pico watts) in order to be able to operate power magnitudes without loosing resolution[4]. However, values are presented to the user in dBm. W (-30) - mW (0) - uW (+30) - nW (+60) - pW (+90)
  $P_{\text{pw}} = 10^{\frac{P_{\text{dBm}}+90}{10}}$

4.2 Miscellaneous

- **Transmitting capability**: we have added a flag to each node that determines if it is able to transmit (1) or not (0), so that we can decide if the node is only listening or both transmitting and listening.

- **Progress bar**: the Komondor simulation progress bar is displayed through a *printf()* command called by any node with *node_id* set to 0. If no node has *node_id* set to 0, the progress bar is not displayed.

References

[1] Sergio Barrachina-Muñoz and Francesc Wilhelmi. Komondor: An IEEE 802.11ax simulator. https://github.com/wn-upf/Komondor, 2017.

[2] G. Chen and B. K. Szymanski. Reusing simulation components: cost: a component-oriented discrete event simulator. In *Proceedings of the 34th conference on Winter simulation: exploring new frontiers*, pages 776–782. Winter Simulation Conference, 2002.

[3] IEEE p802.11ax/d2.0, November 2017.

[4] For instance., the sum of two signals of power values -85 dBm (3.162 pW) and -90 dBm (1 pW), respectively, is -83.803 dBm (4.162 pW).