Komondor User's Guide

Sergio Barrachina-Muñoz and Francesc Wilhelmi

October 31, 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].

This document aims illustrate the installation and execution procedures that must be carried out to successfully perform Komondor simulations. For more detailed information about what is implemented in Komondor, please refer to this document.

2 Tutorial and Development Notes

In this Section we provide a brief tutorial to encourage researchers and other practitioners to use the Komondor simulator for their own experiments, and even to participate in the project.

2.1 Brief Tutorial

Komondor is composed by several modules that allow performing simulations with a high degree of freedom. Here we provide some details on the most important modules, as well as on their practical execution. Figure 1 summarizes the main operations carried out by Komondor.

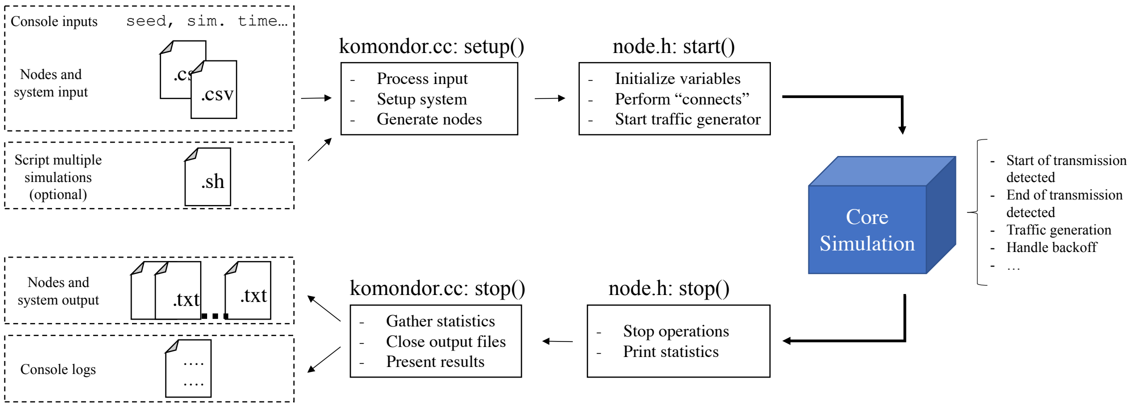

Figure 1: Komondor flowchart

As shown, Komondor receives a set of inputs (nodes information, simulation time, etc.) and initializes the main module, which is in charge of generating the network and gathering useful information regarding the simulation. During the core simulation, nodes interact among each other by sending packets, so that DCF operation is implemented for accessing to the channel. Finally, when the simulation runs out, a set of outputs are generated in order to shed some light on the network performance.

2.1.1 Files Organization

To properly understand the Komondor's operation, it is important to know how the project is organized, which allows obtaining a broader vision of the different modules that constitute it. The code is organized as follows:

- **COST libraries**: constitute the Komondor's primitive operation.
- **Main**: contains the two core files (`komondor.cc` and `node.h`), which are in charge of orchestrating all the simulation. In addition, here we find the inputs and the file that compiles the libraries for executing the code (`build_local`).
- **Methods**: by following clean architecture guidelines, independent methods used by both `komondor.cc` and `node.h` 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 four main header files to carry out its operation. The first one is `wlan.h`, which defines the main characteristics of a WLAN (WLAN id, list of associated STAs, etc.). Furthermore, the `notification.h` object allows to exchange packets between devices. Finally, `logger.h` and `logical_nack.h` are used for auxiliary purposes, which are displaying logs and notifying packet losses causes, respectively.
- **List of macros**: all the static parameters (e.g., constants) are contained in the `list_of_macros.h` 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.

2.1.2 Compilation and Execution

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

1. Set the .csv input files (further defined in Section 2.1.3)

2. Access to the *KomondorSimulator* 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:

   - *arg_1* (INPUT_FILE_SYSTEM_CONFIGURATION): file containing system information (e.g., number of channels available, traffic model, etc.). The file must be a .csv with semicolons as separators.
   - *arg_2* (INPUT_FILE_NODES): file containing nodes information (e.g., position, channels allowed, etc.).The file must be a .csv with semicolons as separators.
   - *arg_3* (OUTPUT_FILE_LOGS): 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).
   - *arg_4* (FLAG_SAVE_SYSTEM_LOGS): flag to indicate whether to save the system logs into a file (1) or not (0).[3]
   - *arg_5* (FLAG_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.
   - *arg_6* (FLAG_PRINT_SYSTEM_LOGS): flag to indicate whether to print the system logs (1) or not (0).
   - *arg_7* (FLAG_PRINT_NODE_LOGS): flag to indicate whether to print the nodes logs (1) or not (0).
   - *arg_8* (SIM_TIME): simulation time in seconds.
   - *arg_9* (SEED): random seed for the experiments.

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 *`.

2.1.3 Input files

To define the simulation environment, the Komondor simulator relies in the following two types of input files:

- **System input:** defines global input parameters such as the number of basic channels considered or the propagation models. System input parameters are defined in Table 1.

- **Nodes input:** defines specific nodes' characteristics such as type, location, or implementing features (e.g., DCB policy). There are two ways of generating nodes, which is indicated in the file name.

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

  – 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).[4]

[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/

[3] Major increases in the execution time may occur if nodes logging is activated. E.g., for a simulation of 4 nodes, simulating 1000 seconds takes 1.127 s and 15.672 s when not logging and when doing so, respectively.

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

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). Table 2 describes the Nodes input parameters for both *nodes* and *aps* files.

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

Table 1: System input parameters description

2.1.4 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.

2.1.5 Output files

A lot of effort has been put on the output generation, since it is a sensitive module that allows understanding and validating the results provided by the simulator. Henceforth, we provide different kinds of outputs, which refer to console and file output logs. Note, as well, that generating output files considerably increases the execution time.

Regarding console output logs, them can be activated through *arg_6* and *arg_7* during the execution, which refer to system and nodes logs, respectively (see Section 2.1.2). Additionally,

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

Table 2: Nodes input parameters description

these logs can be copied into files, which are saved into the *output* folder, only if *arg_4* and *arg_5* are set to 1. While the path of the system's output file must be specified (*arg_3*), nodes' files are automatically created.

Finally, a set of statistics are shown per node and for the entire simulation. Such statistics include throughput experienced, collisions, nodes sent, RTS/CTS sent, etc. An example of nodes and system statistics is shown in Figures 2 and 3

2.1.6 Events Categorization

In order to make output results more understandable, logs are categorized according to the event that generates it. With that, further filtering processes can be carried out by developers. Table 3 describes the codes used for each type of event.

2.2 Code development

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.

```
------- AP_A (N0) ------
  - Throughput = 102.120960 Mbps
  - RTS/CTS sent = 14750 - RTS/CTS lost = 0  (0.00 % lost)
    · RTS lost due to slotted BO = 0 (0.000000 %)
  - Data packets sent = 14750 - Data packets lost = 1453  (9.850847 % lost)
  - num_tx_init_tried = 14750 - num_tx_init_not_possible = 0 (0.000000 % failed)
    · Time EFFECTIVELY transmitting in N channels:
    · - 1: 86.849472 s (86.85 %)
    · Time EFFECTIVELY transmitting in each channel:
    · - 0 = 86.87 s (86.87 %)
    · - 1 = 0.00 s (0.00 %)
    · Number of tx trials per number of channels:
    · - 1: 14750 (100.00 %)
    · - 2: 0 (0.00 %)
  - num_tx_init_not_possible = 0


------- AP_B (N2) ------
  - Throughput = 101.798400 Mbps
  - RTS/CTS sent = 14751 - RTS/CTS lost = 0  (0.00 % lost)
    · RTS lost due to slotted BO = 0 (0.000000 %)
  - Data packets sent = 14750 - Data packets lost = 1495  (10.135593 % lost)
  - num_tx_init_tried = 14751 - num_tx_init_not_possible = 0 (0.000000 % failed)
    · Time EFFECTIVELY transmitting in N channels:
    · - 1: 86.581660 s (86.58 %)
    · Time EFFECTIVELY transmitting in each channel:
    · - 0 = 0.00 s (0.00 %)
    · - 1 = 86.61 s (86.61 %)
    · Number of tx trials per number of channels:
    · - 1: 14751 (100.00 %)
    · - 2: 0 (0.00 %)
  - num_tx_init_not_possible = 0
```

Figure 2: Example of nodes statistics in Komondor

```
 General Statistics:
  - Total number of packets sent = 58999
  - Total throughput = 408.27 Mbps
  - Average number of packets sent per WLAN = 14749
  - Average throughput per WLAN = 102.07 Mbps



- Average throughput per WLAN = 102.07 Mbps
- Proportional Fairness = 32.04
- Jain's Fairness = 1.00
- Average number of data packets successfully sent per WLAN = 14749.75
- Average number of RTS packets lost due to slotted BO = 0.00 (0.00 % loss)

------ FOR COMPARING TO BIANCCI -------
- Prob. collision by slotted BO = 0.000000
- Aggregate throughput = 408.268800 Mbps
- Aggregate number of transmission not possible = 0
--------------------------------------


    · - 1: 14750 (100.00 %)
    · - 2: 0 (0.00 %)
    · - 1: 14751 (100.00 %)
    · - 2: 0 (0.00 %)
    · - 1: 14750 (100.00 %)
    · - 2: 0 (0.00 %)
    · - 1: 14749 (100.00 %)
    · - 2: 0 (0.00 %) SIMULATION 'test' FINISHED
------------------------------------------
# ------------------------------------------------------------------
# CostSimEng with SimpleQueue, stopped at 100.000000
# 590007 events processed in 3.920 seconds, event processing rate: 150523
administrador@ws119785:~/workspace/Komondor/Code/komondor_main$ S
```

Figure 3: Example of system statistics in Komondor

2.2.1 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:

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 3: Node's event logs encoding

- **Power and CCA**: power variables are stored in pW (pico watts) in order to be able to operate power magnitudes without loosing resolution[5]. However, values are presented to the user in dBm. W (-30) - mW (0) - uW (+30) - nW (+60) - pW (+90)
  $P_{\text{pw}} = 10^{\frac{P_{\text{dBm}}+90}{10}}$

2.2.2 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.

3 Conclusions

In this document we provided an overview of the first version of the Komondor simulator, which aims to reproduce the basic operation of IEEE 802.11ax WLANs in addition to allow the utilization of intelligent systems. We introduced the system model considered when building the simulator, as well as the main MAC features implemented. Additionally, due to the open source nature of this project, we provided basic information of interest for developers that are expected to use or even modify this HD WLANs simulator.

[5]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).

Regarding the validation of the simulator, we provided a set of meaningful test scenarios to prove the proper behavior of the simulator. As shown, tests were satisfactory as the throughput computed with Komondor and the CTMN model are pretty similar, and the differences were properly justified.

This project is expected to move forward for including of novel mechanisms such as OFDMA, MU-MIMO, TPC or CST adjustment. In addition, intelligent agents are expected to be included for making operations such as Dynamic CB (DCB).

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.