[TUTORIAL] KOMONDOR: A Wireless Network Simulator Based for Next-Generation IEEE 802.11ax WLANs

Sergio Barrachina and Francesc Wilhelmi

December 13, 2017

Abstract

Komondor is a wireless network simulator that includes novel mechanisms for next-generation WLANs, such as Dynamic Channel Bonding or enhanced Spatial Reuse. One of Komondor's main purposes is to emulate the behavior of IEEE 802.11ax-2019 networks, which main challenge is spectral efficiency in dense deployments. Furthermore, due to the growing popularity of autonomous systems and the tendency of WLANs to use learning, Komondor is intended to include intelligent agents that perform operations like Carrier Sense Threshold adjustment.

Contents

List of Figures

List of Tables

1 Introduction

Komondor is an event-based simulator based in COST [1], a CompC++ library to perform discrete event simulation.[1] The presented simulator is mostly intended to reproduce the novel techniques included in the IEEE 802.11ax-2019 amendment [2], which is called to become a benchmark in next-generation wireless networks. For that, and due to the lack of 11ax-oriented simulators, we aim to develop Komondor. Furthermore, a key feature of our simulator is the inclusion of agents that can decide the behavior of WLANs through Reinforcement Learning (RL). Other popular simulation tools such as ns-3 lack of flexibility and adaptability towards including online learning procedures within their operation.

Komondor has been developed to be an open source tool that contributes to the ongoing research in wireless networks. Henceforth, in this document we describe its main features and system model

[1] COST main website: http://www.ita.cs.rpi.edu/cost.html

considerations, as well as some basic guidelines to run it. To provide detailed technical information regarding the development of the code is out of the scope of this document.

1.1 Next-Generation WLANs

The increasing network requirements in terms of data rate and users capacity has brought the wireless communications community to introduce novel approaches. Regarding IEEE 802.11 WLANs, the 11ax amendment is being developed to improve spectrum efficiency in high density scenarios. To accomplish that, it introduces the concept of High-Efficiency (HE) WLANs, which incorporates new mechanisms such as OFDMA, Dynamic Channel Bonding, Beamforming and Multi-User Multiple-Input Multiple-Output (MU-MIMO). Such advanced mechanisms drastically change the current operation of WLANs and have not been previously implemented with detail in other network simulators. Further information regarding the novel capabilities included in the IEEE 802.11ax standard can be found in [3].

In addition to including such novel techniques, wireless networks are evolving towards autonomous management, which in many cases is achieved through Artificial Intelligence (AI). Its utilization is expected to be key in next-generation complex systems, since it allows solving (or at least approximating) computational-intensive problems. In particular, online learning has been previously applied in well-known problems such as Transmit Power Control (TPC), Carrier Sense Threshold (CST) adjustment and channel allocation [4, 5, 6, 7]. Since most of the literature that applies learning into wireless networks is theoretical in nature, there is a strong need for tools that allow implementing learning algorithms in a realistic simulation environment.

1.2 Komondor Main Features

Komondor aims to thoroughly simulate the operation of wireless networks. Henceforth, it reproduces actual transmissions on a per-packet basis. For that, nodes properties (e.g., location, transmit power, CCA threshold) are taken into account during data exchange procedures.

In overview mode, Komondor is intended to simulate the following wireless networks mechanisms:

- Distributed Coordination Function (DCF)
- Channel Bonding (CB)
- Dynamic transmit power and CST adjustment
- MU-MIMO transmissions
- Directional transmissions
- Packet aggregation
- Dynamic MCS
- RTS/CTS and NAV allocation

Note, as well, that the first version of Komondor introduced by this document includes the basic operation carried out by WLANs and only implements DCF, CB, dynamic MCS and RTS/CTS.

1.3 COST

In order to provide a deep understanding of Komondor, it is important to briefly understand the COST library, which allows building discrete interactions between components, which may represent entities like wireless nodes. Such interaction is achieved through synchronous and asynchronous events. While the former are message explicitly exchanged between components through input/output ports, the later are based on timers. In practice, components perform a set of operations until a significant event occurs. For instance, a node that is decreasing its backoff (i.e., current operation) may freeze it when an overlapping node occupies the channel (i.e., an event). Figure 1 shows the schematic of a COST component, which is characterized by its inports and outports, and a set of timers. While inports and outports allow to directly communicate with other components, timers trigger events specific to the component.

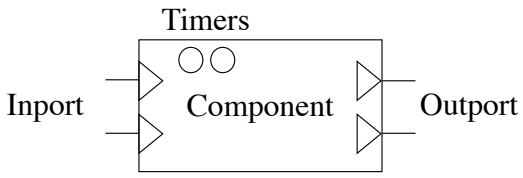


Figure 1: COST component

1.4 Paper Structure

This documents is structured as follows: Section 2 describes the system model. Then, Section 3 defines the implementation of MAC functionalities considered so far.

2 System Model

One of the main tasks regarding the implementation of Komondor is the definition of the system model, which lays the foundations of the simulator. In this Section we describe the models defined for simulating the different communication aspects that wireless device implement.

2.1 Channel Modeling

Channel modeling is one of the most important parts regarding the system model, since it has direct impact on the interactions between wireless devices. For that, and in order to provide the most representative wireless environments, we implemented the following set of path-loss models, which can be further extended by any developer:

- Free Space Path Loss (FSPL): a free-space model is considered, which captures direct line-of-sight and ignores shadowing effects. The experienced loss of power during a transmission that assumes this model is given by:

$$\text{FSPL} = 20\log_{10}(d) + 20\log_{10}(f) + 20\log_{10}(\frac{4\pi}{c}) - G_t - G_r,$$

  where d is the distance between the transmitter and the receiver, f is the frequency used in GHz, c is the light speed in m/s, and G_t and G_r are the gains in dB at the transmitter and receiver, respectively.

- Okumura-Hata model [8]: this well-known model was conceived for predicting the path-loss of cellular transmissions in outside urban and rural environments. Our main concern is related to dense urban areas, so the loss, L_U, that is considered to be experienced during a transmission is given by:

$$\begin{aligned} L_U =& 69.55 + 22.16\log_{10}(f) - 13.82\log_{10}(h_B) + (44.9 - 6.55\log_{10}(h_B))\log_{10}(d) - \\ & 3.2\log_{10}(11.7554h_M)^2 - 4.97, \end{aligned}$$

  where f is the center frequency in MHz, h_B is the height of the transmitter antenna (fixed to 10 m), d is the distance between the transmitter's and the receiver in meters, and h_M is the height of the receiver's antenna (fixed to 10 m).

- Indoor model: this model represents a simple indoor scenario, which is useful to simulate typical scenarios such as flats, schools or restaurants. According to this model, the losses L_{indoor} experienced between a packet transmission are:

$$L_{indoor} = 5 + 10\alpha\log_{10}(d) + h_S + (\frac{d}{f_w})h_O,$$

  where α is a constant that depends on the propagation model (set to 4.4), d is the distance in meters between the transmitter and the receiver, h_S is the shadowing factor, f_w is the frequency of walls (set to one wall each 5 meters), and h_O is the obstacles factor.

- Indoor model with random channel effects: we consider the same model as before, but now we introduce random variables to determine the shadowing and obstacles effects in the power losses.
- Residential path-loss model IEEE 802.11ax: such model is included in the 11ax amendment, and captures the path-loss effects of a typical apartments building. The losses experienced between a packet transmission are:

In addition to path-loss types, we also defined different co-channel interference models to capture several kinds of interactions between devices:

- Without adjacent interference: no power is leaked to adjacent channels.
- Full channel interference: power from other channels i leaked, so that a 20 dBr decrease is noticed for each channel distance. For instance, the power that channel 1 leaks into channel 3 is the actual power in channel 1 minus 40 dBr.
- Limited adjacent interference: in this case, only immediate adjacent channels leak power to the target one.

An example of channels overlapping is shown in Figure 2.

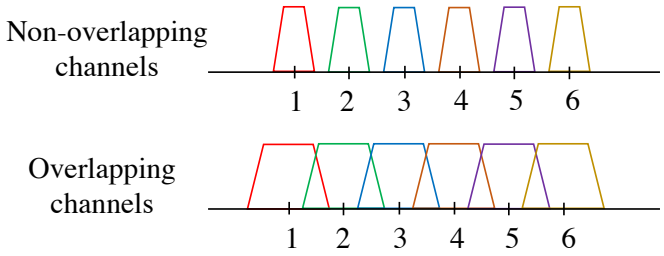


Figure 2: Channel models with and without adjacent interference

Finally, regarding the channel model, it is important to remark that the incoming power is assumed to be the same during the entire transmission. This relaxation allows us to easily determine whenever the channel of interest is busy or not. A direct implication of it affects to path-loss models used, as well as some of them assume random variations of the medium, preventing to obtain the same result with different power received calculations (so far, power received is added and subtracted when the node accesses and leaves the channel, respectively). Thus, for each node we store its incoming power (which is computed only once) for subtracting it at the end of its transmission.

2.2 Traffic Modeling

Traffic modeling refers to the capacity of generating data in higher transmission layers. So far, Komondor only considers downlink traffic, so that data transmissions are initiated by APs. Regarding traffic generation, we have considered three different models:

- Full buffer: transmitters are in a permanent saturation regime, so that they always have packets to be sent.
- Poisson: packets are generated according to a Poisson distribution, so that the time between packets Δ_p is determined by the packet generation rate λ, and is given by $\Delta_{\mathrm{p}} = e^{\frac{1}{\lambda}}$
- Deterministic: packets are generated at fixed time intervals, Δ_d, given by the packet generation rate, $\Delta_{\mathrm{d}} = 1/\lambda$.

Figure 3 illustrates the aforementioned traffic models.

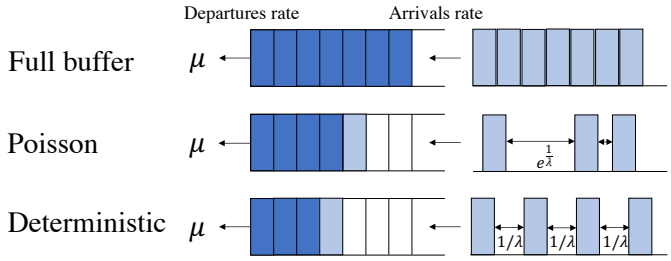


Figure 3: Traffic models used in Komondor

MCS index	SINR interval (dBm)	Modulation type	Coding rate	Data rate (Mbps)			
				20 MHz	40 MHz	80 MHz	160 MHz
0	[-82, -79)	BPSK	1/2	4	8	17	34
1	[-79, -77)	QPSK	1/2	16	33	68	136
2	[-77, -74)	QPSK	3/4	24	49	102	204
3	[-74, -70)	16-QAM	1/2	33	65	136	272
4	[-70, -66)	16-QAM	3/4	49	98	204	408
5	[-66, -65)	64-QAM	2/3	65	130	272	544
6	[-65, -64)	64-QAM	3/4	73	146	306	613
7	[-64, -59)	64-QAM	5/6	81	163	340	681
8	[-59, -57)	256-QAM	3/4	98	195	408	817
9	[-57, -54)	256-QAM	5/6	108	217	453	907
10	[-54, -52)	1024-QAM	3/4	122	244	510	1021
11	≥ 52	1024-QAM	5/6	135	271	567	1143

Table 1: Data rates granted per MCS in IEEE 802.11ax. Guard Intervals (GI) of 1600 ns are only considered.

2.3 Link Modeling

In order to determine the data rate to be used during a given transmission, the MCS defined in the IEEE 802.11ax are considered. Komondor assumes that the MCS used between a pair of devices is determined by the SINR at the received, so that the maximum allowable MCS is used. The required SINR that corresponds to each MCS is defined in Table 1, as well as the granted data rate for each channel width.

So far, link adaptation is not considered[2], so that the MCS to be used between each transmitter-receiver pair is negotiated at the beginning of the first transmission, and remains static during all the simulation. In practice, the highest possible modulation is computed for each number of channels used according to the power received (the transmit power is lower as the number of channels used is higher). With that, we aim to simulate the Receiver Driver Protocol, at which few symbols are transmitted at the lowest bit-rate for all the subcarriers. Thus, according to the SINR perceived at the receiver, the MCS is chosen and communicated to the transmitter.

Finally, regarding the transmission time for sending a packet, it is computed as a function of the data rate and the size of the packet to be transmitted. Packet lengths are defined according to the IEEE 802.11ax specification, which is further described in Section 4.1.

2.4 Collisions Modeling

Collisions are one of the main issues in WLANs, since they harm the experienced performance. In particular, packet losses mostly occur because of backoff collision, hidden-node effects and link asymmetries. An example of a hidden-node collision is shown in Figure 4, in which nodes A and C transmit simultaneously to B, since they do not sense the other's transmission.

In Komondor, a packet loss is considered when the ACK timeout is over. However, it is critical to identify what caused such loss, which may allow to solve the root problem. In particular, the following packet losses categories are provided:

[2]Future work contemplates the inclusion of Minstrel as a rate adaptation scheme.

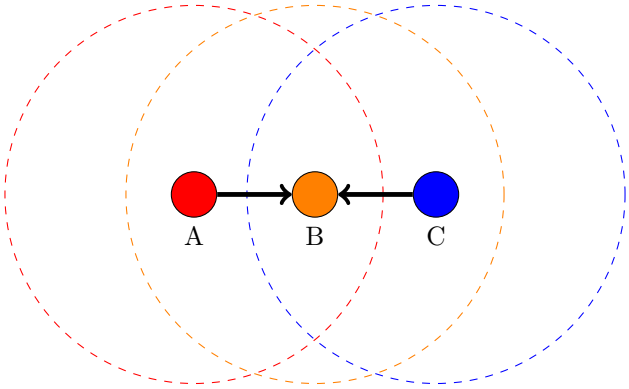


Figure 4: Hidden-node cause of collision

- PACKET_LOST_DESTINATION_TX: packet is discarded because the destination was already transmitting when the packet transmission was attempted.
- PACKET_LOST_LOW_SIGNAL: the packet cannot be decoded because the signal strength is not enough (i.e., it is less than capture effect).
- PACKET_LOST_INTERFERENCE: packet is lost due to interference signals that makes the receiver not accomplish the capture effect condition.
- PACKET_LOST_PURE_COLLISION: packet is lost because two nodes transmit to same destination with signal strengths enough to be decoded (the situation shown in Figure 4).
- PACKET_LOST_LOW_SIGNAL_AND_RX: the destination is already receiving data when a new data transmission starts. In addition, the newest signal strength is not enough to be decoded in normal conditions.
- PACKET_LOST_RX_IN_NAV: packet is lost because the target node is in a NAV period (it previously decoded an RTS/CTS sequence).
- PACKET_LOST_BO_COLLISION: the packet is lost because a backoff collision occurred, i.e., two or more interfering devices ended their backoff simultaneously.

Furthermore, other types of collisions that are uncontrolled may occur. For instance, when applying Channel Bonding, the situation depicted in Figure 5 may occur.

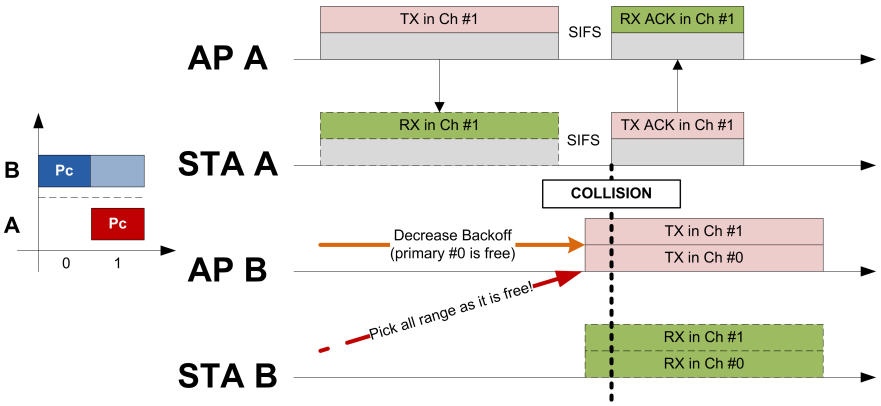


Figure 5: Weird case when collisions occur due to last transmitting AP checks the channel right in the SIFS period of other WLAN and picks a range affecting the previous transmission.

3 MAC Features

In this Section we provide an overview of the main MAC layer features included in Komondor, as well as a glimpse on their practical implementation.

3.1 Channel Access

When an AP has a packet to be sent, it implements Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) to access the medium, which makes use of the Distribution Coordination Function (DCF). With that, transmissions are carried out if the target channel has been empty for a given Backoff (BO) time. A channel is considered to be empty if the interference in it is equal or higher than a Capture Effect (CE) threshold, which allows defining the frequency at which packet losses occur, regardless on the Modulation Coding Scheme used.

3.1.1 Capture Effect

In order to perform transmissions through the radio link, the receiver must perceive that the desired signal strength is bigger than a CE threshold, so that the received data can be distinguished among noise and interference present in the channel. Despite there are two types of interference patterns (*stronger-first* and *stronger-last*), Komondor only considers data transmissions in which the target packet arrives first, just as shown in Figure 6. Otherwise, when a second data packet arrives in the middle of a data transmission, both packets are discarded if the second signal is stronger than the first one.

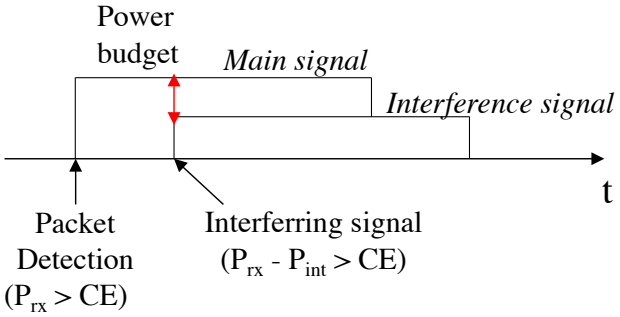


Figure 6: Stronger-First Capture Effect example

3.1.2 Distributed Coordination Function

As previously mentioned, the CSMA/CA operation is based on the DCF, which orchestrates channel access in a distributed manner. Roughly, transmitters (e.g., devices that have data to be transmitted) choose a random BO value, which is decremented only if the channel is sensed as idle due to the CCA condition. Otherwise, the BO is freeze and the transmission is contended. Figure 7 shows an example of the DCF operation in which three nodes listen to each other in the same wireless scenario. As it is shown, STA2 wins the channel access because its BO timer is the first one to reach 0. During the packet transmission STA1 listens and does not decrease its BO. After data transmission is finished and the channel has been idle for a DIFS interval, the BO procedure is activated in all the transmitting devices.

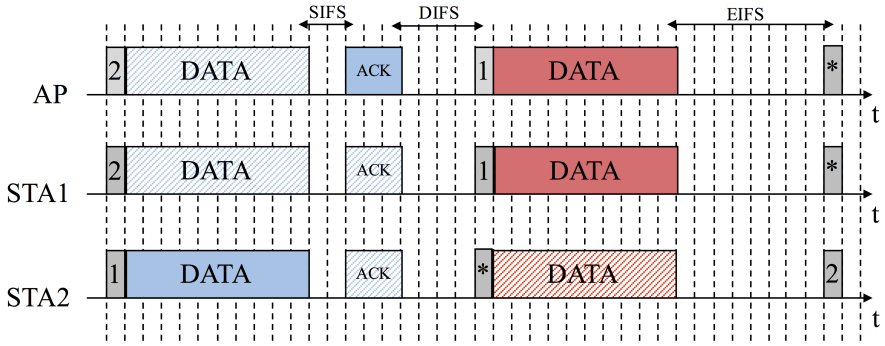


Figure 7: Example of the DCF procedure

The process of generating a BO before a packet transmission is determined by the Congestion Window (CW) and the generation PDF used, which has been considered to support both deterministic and exponential distributions. Komondor implements two kinds of BO pdfs to be used in different situations:

- Uniform: the BO is a number between 0 and CW − 1, and all the values have the same probability.
- Exponential: instead of using an uniform distribution, we use an exponential one, so that the BO is given by a Gaussian distribution with mean $\frac{CW-1}{2}$

An important consideration with respect to discrete BO is that devices are synchronized, which allows reproducing collisions by BO that depend on the congestion window and the number of coexisting nodes.

3.1.3 Contention Window Adaptation

To complement the DCF operation, dynamic CW adaptation is considered on a per-packet basis. Optionally, one can activate CW adaptation so that the CW increases or decreases according to the situation. Given a minimum and a maximum boundaries for CW (CW_{min} and CW_{max}, respectively), the reset operation is performed when a successful transmission is carried out. In such situation, the CW is set to CW_{min}. Otherwise, when packet losses occur, the CW is increased without exceeding CW_{max}. To do so, a counter (namely CW_{count}) is maintained and increased one unit each time a packet loss occurs. Then, the CW is computed as $CW = CW_{min} \times 2^{CW_{count}}$.

3.2 RTS/CTS and NAV Allocation

The Ready-to-Send/Clear-to-Send (RTS/CTS) mechanism is implemented in order to minimize the collisions by hidden node. Through RTS/CTS, transmitting nodes attempt to ensure that the channel would be clear during their transmissions. For that, they send control packets and wait confirmation about the clearness of the channel from the receiver's point of view. Through such packet exchange, overlapping nodes must set a virtual carrier sensing during the transmission duration, which allows reducing the collisions by hidden node. The RTS/CTS operation is exemplified in Figure 8.

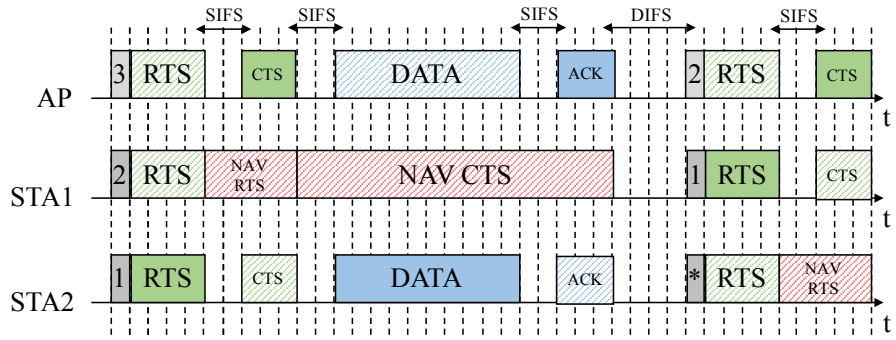


Figure 8: Example of RTS/CTS implementation. The transmitter (STA2) sends an RTS packet before starting a transmission. The receiver (AP) answers with a CTS as it senses the channel free. The other coexisting devices (STA1) listen either the RTS and/or the CTS and set their NAV accordingly.

3.3 Channel Bonding

TO BE COMPLETED BY @SERGIO

Channel Bonding (CB) is one of the most promising techniques to enhance spectral efficiency in IEEE 802.11ax WLANs, since it aims to make the most of medium by transmitting data over several channels. For that, different CB policies are implemented in Komondor, which can be interchangeably applied by the simulated WLANs. To perform CB, one may explicitly define the available range of channels for each WLAN. Then, the following policies can be applied:

- CB disabled: the legacy operation is performed, so that only the primary channel is attempted to be accessed.
- Static CB: carrier sensing is performed at the primary channel. However, when attempting to transmit, all the channels within the CB range must be clear. Otherwise, a new backoff is computed.
  - Aggressive SCB:
  - Power of 2 SCB
- Dynamic:
  - Aggressive DCB
  - Power of 2 DCB[3]: transmit in the larger channel range that is allowed by the *log2* structure shown in Figure **??**.
- Policy-dependent:
  - Sergio's PhD :D

3.4 Packet Aggregation

Packet aggregation aims to reduce transmission overheads such as headers, SIFS and DIFS intervals or backoff periods. For that, it concatenates N MPDUs to be sent over the same packet transmission, so that it can be acknowledged through a block ACK. Komondor allows to define the number of aggregated packets, which value remains static during the entire simulation.

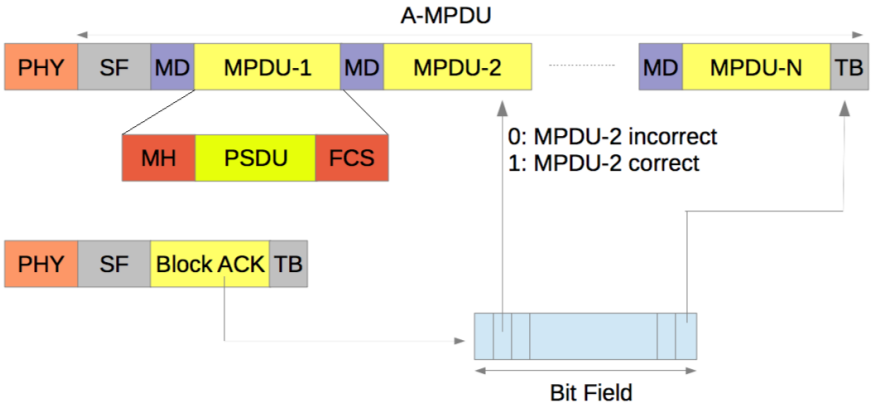


Figure 9: Example of packet aggregation in which N MPDUs are concatenated to be sent during the same packet transmission.

4 Komondor Main Features Validation through CTMN

In this Section we focus on a set of key scenarios, which allow us to validate the Komondor's operation. For that, we use the Continuous Time Markov Networks (CTMNs) model [9] for analytical comparison, which has been further extended and implemented in [10].

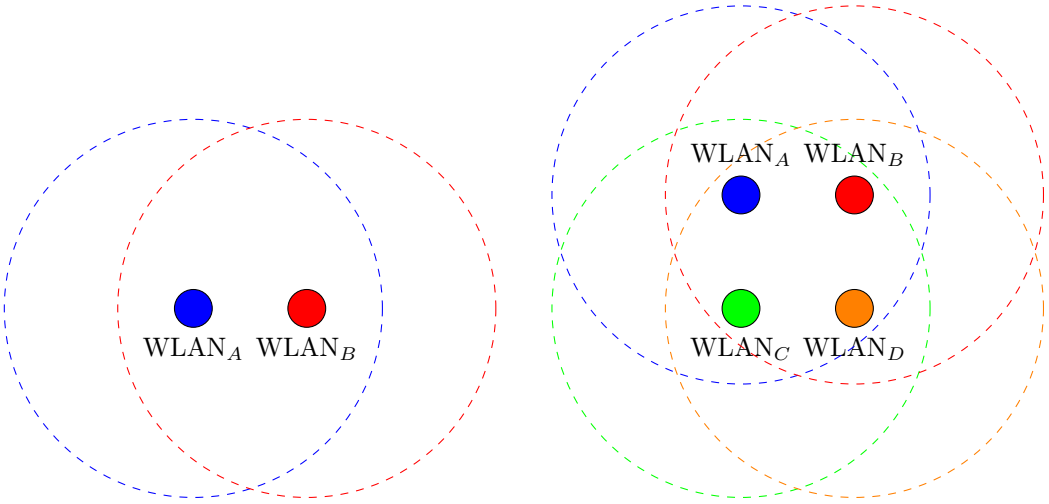
4.1 IEEE 802.11ax Parameters

Before showing the set of Komondor validations, we introduce the IEEE 802.11ax parameters that we have utilized for simulations, and which are recommended to be kept to emulate 11ax's behavior.

[3]In the Matlab code, this model corresponds to the configuration *onlymax* = *true* and *selfloop* = *false*

4.2 Basic Scenarios

Through the following scenarios, we aim to validate the basic operation of Komondor, so that features such as DCF and RTS/CTS can be properly tested. The first validation we aim to show is at a very basic scenario in which two overlapping WLANs transmit by using *a*) the same primary channel, *b*) different non-overlapping channels. Such scenario is depicted in Figure 10a. Furthermore, and in order to provide a more complete validation, we test Komondor's operation in the scenario shown in Figure 10b. The four overlapping WLANs in such scenario are intended to use *a*) the same primary channel, *b*) two different channels, and *c*) four different channels.



(a) Basic scenario 1 (b) Basic scenario 2

Figure 10: Scenarios for basic validations

Results from simulation in each of the basic scenarios are presented in Table 2.

Scenario	**Tool**	Γ_A	Γ_B	Γ_C	Γ_D	Γ
1a	Komondor	a	b	-	-	c
	CTMN	a	b	-	-	c
1b	Komondor	a	b	-	-	c
	CTMN	a	b	-	-	c
2a	Komondor	a	b	c	d	e
	CTMN	a	b	c	d	e
2b	Komondor	a	b	c	d	e
	CTMN	a	b	c	d	e
2c	Komondor	a	b	c	d	e
	CTMN	a	b	c	d	e

Table 2: Summary of the results obtained for the basic scenarios.

4.3 Hidden and Exposed Node Scenarios

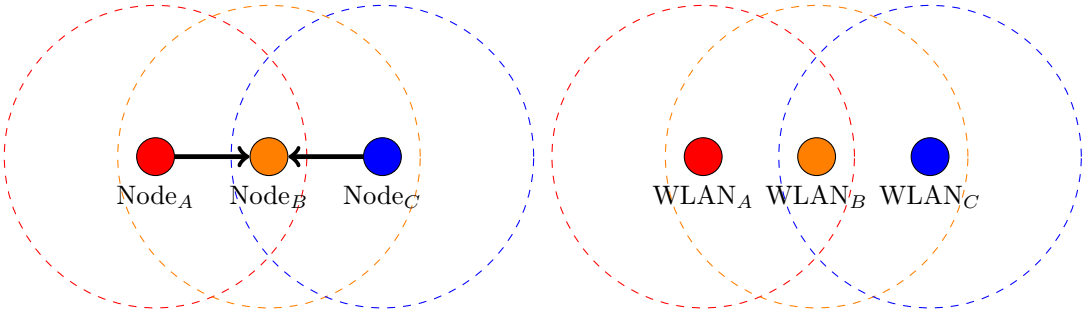
We start by showing

Regarding the exposed node situation, we introduce the scenario depicted in Figure 11b, in which one of the three WLANs suffers starvation because of its location. The fact that WLAN$_A$ and WLAN$_C$ do not sense each other makes that WLAN$_B$ senses the channel busy almost all the time and lacks of opportunities for accessing to it.

Results for both hidden and exposed scenarios are shown in Table 3.

4.4 Channel Bonding Scenarios

Finally, in order to validate the most important technique included in the first version of the Komondor simulator, we now propose a set of scenarios in which CB is applied. For that, we use



(a) Hidden-node (b) Exposed-node

Figure 11: Representative scenarios of performance anomalies in WLANs

Scenario	**Tool**	Γ_A	Γ_B	Γ_C	Γ	Collisions
Hidden node	Komondor	a	b	c	d	x
	CTMN	a	b	c	d	x
Exposed node	Komondor	a	b	c	d	x
	CTMN	a	b	c	d	x

Table 3: Summary of the results obtained for the hidden and exposed node scenarios.

the same scenario as shown in Figure 10b, which allows us to devise several combinations regarding channel allocation. Such combinations are illustrated in Figure 12.

Scenario	**States**	**Tool**	Γ_A	Γ_B	Γ_C	Γ_D	Γ
1	3	ACK	57.1418	57.2815	-	-	114.4234
		CTMN	57.6166	57.6166	-	-	115.2331
2	5	ACK	62.5831	61.3054	-	-	123.8886
		CTMN	62.6145	61.9399	-	-	124.5544
3	7	ACK	31.1883	31.1394	61.7620	-	124.0898
		CTMN	31.2965	31.2965	62.1671	-	124.7601
4	13	ACK	XXX	XXX	XXX	-	XXX
		CTMN	61.9390	63.1841	113.4185	-	238.5416
5	5	ACK	40.8806	40.8506	40.7439	40.7498	163.2251
		CTMN	41.2737	41.2737	41.2737	41.2737	165.0947
6	9	ACK	XXX	XXX	XXX	XXX	XXX
		CTMN	57.6166	57.6166	57.6166	57.6166	230.4663
7	9	ACK	XXX	XXX	XXX	XXX	XXX
		CTMN	57.6166	57.6166	57.6166	57.6166	230.4663
8	5	ACK	XXX	XXX	XXX1	XXX	XXX
		CTMN	41.2737	41.2737	41.2737	41.2737	165.0947

Table 4: Summary of the results obtained for the *log2* DCB scenarios with ACK and WLANs implemented. Simulation time is 10,000 seconds.

5 Tutorial and Development Notes

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

5.1 Brief Tutorial

5.1.1 Files Organization

To properly understand the Komondor's operation, it is important to describe how the project is organized, which provides an overview of the different modules that constitute it. The code is

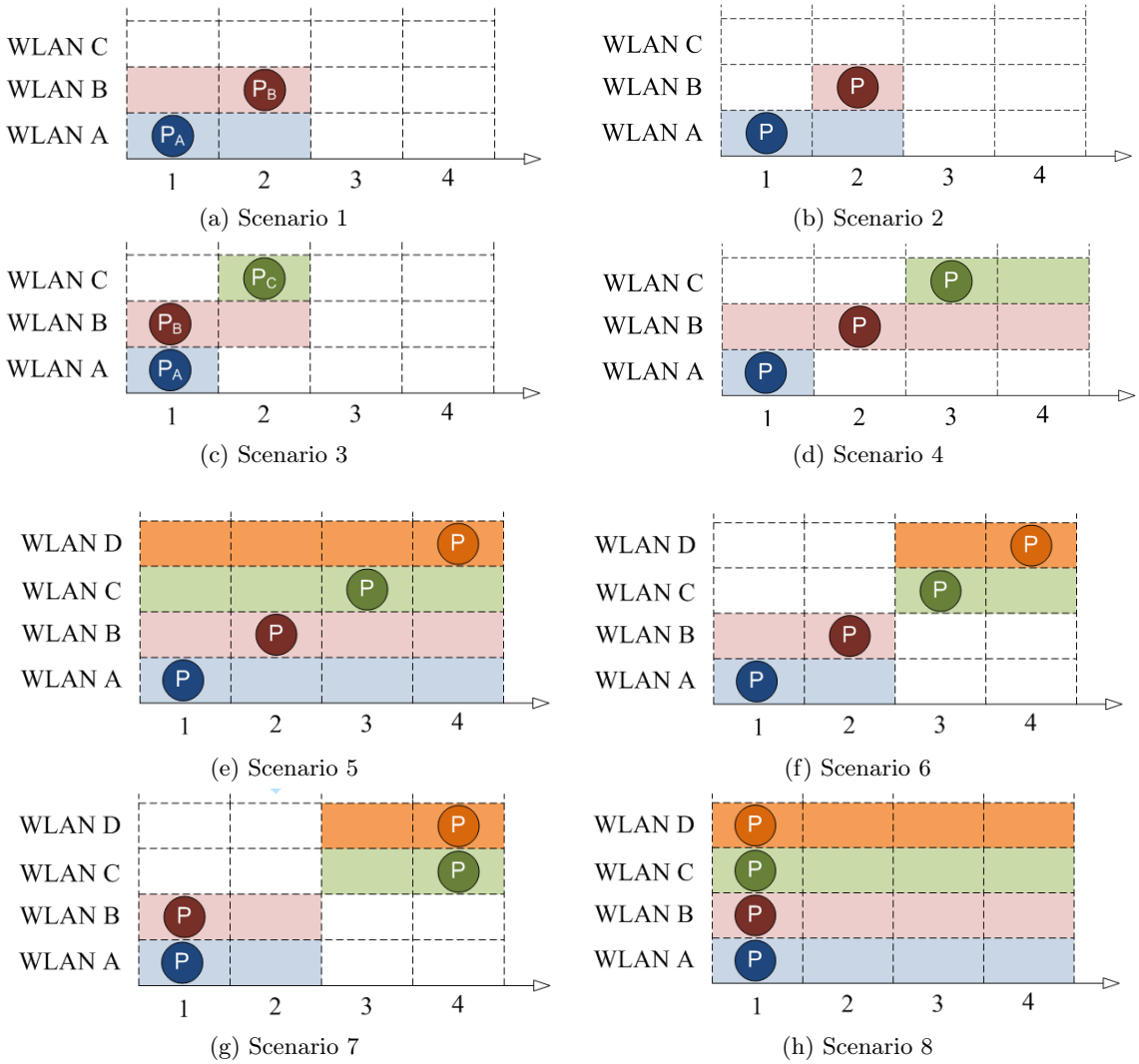


Figure 12: Channel allocations of the different scenarios.

organized as follows:

- COST libraries: contains the necessary COST libraries that allow the Komondor's primitive operation.
- Code: contains the two core files, which are *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.
- Structures: the Komondor simulator considers four main objects 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 saving packet losses causes, respectively.
- List of macros: all the static parameters are contained in the *list_of_macros.h* file.

5.1.2 Compilation and Execution

To compile and execute a Komondor's simulation, the following instructions must be followed:

1. Set the .csv input files (further defined in Section 5.1.3)
2. "cd" to *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[4].
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).[5]
   - *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_8* (SEED): random seed for the experiments.
5. Collect the results either in the output files or in the console.

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 **.

Furthermore, in order to ease experimental procedures, we provide a mechanism for executing several simulations and properly collecting all the results. At folder "scripts multiple executions" one can find few examples for executing a list of input files, thus generating several outputs.

5.1.3 Input files

The Komondor simulator relies in two types of input files for defining the simulation environment:

- **System input:** defines global input parameters such as packets length or propagation models. System input parameters are defined in Table 5.
- **Nodes input:** defines specific nodes' characteristics such as implementing features (e.g., CB policy). There are two ways of generating nodes, which is indicated in the file name. While including the keyword *nodes* all the devices (APs and STAs) must be introduced and described. Otherwise, if including the keyword *aps*, only the APs are defined, so that a STAs are randomly generated under certain introduced parameters (e.g., minimum/maximum number of STAs, maximum distance between APs and STA).[6] As a final remark, in order to ensure a proper execution, it is mandatory to introduce and input file with a list of nodes ordered by *node_id* and starting with *node_id* = 0. It is needed for the array responsible of storing the power perceived by each node (i.e., *power_received_per_node*). Table 6 describes the Nodes input parameters for both *nodes* and *aps* files.

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

[5] 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.

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

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 5: System input parameters description

5.1.4 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 computation 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 5.1.2). Additionally, those 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 show 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 13 and 14

5.1.5 Events Categorization

In order to provide a better understanding when reviewing the results of a given execution, logs are categorized according to the event that generates it. With that, further filtering processes can be carried out by developers. Table 7 describes the codes used for each type of event.

5.2 Code development

This Section is mostly focused in providing the most important clarifications regarding the code to developers that may be interested in using and/or modifying the presented simulator.

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 specifies current operation.
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 6: Nodes input parameters description

5.2.1 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:

- **Power and CCA**: power variables are stored in pW (pico watts) in order to be able to operate power magnitudes without loosing resolution[7]. However, values are presented to the user in dBm. W (-30) - mW (0) - uW (+30) - nW (+60) - pW (+90)
  $P_{\text{pw}} = 10^{\frac{P_{\text{dBm}}+90}{10}}$

5.2.2 Miscellany

- 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

[7]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).

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 7: Node's event logs encoding

```
------- 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 13: 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 14: Example of system statistics in Komondor

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.

5.2.3 Releases

So far, Komondor has experienced two main releases. The first one, Komondor v0.1, included a basic operation in which only data packets were exchanged. Such version included the following features:

- Channel Bonding
- Adjacent power
- Collisions
- Scripting
- Path loss models

Furthermore, the release presented in this document, Komondor v1.0, extends the basic operation of v0.1 and includes all the features previously described.

6 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 provided 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 networks simulator.

Regarding validations, 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.

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] Gilbert Chen and Boleslaw K. Szymanski. COST: a component-oriented discrete event simulator. 2002.

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

[3] Boris Bellalta. Ieee 802.11 ax: High-efficiency wlans. *IEEE Wireless Communications*, 23(1):38–46, 2016.

[4] Francesc Wilhelmi, Boris Bellalta, Cristina Cano, and Anders Jonsson. Implications of decentralized q-learning resource allocation in wireless networks. *arXiv preprint arXiv:1705.10508*, 2017.

[5] Francesc Wilhelmi, Boris Bellalta, Jonsson Anders Neu Gergely Cano, Cristina, and Sergio Barrachina. Collaborative spatial reuse in wireless networks via selfish bandits. *arXiv preprint arXiv:1705.10508*, 2017.

[6] Setareh Maghsudi and Sławomir Stańczak. Joint channel selection and power control in infrastructureless wireless networks: A multiplayer multiarmed bandit framework. *IEEE Transactions on Vehicular Technology*, 64(10):4565–4578, 2015.

[7] Setareh Maghsudi and Sławomir Stańczak. Channel selection for network-assisted d2d communication via no-regret bandit learning with calibrated forecastingtre. *IEEE Transactions on Wireless Communications*, 14(3):1309–1322, 2015.

[8] Masaharu Hata. Empirical formula for propagation loss in land mobile radio services. *IEEE transactions on Vehicular Technology*, 29(3):317–325, 1980.

[9] Boris Bellalta, Alessandro Zocca, Cristina Cano, Alessandro Checco, Jaume Barcelo, and Alexey Vinel. Throughput analysis in csma/ca networks using continuous time markov networks: a tutorial. In *Wireless Networking for Moving Objects*, pages 115–133. Springer, 2014.

[10] Bellalta Boris Barrachina, Sergio and Francesc Wilhelmi. Dynamic spectrum access in high-density WLANs. 2017.