Komondor: an Event-Based Wireless Network Simulator for Next-Generation IEEE 802.11ax WLANs

Sergio Barrachina-Muñoz and Francesc Wilhelmi

November 7, 2018

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 make decisions that allow enhancing the network performance.

In this document we provide an overview of the Komondor simulator, making insight on its main features, its operational mode and its development stages. This document is therefore aimed at providing to potential contributors the necessary information for getting into Komondor, so that its operational mode can be properly understood and the code can be modified/complemented. Komondor has been conceived as an open source tool that contributes to the ongoing research in wireless networks. For that, all the contents are published at the following public repository: `https://github.com/wn-upf/Komondor`. Any interested researcher is invited to collaborate.

Contents

List of Figures

List of Tables

1 Introduction

TODO: modify the introduction according to the new approach (tutorial-like document where code insights are provided).

Komondor [1] is an event-based simulator based on COST [2], a CompC++ library that allows generating discrete event simulations.[1] Komondor is mostly intended to reproduce the novel techniques included in the IEEE 802.11ax-2019 amendment [3], which is called to become a benchmark in next-generation wireless networks. Furthermore, due to the increasing popularity of learning-based approaches in WLANs, our simulator is being built to allow the inclusion of intelligent agents that make decisions in simulation time. The decision of building a new wireless networks simulator is based on *i*) the lack of 11ax-oriented simulators that include novel techniques, *ii*) the need of generating a tool able to simulate intelligent agents behavior and, *iii*) the difficulty in extending other existing solutions (e.g., ns-3) towards our abovementioned main goals.

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

Komondor is a long-term and iterative project which goals are mostly focuses in providing a reliable and accurate IEEE 802.11ax simulator. In this document we present its version v1.0, which includes the core functionalities to provide a basic operation.

1.1 Next-Generation WLANs

The increasing popularity of IEEE 802.11 WLANs has led to new strict requirements in terms of data rate and users capacity. Such situation has brought the wireless communications community to introduce novel approaches. In particular, 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 novel techniques such as OFDMA, Dynamic Channel Bonding, Beamforming and Multi-User Multiple-Input Multiple-Output (MU-MIMO) [4]. Such advanced mechanisms drastically change the current operation of WLANs and have not been previously implemented with detail in other network simulators.

In addition to the novel HE 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 [5, 6, 7, 8]. Since most of the literature that applies learning into wireless networks is theoretical in nature, there is a strong need of tools that allow implementing learning algorithms in a realistic simulation environment.

1.2 Komondor Main Features

Komondor aims to realistically capture 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. The initial version of the Komondor simulator includes the following functionalities:

- **Flexible input files processing with error control**: network capabilities can be introduced into the simulator in a very flexible manner. Moreover, an input checker is provided in order to identify the most prominent errors in the input provided by the user.
- **IEEE 802.11ax WLANs features implemented in version v1.0**:
  - **Distributed Coordination Function (DCF)**: the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) captures the basic Wi-Fi operation for accessing the channel. Moreover, Contention Window (CW) adaptation is considered.
  - **Channel Bonding (CB)**: several channel ranges can be selected during transmissions in order to maximize the spectrum efficiency.
  - **Packet aggregation**: several MPDUs can be aggregated into the same PPDU in order to reduce the generated communication overheads.
  - **Dynamic Modulation Coding Scheme (MCS)**: the MCS is negotiated between any transmitter-receiver pair according to the Signal-to-Interference-and-Noise Ratio (SINR).
  - **Ready-to-Send/Clear-to-Send (RTS/CTS) and Network Allocation Vector (NAV)**: nodes exchange packets before transmitting in order to allocate the channel and prevent collisions.
- **Statistics**: different metrics of interest are gathered and properly presented.

Future development stages are considered to include other features such as OFDMA, MU-MIMO transmissions, beamforming, or dynamic transmit power and CST adjustment.

1.3 COST

In order to provide a deeper understanding of Komondor, it is important to comprehend the COST library, which allows building interactions between components (e.g., wireless nodes). Such interaction is achieved through synchronous and asynchronous events. While the former are messages 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). Moreover, the node may start a transmission when the backoff timer is over (i.e., a trigger). Figure 1 shows the schematic of a COST component, which is characterized by its inports and outports, and a set of timers.

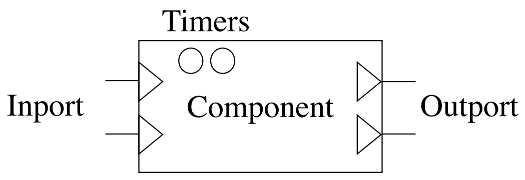


Figure 1: COST component. While inports and outports allow to directly communicate with other components, timers trigger events specific to the component.

1.4 Contributions

In this document we describe the main features of the Komondor simulator and system model considerations, as well as some basic guidelines to run it. The main contributions done in this document are listed below:

- We provide an overview of the first version of the Komondor simulator.
- We describe the implementation done for each of the included functionalities, as well as the main considerations done.
- We provide a set of validations that ensure the proper operation of the presented simulator.
- We provide a tutorial to allow an easy installation and execution of the simulator. Moreover, some implementation details are granted in case the reader is interested in extending the simulator.

Note, as well, that detailed technical information regarding code development is out of the scope of this document. Please, refer to the GitHub repository for more details on code implementation.

1.5 Document Structure

This documents is structured as follows: Section 3 describes the main design principles of Komondor. Then, Section **??** defines the implementation of IEEE 802.11 functionalities considered so far, which are validated in Section **??**. A tutorial and some development notes are provided in Section 7. Finally, some remarks are given in Section 8.

TODO: modify this part

2 Project Overview

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 2 summarizes the functionalities that constitute Komondor. A set of inputs (nodes information, simulation time, etc.) are provided to the main module (`komondor.cc`), which is in charge of initializing and monitoring the simulation. Once nodes are initiated, the core of the simulation takes place, where packet exchanges are held based on the provided wireless design (further detailed in Section 3). Finally, when the simulation runs out, a set of outputs are generated in order to shed some light on the network performance.

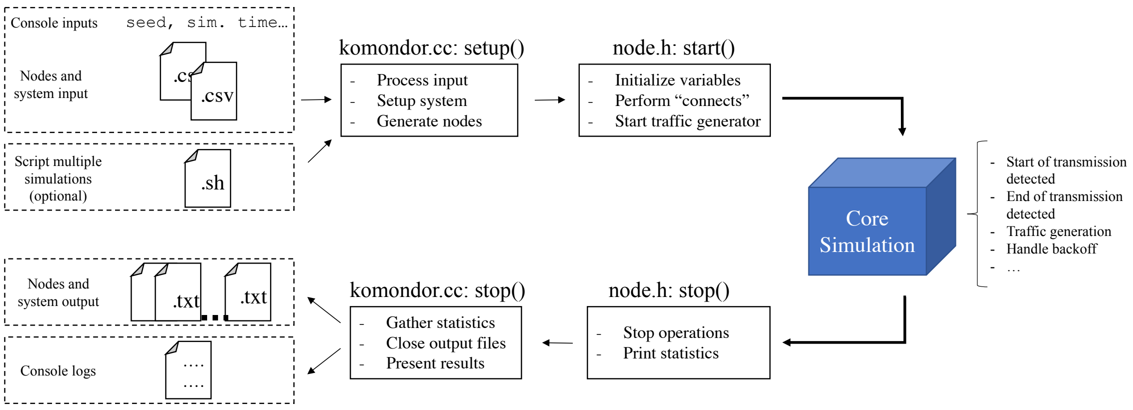
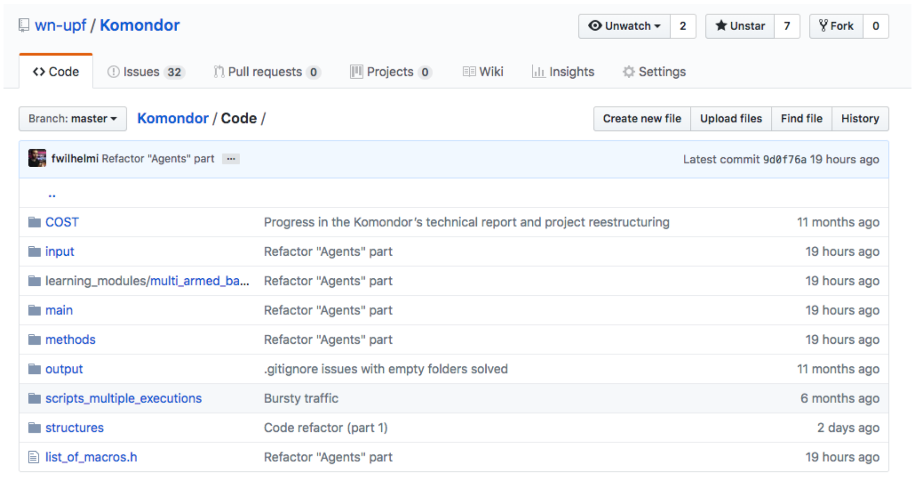


Figure 2: Komondor flowchart

2.1 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 3 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_main.cc`, `node.h`, `agent.h` and `central_controller.h`) that are in charge of orchestrating all the simulation. `komondor_main.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.
- **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.


wn-upf / Komondor

Unwatch 2 | Unstar 7 | Fork 0

Code | Issues 32 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Branch: master | Komondor / Code / | Create new file | Upload files | Find file | History

fwilhelmi Refactor "Agents" part ... Latest commit 9d0f76a 19 hours ago

..

COST	Progress in the Komondor's technical report and project reestructuring	11 months ago
input	Refactor "Agents" part	19 hours ago
learning_modules/multi_armed_ba...	Refactor "Agents" part	19 hours ago
main	Refactor "Agents" part	19 hours ago
methods	Refactor "Agents" part	19 hours ago
output	.gitignore issues with empty folders solved	11 months ago
scripts_multiple_executions	Bursty traffic	6 months ago
structures	Code refactor (part 1)	2 days ago
list_of_macros.h	Refactor "Agents" part	19 hours ago

Figure 3: Komondor files organization

3 Komondor Design Principles

Explain here the types of objects (main, node, etc.), the roles (AP, STA, trasnmitter, etc.), the states management...

In this Section we describe the Komondor design principles, which lay the foundations of the simulator. In particular, we ...

3.1 Main entities

As roughly defined in Section 2, the Komondor simulator is orchestrated by the `komondor_main.cc` class. First of all, it initializes all the entities involved in the simulation (nodes, agents, etc.) according to the input provided by the user. Then, when the simulation ends, the `komondor_main.cc` class collects all the necessary information, displays it and concludes the execution.

Regarding the entities involved in the simulation itself, we currently find the following classes:

- `node.h`: it defines the operation of a wireless node, which can be either of type AP or STA. Differentiating between these two types of nodes has important implications and, until this version, APs are the only type of node that can initiate data transmissions.

- `agent.h`: the agent entity can be optionally enabled for a given WLAN. In practice, an agent may decide the configuration of the WLAN to which it is associated. Further details about the implementation of agents are provided in Section 6.

- `central_controller.h`: this entity is responsible of collecting all the information generated by nodes and/or agents, and to provide a centralized operation to indicate certain configuration parameters. To this date, the central controller is only logically implemented, and no centralized mechanisms are provided to modify the configuration of its attached WLANs.

3.2 States-based operation

To understand the implementation provided by Komondor to simulate the IEEE 802.11 operation, it is important to show the states that are contemplated for a given node. Figure 4 illustrates all the possible states and the interactions between them.

4 Komondor's IEEE 802.11 Core Operation

Explain here the elementary parts of Komonodor that cannot be easily modified.

The core operation of Komondor is described in this Section, thus providing an overview of the most important mechanisms that allow simulating the IEEE 802.11 operation. Note that the contents described in this Section compose the core of the simulator which, from a contributor's point of view, cannot be modified without affecting to the proper system's operation (validated in

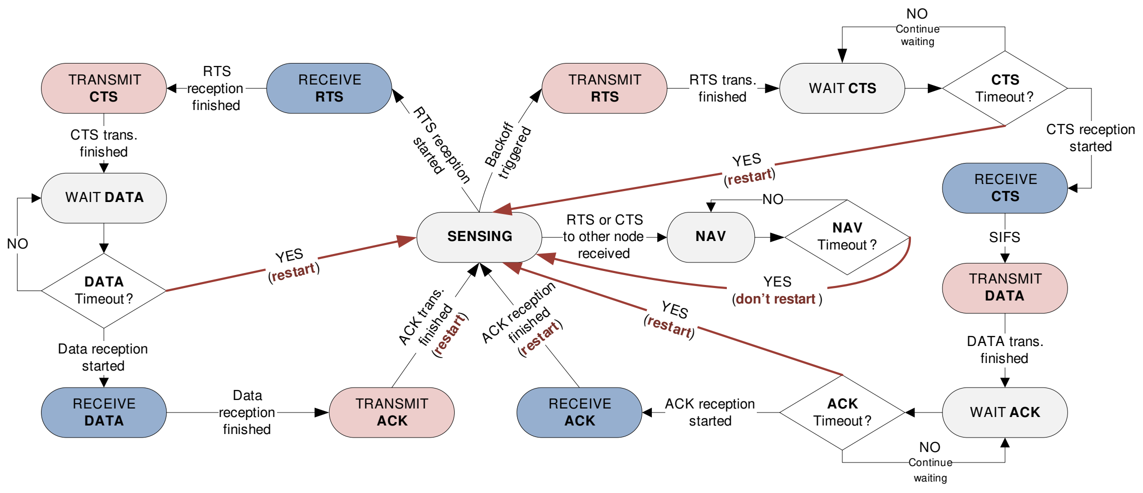


Figure 4: States diagram for Komondor nodes.

[CITE HERE THE VALIDATIONS PAPER]). In particular, we provide the implementation details at both PHY and MAC layers, in addition to the mechanisms involving upper communication layers.

4.1 PHY

Here we describe the implementation of the PHY layer and the main assumptions that have been considered.

4.1.1 Channel representation and signal transmission

Komondor, unlike other network simulators, does not consider an environment-related entity (i.e., the frequency band) with which nodes interact. In contrast, every node maintains its own representation of the channel, according to what they sensed from their physical location. Such a representation is updated each time a new interaction with the channel occurs (i.e., when a node enters or leaves the channel). At the code level, nodes are connected through a set of inports/outports, which allow updating the variables maintained for channel representation. Specifically, there are two methods in the `node.h` class that handle interactions with the channel, which are called `InportSomeNodeStartTX` and `InportSomeNodeFinishTX`. The former is activated when a node starts a transmission, and the latter when is leaves the channel. From the transmitter's point of view, the abovementioned methods are activated at all the other nodes when using `outportSelfStartTX` and `outportSelfFinishTX`, respectively. As shown later in Section 4.2, these outports are ruled by the channel access management methods.

For any transmission, the signal received at all the other nodes depends on the channel effects and propagation losses (further defined in Section 5.1). It is important to note that the same power will be sensed at a given receiver along the entire frame transmission. Such an assumption is reasonable if we consider that frame transmissions typically last a very short time.

4.1.2 Interference management and packet losses

As previously defined, nodes maintain their own channel representation, which is updated each time a third party starts/finishes a packet transmission. In particular, at any time, each node knows the additive interference generated by all the nodes that are currently transmitting. Such an information allows to manage the channel access and to determine whether an incoming transmission can be properly decoded or not.

The method that updates the information of every frequency channel available to the WLAN (note that adjacent channel interference can be also considered) is `UpdateChannelsPower()`. This method is called each time there is a change in the environment, i.e., when either `InportSomeNodeStartTX` or `InportSomeNodeFinishTX` are triggered by their respective outports. Then, according to the

Packet loss type	**Description**
PACKET_LOST_DESTINATION_TX	The destination is already transmitting any frame
PACKET_LOST_LOW_SIGNAL	The signal strength of the attempted transmission is not strong enough to be decoded, i.e., it is lower than the receiver's sensitivity
PACKET_LOST_INTERFERENCE	The sensed interference is greater than the receiver's sensitivity
PACKET_LOST_PURE_COLLISION	Two or more nodes transmit to the same destination and their signal strengths are enough to be decoded
PACKET_LOST_LOW_SIGNAL_AND_RX	The destination is any receiving state and the attempted transmission's signal strength is not strong enough to be decoded
PACKET_LOST_SINR_PROB	The packet lost due to SINR probability, which is associated to the Modulation and Coding Scheme (MCS) used
PACKET_ACK_LOST	Specific for ACK losses
PACKET_LOST_RX_IN_NAV	The intended receiver is in virtual carrier sensing (state \texttt{NAV})
PACKET_LOST_BO_COLLISION	Two nodes transmit simultaneously because they chose the same backoff value
PACKET_LOST_OUTSIDE_CH_RANGE	The frame is transmitted outside the primary channel of the receiver
PACKET_LOST_CAPTURE_EFFECT	The CE condition is not accomplished at the receiver

Table 1: Types of packet losses handled by Komondor.

node's current state, different actions are performed. For instance, in state `SENSING`, a node noticing a new transmission is expected to check whether its backoff countdown can continue or not. Moreover, in case of being in any `RECEIVING` state, the node would check if the currently received frame is lost due to the new generated interference. Related to this, another important assumption is that the Signal to Interference and Noise Ratio (SINR) of every packet transmission must be kept above a capture effect (CE) threshold, which is an input parameter. Otherwise, the packet is lost. Specifically, a *stronger-first* interference pattern is considered, so that any frame is properly decoded only if posterior transmissions do not generate high enough interference to discard it. Note that a dominant data transmission is not going be considered at a given receiver that is already receiving any type of data from another node. In such case, both transmissions are considered to be lost. The stronger-first CE principle is shown in Figure 5.

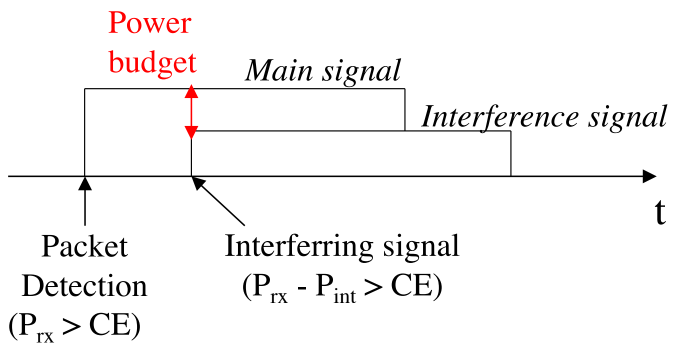

Power budget
Main signal
Interference signal
t
Packet Detection ($P_{rx} > CE$)
Interferring signal (P_{rx} - $P_{int} > CE$)

Figure 5: Capture effect condition for a frame transmission.

The CE condition is one of the reasons why packet losses may occur. In particular, Komondor contemplates several types of packet losses, which are informed to the transmitter in simulation time. The idea is to provide detailed "off-line" information about the interactions between nodes, so that packet losses can be properly understood when the simulation finishes. Note that such an information is not used in any different way than for generating statistics. The type of packet losses handled by Komondor are listed in Table 1.

4.2 MAC

Once the PHY layer has been presented, now we introduce the operation carried out at the MAC, which mostly refer to channel access and frames formatting.

4.2.1 Channel access

In Komondor, the channel access is ruled by the Distributed Coordination Function (DCF), which combines Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) with Binary Exponential Backoff (BEB). Accordingly, transmissions are carried out if the target channel (or set of channels) has been empty for a given Backoff (BO) time, which is computed according to a dynamic Contention Window (CW). The PHY implementation allows the MAC layer to perform the channel access procedures. Figure 6 shows an example of the DCF operation.

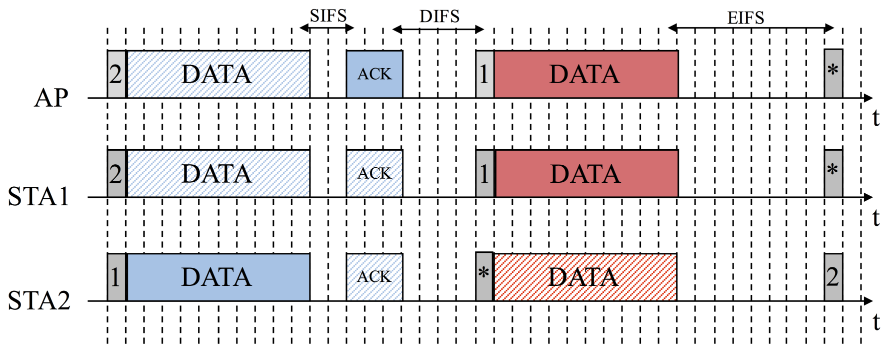


Figure 6: DCF operation in a WLAN. STA2 wins the channel access because its initial BO timer is the lowest one. During the packet transmission, STA1 listens the channel busy and stops its BO operation. Once data transmission is finished and the channel has been idle for a DIFS interval, the BO procedure is activated in all the devices that have a packet to be transmitted and sense the channel idle.

At the code level, provided that there are packets in the buffer, a node computes the backoff counter according to its current CW through the `ComputeBackoff()` function (located at the `backoff_methods.h`). Note that the backoff is paused/resumed according to the channel state, and can be checked through the functions `PauseBackoff()` and `ResumeBackoff()`, respectively. Moreover, two types of BO are considered, according to the temporal domain. On the one hand, we find *continuous BO*, where time is considered to be continuous, so that collisions by backoff cannot occur. On the other hand, we have *slotted BO*, where the time can be divided in slots, and nodes handle their backoff accordingly. In this case, collisions by backoff can occur, but an important assumption is that nodes are synchronized in some sense (slots are the same for everyone). The type of backoff is indicated by an input parameter.

Regarding the computation of the BO value, note that CW is dynamically adapted on a per-packet basis: CW increases or decreases according to successful/failed transmissions. 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 $\text{CW} = \text{CW}_{\text{min}} \times 2^{\text{CW}_{\text{count}}}$. There are two available ways of computing the BO value as a function of the CW: *i)* Uniform: the generated BO is a number between 0 and $\text{CW} - 1$, and all the values have the same probability, and *ii)* Exponential: instead of using an uniform distribution, we use an exponential one, so that the generated BO is given by the mean CW value ($\frac{\text{CW}-1}{2}$).

4.2.2 Virtual carrier sensing

On top of channel access, we find the data transmission, which in this version of Komondor is done through **mandatory** Ready-To-Send/Clear-To-Send (RTS/CTS). The RTS/CTS mechanism is implemented in order to minimize the collisions by hidden node. Through RTS/CTS, transmitting nodes attempt to block the channel for the duration of their transmissions. For that, they send RTS packets and wait for confirmation about the clearness of the channel from the receivers' 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 7.

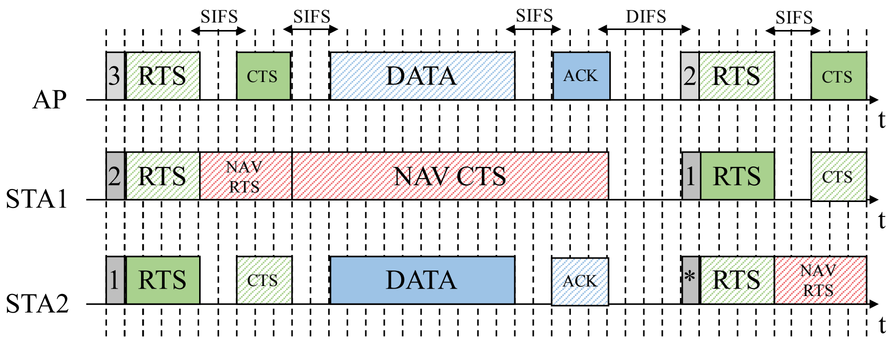


Figure 7: Example of RTS/CTS implementation. The transmitter (STA2) sends an RTS packet before starting a transmission. The receiver (AP) answers with a CTS as it senses the channel free. The other coexisting devices (STA1) that listen either the RTS and/or the CTS, set their NAV accordingly.

In practice, every transmission is initiated with an RTS. Each time a node is able to decode either an RTS or a CTS transmitted to any other recipient, it enters to `NAV` state. The virtual carrier sensing lasts for the entire transmission (including RTS, CTS, DATA and ACK frames), or until a new transmission updates the NAV timer. In case of being in the `NAV` state, a given node cannot transmit or receive any kind of data.

4.2.3 Frames formats

As previously introduced, the RTS/CTS operation is mandatory in the last version of Komondor. As a result, each data transmission needs to follow the next structure: *1)* Send an RTS frame, *2)* receive a CTS frame, *3)* send the DATA, and *4)* receive an ACK. If any of these frames is lost (refer to Section 4.1.2), the entire transmission is considered to have failed. All the abovementioned frames are defined as in the 11ax amendment and, despite the actual data is not being transmitted in Komondor, the size of each field is respected. The size of each frame is defined in Table 2. Note that L_{SF}, L_{MH} and L_{TB} are fields that constitute the headers of certain types of frames.

Parameter	**Description**	**Value**
L_{RTS}	Length of an RTS packet	160 bits
L_{CTS}	Length of a CTS packet	112 bits
L_{D}	Frame size	12000 bits
L_{SF}	Length of service field	16 bits
L_{MH}	Length of MAC header	320 bits
L_{TB}	Length of tail bits	18 bits
L_{BACK}	Length of a (block) ACK	112/432 bits

Table 2: Frames size according to the IEEE 802.11ax amendment.

Apart from standard 11ax frames, packet aggregation is considered, and its utilization can be specified as an input parameter. Roughly, packet aggregation aims at reducing 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.

In Komondor, every time a transmission is attempted to be initiated, the number of packets to be aggregated is checked, so as the packets available at the buffer. In addition, due to the PPDU duration limitation,[2] the number of packets aggregated can be lower than the intended one. In particular, the function `findMaximumPacketsAggregated()` is responsible to find the maximum number of packets that can be aggregated in a given transmission.

[2]The maximum PPDU duration defined in the IEEE 802.11 amendment is 5,484 ms.

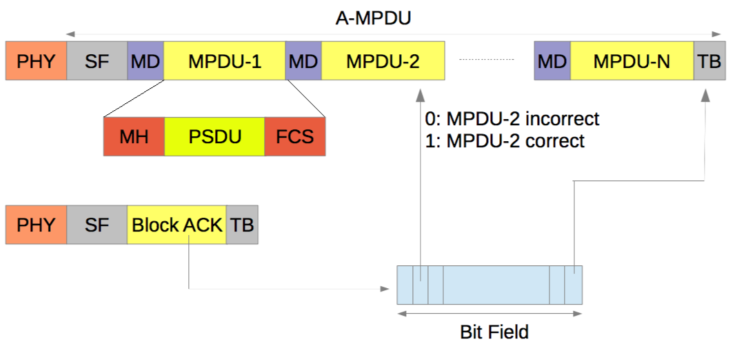


Figure 8: Example of packet aggregation in which N MPDUs are concatenated to be sent during the same packet transmission.

4.2.4 Transmission times

Explain here the transmission times and the data rates.

4.3 Upper communication layers

4.3.1 Traffic generator

5 Modules and Models Available in Komondor

Explain here the implementations that can be complemented by an interested researcher. The explanation should be oriented according to the code structure.

5.1 Physical Propagation module

5.2 Traffic generation module

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 process, so that the average time between packets is determined by the packet generation rate λ, and which is given by $\Delta_{\text{p}} = \frac{1}{\lambda}$.
- **Deterministic**: packets are generated at fixed time intervals given by the packet generation rate, $\Delta_{\text{d}} = 1/\lambda$.

Figure 9 illustrates the aforementioned traffic models.

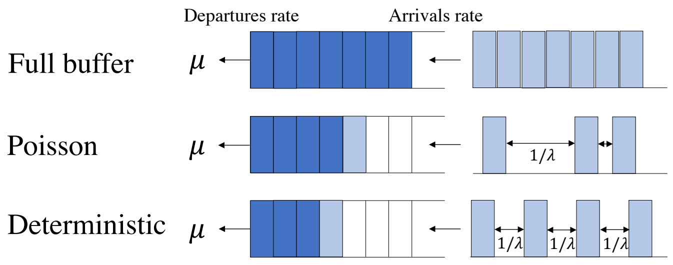


Figure 9: Traffic models used in Komondor

5.3 Dynamic Channel Bonding module

6 Agents in Komondor

The Komondor simulator includes a set of modules that allow simulating the operation of WLANs with embedded agents. So far, for the release v.2.0 (TODO: clarify this), decentralized agents are fully implemented. Note, as well, that the communication infrastructure for enabling both distributed and centralized is available. However, Komondor is not yet ready to support these operational modes. Intensive work about a Machine Learning (ML) architecture is currently in progress, where all the learning modes are being defined in the context of Komondor.

Komondor incorporates agent entities (`agent.h` class), which are associated to nodes and are responsible of monitoring their performance and performing recommendations for certain configurable parameters. Accordingly, Komondor seeks to simulate an agent-based infrastructure where APs contain the necessary intelligence to learn in a decentralized manner.[3] Note, as well, that a single agent can be associated to a given WLAN, and the communication is held only with the AP.

The agent entity is therefore responsible of the following functionalities:

- Make queries to the agent to retrieve information regarding its current performance (e.g., packets sent/lost, delay, etc.) and configuration (e.g., transmission power, number of associated STAs, etc.).
- Process the information obtained from the AP and execute a given learning-based operation.
- Generate a new configuration to be suggested to the AP.

Figure 10 illustrates the operations held by an AP and its embedded agent during the learning procedure. Both entities communicate each other through a set of inports/outports. Note, as well, that such a communication occurs at certain intervals that can be indicated by the input. In addition, delays can be added to simulate different architectural limitations (e.g., hardware delay).

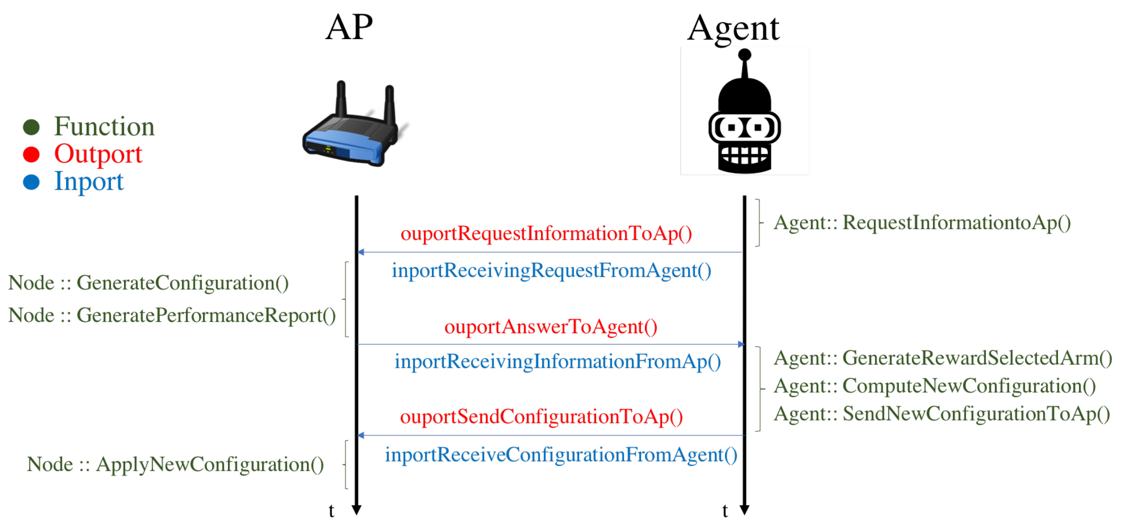

AP
Agent
● Function
● Outport
● Inport
ouportRequestInformationToAp()
Agent:: RequestInformationtoAp()
inportReceivingRequestFromAgent()
Node :: GenerateConfiguration()
Node :: GeneratePerformanceReport()
ouportAnswerToAgent()
inportReceivingInformationFromAp()
Agent:: GenerateRewardSelectedArm()
Agent:: ComputeNewConfiguration()
Agent:: SendNewConfigurationToAp()
ouportSendConfigurationToAp()
inportReceiveConfigurationFromAgent()
Node :: ApplyNewConfiguration()
t
t

Figure 10: AP-agent communication-based operation in Komondor.

It is important to note that `agent.h` is responsible of orchestrating the learning operation through the communication with the AP. The learning mechanisms are independent modules that are used by the agent and which must be placed at the *Komondor/main/learning_modules* folder. So far, we provided the `multi_armed_bandits.h` class, which includes the necessary functions to apply the Multi-Armed Bandits framework [10]. Such a class is responsible of handling the environment through which MAB-based strategies can be applied. In other words, it processes the information provided by the agent (performance and configuration) to generate the necessary data structures that allow implementing certain action-selection strategies. Once the data is prepared

[3]Decentralized learning frames a situation in which WLANs are able to learn based on their own performance information (further information can be found in [5, 6, 9]).

for the MAB operation, the new configuration is obtained according to the chosen action-selection strategy. For code modularization purposes, we provided a subfolder (i.e., *Komondor/main/learning_modules/action_selection_strategies*) containing different action-selection strategies such as ε-greedy [11] or Thompson sampling [12].

The project has been therefore conceived so that any interested researcher can easily implement a given learning mechanism (either online or offline), thus making use of a flexible infrastructure.

7 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.

7.1 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.

7.1.1 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)
  $P_{\text{pw}} = 10^{\frac{P_{\text{dBm}}+90}{10}}$

7.1.2 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.

8 Conclusions Future Development Notes

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.

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).

[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).

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Francesc Wilhelmi, Sergio Barrachina-Muñoz, Cristina Cano, Boris Bellalta, Anders Jonsson, and Gergely Neu. Potential and pitfalls of multi-armed bandits for decentralized spatial reuse in wlans. *arXiv preprint arXiv:1805.11083*, 2018.

[10] Peter Auer, Nicoló Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[12] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.