## General description

WSNet is an event-driven simulator for wireless networks on a large scale. It is written in C/ModernC++, uses free (under CeCILL license) and accessible libraries and it is configured using XML files. In comparison with the existing network simulators at the time of its conception, WSNet was designed to stand out on two particular points:

* Management of a massive number of nodes, up to several thousand, to meet the specific needs of the wireless sensor networks.
* A “realistic” physical layer for researchers and engineers working on the development of algorithms for the upper layers of the networks.

Its architecture consists in different blocks that model characteristics and properties of the radio medium. As an example, separate blocks are defined for the radio propagation, the interference computation or the radio modulation.

A number of secondary constraints were placed on the development of WSNet and are now as features of the simulator:

* Complete and modular model for the software architecture of nodes, i.e. the different layers of the OSI model.
* Modeling spectrum
* Modeling of collisions and interference.
* Modeling of mobility and physical environment - such as the presence of obstacles.

WSNet uses a mode of simulation said in discrete events. The principle is to consider that the system fits its behavior only to the appearance of events in time. By opposition to a simulation at continuous time, a simulator with discreet events can so save up a great deal of intermediate calculations, to focus only on the critical moments of the simulation. It supposes nevertheless that the evolution and the behavior of the system is in agreement with the hypothesis of departure.



Figure 1: Appearance of events at specific times of the temporal evolution line

In the last figure, we can see the discreet moments when the simulator is going "to wake up" and to update the state of the system. The temporal notion is considered, but no update of the simulated system is sensible to be made outside these moments of awakening. Within the framework of simulation of network, these events will be mainly the sending and the reception of packages, and the evolution of the position of nodes and the physical environment within the framework of simulation with mobility.

## Node Architecture

WSNet possesses a modular structure, which is partially represented on Figure 2. Some blocks model the properties of the radio medium, the simulation core and the characteristics of simulated nodes.

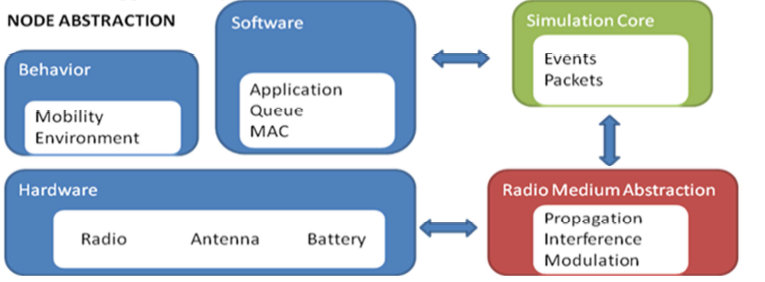


Figure 2: WSNet architecture.

The radio medium abstraction consists of three sub-blocks: propagation, interference and modulation. The simulator core allows generating and processing the events and the packets of the simulation. This is a key element of the simulator performance because it determines the speed degree and the effectiveness of simulations. The node abstraction is composed of three sub-blocks:

* Behavior describes the mobility and the environment in which the nodes can evolve (fire, obstacles, etc)
* Hardware defines the different physical properties of each node (TX/RX frequency, TX power, battery)
* Software describes the upper layer of the simulation (MAC, NWK and application layer)

As shown in Figure 3, a node is a set of superimposed layers, which communicate directly with the lower or upper layers by the intermediaries of functions TX and RX. In every layer corresponds a particular model, which must be written and compiled in C. These models include at the same time either the instances of the TCP/IP protocol for high layers, or else simple mathematical calculations of BER (Bit Error Rate) for a modulation layer.

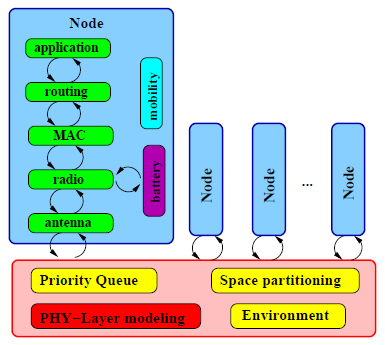


Figure 3: Node architecture.

Furthermore, WSNet also provides the existence of multiple radio access on a single mobile terminal, enabling communication using different frequencies and channels. This feature can be used to simulate nodes with several antennas and technologies. An example would be a node with both a Bluetooth and a WiFi transceiver attached on the device.



Figure 4: WSNET architecture of different nodes with their different radio access.

### Software

Practical communication system design is aided by a well-defined conceptual framework called the Open System Interconnection (OSI) reference model. The OSI model is a seven-layer architecture, where each layer is responsible for specific sub-systems.



Figure 5: Open System Interconnection (OSI) reference model

Differently from the conventional networks layers, WSNet and WSN consider only the following layers: application (APP), network or routing (NET), medium access control (MAC), and physical (PHY). Although there is no transport layer, since it is complex and it would waste sensors energy, some WSN protocols have been designed for congestion control and reliable end-to-end communication (functions of the transport layer).



Figure 6: WSNET protocol stack

#### Generic software module

Each layer is defined as an independent module with inputs and outputs allowing to communicate with upper or lower modules.

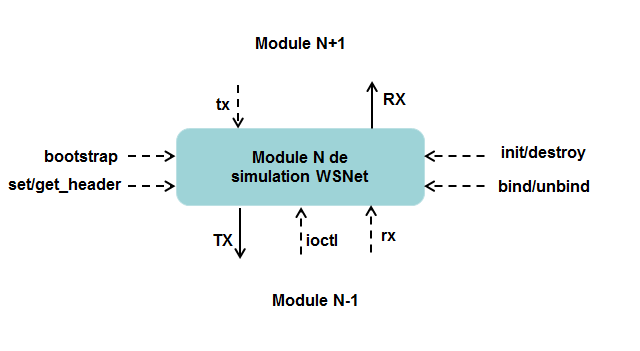


Figure 7: WSNet module

Each module defines generic interfaces:

* Init: this function is called to create the module, allocate the memory and initialize the parameters and global variables.
* Destroy: this function is called for the destruction of the module, to free memory and to remove parameters and global variables.
* Bind: this function is called to create the module, to allocate memory and initialize the parameters and variables specific to the node (local).
* Unbind: this function is called for the destruction of the module to free memory and remove parameters and local variables.
* Bootstrap: This function is called at the start of simulation to install and create the first event.
* set\_header and get\_header: These functions are used to add headers to the packets
* Ioctl: This function is used to share information between independent layers.
* TX: this function is called when the upper layer transmits a data packet.
* RX: This function is called automatically when a packet is received from the lower layer**.**

In addition to these interfaces, each module can use the TX and RX functions to communicate with upper and lower layers. When a module wishes to send a packet to a lower module, it calls the TX function that automatically calls the tx function of the lower layer. When a module wishes to send a packet to the upper module, it calls the function RX automatically calling the rx function of the upper layer

Global variables are saved in classdata structure whereas local variables in nodedata structure.

#### Shared memory

The implementation of a shared data mechanism is necessary for a cross-layer approach due to the layered and disjoint structure WSNET firstly propose for its modules. As modules are implemented separately and do not share information, task to implement a protocol, which is based on the mutual agreement of two layers, becomes difficult. Thus, a new module have been proposed, called the shared data module. It is designed to enable access to the same information from different blocks. As the WSNet simulator is based on one single process, the mutual exclusion problem is not an issue. Therefore, any modification made by any layer will be available for other layers leading to the best implementation scenario for cross-layer protocols.

### Physical Layer for WSNET before version 3.1

The modeling of the radio takes care of the decision of reception or non-reception of packets, leaving the MAC layer to arbitrate in the access to the channel. The models existing for the MAC layer concern mainly the protocol 802.15.4, in 868MHz, 902MHz and 2.4GHz. These models implement particularly the back-off described in the protocol. The modeling of the physical layer is common to all the nodes and is composed of following modules:

* Propagation: this module is the lowest of the layer, and represents the path loss and all the effects bound to the power of the signal, such as the shadowing or the fading. The interferences are calculated by another module.
* Interference: this module gets back the value of the SNR of the propagation module and calculates the SINR, the ratio Signal on Interferences + Noise. This calculation can be made with an arbitrary granularity, which is once for the entire package, or once on every bit, or for any intermediate division.
* Modulation: this module sends back the BER following the value of the SNR or SINR of the propagation or interference layers.
* Antenna: this module allows integrating the properties of antennas, especially the different gains following the angle of transmission between the nodes.

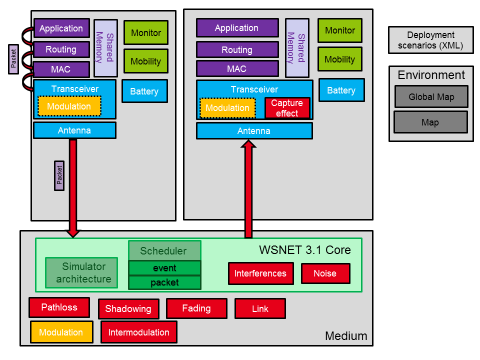


Figure 8: Architecture of WSNet 3.1

In WSNet 3.1, communication between nodes on the network is via packets.

|  |  |  |
| --- | --- | --- |
| Packet | Id | Id of the packet |
| size | Size of the packet (Byte) |
| Real\_size | real size of the packet data (Bit) |
| Type | Type of the packet |
| node | node that has created the packet |
| Duration | packet TX/RX duration |
| Interface | interface that has emitted the packet |
| Clock0 | packet RX start |
| Clock1 | packet RX end |
| PER | Packet Error Rate |
| RxmW | RX power in mW |
| RxdBm | RX power in dBm |
| RSSI | RSSI indicator in dBm |
| BER | List of Bit Error Rate for each segment |
| Noise\_mW | List of Noise for each segment |
| LQI | Link Quality Indicator |
| txdBm | TX power in dBm |
| channel | Id of the channel used |
| modulation | Modulation class |
| Tb | Time to send a bit |
| destination | Destination for encapsulation in lower layer |
| fields | Packet Data |

The various packet fields are filled during the passage of the packet structure in the corresponding layer. The packet is created by the application layer when a node decides to send to all nodes within communication range. Application layer fills the information like the packet identifier (id), the size and the type and the Id of the node sending the packet. The packet is then transmitted to the lower layer thanks to the TX function. The PHY layer is modeled using radio and antenna modules that are instantiated for each node, as well as a medium module that is instantiated once for all nodes. The RX, TX, and MEDIA\_RX MEDIA\_TX functions allow the exchange of packets between the different modules.

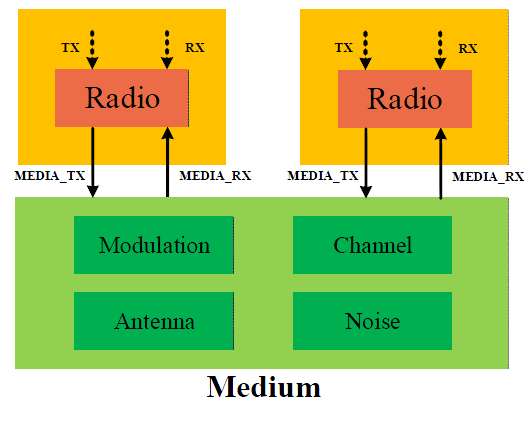
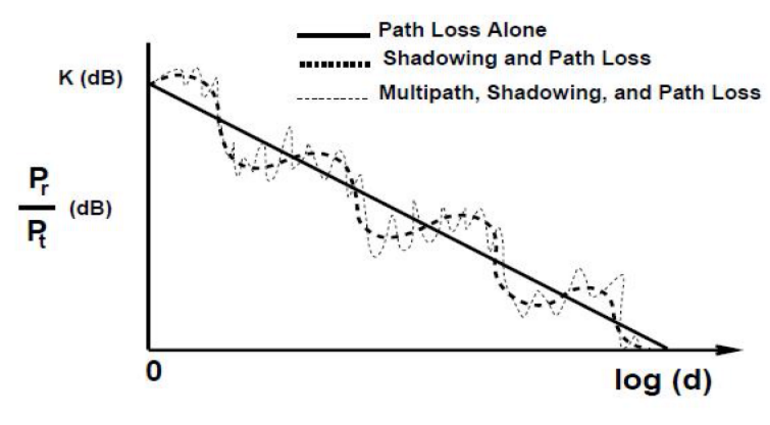


Figure 9: Packet exchange in WSNet

The functionality of the radio module is to initialize the radio filling the packet fields that characterize the radio signal according to the standard used. It receives the packet from the upper layer (MAC), checks if the node is not able to receive or transmit before sending the packet to the lower module (antenna). When receiving a packet, it decides to accept or reject based on the Packet Error Rate PER. Finally, it decrements the energy of the battery based on the operation performed.

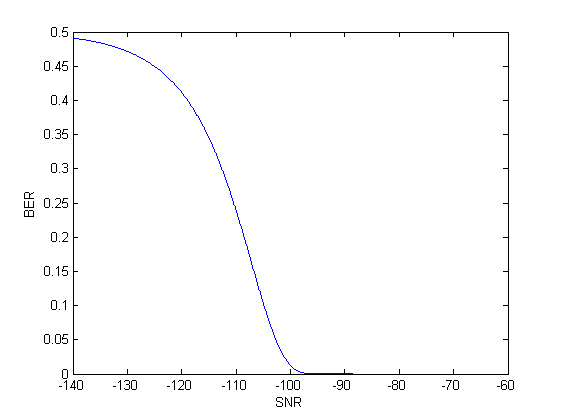
The medium module is not a node module but is located in the core of the simulator. Its task is to calculate the received power (rxdBm), the SNR, RSSI, BER and PER based on the parameters of the channel, to update the fields in the packet structure and forward the packet to receiving antennas. The medium module uses a set of blocks representing different properties of the radio medium:

* Channel library to model the attenuation due to the radio channel. This library contains the following models: path-loss, shadowing and fading.



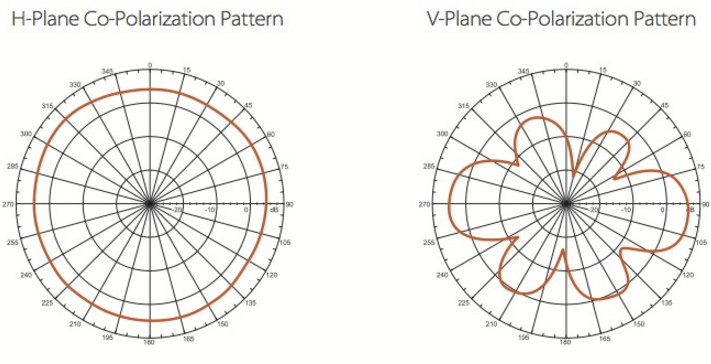
**Figure 10:** **Propagation models: Pathloss, Shadowing, Fading**

* Interference block that calculates the SINR based on the received signal strength, noise and interference.
* Modulation calculating the BER as a function of SINR.



**Figure 11: Example of modulation library: BER as a function of SNR**

* The antenna module allows taking into account the properties of the antenna, for example, gain, loss, orientation, etc.



**Figure 12: Example of antenna pattern**

Upon packet emission, by an internal node or an external program, WSNet generates two events:

* a carrier sense event that occurs at transmission start;
* a packet reception event that occurs at transmission end.

The carrier sense event is used to notify all nodes with the apparition of a new signal. This event provides each node with the radio resource (frequency, code, modulation, etc.), the SINR (Signal to Interference plus Noise Ratio), the reception power, and the BER (Bit Error Rate) of the signal computed for the considered node. This event enables the radio to lock on the packet when its transmission starts with the knowledge of the signal strength and the SINR. With this event, one may easily simulate capture effects, loss of a signal, etc. The packet reception event notifies the reception of the packet.

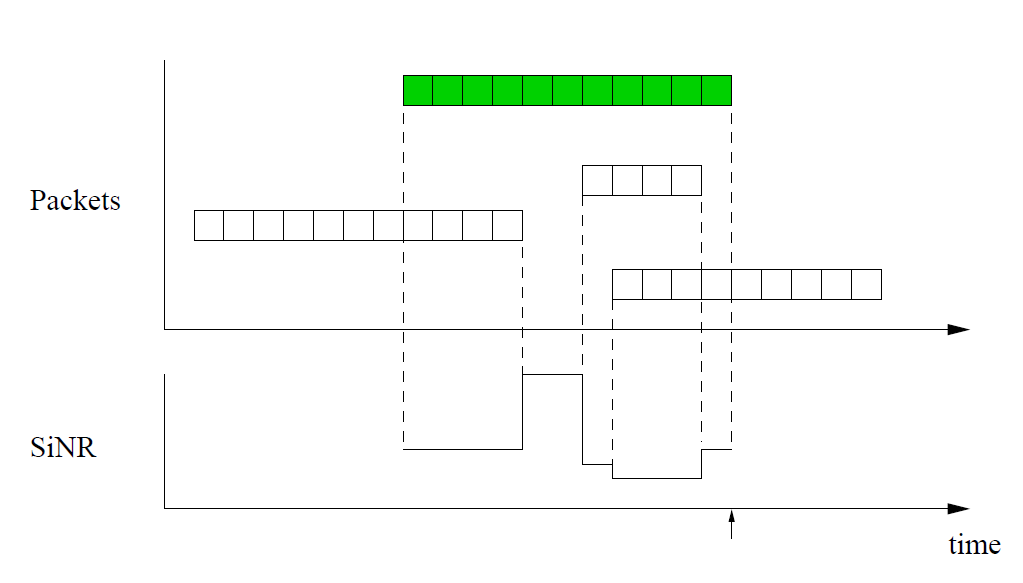
Given a receiving node, the SINR, the reception power as well as the BER are computed for each bit of the packet. This information is provided to the receiving node. Depending on the user configuration, the simulator either drops the packet as a function of the packet PER (Packet Error Rate) or introduces errors in the packet according to the BER.

To compute the reception power of a packet, several factors are considered:

* The transmission power;
* The radiation pattern of the emitter's antenna;
* The propagation model to compute the signal power at the receiver's location;
* The radiation pattern of the receiver's antenna;
* The sensibility of the receiver's antenna.

To compute the SINR at a given time, we consider the following factors:

* The white noise of the receiver's antenna;
* For each packet that is being transmitted at that time:
  + The reception power of the colliding packet;
  + The interference model and the radio resources that are used by the considered and the colliding packets.



**Figure 13:** **SINR computation in WSNet**

Once the SINR is known, the corresponding BER is computed using the modulation model.

This is modeled in kernel\model\_handlers\media\_rxtx and noise files.

### Physical Layer for WSNET 4.0

WSNet 4.0 is now signal-oriented and is implemented in Modern C++. WSNET simulator has been extended in several aspects (e.g., spectrum use, interference, capture effect) to take into account flexibility and specificity of PHY/MAC layers. The core of WSNet simulator is now separated from the medium and is dedicated only to the load, startup, configuration and orchestration of the models and the simulation scheduling. The following figure shows the new architecture of WSNet.

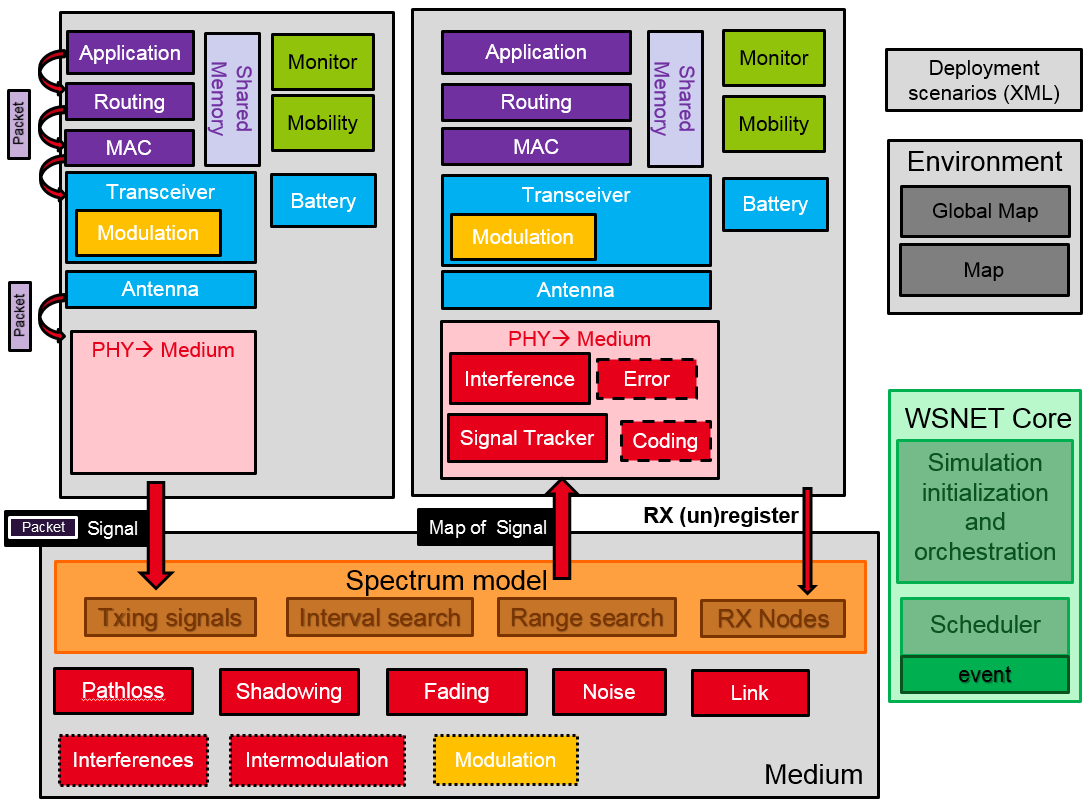
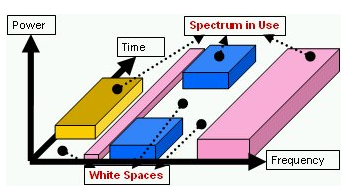


Figure 14: Architecture of WSNet 4.0

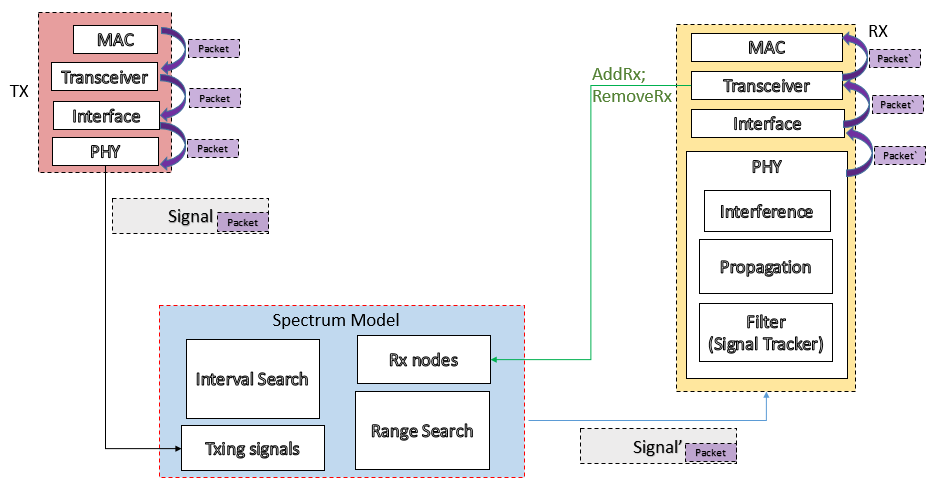
To support PHY layer heterogeneity and flexibility, we have modified the core of the WSNet simulator by including a spectrum model. This new model provides a support for modeling the both frequency and time-dependent aspects of communications. Moreover, it exploits the spectrum in terms of spectral resources instead of only logical channels.



***Figure 15:******Spectrum model***

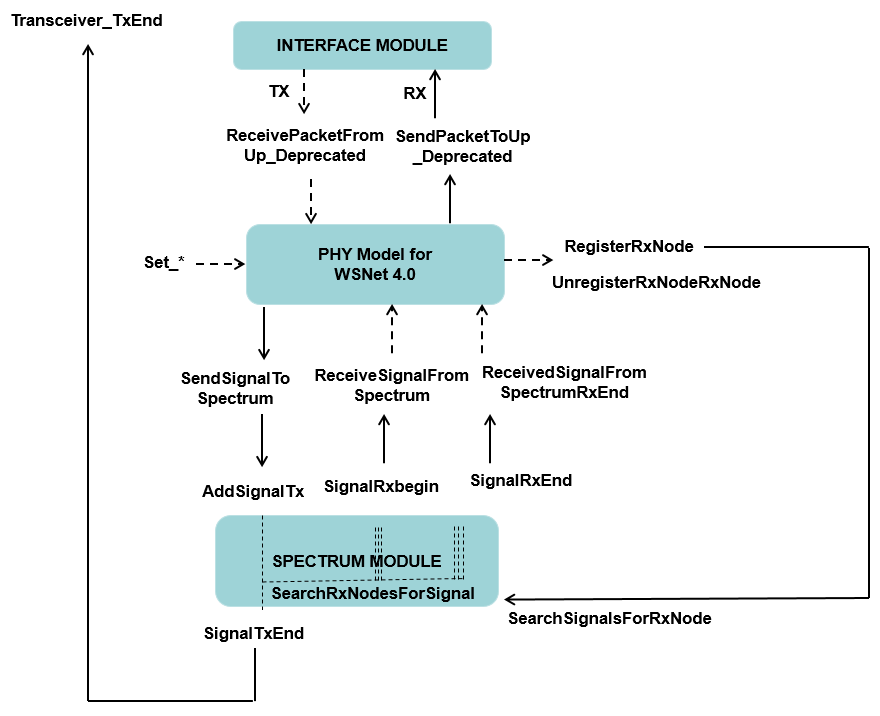
It provides more accurate PHY models (several waveforms with different configurations) and a more accurate interference model for heterogeneous simulations.

It is important to notice that all layers above transceiver have full backward compatibility. Thus, packets can be generated and used as before without need of redesigning. Furthermore, a transceiver model from a 3.0 version needs little to none change to work currently, however not taking the full potential uses of the new features. A packet typically created on the application layer is transmitted to the lower layers as it was done on the 3.0 version. When the packet arrives on the interface (antenna), it then forward the packet to the PHY model which received the packet on its ReceiveFromUp\_Deprecated interface (API) call. The PHY will then create a Signal (updating the PSD, time, frequency) and insert the packet inside. After that, the PHY will call all the medium-related models (e.g. coding) which will then perform their tasks on the Signal and, once back to the PHY model, the PHY transmit the Signal to the spectrum model on which it was connected by using the SendToSpectrum API.



***Figure 16:******New WSNet kernel supporting massive scenarios and PHY layer flexibility.***

In order to receive the Signals created by others, nodes need to register on the spectrum specifying the frequency intervals on which they will be listening. The registration of nodes is done by the PHY of a node willing to listen the channel for signals by calling the Spectrum API RegisterRxNode. The spectrum will then register the node and the frequency intervals on which this nodes will be listening. When a signal is transmitted, the spectrum model will then look for all nodes that are registered to RX a frequency interval that intersects with the signal frequency interval. Once the nodes are identified, the spectrum demands the Scheduler to add events of RX\_BEGIN and RX\_END on the appropriate timing.



***Figure 17:******Packet/Signal exchange in WSNET 4.0.***

The RX\_BEGIN event will call the ReceiveSignalFromSpectrum API of the receiving PHY at the beginning of the reception and the RX\_END event will call the ReceiveSignalFromSpectrumRxEnd API of the receiving PHY when the reception is over. On the call of the ReceiveSignalFromSpectrum API, the PHY of the receiver side will then call all the medium-related models (e.g. interference, modulation, error, coding, signal tracker) to verify whether the signal should be treated and the node is able to successfully receive the signal. After that, if the models decide that the node is able to receive the signal, the PHY model will transmit the packet inside this signal to the interface (antenna) model by using the SendPacketToUP\_Deprecated API which will forward the packet to the interface connected to the PHY model. From this point forward, it is very compatible to what happened before, i.e. the models will transmit the packets to the upper layers until the packet reaches it correct layer destination.

Two new models have been introduced: signal tracker and interference. The signal tracker model is responsible for verifying which signal is going to be the signal of interest, or the tracked signal. Thus it verifies whether the signal will be the one that the transceiver will decode, or it will be considered as an interferer for another signal already in track. Concerning the interference model, we are now evaluating the Interference between two signals and not two packets. The amount of energy coming from the interference in a time-frequency proportion of the received packet is computed as follows:

\* psd \*

**∆t**

**∆f**

**Tpacket**

**Interference**

Figure 18: Interference model based on signal

is the packet duration of the received packet

is the bandwidth of the interference

is the duration of the interference

is the power spectral density value of the interference

Then, we estimate the **SINR** according to received power (), the noise (N) and interferences () as:

Once the SINR is known, the corresponding BER is computed using the modulation model.