ns-3 training: Wi-Fi

Sébastien Deronne, June 2018

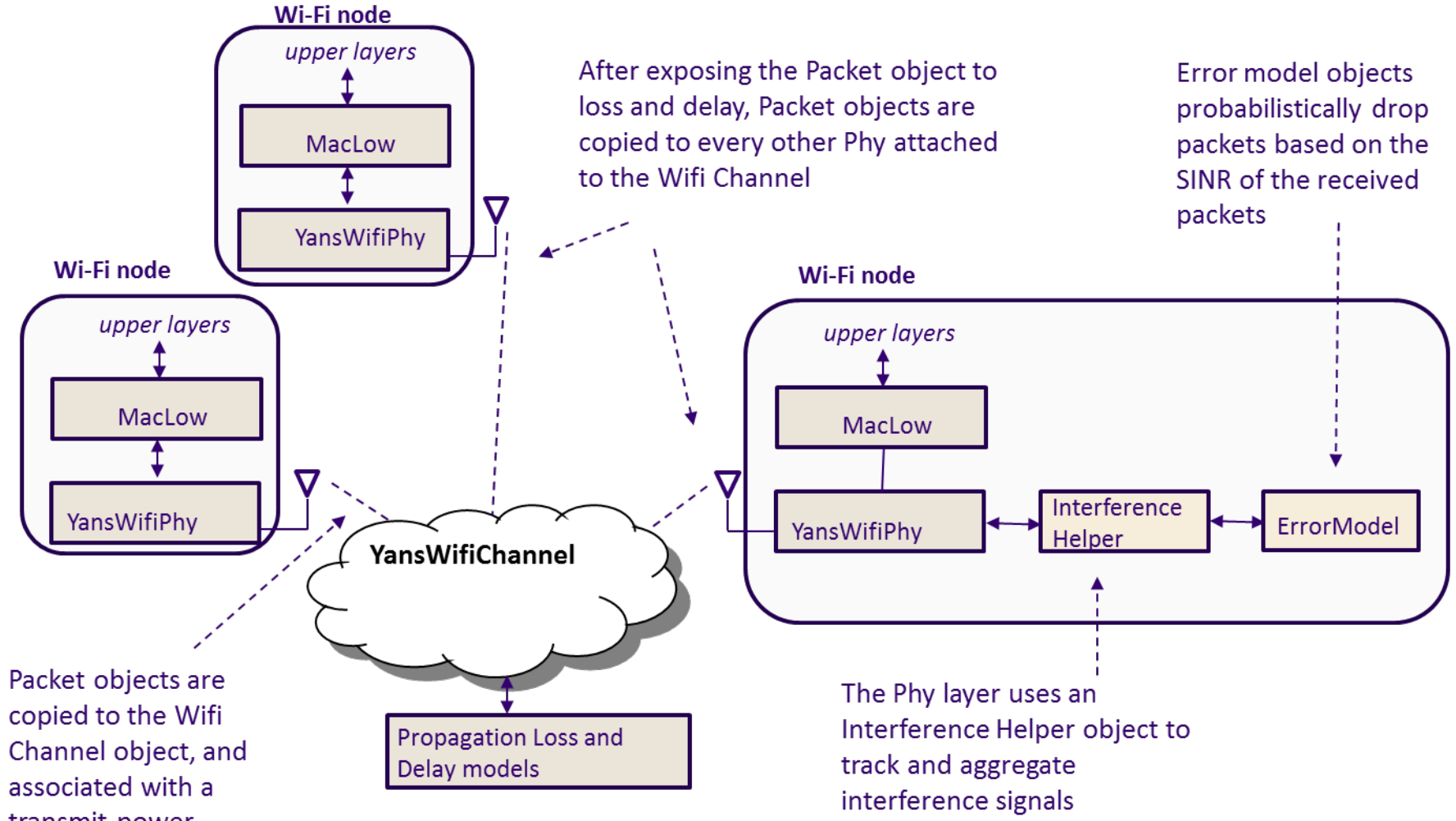
Outline

- Wi-Fi in detail
  - Overview
  - MAC and PHY architecture
  - Yans and Spectrum models
  - Mobility models
  - Propagation models
  - Energy Model
- Configurations via helpers
- Current activities and future developments

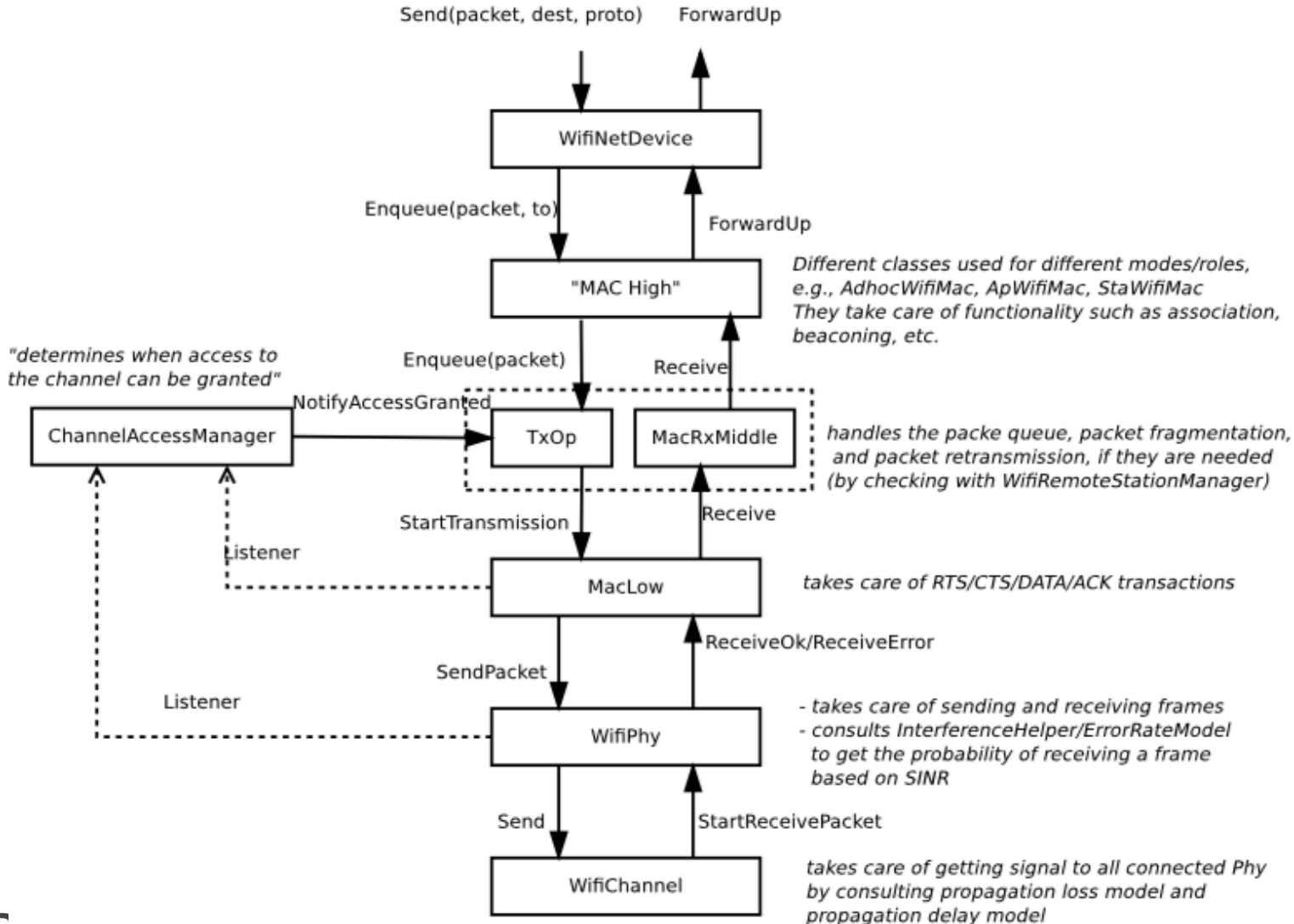
Wi-Fi Overview

- Wi-Fi module features
  - 802.11 a/b/g/n/ac/ax (2.4 & 5 GHz) PHY
  - DCF implementation (Basic + RTS/CTS)
  - PCF implementation (since ns-3.29!)
  - QoS support (EDCA)
  - MSDU/MPDU aggregation and block ACK
  - Infrastructure and ad-hoc modes
  - Many rate adaptation algorithms
  - AWGN-based error models
- Unsupported features
  - 11ac advanced features (TX beamforming, MU-MIMO, …)
  - Most of 11ax features (OFDMA, spatial reuse, …)
  - HCF/HCCA implementation

Wi-Fi abstraction elements



Wi-Fi architecture



MAC High

- Presently, 3 MAC high models:
  - AdhocWifiMac: simplest one, for adhoc mode.
  - ApWifiMac: access point in infrastructure mode, sends beacons and handles associations requests.
  - StaWifiMac: station in infrastructure mode, keeps associated to the access point.

MAC Middle and MAC Low

- MacTxMiddle/MaxRxMiddle:
  - Attaches sequence numbers;
  - Reassembles fragmented packets ;
  - Discards duplicated frames by verifying the received sequence numbers.
- Txop/QosTxop:
  - Queue data and management packets;
  - Fragmentation/Retransmissions;
  - MSDU aggregation and block ACK sessions;
  - 1 Txop, 4 QosTxop (one per TID).
- ChannelAccessManager: determines when a transmission can start based on listeners (physical and virtual).
- MacLow
  - RTS/CTS/DATA/ACK transactions;
  - MPDU Aggregation and block ACKs

Rate control (1)

- Rate control algorithm is used by the MAC low layer to determine at which rate a DATA (or a RTS) frame should be sent by the PHY.
- All rate managers inherit from WifiRemoteStationManager, which is the interface towards MacLow.
- Supported rate control algorithms:
  - Algorithms found in real devices:
    ArfWifiManager (default for WifiHelper), OnoeWifiManager, ConstantRateWifiManager, MinstrelWifiManager, MinstrelHtWifiManager (for 802.11n/ac).
  - Algorithms found in literature:
    IdealWifiManager, AarfWifiManager, AmrrWifiManager, CaraWifiManager, RraaWifiManager, AarfcdWifiManager, ParfWifiManager, AparfWifiManager.

Rate control (2)

- 802.11n/ac can use Constant, Ideal or MinstrelHT, 802.11ax is currently limited to be used with Constant!
- For management frames, WifiRemoteStationManager does not call the corresponding rate manager, it selects the lowest basic rate.
- For response frames, the specific rate manager is also not called, response rate selection is also handled by WifiRemoteStationManager and follows IEEE 802.11 standard rules.

Physical layer

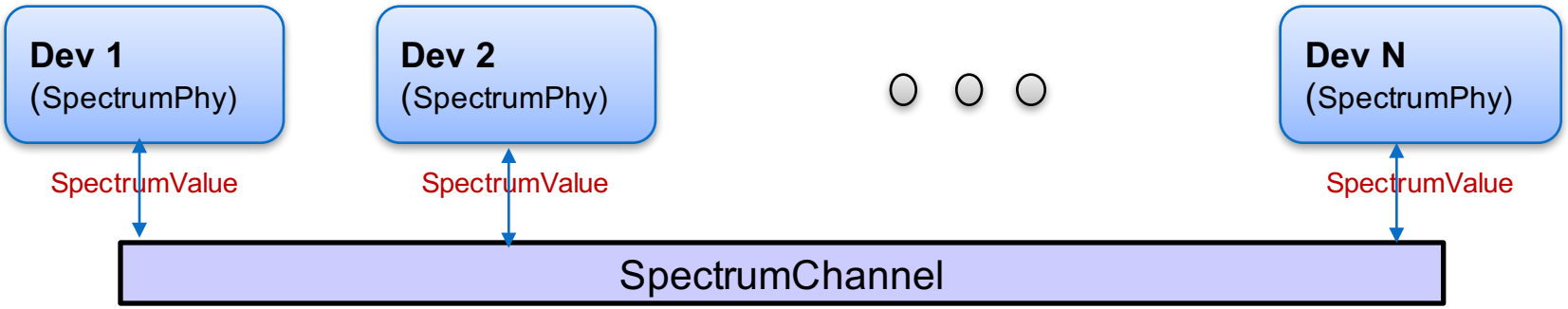
- Responsible for modeling the TX/RX of packets:
  - each packet received is probabilistically evaluated for successful or failed reception (random number versus PER);
  - an object exists to track (InterferenceHelper) all received signals so that the correct interference power for each packet can be computed when a reception decision has to be made;
  - error models corresponding to the modulation and standard are used to look up probability of successful reception
- Capture effect supported: can re-sync on a stronger packet even if already in RX state.
- 2 Wi-Fi PHY models in ns-3: Yans and Spectrum.

YansWifiPhy

- Default and legacy PHY layer implementation.
- Based on the model described in "Yet Another Network Simulator".
- Works together with YansWifiChannel.
- YansWifiChannel holds:
  - Channel loss model:
    Determine RX power at each receiver, based on TX power, nodes positions and the selected propagation loss model.
  - Channel delay model:
    Determine the moment the receive function of a given receiver should be called, based on nodes positions and the selected propagation delay model.

SpectrumWifiPhy (1)

- Spectrum model is made of 3 key pieces:
  - SpectrumValue: the signal abstraction being passed through to the channel and to the receivers. It consists in a vector of sub-bands representing center frequency, bandwidth, and sub-band power specral density.
  - SpectrumChannel: delays/attenuates/modifies the transmitted signal as specified before providing copies to all receivers. It automatically converts signals with different resolutions.
  - SpectrumPhy: Inheriting from SpectrumPhy allows different types of devices to interact through the wireless channel.



SpectrumWifiPhy (2)

- SpectrumWifiPhy class reuses the existing interference manager and error rate models originally built for YansWifiPhy.
- The spectrum is sub-divided into sub-bands (the width of an OFDM subcarrier, depends on the technology).
- The power allocated to a particular channel is spread across the sub-bands.
- OFDM transmit spectrum masks model adjacent channels.
- WifiSpectrumPhyInterface act as a shim between the SpectrumWifiPhy and the Spectrum channel (avoid multiple inheritance).

YansWifiPhy versus SpectrumWifiPhy

	YansWifiPhy	SpectrumWifiPhy
Signals	Wi-Fi only	Permits to include multiple technologies coexisting on the same channel
Frequency-level decomposition	No	Yes
Simulation time	Fast	Slow

Which one to choose?

- Simulations involving mixed technologies (Wi-Fi, LTE, …) => **Spectrum**
- Simulation involving frequency dependent effects => **Spectrum**
- Simulation involving other Wi-Fi networks working in adjacent channels or using different channel bonding configurations => **Spectrum**
- Otherwise => **Yans (faster) or Spectrum**

Propagation Models (1)

- Propagation module defines:
  - Propagation delay models:
    Calculate the time for signals to travel from the TX antennas to RX antennas.
  - Propagation loss models:
    Calculate the Rx signal power considering the Tx signal power and the mutual Rx and Tx antennas positions.

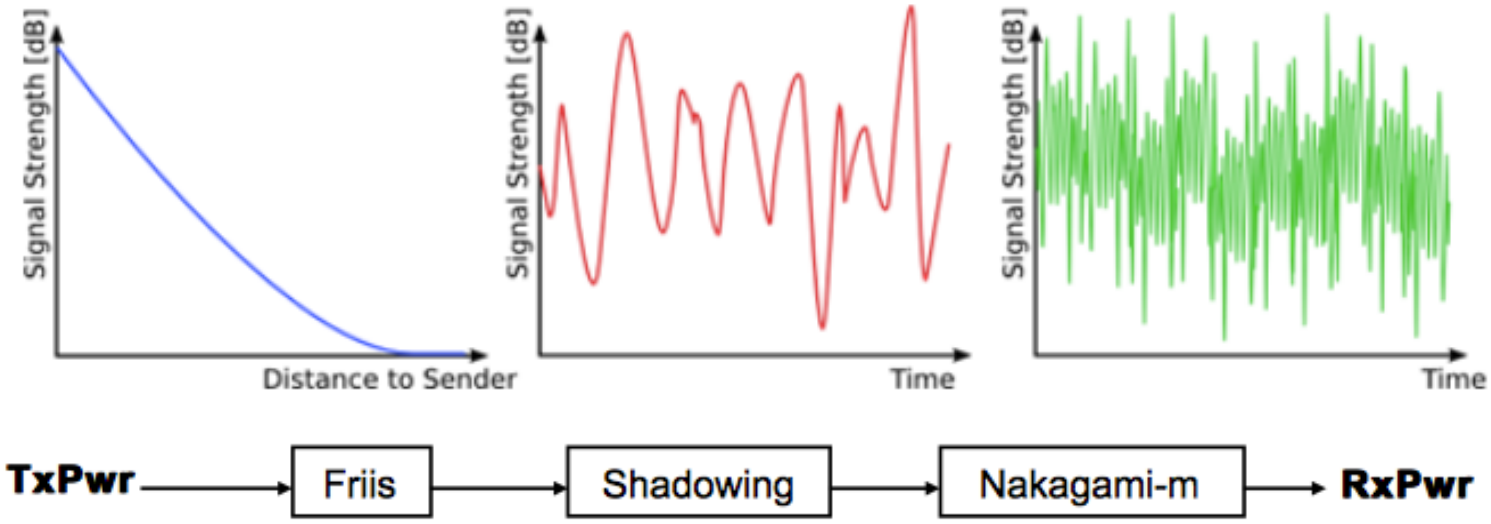
- Propagation delay models:
  - ConstantSpeedPropagationDelayModel: In this model, the signal travels with constant speed. The delay is calculated according with the transmitter and receiver positions.
  - RandomPropagationDelayModel: The propagation delay is totally random, and it changes each time the model is called. All the packets experience a random delay. As a consequence, the packets order is not preserved.

Propagation Models (2)

- Propagation loss models:
  - Many propagation loss models are implemented:
    - ✓ Abstract propagation loss models:
      FixedRss, Range, Random, Matrix, …
    - ✓ Deterministic path loss models:
      Friis, LogDistance, ThreeLogDistance, TwoRayGround, …
    - ✓ Stochastic fading models:
      Nakagami, Jakes, …
  - Extension for Spectrum:
    Some of the models have been extended for Spectrum in order to provide a different RX power per sub-band.

Propagation Models (3)

– A propagation loss model can be “chained” to another one, making a list. The final Rx power takes into account all the chained models.
Example: path loss model + shadowing model + fading model



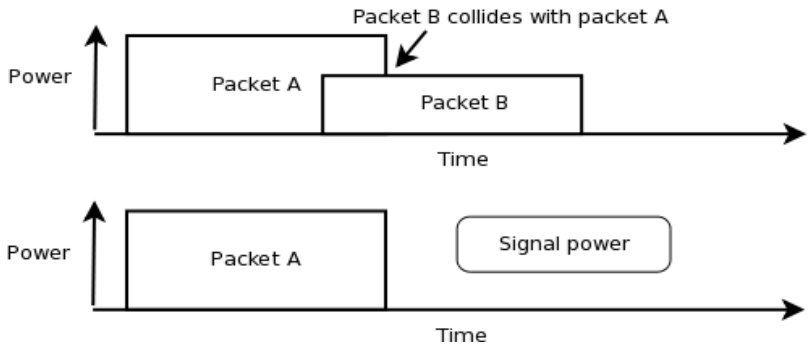
(!) Be careful when using YansWifiChannelHelper::Default(),
the default LogDistance propagation model is added.
Calling AddPropagationLoss() again will add a second propagation loss model.

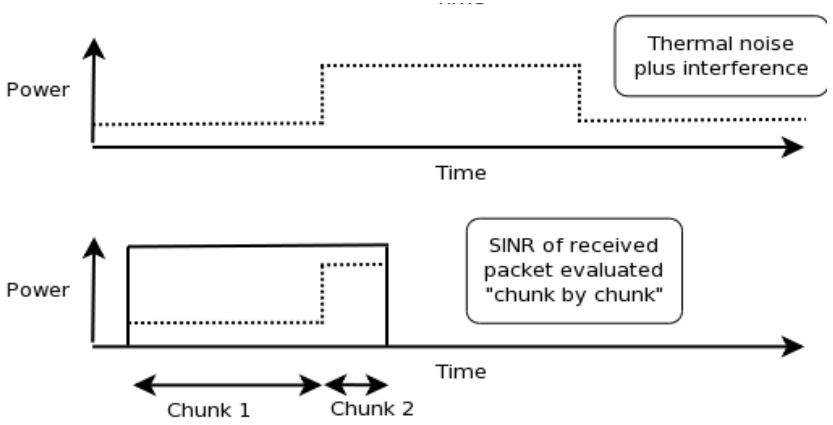
Packet reception

- The packet reception occurs in two stages:
  - PLCP preamble and header (lower modulation rate than the payload);
  - Payload of the packet (generally use a higher modulation).
- If both the preamble/header and payload are successfully received, the packet is passed up to the MacLow object.
- Even if packet objects received by the PHY are not part of the reception process, they are remembered by the InterferenceHelper object for purposes of SINR computation and making clear channel assessment decisions.

Interference helper

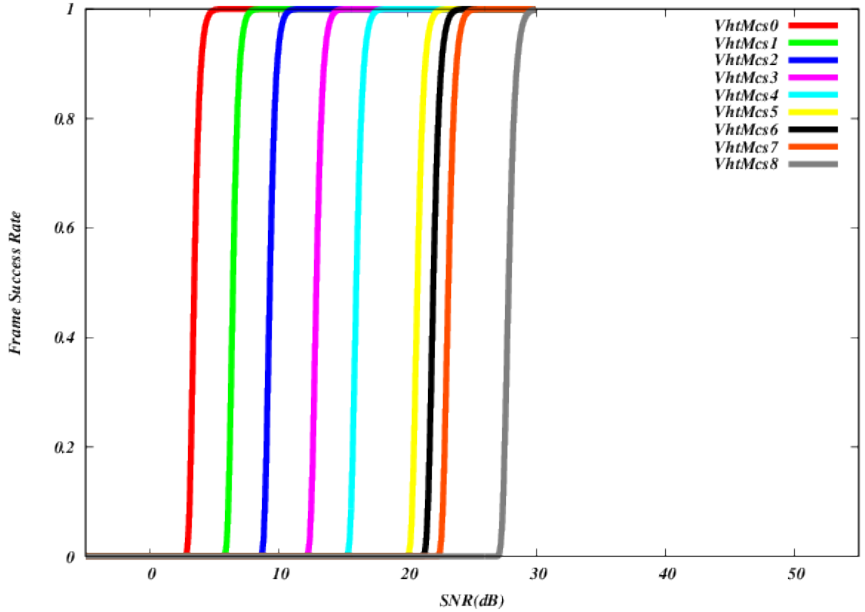
- Object that tracks all incoming packets (power of their interference!)
- Calculates probability of error values for packet being received:
  - SINR is evaluated on chunk-by-chunk basis.
  - Look at its position in the received packet (preamble, header or payload). (!) can have different modulations in a same chunk…
  - Call the error rate model to determine PER of a chunk.
  - PER = product of the PER of each chunk.
- Evaluates whether energy on the channel rises above the CCA threshold.
- Example: Rx packet A (no capture mode)



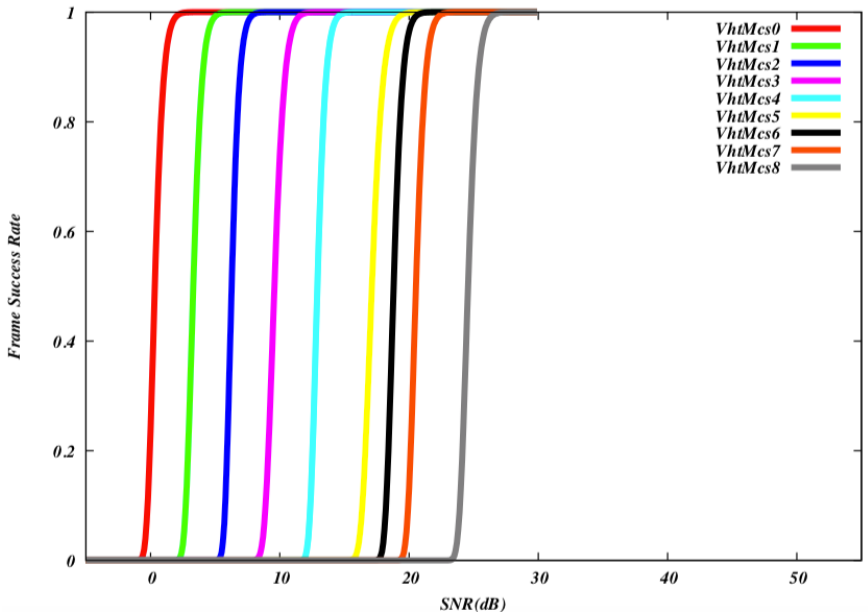


Error models

- Computes a probability of error for a given SINR.
- Based on analytical models with error bounds.
- Supports additive white gaussian noise channels (AWGN) only; any potential fast fading effects are not modeled.
- Three implementations with different bounds: YansErrorRateModel, NistErrorRateModel (default), DsssErrorRateModel (802.11b).



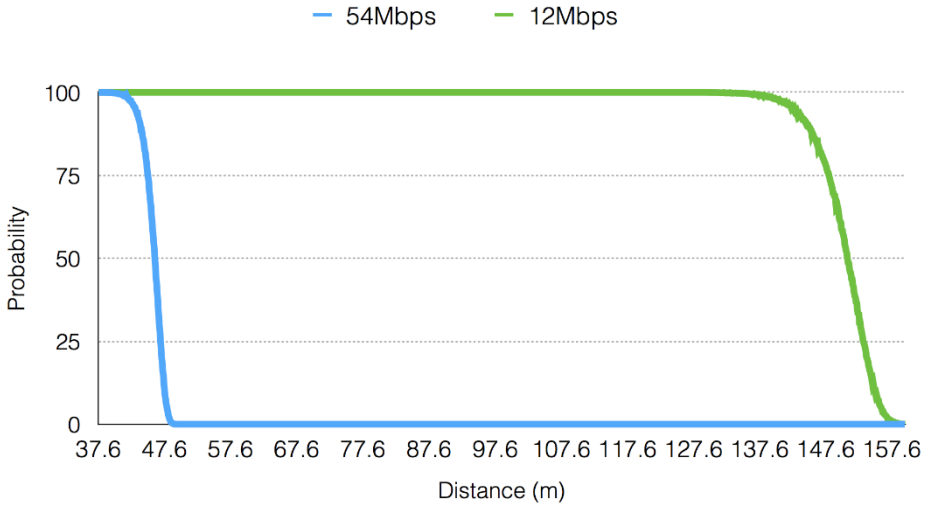
NistErrorRateModel (802.11ac)

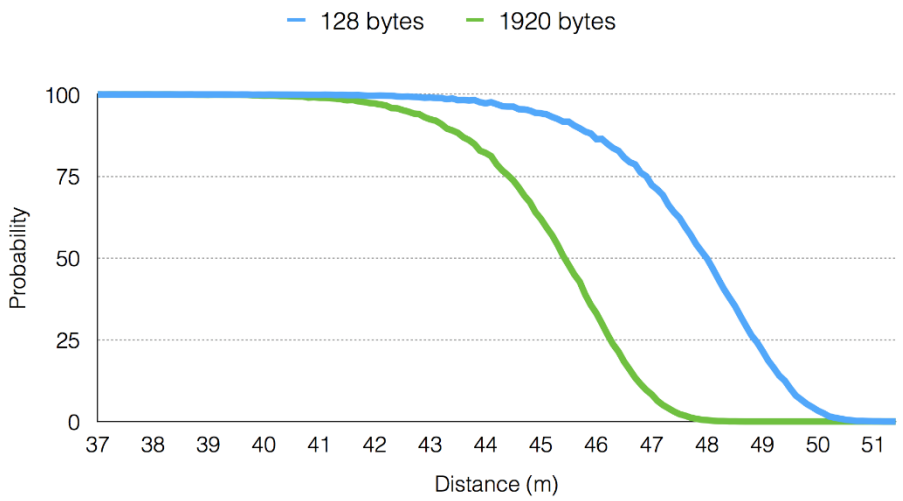


YansErrorRateModel (802.11ac)

Communication Range

- Depends on many factors
  - Propagation loss model
  - PHY configuration (TX power, antennas gains, …)
  - Frame size (big vs small)
  - Transmission mode (6Mbps vs 54 Mbps)



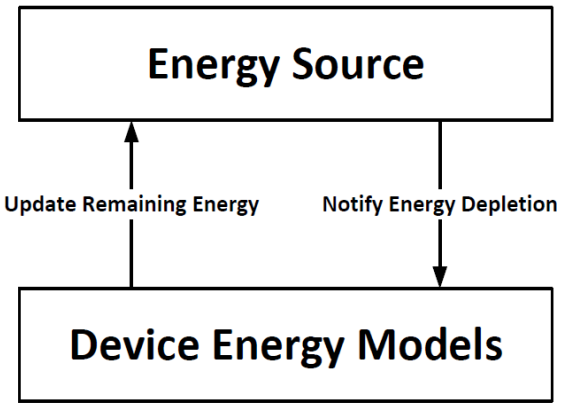


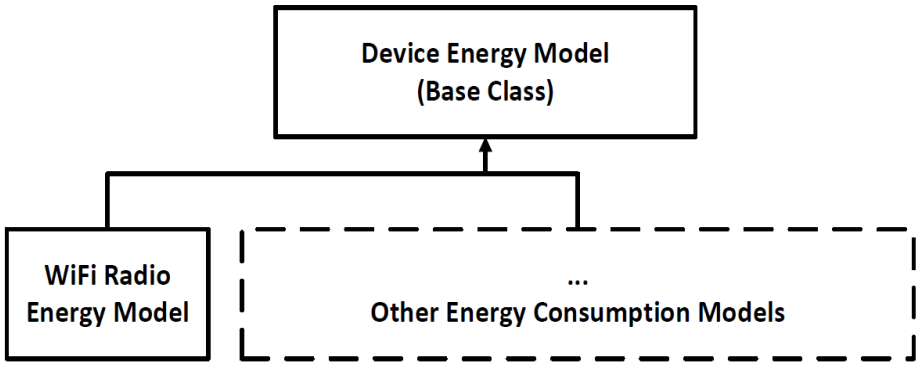
PHY state machine

- WifiPhyStateHelper manages the state machine of the PHY layer.
- It allows other objects to hook as listeners to monitor PHY state (MAC layer, energy module, …).
- The PHY layer can be in one of six states:
  - **TX**: the PHY is currently transmitting a signal on behalf of its associated MAC
  - **RX**: the PHY is synchronized on a signal and is waiting until it has received its last bit to forward it to the MAC.
  - **CCA Busy**: the PHY is not in TX or RX state but the measured energy is higher than the energy detection threshold.
  - **IDLE**: the PHY is not in the TX, RX, or CCA BUSY states.
  - **SWITCHING**: the PHY is switching channels.
  - **SLEEP**: the PHY is in a power save mode and cannot send nor receive frames.
  - **OFF** (since ns-3.28): the PHY is switched off and cannot send nor receive frames.

Wifi Radio Energy Model (1)

- Wi-Fi devices have different energy consumption based on state => current per state configurable via WifiRadioEnergyModelHelper.
- Wi-Fi handles energy depletion and recharge:
  - Energy depletion callback will move wifi state to OFF;
  - Energy recharged callback move wifi state to IDLE and restart channel access manager;
  - When state change, compute time the device can stay ON and re-schedule OFF state.
- Model architecture:





Wifi Radio Energy Model (2)

- Usage:

```
/* energy source */
BasicEnergySourceHelper basicSourceHelper;
// configure energy source
basicSourceHelper.Set ("BasicEnergySourceInitialEnergyJ", DoubleValue
(0.1));
// install source
EnergySourceContainer sources = basicSourceHelper.Install (c);
/* device energy model */
WifiRadioEnergyModelHelper radioEnergyHelper;
// configure radio energy model
radioEnergyHelper.Set ("TxCurrentA", DoubleValue (0.0174));
// install device model
DeviceEnergyModelContainer deviceModels;
deviceModels = radioEnergyHelper.Install (devices, sources);
```

How to extend?

- Create/modify the Wi-Fi frames/headers by making changes to wifi-mac-header.*
- Create/modify rate control algorithms can be done by creating a new child class of Wi-Fi remote station manager or modifying the existing ones.
- Handle new/modified management frames by adding changes to regular-wifi-mac.*, infrastructure-wifi-mac.*, sta-wifi-mac.*, ap-wifi-mac.*, or adhoc-wifi-mac.*
- Handle new/modified control frames by making changes to mac-low.*
- Add information element to management frames by adding a new class and making use of it in mgt-headers.*
- …

MobilityHelper: mobility and position

- The MobilityHelper combines a mobility model and position allocator.
- Position Allocators setup initial position of nodes (only used when simulation starts):
  - List: allocate positions from a deterministic list specified by the user;
  - Grid: allocate positions on a rectangular 2D grid (row first or column first);
  - Random position allocators: allocate random positions within a selected form (rectangle, circle, …).
- Mobility models specify how nodes will move during the simulation:
  - Constant: position, velocity or acceleration;
  - Waypoint: specify the location for a given time (time-position pairs);
  - Trace-file based: parse files and convert into ns-3 mobility events, support mobility tools such as SUMO, BonnMotion (using NS2 format) , TraNS, …

Position Allocation Examples

- List

```
MobilityHelper mobility;
// place two nodes at specific positions (100,0) and (0,100)
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
positionAlloc->Add (Vector (100, 0, 0));
positionAlloc->Add (Vector (0, 100, 0));
mobility.SetPositionAllocator(positionAlloc);
```

- Grid Position

```
MobilityHelper mobility;
// setup the grid itself: nodes are laid out started from (-100,-100) with 20 per row, the x
// interval between each object is 5 meters and the y interval between each object is 20 meters
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (-100.0), "MinY", DoubleValue (-100.0),
                               "DeltaX", DoubleValue (5.0), "DeltaY", DoubleValue (20.0),
                               "GridWidth", UintegerValue (20), "LayoutType", StringValue ("RowFirst"));
```

- Random Rectangle Position

```
// place nodes uniformly on a straight line from (0, 100)
MobilityHelper mobility;
Ptr<RandomRectanglePositionAllocator> positionAloc = CreateObject<RandomRectanglePositionAllocator>();
positionAloc->SetAttribute("X", StringValue("ns3::UniformRandomVariable[Min=0.0|Max=100.0]"));
positionAloc->SetAttribute("Y", StringValue("ns3::ConstantRandomVariable[Constant=50.0]"));
mobility.SetPositionAllocator(positionAloc);
```

Mobility Model Example

- Constant Position

```
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);
```

- Constant Speed

```
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
mobility.Install (nodes);
Ptr<UniformRandomVariable> rvar = CreateObject<UniformRandomVariable>();
for (NodeContainer::Iterator i = nodes.Begin (); i != nodes.End (); ++i) {
  Ptr<Node> node = (*i);
  double speed = rvar->GetValue(15, 25); //random speed between 15 and 25 m/s
  node->GetObject<ConstantVelocityMobilityModel>()->SetVelocity(Vector(speed,0,0));
}
```

- Trace-file based

```
// Create Ns2MobilityHelper with the specified trace log file as parameter
Ns2MobilityHelper ns2 = Ns2MobilityHelper (“mobility_trace.txt”);
ns2.Install (); // configure movements for each node, while reading trace file
```

Wi-Fi configuration via helpers (1)

- Wi-Fi helpers are available for users to create Wi-Fi devices and channels with only a few lines of code.

- To create a Wi-Fi network, users need to follow these steps:
  - Decide if SpectrumWifiPhy or YansWifiPhy.
  - Configure the wireless Channel: propagation loss model and propagation delay model.
  - Configure PHY: error rate model and other PHY attributes (channel width, guard intervals, …) can be set.
  - Configure MAC: architecture (ad-hoc, infrastructure, station or access point), device level (QoS, HT, VHT, HE, …) and select rate control algorithm.
  - Create devices: install MAC and PHY on nodes.
  - Configure mobility: specify initial locations and how nodes will move during simulation.

Wi-Fi configuration via helpers (2)

- Different ways to configure some parameters (e.g. channel number):
  - Via global configuration default:
    ```
    Config::SetDefault ("ns3::WifiPhy::ChannelNumber", UintegerValue (3));
    ```
  - Via the helper:
    ```
    wifiPhyHelper.Set ("ChannelNumber", UintegerValue (3));
    ```
  - By selecting the standard;
  - By performing post-installation configuration:
    ```
    Config::Set("/NodeList/0/DeviceList/*/$ns3::WifiNetDevice/Phy/$ns3::WifiPhy/ChannelNumber", Uinteger (3));
    ```
- But be carefull:
  - Some parameters are NOT INDEPENDANT:
    frequency, channel number, channel width and standard.
  - The SetStandard method set default parameters (mainly timing parameters) as defined in the selected standard, OVERWRITING values that may have been configured. In order to change those parameters, one should use post-installation method.

  => Always make sure you correctly set your settings!

Typical configuration

```
// Create 1 access point and 2 stations
NodeContainer ap, stas;
ap.Create (1);
stas.Create (2);

// Select 802.11b standard (802.11a is default)
WifiHelper wifi; //used to install 802.11 PHY and MAC on the nodes
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

// Setup PHY layer (YANS with default NIST error rate model)
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
// If one wants to use YANS:
//wifiPhy.SetErrorRateModel ("ns3::YansErrorRateModel");
// ns-3 supports RadioTap and Prism tracing extensions for 802.11
wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

YansWifiChannelHelper wifiChannel; //no call to default here, since we set
propagation delay and propagation loss models ourselves

// reference loss must be changed since 802.11b is operating at 2.4GHz!
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel",
                                "ReferenceLoss", DoubleValue (40.0459));

wifiPhy.SetChannel (wifiChannel.Create ());
```

Typical configuration (cont.)

```
// Setup a non-QoS MAC layer, with rate control disabled (11 Mbit/s
constant rate for data frames, 1 Mbit/s constant rate for RTS frames)
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                              "DataMode", StringValue (“DsssRate11Mbps”),
                              "ControlMode",StringValue (“DsssRate1Mbps”));

// Setup access point
WifiMacHelper wifiMac;
wifiMac.SetType ("ns3::ApWifiMac",
                 "Ssid", SsidValue (“wifi-ns-3”));
NetDeviceContainer apDevice = wifi.Install (wifiPhy, wifiMac, ap);
NetDeviceContainer devices = apDevice;

// Setup stations
wifiMac.SetType ("ns3::StaWifiMac",
                 "Ssid", SsidValue (“wifi-ns-3”));
NetDeviceContainer staDevice = wifi.Install (wifiPhy, wifiMac, stas);
devices.Add (staDevice);
```

Typical configuration (cont.)

```
// Configure mobility
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc =
CreateObject<ListPositionAllocator> ();
positionAlloc->Add (Vector (0.0, 0.0, 0.0)); //AP
positionAlloc->Add (Vector (5.0, 0.0, 0.0)); //STA 1
positionAlloc->Add (Vector (0.0, 5.0, 0.0)); //STA 2
mobility.SetPositionAllocator (positionAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (ap);
mobility.Install (sta);

// other set up (e.g. Internet stack, Application, …)
```

QoS configuration

- Example programs include:
  - examples/wireless/80211e-txop.cc
  - examples/wireless/wifi-blockack.cc
  - examples/wireless/wifi-multi-tos.cc
- Select QoS or non-Qos for 802.11a/b/g, QoS is always enabled for 802.11n/ac/ax.
- Example with QoS enabled:

```
wifiMac.SetType ("ns3::StaWifiMac",
                 "Ssid", SsidValue (“wifi-ns-3”),
                 ”QosSupported", BooleanValue (true));
NetDeviceContainer staDevice = wifi.Install (wifiPhy, wifiMac, stas);
devices.Add (staDevice);

wifiMac.SetType ("ns3::ApWifiMac",
                 "Ssid", SsidValue (“wifi-ns-3”),
                 ”QosSupported", BooleanValue (true));
NetDeviceContainer apDevice = wifi.Install (wifiPhy, wifiMac, ap);
NetDeviceContainer devices = apDevice;
```

QoS configuration (cont.)

- Example to tune QoS settings per AC (post-installation):

```
// Modify EDCA configuration (TXOP limit) for AC_BE
 Ptr<NetDevice> dev = ap.Get (0)->GetDevice (0);
 Ptr<WifiNetDevice> wifi_dev = DynamicCast<WifiNetDevice> (dev);
 Ptr<WifiMac> wifi_mac = wifi_dev->GetMac ();
 PointerValue ptr;
 Ptr<QosTxop> edca;
 wifi_mac->GetAttribute ("BE_Txop", ptr);
 edca = ptr.Get<QosTxop> ();
 edca->SetAifsn (1);
 edca->SetMinCw (3);
 edca->SetMaxCw (7);
 edca->SetTxopLimit (MicroSeconds (3008));
```

PCF configuration

- Example available in examples/wireless/wifi-pcf.cc
- Only use PCF with non-QoS 802.11a/b/g!
- Example: specify whether a device does support PCF and set parameters:

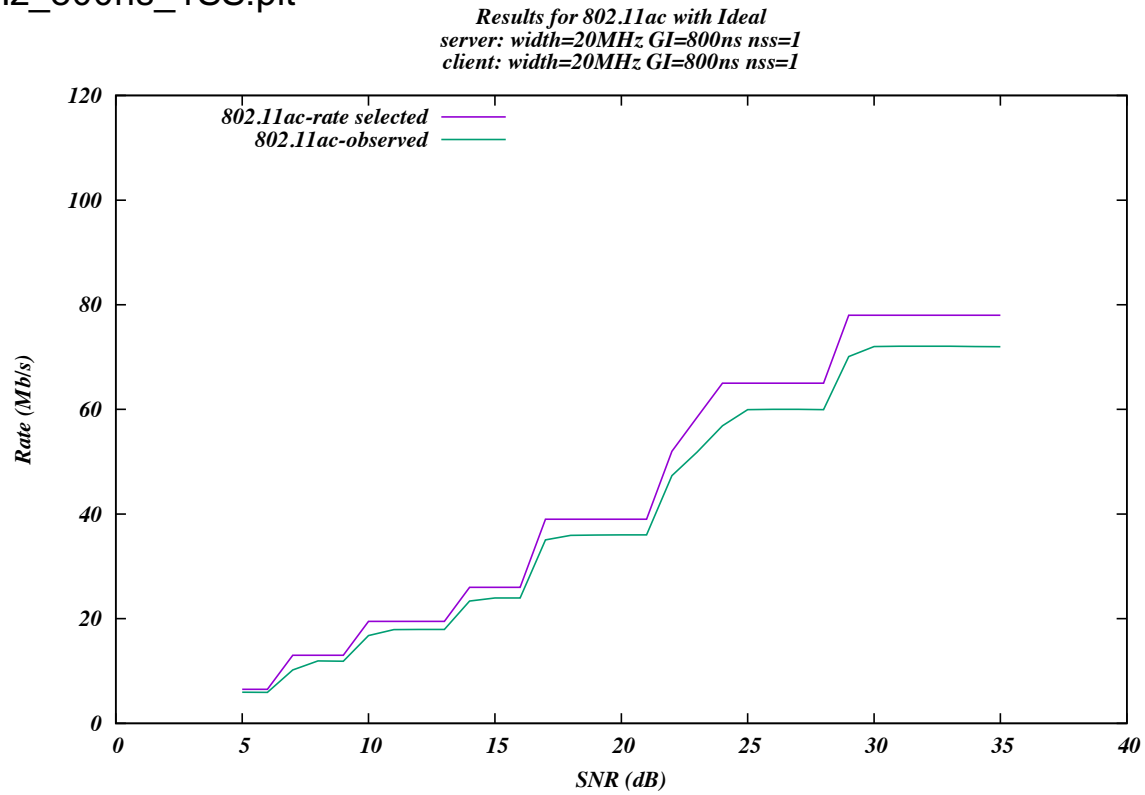
```
…
// Setup stations
wifiMac.SetType ("ns3::StaWifiMac",
                 "Ssid", SsidValue (“wifi-ns-3”),
                 "PcfSupported", BooleanValue (true));
NetDeviceContainer staDevice = wifi.Install (wifiPhy, wifiMac, stas);
devices.Add (staDevice);

// Setup access point
wifiMac.SetType ("ns3::ApWifiMac",
                 "Ssid", SsidValue (“wifi-ns-3”),
                 "PcfSupported", BooleanValue (true),
                 "CfpMaxDuration", TimeValue (MicroSeconds (20480)));
NetDeviceContainer apDevice = wifi.Install (wifiPhy, wifiMac, ap);
NetDeviceContainer devices = apDevice;
…
```

Rate managers example

- Example script to test any rate control algorithms: src/wifi/examples/wifi-manager-example.cc

$ ./waf --run "wifi-manager-example --wifiManager=Ideal --standard=802.11ac --serverChannelWidth=20 --clientChannelWidth=20 --serverShortGuardInterval=800 --clientShortGuardInterval=800 --serverNss=1 --clientNss=1”

$ gnuplot wifi-manager-example-Ideal-802.11ac-server=20MHz_800ns_1SS-client=20MHz_800ns_1SS.plt



Wi-Fi timing settings

- Unaware people tend to configure Wi-Fi timing values (slot, timeouts, …) at the begin of the simulation script…
- But those values gets overwritten at the moment the call the Install method of the helper (this will call Configure80211X).
- The solution is to set those timing attributes AFTER the Install method is called:

```
NetDeviceContainer apDevice;
apDevice = wifi.Install (phy, mac, ap);
Config::Set
("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/Slot",
TimeValue (MicroSeconds (18)));
```

- Full example available in examples/wireless/wifi-timing-attributes.cc

Configuring 802.11n/ac (1)

- Example programs include
  - examples/wireless/ht-wifi-network.cc
  - examples/wireless/vht-wifi-network.cc
  - examples/wireless/wifi-aggregation.cc
- Setting the WifiPhyStandard will set most defaults reasonably

```
WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211ac);
```

- 802.11ac/ax uses 80 MHz channel by default; 802.11n uses 20 MHz

Configuring 802.11n/ac (2)

- Maximum supported channel width and whether short guard interval is enabled can be modified through wifi PHY helpers (yans and spectrum):

```
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

// Select channel width of 40 MHz
phy.Set (”ChannelWidth", UintegerValue (40));

// Enable guard interval
phy.Set ("ShortGuardEnabled", BooleanValue (true));
```

Configuring aggregation

- Supported aggregation schemes (802.11n/ac):
  - A-MSDU
  - A-MPDU
  - Mixed/hybrid (A-MSDU together with A-MPDU)
- Example: configure aggregation for AC_BE queue

```
…
WifiMacHelper mac;
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "BE_MaxAmpduSize",UintegerValue(32768),
             "BE_MaxAmsduSize",UintegerValue (3839));
…
```

MIMO (1)

- For 802.11n/ac/ax devices, it is possible to configure MIMO.
- Up to 4 SDM streams are supported.
- Relevant parameters (WifiPhy):
  - Antennas: specify the number of physical antennas/chains;
  - MaxSupportedTxSpatialStreams: Tell the maximum number of spatial streams that can be transmitted by the device;
  - MaxSupportedRxSpatialStreams: Tell the maximum number of spatial streams that can be received by the device.
- Obviously, MaxSupportedTxSpatialStreams and MaxSupportedTxSpatialStreams have to be smaller or equal to Antennas.

MIMO (2)

- Different configurations possible (case of 2 spatial streams):
  - Symmetrical: 2 x 2 : 2
  - Asymmetrical with more antennas at TX side: 3 x 2 : 2
  - Asymmetrical with more antennas at RX side: 2 x 3 : 2
  - Limit number of streams: 3 x 3 : 2
- If more antennas than the number of streams, it will make your wireless link more robust (diversity, …).
- A rough approximation is done as a 3dB SNR gain per additional antenna/chain (according to some measurements at UW):
  - 2 x 2 : 2 => 3 x 2 : 2 = + 3dB
  - 2 x 2 : 2 => 3 x 3 : 2 = + 6dB
  - …

MIMO (3)

- Usage:

```
…
YansWifiChannelHelper channel =
YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

//Configure MIMO
phy.Set ("Antennas", UintegerValue (3));
phy.Set ("MaxSupportedTxSpatialStreams",
         UintegerValue (2));
phy.Set ("MaxSupportedRxSpatialStreams",
         UintegerValue (3));
…
```

802.11ax example

- PHY has been changed in 802.11ax: longer symbols, different carrier spacing, 3 possible guard intervals, …

  => New parameter for 802.11ax guard interval: GuardInterval
  (! different than ShortGuardEnabled used for 802.11n/ac !)

- 80211ax example:

```
YansWifiChannelHelper ch = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (ch.Create ());

// Set HE guard interval to use 1600 ns GI
phy.Set ("GuardInterval", TimeValue (NanoSeconds (1600)));

WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211ax_5GHZ);

//802.11ax only supports constant rate manager
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
"DataMode", StringValue (HeMcs4), "ControlMode", StringValue (HeMcs0));
```

Multiple access points and inter-channel interference (Spectrum)

```
…

// Create Spectrum channel and PHY
Ptr<MultiModelSpectrumChannel> channel =
CreateObject<MultiModelSpectrumChannel> ();
Ptr<FriisPropagationLossModel> lossModel =
CreateObject<FriisPropagationLossModel> ();
channel->AddPropagationLossModel (lossModel);
Ptr<ConstantSpeedPropagationDelayModel> delayModel =
CreateObject<ConstantSpeedPropagationDelayModel> ();
channel->SetPropagationDelayModel (delayModel);

SpectrumWifiPhyHelper phy = SpectrumWifiPhyHelper::Default ();
phy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
phy.SetChannel (channel);

//Configure 802.11ac
WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211ac);
wifi.SetRemoteStationManager ("ns3::IdealWifiManager");
WifiMacHelper mac;
Ssid ssid;
```

Multiple access points and inter-channel interference (cont.)

```
// Network A: frequency primaryChannelA, transmitted power txPowerA
// Network B: frequency primaryChannelB, transmitted power txPowerB
NetDeviceContainer staDeviceA, staDeviceB, apDeviceA, apDeviceB;

ssid = Ssid ("network-A");
phy.Set ("TxPowerStart", DoubleValue (txPowerA));
phy.Set ("TxPowerEnd", DoubleValue (txPowerA));
phy.Set ("ChannelNumber", UintegerValue (primaryChannelA));
phy.Set ("ChannelWidth", UintegerValue (channelWidthA));
mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid));
staDeviceA = wifi.Install (phy, mac, wifiStaNodes.Get (0));
mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));
apDeviceA = wifi.Install (phy, mac, wifiApNodes.Get (0));

ssid = Ssid ("network-B");
phy.Set ("TxPowerStart", DoubleValue (txPowerB));
phy.Set ("TxPowerEnd", DoubleValue (txPowerB));
phy.Set ("ChannelNumber", UintegerValue (primaryChannelB));
phy.Set ("ChannelWidth", UintegerValue (channelWidthB));
mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid));
staDeviceB = wifi.Install (phy, mac, wifiStaNodes.Get (1));
mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));
apDeviceB = wifi.Install (phy, mac, wifiApNodes.Get (1));
```

Current Wi-Fi developments

- 802.11ad: review code of Hany Assasa, merge with ns-3-dev and split in separate wigig module (work in progress by Sébastien).
- 802.11ax:
  - A team at UW is working on spatial reuse features and the PHY abstraction.
  - Rediet and colleages are working on modeling OFDMA.
- 802.11n/ac: PHY layer abstraction models based on Matlab link simulator (work in progress at University of Washington).

Roadmap and future work (1)

- 802.11ah: Once wigig work is done, Sébastien will start to review code of Le Tian, merge with ns-3-dev and split in separate sub1ghz module.
- 802.11ax: UW team plans to work on multi-user aspects in the near future.
- 802.11ac: Some work should be planned to support TX beamforming and MU-MIMO.
- HCF/HCCA: Sébastien will extend his PCF implemention to support HCF/HCCA.

Q&A