Session 6. Wireless LAN in ns-3

곽철영
Multimedia & Wireless Networking Laboratory, SNU
cykwak@mwnl.snu.ac.kr

Contents

- IEEE 802.11 Wireless LAN
- WLAN Basic Operation in ns-3
- Frame Aggregation (A-MPDU)

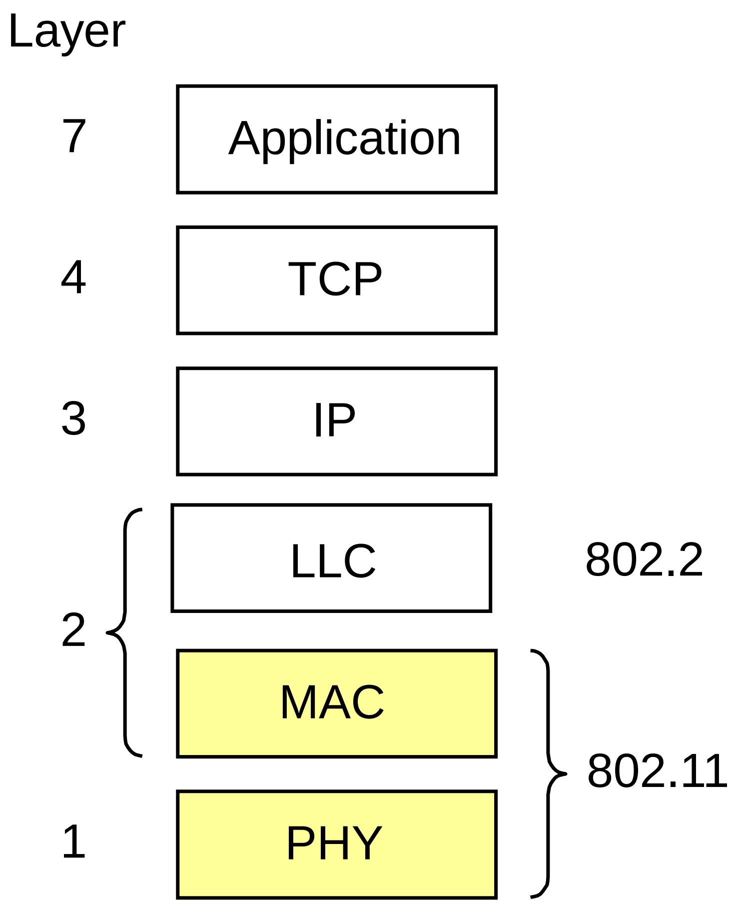
IEEE 802.11 Wireless LAN

IEEE 802.11 Wireless LAN

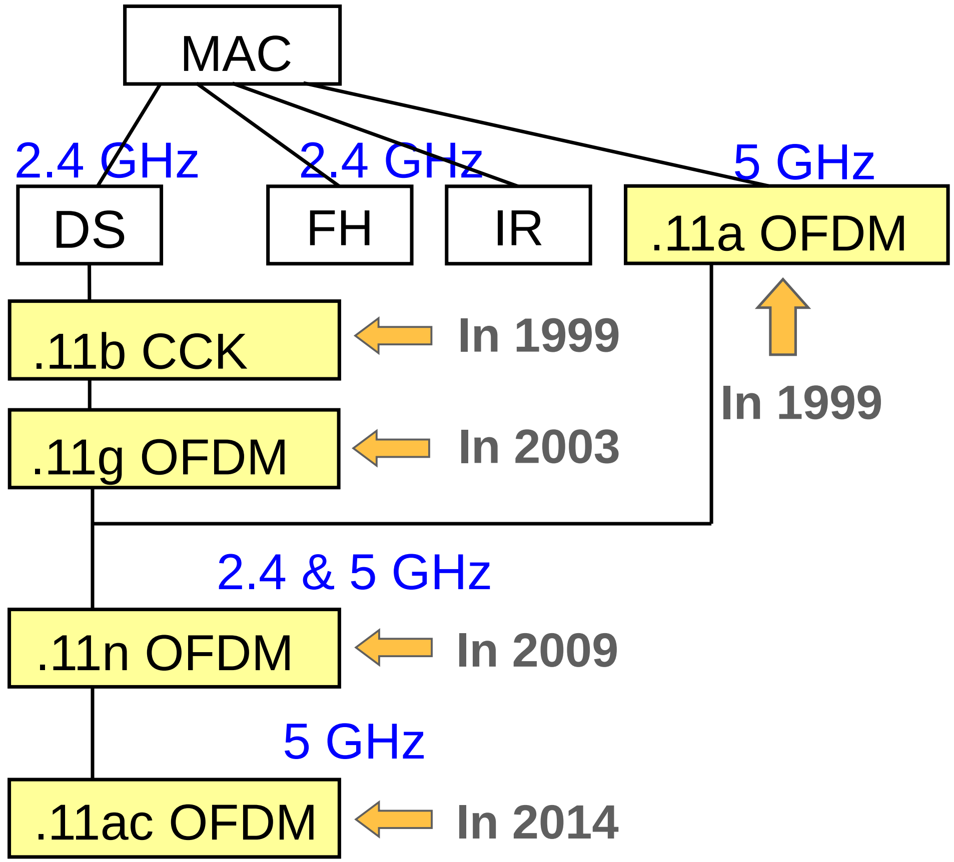
- Popular for enterprise, home, “hot-spot”
- Enables (indoor) wireless and mobile high-speed networking
- “Wireless Ethernet” with comparable speed
  - Supports up to 600 Mbps within ~100 m range
- Runs at unlicensed bands at 2.4 GHz and 5 GHz
  - Emerging extensions operate at 50~700 MHz (TV white space), 900 MHz, 3.65 GHz (Satellite band), 60 GHz, etc.
- MAC and PHY layer technology

IEEE 802.11 Standard Overview

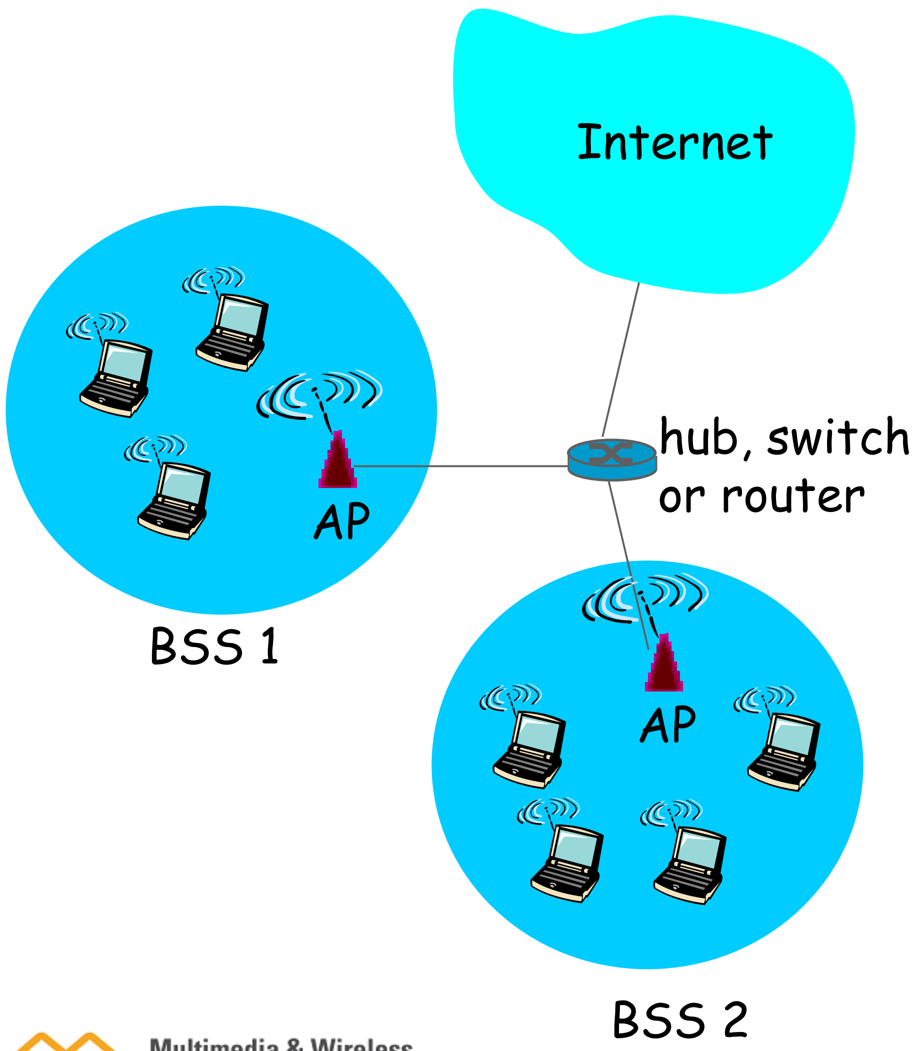
- Layers 1 and 2



- One MAC and multiple PHYs



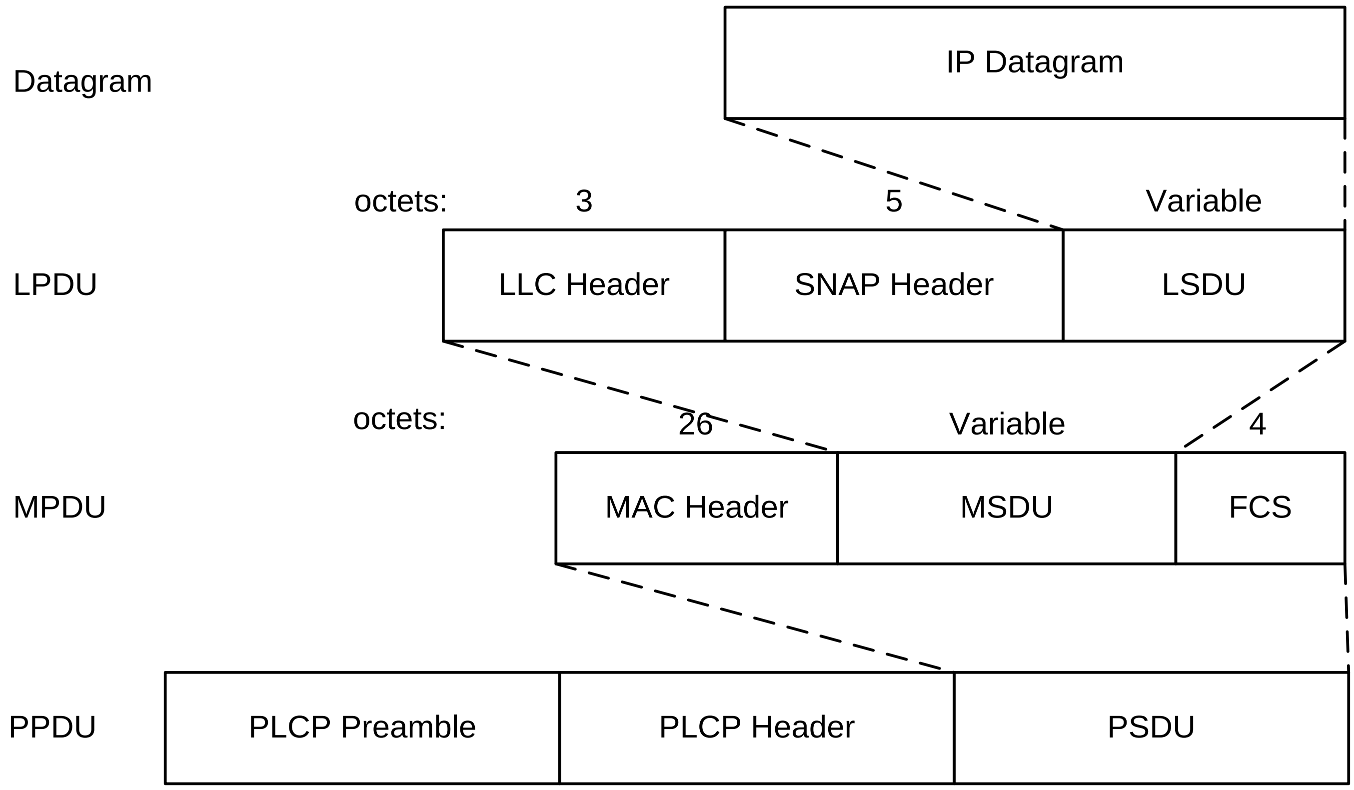
IEEE 802.11 Architecture



- Infrastructure mode
  - Infrastructure Basic Service Set → BSS
  - An access point (AP) and multiple stations (STAs)
  - Every transmission is with AP; no peer-to-peer communication
- Ad hoc mode
  - Independent Basic Service Set → IBSS
  - Multiple stations (STAs), and no AP
  - Peer-to-peer communication only

Layer Interactions

- Datagram vs. LPDU vs. MPDU (MAC frame) vs. PPDU (PHY frame)



IEEE 802.11 Frame Format

duration of reserved transmission time (RTS/CTS)

frame seq #

Octets:	2	2	6	6	6	2	6	0 - 2312	4
	frame control	duration	address 1	address 2	address 3	seq control	address 4	payload	CRC

Bits:	2	2	4	1	1	1	1	1	1	1	1
	Protocol version	Type	Subtype	To AP	From AP	More frag	Retry	Power mgt	More data	WEP	Rsvd

frame type
(RTS, CTS, ACK, data)

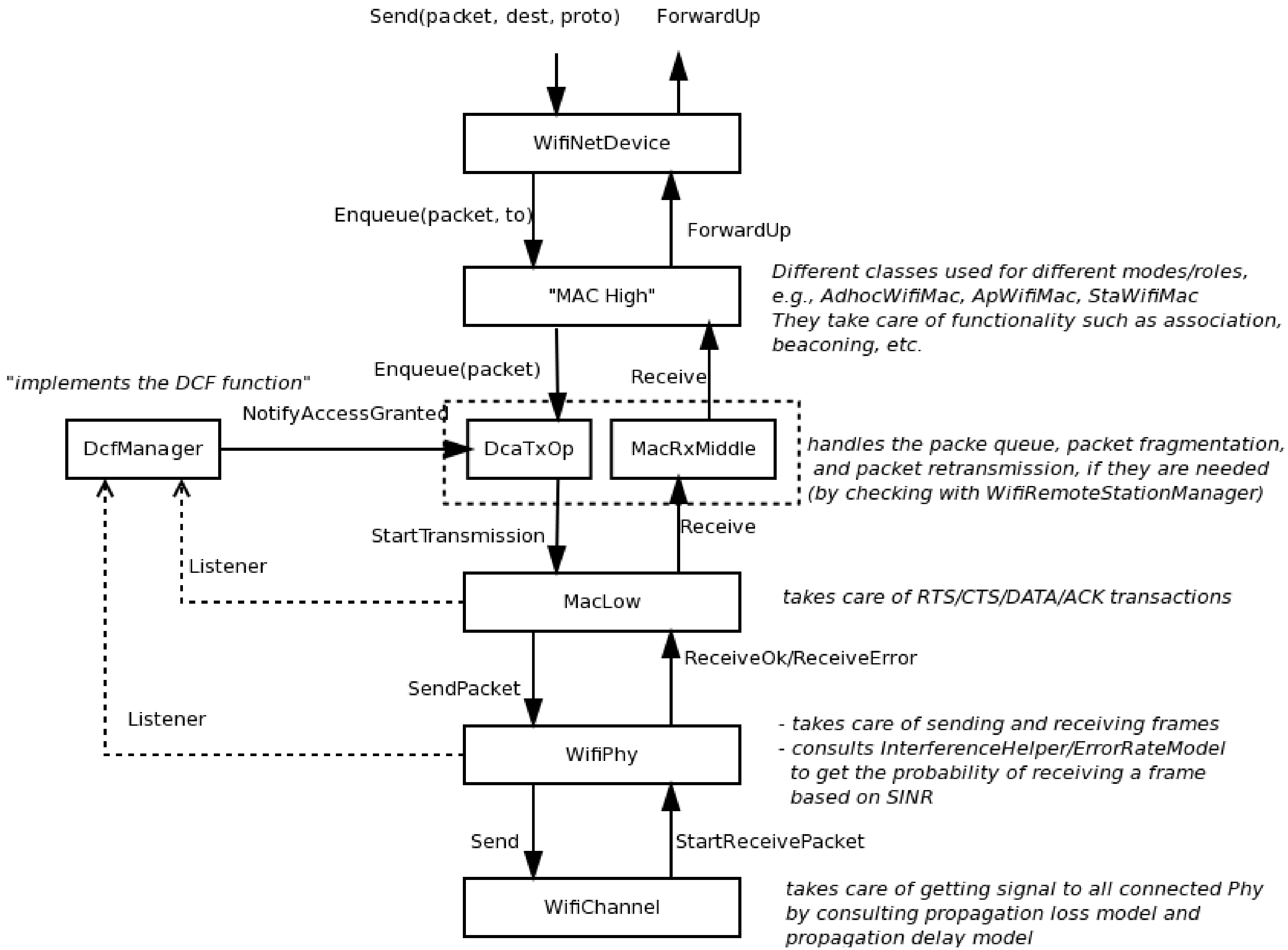
Overview of WiFi Model in ns-3

- ns-3 provides models for 802.11
- WifiNetDevice models a wireless network interface
- ns-3 supports
  - basic 802.11 DCF with infrastructure and ad hoc modes
  - 802.11a/b/g/n/ac physical layers
  - QoS-based EDCA and queueing extensions of 802.11e
  - various propagation loss models including Nakagami, Rayleigh, Friis, LogDistance, FixedRss, Random
  - various rate control algorithms including Aarf, Arf, Cara, Onoe, Rraa, ConstantRate, and Minstrel
  - 802.11s (mesh)

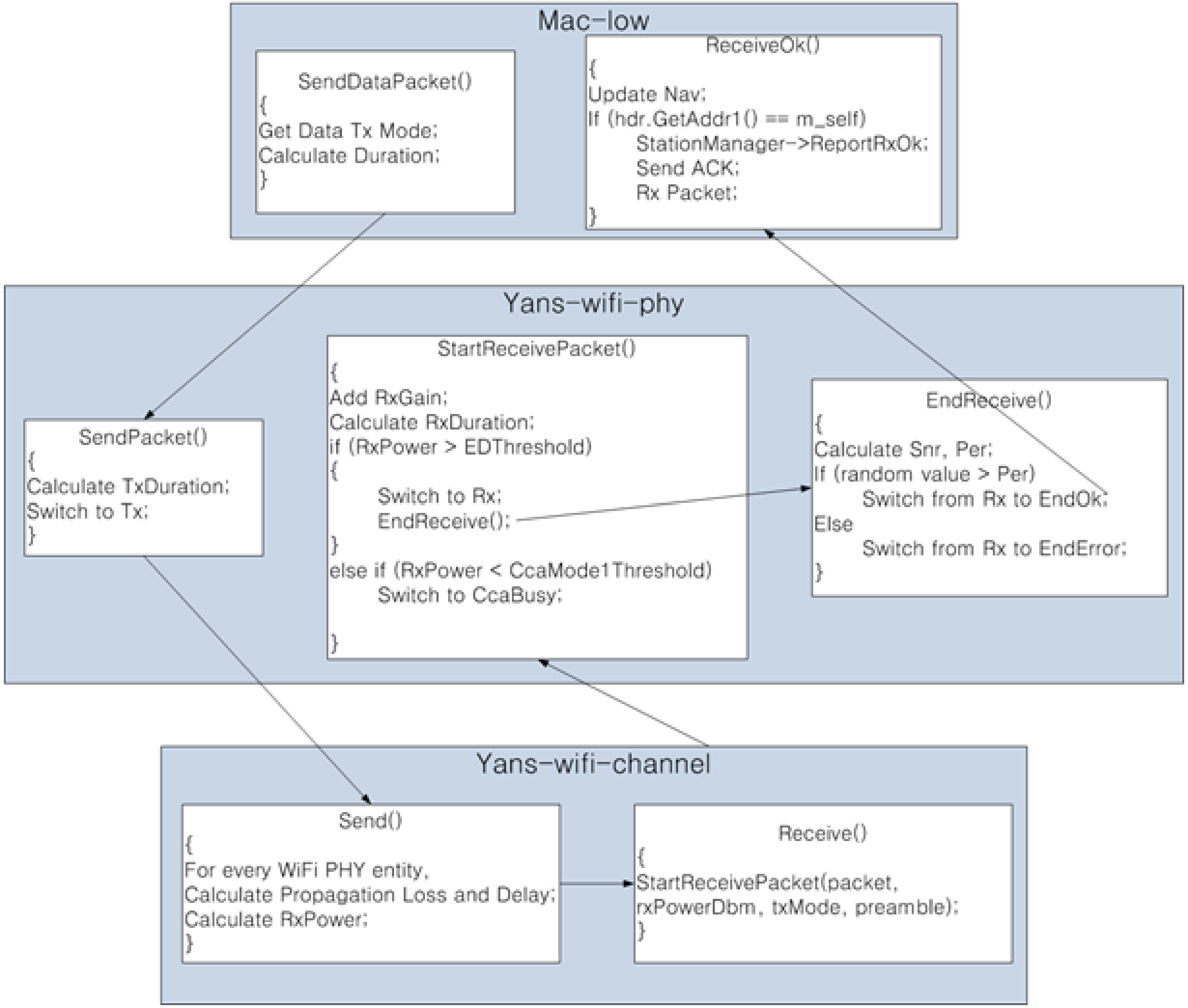
Overview of WiFi Model

- Four levels of models
  - PHY layer model
    - ns3::WifiPhy
  - MAC low models
    - ns3::MacLow
      - Takes care of RTS/CTS/DATA/ACK transmission
    - ns3::DcfManager, ns3::DcfState
      - Implements DCF and EDCAF function
    - ns3::DcaTxop, ns3::EdcaTxopN
      - Handles packet queue, packet fragmentation, retransmission
  - MAC high models
    - ns3::RegularWifiMac
      - ns3::ApWifiMac, ns3::StaWifiMac, ns3::AdhocWifiMac
  - Rate control algorithms
    - ns3::WifiRemoteStationManager

Overview of WiFi Model



WiFi Transmission Flow



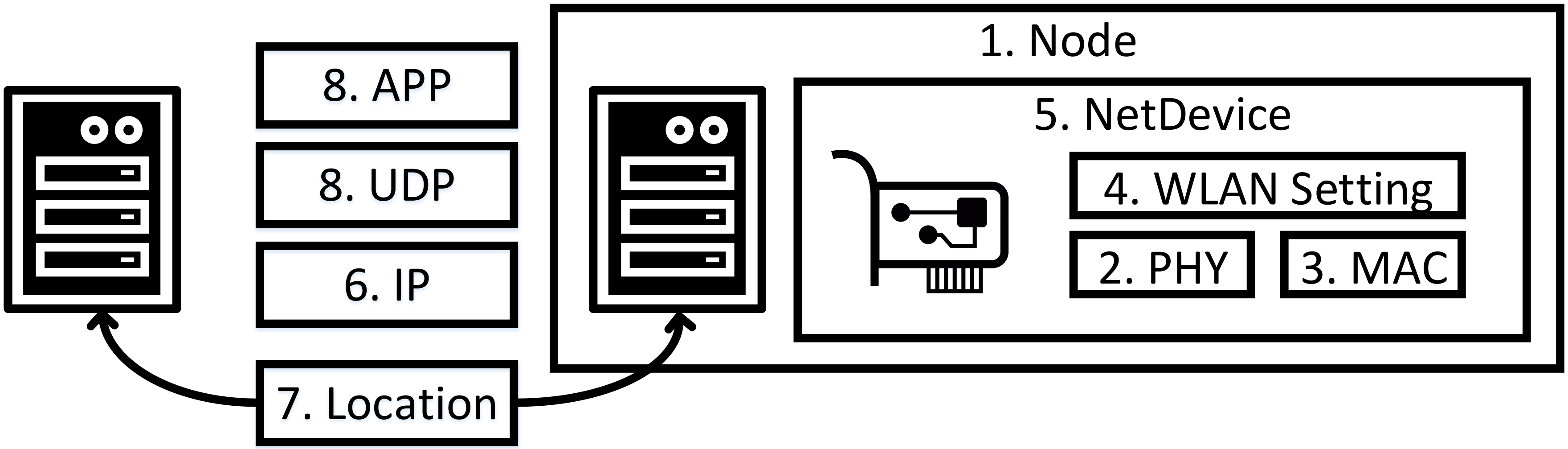
Multimedia & Wireless Networking Laboratory

WLAN Basic Operation in ns-3

Inclass Assignment 1

1. Create **Nodes**: NodeContainer
2. Create **PHY**: YansWifiPhyHelper
3. Create **MAC**: WifiMacHelper
4. **WLAN Setting**: WifiHelper
5. Create **NetDevices**: NetDeviceContainer
6. Create **Network Layer**: InternetStackHelper
7. **Locate Nodes**: MobilityHelper
8. Create **UDP Application**:
   UdpServerHelper, UdpClientHelper
9. Simulation **Run** and **Calc. Throughput**

Inclass Assignment 1



1. Create Nodes

- Create two NodeContainers: wifiStaNode, wifiApNode
  - Let's make 1 STA and 1 AP

ns3::NodeContainer::NodeContainer ( )

Create an empty **NodeContainer**.

Definition at line **26** of file **node-container.cc**.

void ns3::NodeContainer::Create (uint32_t n)

Create n nodes and append pointers to them to the end of this **NodeContainer**.

Nodes are at the heart of any ns-3 simulation. One of the first tasks that any simulation needs to do is to create a number of nodes. This method automates that task.

Parameters

n The number of Nodes to create

Definition at line **93** of file **node-container.cc**.

2. Create PHY Layer

- YansWiFiChannel
  - This channel implements the propagation model described in "Yet Another Network Simulator"
  - ns3::YansWifiChannel class
    - Used in tandem with the ns3::YansWifiPhy class
  - Contain a ns3::PropagationLossModel and a ns3::PropagationDelayModel
    - By default, no propagation models are set so, it is the caller's responsibility to set them before using the channel.

2. Create PHY Layer

- Create channel

YansWifiChannelHelper ns3::YansWifiChannelHelper::Default (void) static

Create a channel helper in a default working state.

By default, we create a channel model with a propagation delay equal to a constant, the speed of light, and a propagation loss based on a log distance model with a reference loss of 46.6777 dB at reference distance of 1m.

Definition at line 41 of file yans-wifi-helper.cc.

Ptr< YansWifiChannel > ns3::YansWifiChannelHelper::Create (void) const

Returns

a new channel

Create a channel based on the configuration parameters set previously.

Definition at line 98 of file yans-wifi-helper.cc.

2. Create PHY Layer

- Create PHY
- Set channel to PHY

YansWifiPhyHelper ns3::YansWifiPhyHelper::Default (void)

Create a phy helper in a default working state.

Returns

a default **YansWifiPhyHelper**

Definition at line 133 of file yans-wifi-helper.cc.

void ns3::YansWifiPhyHelper::SetChannel (**Ptr**< **YansWifiChannel** > channel)

Parameters

channel the channel to associate to this helper

Every PHY created by a call to Install is associated to this channel.

Definition at line 141 of file yans-wifi-helper.cc.

3. Create MAC Layer

- Create MAC
  - WifiMacHelper
- Create SSID
  - SSID
- Set MAC (type="ns3::StaWifiMac")
  - Set attribute "Ssid" to SsidValue(...)
  - Set attribute "ActiveProbing" to BooleanValue (false)

ns3::Ssid::Ssid (std::string s)

Create SSID from a given string.

Parameters

s SSID in string

```
void ns3::WifiMacHelper::SetType ( std::string              type,
                                   std::string              n0 = "",
                                   const AttributeValue &   v0 = EmptyAttributeValue (),
                                   std::string              n1 = "",
                                   const AttributeValue &   v1 = EmptyAttributeValue (),
                                   std::string              n2 = "",
```

4. WLAN Setting

- Set Standard

```
void ns3::WifiHelper::SetStandard ( enum WifiPhyStandard  standard )
```

Parameters

standard the phy standard to configure during installation

WIFI_PHY_STANDARD_80211n_2_4GHZ	HT OFDM PHY for the 2.4 GHz band (clause 20)
WIFI_PHY_STANDARD_80211n_5GHZ	HT OFDM PHY for the 5 GHz band (clause 20)
WIFI_PHY_STANDARD_80211ac	VHT OFDM PHY (clause 22)

4. WLAN Setting

- Set TX rate (type=“ns3::ConstantRateWifiManager”)
  - Set attribute “DataMode” to StringValue (...)
  - Set attribute “ControlMode” to StringValue (...)
  - 11n MCSs: HtMcs7 ~ HtMcs0

```
void ns3::WifiHelper::SetRemoteStationManager ( std::string              type,
                                                std::string              n0 = "",
                                                const AttributeValue &   v0 = EmptyAttributeValue (),
                                                std::string              n1 = "",
                                                const AttributeValue &   v1 = EmptyAttributeValue (),
```

5. Create NetDevices

- Create NetDevice for STA

ns3::NetDeviceContainer::NetDeviceContainer ( )

Create an empty **NetDeviceContainer**.

- Install WLAN (setting, PHY, MAC, wifiStaNode) to STA NetDevice

```
NetDeviceContainer ns3::WifiHelper::Install ( const WifiPhyHelper &  phy,
                                              const WifiMacHelper &  mac,
                                              NodeContainer          c
                                            )                        const
```

Parameters

phy the PHY helper to create PHY objects

mac the MAC helper to create MAC objects

c the set of nodes on which a wifi device must be created

Returns

a device container which contains all the devices created by this method.

5. Create NetDevices

- Change MAC type and setting (type=“ns3::ApWifiMac”)
  - Set attribute “Ssid” to SsidValue(...)
  - Set attribute “BeaconInterval” to TimeValue (MicroSeconds (102400))
  - Set attribute “BeaconGeneration” to BooleanValue (true)

```
void ns3::WifiMacHelper::SetType ( std::string                type,
                                   std::string                n0 = "",
                                   const AttributeValue &     v0 = EmptyAttributeValue (),
                                   std::string                n1 = "",
                                   const AttributeValue &     v1 = EmptyAttributeValue (),
                                   std::string                n2 = "",
```

- Create NetDevice for AP
- Install WLAN (setting, PHY, MAC, wifiApNode) to AP NetDevice

6. Create Network Layer

- Create InternetStack
- Install wifiApNode and wifiStaNode to InternetStack

ns3::InternetStackHelper::InternetStackHelper (void)

Create a new **InternetStackHelper** which uses a mix of static routing and global routing by default.

void ns3::InternetStackHelper::Install (NodeContainer c) const

For each node in the input container, aggregate implementations of the **ns3::Ipv4**, **ns3::Ipv6**, ns3::Udp, and, ns3::Tcp classes.

The program will assert if this method is called on a container with a node that already has an **Ipv4** object aggregated to it.

Parameters

c **NodeContainer** that holds the set of nodes on which to install the new stacks.

6. Create Network Layer

- Create Ipv4 address
- Set network, mask, and base address

```
ns3::Ipv4AddressHelper::Ipv4AddressHelper ( )
```

Construct a helper class to make life easier while doing simple IPv4 address assignment in scripts.

```
void ns3::Ipv4AddressHelper::SetBase ( Ipv4Address  network,
                                       Ipv4Mask     mask,
                                       Ipv4Address  base = "0.0.0.1"
                                     )
```

Set the base network number, network mask and base address.

The address helper allocates IP addresses based on a given network number and mask combination along with an initial IP address.

6. Create Network Layer

- Create Ipv4InterfaceContainer (for STA and AP)

```
ns3::Ipv4InterfaceContainer::Ipv4InterfaceContainer ( )
```

Create an empty **Ipv4InterfaceContainer**.

- Assign Ipv4 address to NetDevices (for STA and AP)

```
Ipv4InterfaceContainer ns3::Ipv4AddressHelper::Assign ( const NetDeviceContainer & c )
```

Assign IP addresses to the net devices specified in the container based on the current network prefix and address base.

7. Locate Nodes

- Create MobilityHelper
- Create pointer of ListPositionAllocator
- Add new Vectors to ListPositionAllocator (for AP and STA)
  - Vector (0.0, 0.0, 0.0) for AP, Vector (1.0, 0.0, 0.0) for STA

ns3::MobilityHelper::MobilityHelper ( )

Construct a Mobility Helper which is used to make life easier when working with mobility models.

ns3::ListPositionAllocator::ListPositionAllocator ( )

Allocate positions from a deterministic list specified by the user.

void ns3::ListPositionAllocator::Add (Vector v)

Add a position to the list of positions.

7. Locate Nodes

- Set ListPositionAllocator to MobilityHelper
- Set type of mobilitymodel to MobilityHelper
  - Set to “ns3::ConstantPositionMobilityModel”

```
void ns3::MobilityHelper::SetPositionAllocator ( Ptr< PositionAllocator >  allocator )
```

Set the position allocator which will be used to allocate the initial position of every node initialized during MobilityModel::Install.

Parameters

allocator allocate initial node positions

```
void ns3::MobilityHelper::SetMobilityModel ( std::string                type,
                                             std::string                n1 = "",
                                             const AttributeValue &     v1 = EmptyAttributeValue (),
                                             std::string                n2 = "",
                                             const AttributeValue &     v2 = EmptyAttributeValue ()
```

7. Locate Nodes

- Install nodes to MobilityHelper

void ns3::MobilityHelper::Install (NodeContainer container) const

Layout a collection of nodes according to the current position allocator type.

For each node in the provided **NodeContainer**, this method creates an instance of a **ns3::MobilityModel** subclass (the type of which was set with **MobilityHelper::SetMobilityModel**), aggregates it to the node, and sets an initial position based on the current position allocator (set through **MobilityHelper::SetPositionAllocator**).

8. Create UDP Application

- Create UDP server
- Create server application on server node

ns3::UdpServerHelper::UdpServerHelper (uint16_t port)

Create **UdpServerHelper** which will make life easier for people trying to set up simulations with udp-client-server application.

Parameters

port The port the server will wait on for incoming packets

ApplicationContainer ns3::UdpServerHelper::Install (NodeContainer c)

Create one UDP server application on each of the Nodes in the **NodeContainer**.

Parameters

c The nodes on which to create the Applications. The nodes are specified by a **NodeContainer**.

Returns

The applications created, one **Application** per **Node** in the **NodeContainer**.

8. Create UDP Application

- Start and stop server application on server node
  - Start server app at Seconds(0.0)
  - Stop server app at Seconds(simulationTime +1)

void ns3::ApplicationContainer::Start (Time start)

Arrange for all of the Applications in this container to **Start()** at the **Time** given as a parameter.

void ns3::ApplicationContainer::Stop (Time stop)

Arrange for all of the Applications in this container to **Stop()** at the **Time** given as a parameter.

8. Create UDP Application

- Create UDP client

```
ns3::UdpClientHelper::UdpClientHelper ( Ipv4Address  ip,
                                        uint16_t     port
                                      )
```

Create **UdpClientHelper** which will make life easier for people trying to set up simulations with udp-client-server.

Parameters

ip The IPv4 address of the remote UDP server

port The port number of the remote UDP server

8. Create UDP Application

- Set UDP client attributes
  - Set “MaxPackets” to 4294967295 (=2^32 -1)
  - Set “Interval” to TimeValue (Time (“0.00002”))
  - Set “PacketSize” to UintegerValue (payloadSize)

```
void ns3::UdpClientHelper::SetAttribute ( std::string            name,
                                          const AttributeValue & value
                                        )
```

Record an attribute to be set in each **Application** after it is is created.

Parameters

name the name of the attribute to set

value the value of the attribute to set

8. Create UDP Application

- Create client application on server node

ApplicationContainer ns3::UdpClientHelper::Install (NodeContainer c)

Parameters

c the nodes

Create one UDP client application on each of the input nodes

Returns

the applications created, one application per input node.

- Start and stop server application on server node
  - Start server app at Seconds(1.0)
  - Stop server app at Seconds(simulationTime +1)

void ns3::ApplicationContainer::Start (Time start)

Arrange for all of the Applications in this container to **Start()** at the **Time** given as a parameter.

void ns3::ApplicationContainer::Stop (Time stop)

Arrange for all of the Applications in this container to **Stop()** at the **Time** given as a parameter.

9. Simulation Run and Calc. Thpt.

- Run simulation

```
void ns3::Simulator::Run ( void )
```

Run the simulation.

- Schedule simulation stop time

```
void ns3::Simulator::Stop ( Time const & delay )
```

Schedule the time delay until the **Simulator** should stop.

- Destroy simulation

```
void ns3::Simulator::Destroy ( void )
```

Execute the events scheduled with **ScheduleDestroy()**.

9. Simulation Run and Calc. Thpt.

- Throughput calculation
  - Get # of total received packets at UDP server

uint32_t ns3::UdpServer::GetReceived (void) const

Returns the number of received packets.

  - Calculate throughput (Mbps)
    - Thpt = totalPacketsRecv * payloadSize * 8 / (simulationTime *10^6)
  - Screen out the result
    - std::cout << “Throughput= ” << Thpt << “ Mbps” << ‘\n’;

Exercise 1

1. Fill in the appropriate code for "to do #1~#4“

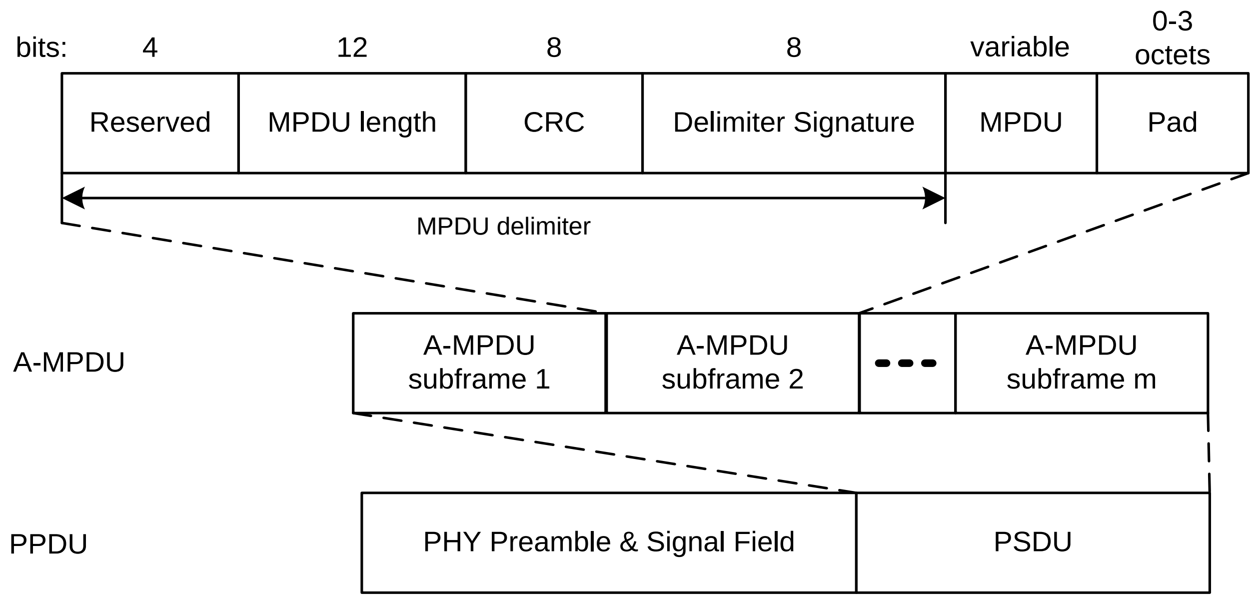
2. Make command line arguments for **packet size**, **simulation time**

3. Change **packet size** and observe throughput change.

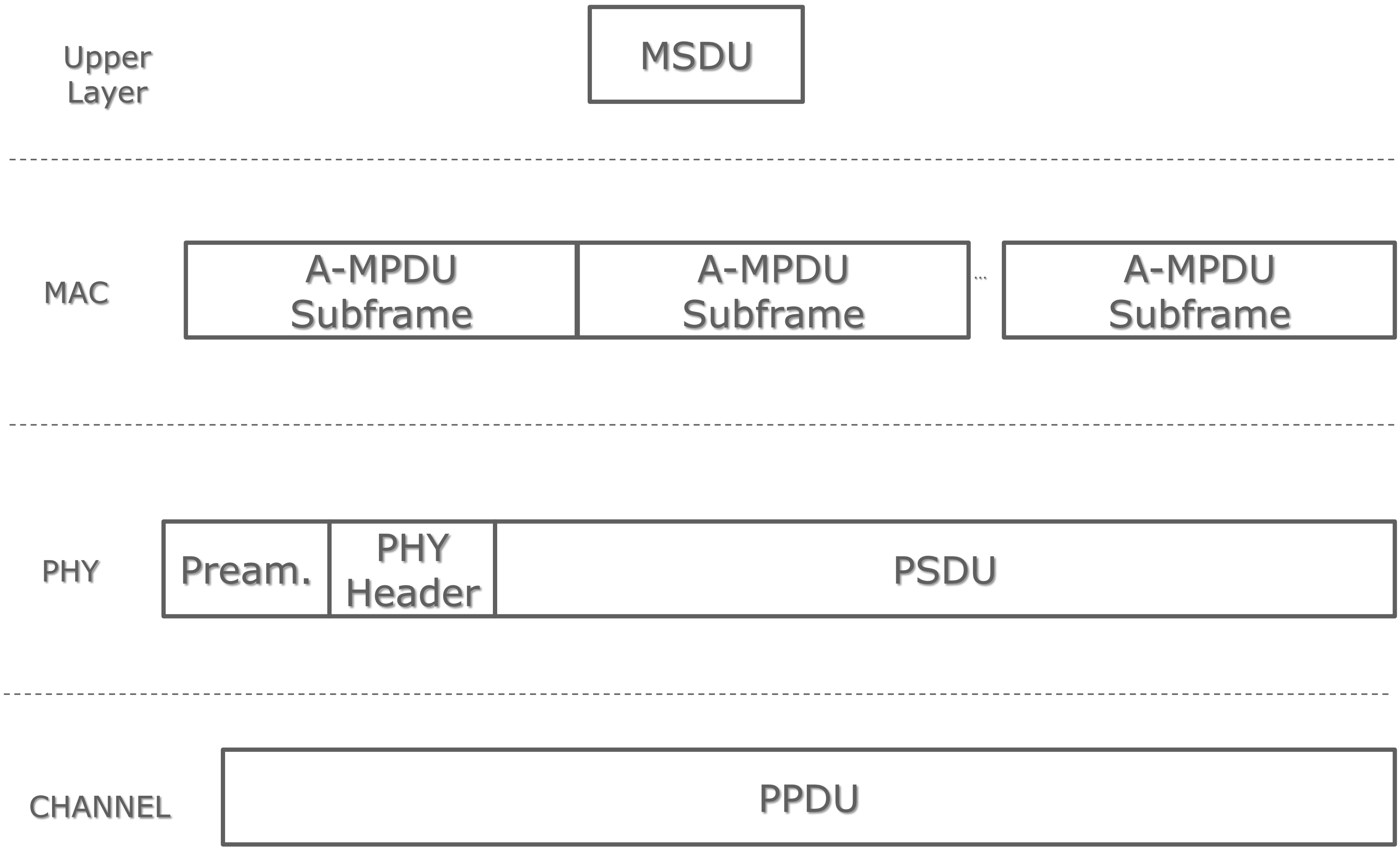
A-MPDU

A-MPDU

- WLAN protocol overhead limits its performance
  - PLCP preamble/header (40 us for HT)
  - Contention overhead
- A-MPDU: Only one PHY overhead for multiple MPDUs



A-MPDU


Upper Layer
MSDU
MAC
A-MPDU Subframe
A-MPDU Subframe
...
A-MPDU Subframe
PHY
Pream.
PHY Header
PSDU
CHANNEL
PPDU

A-MPDU Aggr. Limits (802.11n)

1. BlockAck limit:

   64 MPDUs

2. PSDU size limit:

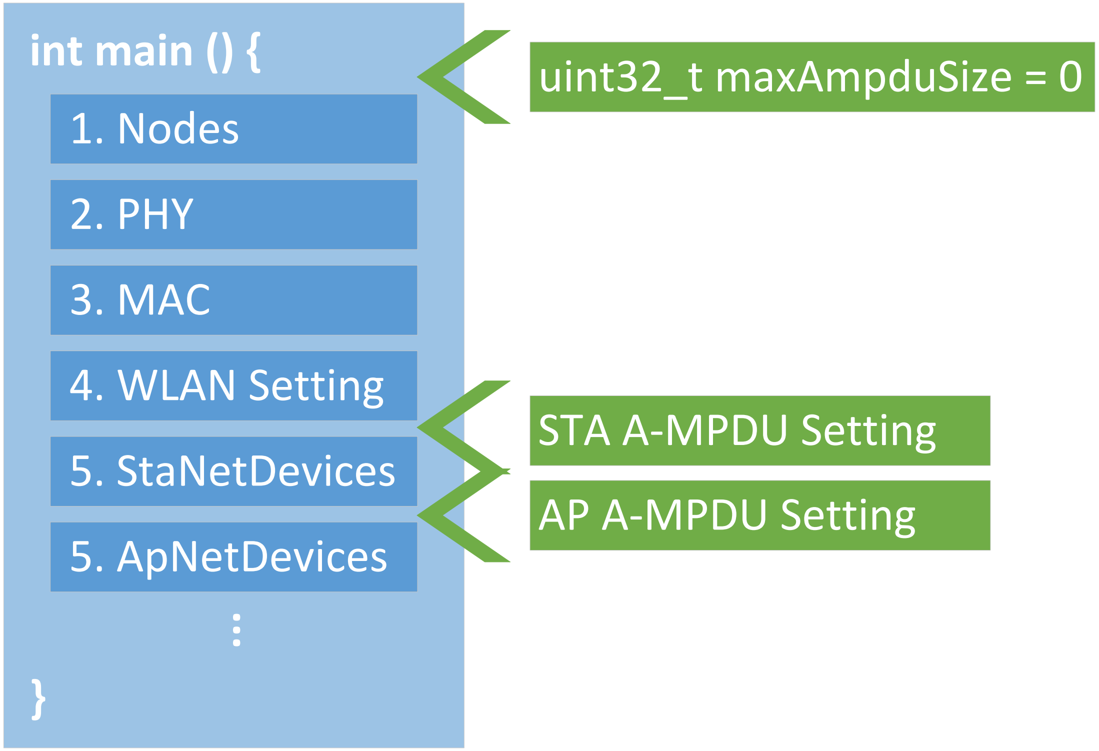
   65,535 bytes

3. PPDU duration limit:

   10 ms

Inclass Assignment 2

1. Create uint32_t variable **maxAmpduSize=0**
2. Create **A-MPDU setting** on MAC for STA/AP NetDevices



A-MPDU Setting

- Set MPDU aggreagator
  - Attribute name = “BE_MaxAmpduSize”,
  - Attribute value = “maxAmpduSize” variable

```
void ns3::WifiMacHelper::SetType ( std::string              type,
                                   std::string              n0 = "",
                                   const AttributeValue &   v0 = EmptyAttributeValue (),
                                   std::string              n1 = "",
                                   const AttributeValue &   v1 = EmptyAttributeValue (),
                                   std::string              n2 = "",
```

Exercise 2

1. Make command line arguments for **maximum A-MPDU frame size (maxAmpduSize)**

2. Change **maxAmpduSize** and observe throughput change.

Q & A