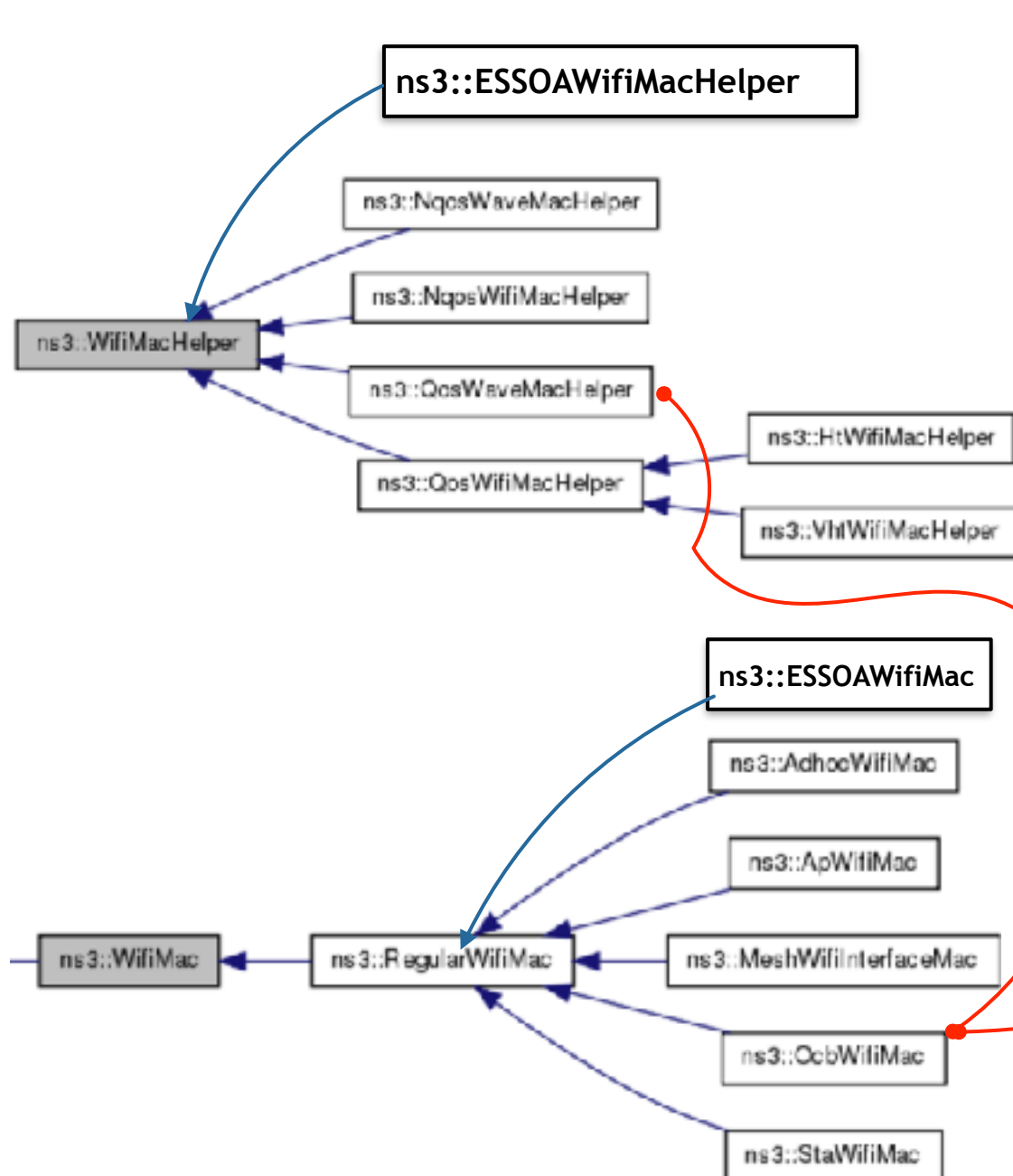


# New MAC model on Wifi

M.S. Francisco Eduardo Balart Sanchez



```
QosWaveMacHelper waveMac = QosWaveMacHelper::Default ();
```

```
devices = waveHelper.Install (wavePhy, waveMac, nodes);
```

```
QosWaveMacHelper
QosWaveMacHelper::Default (void)
{
    QosWaveMacHelper helper;

    // We're making QoS-enabled Wi-Fi MACs here, so we set the necessary
    // attribute. I've carefully positioned this here so that someone
    // who knows what they're doing can override with explicit
    // attributes.
    helper.SetType ("ns3::OcbWifiMac", "QosSupported", BooleanValue (true));
}
```

# WifiMAC

Go to:

[https://github.com/balart40/big\\_data\\_iteso\\_phd\\_public/wiki/NS3 with MANET#creating-a-new-wifi-mac-model](https://github.com/balart40/big_data_iteso_phd_public/wiki/NS3%20with%20MANET#creating-a-new-wifi-mac-model)

AODV

# AODV

```
void
AodvExample::Run ()
{
// Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThresho
CreateNodes ();
CreateDevices ();
InstallInternetStack ();
InstallApplications ();

std::cout << "Starting simulation for " << totalTime << " s ... \n";

Simulator::Stop (Seconds (totalTime));
Simulator::Run ();
Simulator::Destroy ();
}
```

WifiNetDevices

```
void
AodvExample::CreateDevices ()
{
WifiMacHelper wifiMac;
wifiMac.SetType ("ns3::AdhocWifiMac");
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
wifiPhy.SetChannel (wifiChannel.Create ());
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode", StringValue ("OfdmRate6Mbps"), "RtsCtsThreshold", UIntegerValue (0));
devices = wifi.Install (wifiPhy, wifiMac, nodes);

if (pcap)
{
wifiPhy.EnablePcapAll (std::string ("aodv"));
}
}
```

```
void
AodvExample::CreateNodes ()
{
std::cout << "Creating " << (unsigned)size << " nodes " << step << " m apart. \n";
nodes.Create (size);
// Name nodes
for (uint32_t i = 0; i < size; ++i)
{
std::ostringstream os;
os << "node-" << i;
Names::Add (os.str (), nodes.Get (i));
}
// Create static grid
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (step),
                               "DeltaY", DoubleValue (0),
                               "GridWidth", UIntegerValue (size),
                               "LayoutType", StringValue ("RowFirst"));
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);
}
```

```
void
AodvExample::InstallInternetStack ()
{
AodvHelper aodv;
// you can configure AODV attributes here using aodv.Set(name, value)
InternetStackHelper stack;
stack.SetRoutingHelper (aodv); // has effect on the next Install ()
stack.Install (nodes);
Ipv4AddressHelper address;
address.SetBase ("10.0.0.0", "255.0.0.0");
Interfaces = address.Assign (devices);

if (printRoutes)
{
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("aodv.routes", std::ios::out);
aodv.PrintRoutingTableAllAt (Seconds (8), routingStream);
}
}
```

# OSI

data unit

layers

**Routing**

InternetStack

Host Layers

Media Layers

data

**application**  
Network Process to Application

data

**presentation**  
Data Representation & Encryption

data

**session**  
Interhost Communication

segments

**transport**  
End-to-End Connections and Reliability

packets

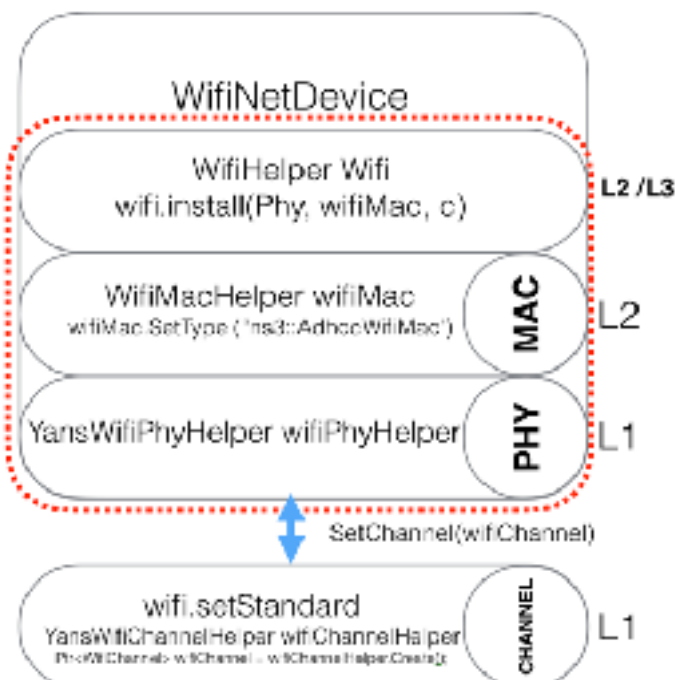
**network**  
Path Determination & Logical Addressing (IP)

frames

**data link**  
Physical Addressing (MAC & LLC)

bits

**physical**  
Media, Signal and Binary Transmission



# AODV routing

aodv-routing-protocol.h

```
namespace ns3
{
namespace aodv
{
/**
 * \ingroup aodv
 *
 * \brief AODV routing protocol
 */
class RoutingProtocol : public Ipv4RoutingProtocol
private:
    Neighbors m_nb;
```



# AODV routing

[aodv-routing-protocol.cc](http://aodv-routing-protocol.cc)

```
RoutingProtocol::RoutingProtocol () :
    m_nb (HelloInterval),
    {
        m_nb.SetCallback (MakeCallback (&RoutingProtocol::SendRerrWhenBreaksLinkToNextHop, this));
    }

void
RoutingProtocol::Start ()
{
    if (EnableHello)
    {
        m_nb.ScheduleTimer ();
    }
}

bool
RoutingProtocol::RouteInput (Ptr<const Packet> p, const Ipv4Header &header,
                             Ptr<const NetDevice> ldev, UnicastForwardCallback ucb,
                             MulticastForwardCallback mcb, LocalDeliverCallback lcb, ErrorCallback ecb)
{
    // Unicast local delivery
    if (m_ipv4->IsDestinationAddress (dst, ldev))
    {
        UpdateRouteLifeTime (origin, ActiveRouteTimeout);
        RoutingTableEntry toOrigin;
        if (m_routingTable.LookupValidRoute (origin, toOrigin))
        {
            UpdateRouteLifeTime (toOrigin.GetNextHop (), ActiveRouteTimeout);
            m_nb.Update (toOrigin.GetNextHop (), ActiveRouteTimeout);
        }
    }

    bool
RoutingProtocol::Forwarding (Ptr<const Packet> p, const Ipv4Header & header,
                             UnicastForwardCallback ucb, MulticastForwardCallback mcb, LocalDeliverCallback lcb, ErrorCallback ecb)
{
    m_nb.Update (route->GetGateway (), ActiveRouteTimeout);
    m_nb.Update (toOrigin.GetNextHop (), ActiveRouteTimeout);
}

void
RoutingProtocol::NotifyInterfaceUp (uint32_t i)
{
    if (l3->GetInterface (i)->GetArpCache ())
    {
        m_nb.AddArpCache (l3->GetInterface (i)->GetArpCache ());
    }
    mac->TraceConnectWithoutContext ("TxErrHeader", m_nb.GetTxErrorCallback ());
}
```



# AODV routing

[aodv-routing-protocol.cc](http://aodv-routing-protocol.cc)

```
void
RoutingProtocol::NotifyInterfaceDown (uint32_t i)
{
  if (wifl != 0)
  {
    Ptr<WifiMac> mac = wifl->GetMac ()->GetObject<AdhocWifiMac> ();
    if (mac != 0)
    {
      mac->TraceDisconnectWithoutContext ("TxErrHeader",
      m_nb.GetTxErrorCallback ());
      m_nb.DelArpCache (l3->GetInterface (i)->GetArpCache ());
    }
  }

  if (m_socketAddresses.empty ())
  {
    NS_LOG_LOGIC ("No aodv interfaces");
    m_htimer.Cancel ();
    m_nb.Clear ();
    m_routingTable.Clear ();
    return;
  }
}

void
RoutingProtocol::NotifyRemoveAddress (uint32_t i, Ipv4InterfaceAddress address)
{
  if (m_socketAddresses.empty ())
  {
    NS_LOG_LOGIC ("No aodv interfaces");
    m_htimer.Cancel ();
    m_nb.Clear ();
    m_routingTable.Clear ();
    return;
  }
}

void
RoutingProtocol::RecvRequest (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src)
{
  m_nb.Update (src, Time (AllowedHelloLoss * HelloInterval));
}

void
RoutingProtocol::ProcessHello (RrepHeader const & rrepHeader, Ipv4Address receiver )
{
  if (EnableHello)
  {
    m_nb.Update (rrepHeader.GetDst (), Time (AllowedHelloLoss * HelloInterval));
  }
}
```

# aodv-neighbor class

```
balart40@balart40-VirtualBox:~/Desktop/balart40/ns/ns-allinone-3.25/ns-3.25/src/aodv$ ls
bindings  doc  examples  helper  model  test  wscript
balart40@balart40-VirtualBox:~/Desktop/balart40/ns/ns-allinone-3.25/ns-3.25/src/aodv$ cd model/
balart40@balart40-VirtualBox:~/Desktop/balart40/ns/ns-allinone-3.25/ns-3.25/src/aodv/model$ ls
aodv-dpd.cc  aodv-id-cache.cc  aodv-neighbor.cc  aodv-packet.cc  aodv-routing-protocol.cc  aodv-rqueue.cc  aodv-rtable.cc
aodv-dpd.h   aodv-id-cache.h   aodv-neighbor.h   aodv-packet.h   aodv-routing-protocol.h  aodv-rqueue.h   aodv-rtable.h
balart40@balart40-VirtualBox:~/Desktop/balart40/ns/ns-allinone-3.25/ns-3.25/src/aodv/model$
```

\*.h

```
#ifndef AODVNEIGHBOR_H
#define AODVNEIGHBOR_H

#include "ns3/simulator.h"
#include "ns3/timer.h"
#include "ns3/ipv4-address.h"
#include "ns3/callback.h"
#include "ns3/wifi-mac-header.h"
#include "ns3/arp-cache.h"
#include <vector>

namespace ns3
{
namespace aodv
{
class RoutingProtocol;
/**
 * \ingroup aodv
 * \brief maintain list of active neighbors
 */
class Neighbors
{
public:
    /// c-tor
    Neighbors (Time delay);
    /// Neighbor description
    struct Neighbor
    {
        Ipv4Address m_neighborAddress;
        Mac48Address m_hardwareAddress;
        Time m_expireTime;
        bool close;

        Neighbor (Ipv4Address ip, Mac48Address mac, Time t) :
            m_neighborAddress (ip), m_hardwareAddress (mac), m_expireTime (t),
            close (false)
        {
        }
    };
};
```

```
/// Return expire time for neighbor node with address addr, if exists, else return 0.
Time GetExpireTime (Ipv4Address addr);
/// Check that node with address addr is neighbor
bool IsNeighbor (Ipv4Address addr);
/// Update expire time for entry with address addr, if it exists, else add new entry
void Update (Ipv4Address addr, Time expire);
/// Remove all expired entries
void Purge ();
/// Schedule m_ntimer.
void ScheduleTimer ();
/// Remove all entries
void Clear () { m_nb.clear (); }

/// Add ARP cache to be used to allow layer 2 notifications processing
void AddArpCache (Ptr<ArpCache>);
/// Don't use given ARP cache any more (interface is down)
void DelArpCache (Ptr<ArpCache>);
/// Get callback to ProcessTxError
Callback<void, WifiMacHeader const &> GetTxErrorCallback () const { return m_txErrorCallback; }

/// Handle link failure callback
void SetCallback (Callback<void, Ipv4Address> cb) { m_handleLinkFailure = cb; }
/// Handle link failure callback
Callback<void, Ipv4Address> GetCallback () const { return m_handleLinkFailure; }

private:
    /// link failure callback
    Callback<void, Ipv4Address> m_handleLinkFailure;
    /// TX error callback
    Callback<void, WifiMacHeader const &> m_txErrorCallback;
    /// Timer for neighbor's list. Schedule Purge().
    Timer m_ntimer;
    /// vector of entries
    std::vector<Neighbor> m_nb;
    /// list of ARP cached to be used for layer 2 notifications processing
    std::vector<Ptr<ArpCache> > m_arp;

    /// Find MAC address by IP using list of ARP caches
    Mac48Address LookupMacAddress (Ipv4Address);
    /// Process layer 2 TX error notification
    void ProcessTxError (WifiMacHeader const &);
};

}

}
```

```
#endif /* AODVNEIGHBOR_H */
```

# aodv-neighbor class

```
#include "aodv-neighbor.h"
#include "ns3/log.h"
#include <algorithm>

namespace ns3
{
    NS_LOG_COMPONENT_DEFINE ("AodvNeighbors");
    namespace aodv
    {
        Neighbors::Neighbors (Time delay) :
        m_ntimer (Timer::CANCEL_ON_DESTROY)
        {
            m_ntimer.SetDelay (delay);
            m_ntimer.SetFunction (&Neighbors::Purge, this);
            m_txErrorCallback = MakeCallback (&Neighbors::ProcessTxError, this);
        }
        bool Neighbors::IsNeighbor (Ipv4Address addr)
        {
            Purge ();
            for (std::vector<Neighbor>::const_iterator i = m_nb.begin ();
                 i != m_nb.end (); ++i)
            {
                if (i->m_neighborAddress == addr)
                    return true;
            }
            return false;
        }
        Time Neighbors::GetExpireTime (Ipv4Address addr)
        {
            Purge ();
            for (std::vector<Neighbor>::const_iterator i = m_nb.begin (); i
                 != m_nb.end (); ++i)
            {
                if (i->m_neighborAddress == addr)
                    return (i->m_expireTime - Simulator::Now ());
            }
            return Seconds (0);
        }
        void Neighbors::Update (Ipv4Address addr, Time expire)
        {
            for (std::vector<Neighbor>::iterator i = m_nb.begin (); i != m_nb.end (); ++i)
                if (i->m_neighborAddress == addr)
                {
                    i->m_expireTime
                        = std::max (expire + Simulator::Now (), i->m_expireTime);
                    if (i->m_hardwareAddress == Mac48Address ())
                        i->m_hardwareAddress = LookupMacAddress (i->m_neighborAddress);
                    return;
                }
            NS_LOG_LOGIC ("Open link to " << addr);
            Neighbor neighbor (addr, LookupMacAddress (addr), expire + Simulator::Now ());
            m_nb.push_back (neighbor);
            Purge ();
        }
        struct CloseNeighbor
        {
            bool operator() (const Neighbors::Neighbor & nb) const
            {
                return ((nb.m_expireTime < Simulator::Now ()) || nb.close);
            }
        };

        void Neighbors::Purge ()
        {
            if (m_nb.empty ())
                return;

            CloseNeighbor pred;
            if (!m_handleLinkFailure.IsNull ())
            {
                for (std::vector<Neighbor>::iterator j = m_nb.begin (); j != m_nb.end (); ++j)
                {
                    if (pred (*j))
                    {
                        NS_LOG_LOGIC ("Close link to " << j->m_neighborAddress);
                        m_handleLinkFailure (j->m_neighborAddress);
                    }
                }
            }
            m_nb.erase (std::remove_if (m_nb.begin (), m_nb.end (), pred), m_nb.end ());
            m_ntimer.Cancel ();
            m_ntimer.Schedule ();
        }
        void Neighbors::ScheduleTimer ()
        {
            m_ntimer.Cancel ();
            m_ntimer.Schedule ();
        }
        void Neighbors::AddArpCache (Ptr<ArpCache> a)
        {
            m_arp.push_back (a);
        }
        void Neighbors::DelArpCache (Ptr<ArpCache> a)
        {
            m_arp.erase (std::remove (m_arp.begin (), m_arp.end (), a), m_arp.end ());
        }
        Mac48Address Neighbors::LookupMacAddress (Ipv4Address addr)
        {
            Mac48Address hwaddr;
            for (std::vector<Ptr<ArpCache> >::const_iterator i = m_arp.begin ();
                 i != m_arp.end (); ++i)
            {
                ArpCache::Entry * entry = (*i)->Lookup (addr);
                if (entry != 0 && (entry->IsAlive () || entry->IsPermanent ()) && !entry->IsExpired ())
                {
                    hwaddr = Mac48Address::ConvertFrom (entry->GetMacAddress ());
                    break;
                }
            }
            return hwaddr;
        }
        void Neighbors::ProcessTxError (WifiMacHeader const & hdr)
        {
            Mac48Address addr = hdr.GetAddr1 ();

            for (std::vector<Neighbor>::iterator i = m_nb.begin (); i != m_nb.end (); ++i)
            {
                if (i->m_hardwareAddress == addr)
                    i->close = true;
            }
            Purge ();
        }
    }
}
```

# Reviewing MAC on NS3

- Basically we put a PRINT message of All the functions of the ESSOA MAC WIFI Class which is a clone of the adhoc to see when their functions are triggered
  - This with the objective of knowing if MAC has a functions that is always running which would be ideal for the ESSOA algorithm to be placed
  - To understand more how the mac is used
- The aforementioned was done with the test run with and without traffic

# Reviewing MAC on NS3: Put Print messages in all functions of the mac class functions\*

```
TypeId
EssoaWifiMac::GetTypeId (void)
{
    NS_LOG_UNCOND ("%INFO: Entered to GetTypeId function of ESSOA MAC CLASS");
}

EssoaWifiMac::~EssoaWifiMac ()
{
    NS_LOG_FUNCTION (this);
    NS_LOG_UNCOND ("%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS");
}

void
EssoaWifiMac::SetAddress (Mac48Address address)
{
    NS_LOG_UNCOND ("%INFO: Entered to SetAddress function of ESSOA MAC CLASS");
}

EssoaWifiMac::EssoaWifiMac ()
{
    NS_LOG_UNCOND ("%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS");
    // Get the MAC address from the config file as follows:
}

void
EssoaWifiMac::Enqueue (Ptr<const Packet> packet, Mac48Address to)
{
    NS_LOG_UNCOND ("%INFO: Entered to Enqueue function of ESSOA MAC CLASS");
}

void
EssoaWifiMac::SetLinkUpCallback (Callback<void> linkUp)
{
    NS_LOG_FUNCTION (this << &linkUp);
    NS_LOG_UNCOND ("%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS");
}

void
EssoaWifiMac::Receive (Ptr<Packet> packet, const WifiMacHeader *hdr)
{
    NS_LOG_FUNCTION (this << packet << hdr);
    NS_LOG_UNCOND ("%INFO: Entered to Receive function of ESSOA MAC CLASS");
}
```



# No traffic sim

```
NS_LOG_UNCOND ("%INFO: Configuring MAC...");
// MAC layer configuration
WifiMacHelper wifiMac;
// Setting the type with Adhoc we will wrap and inherit the RegularMac and WifiMac classes properties
wifiMac.SetType ("ns3::EssoaWifiMac", "QosSupported", BooleanValue (false));

// Make a device nodes with phy, mac and nodes already configured
// Below will create WifiNetDevices *****
NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

NS_LOG_UNCOND ("%INFO: configuring mobility...");
```

\*.h, due to the include?

```
class EssoaWifiMac : public RegularWifiMac
{
public:
    static TypeId GetTypeId (void);
```

```
EssoaWifiMac::GetTypeId (void)
{
```

```
NS_LOG_UNCOND ("%INFO: Entered to GetTypeId function of ESSOA MAC CLASS");
```

```
%INFO: Entered to GetTypeId function of ESSOA MAC CLASS
%INFO: Starting Test now...
%INFO: Creating Nodes...
%INFO: Size of vector of vectors 3
%INFO: Configuring PHY Loss model and connecting to PHY...
%INFO: Configuring PHY Loss model...
%INFO: Connecting PHY with Channel...
%INFO: Configuring PHY STD and RSM...
%INFO: Configuring MAC...
%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS
%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS
%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS
%INFO: configuring mobility...
%INFO: configuring Energy...
%INFO: Assign Mac48Address Addresses & Generating Traffic hello world broadcast
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
%INFO: TimeStamp: 1 segs Node: ID: 0    Current Remaining energy = 4.181 Joules
%INFO: TimeStamp: 1 segs Node: ID: 1    Current Remaining energy = 4.181 Joules
%INFO: TimeStamp: 1 segs Node: ID: 2    Current Remaining energy = 4.181 Joules
%INFO: TimeStamp: 2 segs Node: ID: 0    Current Remaining energy = 3.362 Joules
%INFO: TimeStamp: 7 segs Node: ID: 0    Current Remaining energy = 0 Joules
%INFO: TimeStamp: 7 segs Node: ID: 1    Current Remaining energy = 0 Joules
%INFO: TimeStamp: 7 segs Node: ID: 2    Current Remaining energy = 0 Joules
%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS
%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS
%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS
```

# No traffic sim

```
namespace ns3 {
```

```
class WifiMac;
```

```
class WifiMacHelper  
{  
public:
```

```
WifiMacHelper ();
```

```
virtual void SetType (std::string type,  
std::string n0 = "", const AttributeValue &v0 = EmptyAttributeValue (),
```

```
virtual Ptr<WifiMac> Create (void) const;
```

```
protected:
```

```
ObjectFactory m_mac;
```

```
Ptr<WifiMac>  
WifiMacHelper::Create (void) const  
{  
Ptr<WifiMac> mac = m_mac.Create<WifiMac> ();  
return mac;  
}
```

```
WifiMacHelper::WifiMacHelper ()  
{  
//By default, we create an AdHoc MAC layer without QoS.  
SetType ("ns3:AdhocWifiMac",  
"QosSupported", BooleanValue (false));  
}
```

```
namespace ns3 {
```

```
class Dcf;
```

```
class WifiMac : public Object  
{  
public:  
static TypeId GetTypeId (void);
```

```
NS_LOG_UNCOND ("%INFO: Configuring MAC...");  
// MAC layer configuration  
WifiMacHelper wifiMac;  
// Setting the type with AdHoc we will wrap and inherit the RegularMac and WifiMac classes properties  
wifiMac.SetType ("ns3:EssoaWifiMac", "QosSupported", BooleanValue (false));  
  
// Make a device nodes with phy, mac and nodes already configured  
// Below will create WifiNetDevices *****  
NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);  
  
NS_LOG_UNCOND ("%INFO: configuring mobility...");
```

```
void  
WifiMacHelper::SetType (std::string type,  
std::string n0, const AttributeValue &v0,  
m_mac.SetTypeId (type)  
m_mac.Set (n0, v0);
```

```
NS_LOG_UNCOND ("%INFO: Configuring PHY STD and RSM...");  
// Create WifiHelper to be able to setup the PHY  
WifiHelper wifi; // = WifiHelper::Default ();  
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
```

```
%INFO: Entered to GetTypeID function of ESSOA MAC CLASS  
%INFO: Starting Test now...  
%INFO: Creating Nodes...  
%INFO: Size of vector of vectors 3  
%INFO: Configuring PHY Loss model and connecting to PHY...  
%INFO: Configuring PHY Loss model...  
%INFO: Connecting PHY with Channel...  
%INFO: Configuring PHY STD and RSM...  
%INFO: Configuring MAC...  
%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS  
%INFO: Entered to SetAddress function of ESSOA MAC CLASS  
%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS  
%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS  
%INFO: Entered to SetAddress function of ESSOA MAC CLASS  
%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS  
%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS  
%INFO: Entered to SetAddress function of ESSOA MAC CLASS  
%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS  
%INFO: configuring mobility...  
%INFO: configuring Energy...  
%INFO: Assign Mac48Address Addresses & Generating Traffic hello world broadcast  
%INFO: Entered to SetAddress function of ESSOA MAC CLASS  
%INFO: Entered to SetAddress function of ESSOA MAC CLASS  
%INFO: Entered to SetAddress function of ESSOA MAC CLASS  
%INFO: TimeStamp: 1 segs Node: ID: 0 Current Remaining energy = 4.181 Joules  
%INFO: TimeStamp: 1 segs Node: ID: 1 Current Remaining energy = 4.181 Joules  
%INFO: TimeStamp: 1 segs Node: ID: 2 Current Remaining energy = 4.181 Joules  
%INFO: TimeStamp: 2 segs Node: ID: 0 Current Remaining energy = 3.362 Joules
```

```
%INFO: TimeStamp: 7 segs Node: ID: 0 Current Remaining energy = 0 Joules  
%INFO: TimeStamp: 7 segs Node: ID: 1 Current Remaining energy = 0 Joules  
%INFO: TimeStamp: 7 segs Node: ID: 2 Current Remaining energy = 0 Joules  
%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS  
%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS  
%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS
```



# No traffic sim

```
Ptr<WifiMac>
WifiMacHelper::Create (void) const
{
    Ptr<WifiMac> mac = m_mac.Create<WifiMac> ();
    return mac;
}
```

```
class WifiMac : public Object
{
public:
    static TypeId GetTypeId (void);
```

```
namespace ns3 {
class Dcf;
```

```
void
WifiMacHelper::SetType (std::string type,
                        std::string n0, const AttributeValue &v0,
```

```
{
    m_mac.SetTypeId (type);
    m_mac.Set (n0, v0);
```

```
Ptr<Object>
ObjectFactory::Create (void) const
{
    NS_LOG_FUNCTION (this);
    Callback<ObjectBase *> cb = m_tid.GetConstructor ();
    ObjectBase *base = cb ();
    Object *derived = dynamic_cast<Object *> (base);
    NS_ASSERT (derived != 0);
    derived->SetTypeId (m_tid);
    derived->Construct (m_parameters);
    Ptr<Object> object = Ptr<Object> (derived, false);
    return object;
}
```

```
void
ObjectFactory::SetTypeId (std::string tid)
{
    NS_LOG_FUNCTION (this << tid);
    m_tid = TypeId::LookupByName (tid);
}
```

```
void
ObjectFactory::Set (std::string name, const AttributeValue &value)
{
    NS_LOG_FUNCTION (this << name << &value);
    if (name == "")
    {
        return;
    }

    struct TypeId::AttributeInformation info;
    if (!m_tid.LookupAttributeByName (name, &info))
    {
        NS_FATAL_ERROR ("Invalid attribute set (" << name << ") on " << m_tid.GetName ());
        return;
    }
    Ptr<AttributeValue> v = info.checker->CreateValidValue (value);
    if (v == 0)
    {
        NS_FATAL_ERROR ("Invalid value for attribute set (" << name << ") on " << m_tid.GetName ());
        return;
    }
    m_parameters.Add (name, info.checker, value.Copy ());
}
```

# No traffic sim

```
NS_LOG_UNCOND ("%INFO: Configuring MAC...");
// MAC layer configuration
WifiMacHelper wifiMac;
// Setting the type with Adhoc we will wrap and inherit the RegularMac and WifiMac classes properties
wifiMac.SetType ("ns3::EssoaWifiMac", "QosSupported", BooleanValue (false));

// Make a device nodes with phy, mac and nodes already configured
// Below will create WifiNetDevices *****
NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

NS_LOG_UNCOND ("%INFO: configuring mobility...");
```

```
NetDeviceContainer
WifiHelper::Install (const WifiPhyHelper &phyHelper,
                    const WifiMacHelper &macHelper, NodeContainer c) const
{
    NetDeviceContainer devices;
    for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i)
    {
        Ptr<Node> node = *i;
        Ptr<WifiNetDevice> device = CreateObject<WifiNetDevice> ();
        Ptr<WifiRemoteStationManager> manager = m_stationManager.Create<WifiRemoteStationManager> ();
        Ptr<WifiMac> mac = macHelper.Create ();
        Ptr<WifiPhy> phy = phyHelper.Create (node, device);
        mac->SetAddress (Mac48Address::Allocate ());
        mac->ConfigureStandard (m_standard);
        phy->ConfigureStandard (m_standard);
        device->SetMac (mac);
        device->SetPhy (phy);
        device->SetRemoteStationManager (manager);
        node->AddDevice (device);
        devices.Add (device);
        NS_LOG_DEBUG ("node=" << node << ", mob=" <<
    }
    return devices;
}
```

```
%INFO: Entered to GetTypeID function of ESSOA MAC CLASS
%INFO: Starting Test now...
%INFO: Creating Nodes...
%INFO: Size of vector of vectors 3
%INFO: Configuring PHY Loss model and connecting to PHY...
%INFO: Configuring PHY Loss model...
%INFO: Connecting PHY with Channel...
%INFO: Configuring PHY STD and RSM...
%INFO: Configuring MAC
```

```
%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS
%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS
%INFO: Entered to EssoaWifiMac constructor function of ESSOA MAC CLASS
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
%INFO: Entered to SetLinkUpCallback function of ESSOA MAC CLASS
```

```
%INFO: configuring mobility...
```

```
%INFO: configuring Energy...
```

```
%INFO: Assign Mac48Address Addresses & Generating Traffic hello world broadcast
```

```
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
```

```
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
```

```
%INFO: Entered to SetAddress function of ESSOA MAC CLASS
```

```
%INFO: TimeStamp: 1 segs Node: ID: 0 Current Remaining energy = 4.181 Joules
```

```
%INFO: TimeStamp: 1 segs Node: ID: 1 Current Remaining energy = 4.181 Joules
```

```
%INFO: TimeStamp: 1 segs Node: ID: 2 Current Remaining energy = 4.181 Joules
```

```
%INFO: TimeStamp: 2 segs Node: ID: 0 Current Remaining energy = 3.362 Joules
```

```
Simulator::Stop (Seconds (70.0));
Simulator::Run ();
Simulator::Destroy ();
```

```
%INFO: TimeStamp: 7 segs Node: ID: 0 Current Remaining energy = 0 Joules
%INFO: TimeStamp: 7 segs Node: ID: 1 Current Remaining energy = 0 Joules
%INFO: TimeStamp: 7 segs Node: ID: 2 Current Remaining energy = 0 Joules
%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS
%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS
%INFO: Entered to EssoaWifiMac destructor function of ESSOA MAC CLASS
```



# traffic sim

```
for (uint32_t i = 0; i < nDevices; ++i)
{
    Ptr<WifiNetDevice> wifiNetDevicePointer = DynamicCast<WifiNetDevice> (devices.Get (nodeIndexx));
    wifiNetDevicePointer->SetAddress(Mac48Address::Allocate ());
    devices.Get(nodeIndexx)->SetReceiveCallback(MakeCallback(&ReceivePacket));
    //wifiNetDevicePointer->GetPhy()->TraceConnectWithoutContext ("MonitorsnifferRX", MakeCallback (&ReceivePacketWithRss));
    //Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (sources.Get (nodeIndexx));
    Ptr<BasicEnergySource> basicSourcePtr = wifiNetDevicePointer->GetNode ()->GetObject<BasicEnergySource>();
    NS_ASSERT (basicSourcePtr != NULL);
    basicSourcePtr->TraceConnect ("RemainingEnergy",PrintID(wifiNetDevicePointer->GetNode ()), MakeCallback (&RemainingEnergyTrace));
    // Here te basicRadioModelPtr already is pointing to the ith device so we will want to get the radiomodelattached to it
    // in this case we only have attached one so that why the 0 if we had attached more we could put 1..n
    Ptr<DeviceEnergyModel> basicRadioModelPtr = basicSourcePtr->FindDeviceEnergyModels ("ns3::WifiRadioEnergyModel").Get (0);
    //Ptr<WifiRadioEnergyModel> basicRadioModelPtr = basicSourcePtr->GetObject<WifiRadioEnergyModel>();
    NS_ASSERT (basicRadioModelPtr != NULL);
    basicRadioModelPtr->TraceConnect ("TotalEnergyConsumption",PrintID(wifiNetDevicePointer->GetNode ()), MakeCallback (&TotalEnergy));
    // This is temporal since at end Nodes will have its own broadcast function below can lead to dead lock, need to see Mutual Exclusion
    features
    double r = 1.0 / ((double) rand() / (RAND_MAX));
    Simulator::ScheduleWithContext (wifiNetDevicePointer->GetNode ()->GetId (),Seconds (r), &GenerateHelloBroadcast, wifiNetDevicePointer,
    packetSize, InterPacketInterval, numPackets);
    nodeIndexx++;
}

static void GenerateHelloBroadcast(Ptr<WifiNetDevice> wifinetdevice, uint32_t pktSize, Time pktInterval, uint32_t numPackets)
{
    r
```

```
static void GenerateHelloBroadcast(Ptr<WifiNetDevice> wifinetdevice, uint32_t pktSize, Time pktInterval, uint32_t numPackets)
{
    wifinetdevice->Send (packet, broadcast,1);
    NS_LOG_UNCOND ("%INFO: Packet sent Generate traffic Broadcast call done");
}
```

```
static void GenerateTraffic (Ptr<WifiNetDevice> wifinetdevice, uint32_t pktSize, Time pktInterval, HelloHeader helloheaderToSend, const Address& destAddress)
{
    wifinetdevice->Send (packet, destAddress,1);
    NS_LOG_UNCOND ("%INFO: Packet sent Generate traffic call done");
}
```

```
bool ReceivePacket( Ptr<NetDevice> netdevice, Ptr<const Packet> packet, uint16_t protocol, const Address& sourceAddress)
{
    packet->EnablePrinting ();

    NS_LOG_UNCOND ("%INFO: sending packet response due to callback of BROADCAST REQUEST ANSWER with protocol: " << protocol);
    Simulator::Schedule (Seconds (r), &GenerateTraffic, wifinetdevice, pktSize, Seconds(r),header,sourceAddress);

    NS_LOG_UNCOND ("%INFO: sending packet response due to callback RESPONSE TO RESPONSE of BROADCAST NON REQUEST RESP with protocol: " << protocol);
    Simulator::Schedule (Seconds (r), &GenerateTraffic, wifinetdevice, pktSize, Seconds(r),header,sourceAddress);
}
}
```



```
*****
%INFO: Start of Recursive broadcast traffic 10 PACKETS REMAINING
%INFO: Sending Broadcast packet! I am node 2 my MAC is: 02-06-00:00:00:00:00:06
%INFO: Entered to Enqueue function of ESSOA MAC CLASS
%INFO: Packet sent Generate traffic Broadcast call done
%INFO: Decrementing timers
*****
%INFO: Node Id: 2
%INFO: Printing 0 neighbor(s)
%INFO: Entered to Receive function of ESSOA MAC CLASS
*****
%INFO: Rx packet for Node: 1 my MAC: 02-06-00:00:00:00:00:05 RX packet from MAC: 02-06-00:00:00:00:00:06
*****
%INFO: Tx Packet Info, Source Node Id: 1
%INFO: Tx Packet Info, Destination Node Id: 2
%INFO: Tx Packet Info, Sent Unique ID: 1
%INFO: Tx Packet Info, Rx Unique ID: 0
%INFO: sending packet response due to callback of BROADCAST REQUEST ANSWER with protocol: 1
%INFO: Entered to Receive function of ESSOA MAC CLASS
*****
%INFO: Tx Packet Info, Source Node Id: 0
%INFO: Tx Packet Info, Destination Node Id: 2
%INFO: Tx Packet Info, Sent Unique ID: 1
%INFO: Tx Packet Info, Rx Unique ID: 0
%INFO: sending packet response due to callback of BROADCAST REQUEST ANSWER with protocol: 1
*****
%INFO: Sending packet! I am node 1 my MAC is: 02-06-00:00:00:00:00:05
%INFO: source address: 02-06-00:00:00:00:00:06 to node Id: 2
%INFO: Entered to Enqueue function of ESSOA MAC CLASS
%INFO: Packet sent Generate traffic call done
%INFO: Entered to Receive function of ESSOA MAC CLASS
*****
*****
%INFO: RX packet for Node: 2 my MAC: 02-06-00:00:00:00:00:06 RX packet from MAC: 02-06-00:00:00:00:00:04
*****
%INFO: Tx Packet Info, Source Node Id: 2
%INFO: Tx Packet Info, Destination Node Id: 0
%INFO: Tx Packet Info, Sent Unique ID: 2
%INFO: Tx Packet Info, Rx Unique ID: 0
%INFO: sending packet response due to callback RESPONSE TO RESPONSE of BROADCAST NON REQUEST RESP with protocol: 1
*****
%INFO: Sending packet! I am node 2 my MAC is: 02-06-00:00:00:00:00:06
%INFO: source address: 02-06-00:00:00:00:00:05 to node Id: 1
*****
```

# AODV Explained

...