Wi-Fi Module in ns-3

Daniel Lertpratchya
Outline

• Supported Features

• Architecture

• Using Wi-Fi Module

• FAQs (using Wi-Fi)

• Future work in 2014
Wi-Fi Module Features
Wi-Fi Module Features

• DCF implementation (Basic + RTS/CTS)

• 802.11 a/b/g

• QoS support (EDCA only)

• Some 802.11n (block ACK, frame aggregation)

• Infrastructure and ad-hoc modes

• Rate adaptation algorithms

• Channel models

• Mobility models
Wi-Fi Module Architecture
Wi-Fi Module Architecture

- PHY layer model
- MAC low models: implement DCF and EDCA
- MAC high models
  - Specialized models: ad-hoc and infrastructure
- Rate control algorithms
Wi-Fi Module Architecture

- **WifiNetDevice**
  - Send (packet, dest, proto)
  - Enqueue (packet, to)
  - Enqueue (packet)
  - StartTransmission
  - SendPacket
  - Send

- **MAC High**
  - DcaTxop
  - MacRxMiddle
  - StartTransmission

- **MAC Low**
  - WifiPhy
  - Receive
  - ReceiveOk/ReceiveError
  - StartReceivePacket

- **WifiChannel**

**ApWifiMac, AdhocWifiMac, StaWifiMac**
- Beacon generation, association etc.
- Packet queue, fragmentation, retransmission (if needed)
- RTS/CTS/DATA/ACK transaction

For more detailed info, see ns-3 model.
MAC High

• Presently, three MAC high models
  • AdhocWifiMac: simplest one
  • ApWifiMac: beacon, associations by STAs
  • StaWifiMac: association based on beacons
• All inherit from RegularWifiMac, which handles QoS and non-QoS support
MAC Low

- Three components
  - MacLow
    - RTS/CTS/DATA/ACK transactions
  - DcfManager
    - implements the DCF
  - DcaTxop and EdcaTxopN:
    - One for NQoS, the other for QoS
    - Packet queue
    - Fragmentation/Retransmissions
Wi-Fi Rate Adaptation

• Both low-latency and high-latency Rate Adaptation is supported

• Algorithms in real devices (partial list)
  • ArfWifiManager (default of WifiHelper)
  • OnoeWifiManager
  • ConstantRateWifiManager

• Algorithms in literature (partial list)
  • IdealWifiManager
  • AarfWifiManager
  • AmrrWifiManager
Wi-Fi PHY

- Work at packet (frame) level
- Different models available
  - PhySimWifi — symbol-level, too slow for practical use
  - YansWifiPhy — default & most widely used
  - SpectrumWifiPhy (unfinished model, work stalled)
    - Models power spectral density of each transmission
    - Supports inter-technology interference
  - Some details are not implemented (e.g. preamble/capture effect)
Synchronization Model

- Energy Detection only
  - no preamble detection model
  - no AGC model
- Sync on first RX with energy > threshold
- Collision: the error model would likely drop the packet
- No capture effect: won’t re-sync on a stronger packet
SNIR Model [1]

• Each TX attempt modeled with single power value
• SINR calculation assuming gaussian interference
• «Chunk» error rate model
  • Basically, packet error rate model
  • But accounts for partially overlapping packets in time
  • «chunk»: interval of RX with constant SINR
• Orthogonal channel model only
  • No adjacent channel interference model
SNIR Calculation [1]
Error Rate Models

- One 802.11b model
  - Validated with Clear Channel experiments
- Two OFDM models
  - YANS model — A bit optimistic
  - NIST model — Better match with experiment [2]
Using Wi-Fi in ns-3
Setting up Wi-Fi Simulations

- Nodes
- Wi-Fi
- Mobility
- Routing
- Internet stack
- Application
Decisions

- Wi-Fi Standard 802.11a/b/g
- QoS or non-QoS
- Infrastructure or ad hoc
- Rate control
- WifiChannel — propagation loss/delay model
- Routing
- Mobility model
Sample Code

```cpp
NodeContainer c;
c.Create (2);

WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211a);

// Set to a non-QoS upper mac
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();

// Set it to adhoc mode
wifiMac.SetType ("ns3::AdhocWifiMac");

// Set Wi-Fi rate manager
std::string phyMode ("OfdmRate54Mbps");
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                           "DataMode",StringValue (phyMode),
                           "ControlMode",StringValue (phyMode));
```
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
// ns-3 supports RadioTap and Prism tracing extensions for 802.11
wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel",
    "Exponent", DoubleValue (3.0));
wifiPhy.SetChannel (wifiChannel.Create ());

NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

// Enable PCAP
wifiPhy.EnablePcap ("wifi-adhoc", devices);

// Configure mobility
MobilityHelper mobility;
Ptr<ListPositionAllocator> posAlloc = CreateObject<ListPositionAllocator> ();
posAlloc->Add (Vector (0.0, 0.0, 0.0));
posAlloc->Add (Vector (5.0, 0.0, 0.0));
mobility.SetPositionAllocator (posAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (c);
Selecting Wi-Fi Standard

NodeContainer c;
c.Create (2);

WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211a);

• Create WifiHelper

• WifiHelper::SetStandard (enum WifiPhyStandard s);

<table>
<thead>
<tr>
<th>WIFI_PHY_STANDARD_80211a</th>
<th>OFDM PHY 5GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIFI_PHY_STANDARD_80211b</td>
<td>DSSS PHY and HR/DSSS PHY</td>
</tr>
<tr>
<td>WIFI_PHY_STANDARD_80211g</td>
<td>EPR-OFDM PHY</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>
QoS or non-QoS

NodeContainer c;
c.Create (2);

WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211a);

// Set to a non-QoS upper mac
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();

- NqosWifiMacHelper for non-QoS
- QosWifiMacHelper for QoS
Ad hoc or Infrastructure

NodeContainer c;
c.Create (2);

WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211a);

// Set to a non-QoS upper mac
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();

// Set it to adhoc mode
wifiMac.SetType ("ns3::AdhocWifiMac");

<table>
<thead>
<tr>
<th>ns3::AdhocWifiMac</th>
<th>simplest, no beacon/assoc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns3::StaWifiMac</td>
<td>generates beacon, assoc. resp.</td>
</tr>
<tr>
<td>ns3::ApWifiMac</td>
<td>assoc. request</td>
</tr>
</tbody>
</table>
Rate Manager

// Set Wi-Fi rate manager
std::string phyMode ("OfdmRate54Mbps");
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
  "DataMode",StringValue (phyMode),
  "ControlMode",StringValue (phyMode));

- Select Wi-Fi rate manager
- This example — constant rate
- Unicast DATA @ 54Mbps
- Control “request” @ 54Mbps

<table>
<thead>
<tr>
<th>Standard</th>
<th>Rate Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.11a</td>
<td>OfdmRateXXMbps</td>
</tr>
<tr>
<td>802.11b</td>
<td>DsssRateXXMbps</td>
</tr>
<tr>
<td>802.11g</td>
<td>ErpOfdmRateXXMbps</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>
Rate Manager (cont.)

• For Non-unicast DATA

  Config::SetDefault("ns3::WifiRemoteStationManager::NonUnicastMode", 
  StringValue (phyMode));

• Control “reply” modes are not directly set (see [3])

• Other rate managers may not require manual setting as they automatically select the “best” rate
Rate Manager (cont.)

- Other interesting attributes
  - MaxSsrc — retransmission limit for RTS
  - MaxSlfrc — retransmission limit for DATA
  - RtsCtsThreshold
  - FragmentationThreshold
- May not have effect on some rate control algorithms
WifiPhy

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();

• Select ErrorRateModel here

• YansWifiPhyHelper::Default () is NistErrorRateModel

• To change to YansErrorRateModel, do
  YansWifiPhyHelper::SetErrorRateModel ("ns3::YansErrorRateModel");

• Nist/Yans ErrorRateModels are for ERP-OFDM and OFDM, both models call DsssErrorRateModel automatically for DSSS
WifiPhy

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
// ns-3 supports RadioTap and Prism tracing extensions for 802.11
wifiPhy.SetPcapDataLinkType(YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

• (Optional) set PCAP data link type
• Only “set” PCAP data link type, “enable” later

<table>
<thead>
<tr>
<th>DLT_IEEE802_11 (default)</th>
<th>802.11 headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLT_PRISM_HEADER</td>
<td>Include Prism monitor mode info.</td>
</tr>
<tr>
<td>DLT_IEEE802_11_RADIO</td>
<td>Include Radiotap link layer info.</td>
</tr>
<tr>
<td>No.</td>
<td>Time</td>
</tr>
<tr>
<td>-----</td>
<td>---------------</td>
</tr>
<tr>
<td>1</td>
<td>0000000000</td>
</tr>
</tbody>
</table>

Frame 1: 1086 bytes on wire (8688 bits), 1086 bytes captured (8688 bits)

- Radiotap Header v0, Length 22
  - Header revision: 0
  - Header pad: 0
  - Header length: 22
  - Present flags
    - MAC timestamp: 10000000
  - Flags: 0x10
    - Data Rate: 6.0 Mb/s
    - Channel frequency: 5005 [A 1]
  - Channel type: 802.11a (0x0140)

- IEEE 802.11 Data, Flags: o........
- Logical-Link Control
- User Datagram Protocol, Src Port: 49153 (49153), Dst Port: http (80)
- Data (1000 bytes)
signal and noise
WifiChannel (Loss/Delay)

YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel",
   "Exponent", DoubleValue (3.0));
wifiPhy.SetChannel (wifiChannel.Create ());

- “Blank” YansWifiChannelHelper
- Manually set PropogationDelayModel to ConstantSpeed
- Manually “add” LogDistancePropagationLossModel to the chain of PropagationLossModels
- ns3::YansWifiChannelHelper::Default () sets Delay to ConstantSpeed and already has LogDistance in the chain of PropagationLossModels (e.g. calling AddPropagationLoss again will add to the chain)
Wi-Fi Device

NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

// Enable PCAP
wifiPhy.EnablePcap ("wifi-adhoc", devices);

• Finally install the device on nodes

• Enable PCAP output if needed
Mobility Model

• Initial position (partial list)
  • List
  • RandomRectangle

• Mobility model (partial list)
  • Constant
  • Random Waypoint (see [6])
Mobility Model — Initial

// Configure mobility
MobilityHelper mobility;
Ptr<ListPositionAllocator> posAlloc = CreateObject<ListPositionAllocator> ();
posAlloc->Add (Vector (0.0, 0.0, 0.0));
posAlloc->Add (Vector (5.0, 0.0, 0.0));
mobility.SetPositionAllocator (posAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (c);

• Here: ListPositionAllocator

• Other options (partial list):
  • RandomRectanglePositionAllocator
  • GridPositionAllocator
Mobility Model — Mobility

// Configure mobility
MobilityHelper mobility;
Ptr<ListPositionAllocator> posAlloc = CreateObject<ListPositionAllocator>();
posAlloc->Add(Vector (0.0, 0.0, 0.0));
posAlloc->Add(Vector (5.0, 0.0, 0.0));
mobility.SetPositionAllocator (posAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (c);

• Here: ConstantPositionMobilityModel

• Other options (partial list):
  • RandomWalk2dMobilityModel
  • RandomWaypointMobilityModel
Others…

• Internet stack
• Routing
• Application
• Statistics e.g. FlowMonitor
Routing Protocols

• Ad-hoc routing protocol examples (IPv4 only):
  • DSR
  • AODV
  • OLSR

• Different set up (see manet-routing-compare.cc)

• Some are sensitive to start time (more later)

• Also possible: calculate routes offline and use static routes
Frequently Asked Questions
Transmission Range

• Related questions:
  • How many nodes do I need?
  • Planned network (e.g. grid/linear)
Transmission Range

- Depends on many factors
- Transmission mode (6Mbps vs. 54Mbps)
- Propagation loss model
- Frame size (big vs. small)
- etc.
Transmission Range (cont.)

• Simple set up with two nodes — source and sink

• Source — broadcast UDP @100pkt/s for 1000s

• Places the two nodes at different distances, run the application then count the number of packets received at different distances
Changing Wi-Fi Mode

Distance (m)

Probability

54Mbps

12Mbps
Changing Frame Size

- 128 bytes
- 1920 bytes

Distance (m)

Probability

37 38 39 40 41 42 43 44 45 46 47 48 49 50 51

Distance (m)
Simulation Setup

• Higher data rate = shorter range
  • e.g. 802.11g @ 54Mbps ~ 40 meters
• Need (very!) high density network
• More nodes = more simulation time
  • Simulation can be really, really slow
Implications on Routing

• Broadcast is sent using the slowest rate while unicast is usually sent using higher rate

• Common: build routes using broadcast, send data using unicast

• Route is OK when “build”, bad when “use”
Examples (Grid Network)

- Select propagation loss model
- e.g. log-distance is tricky
- Do you want diagonal links?
- No? Range or matrix may work
- Also applies to linear network
Propagation Loss Models

- Some propagation loss models have parameters that depend on the channel frequency
  - e.g. reference loss of log-distance
- Check for frequency-dependent variables!
- Most default values of ns-3 are for 802.11a (5 GHz)
- Setting up the loss model correctly
Wi-Fi Rate Manager

• Multiple Wi-Fi rate managers are available

• Depends on your simulation setup

• Most common is probably ConstantRate
  • Settable: rate for Data and Control (RTS)
  • CTS/ACK are not “directly” settable (see [3])
  • Rate of non-unicast is set at WifiRateManager
Setting up Applications

• Timing is very important in wireless simulations

• Setting multiple applications to start at the same time is not a good idea in general
  (ApplicationContainer::Start () sets all in container)

• Nodes are likely to initiate route building process at the exact same time
  • collision is almost guaranteed
Maximum “Application” Rate

- The maximum achievable “application” data rate is not the same as the “Wi-Fi” rate due to:
  
  - Timing (IFS)
  
  - RTS/CTS/ACK
  
  - Multihop (shared medium)
  
- Avoid setting application rate == Wi-Fi rate
Mixed-mode Simulation

• Currently 802.11g is compatible with 802.11b
• E.g. timing
• DSSS preamble + OFDM frame is not supported
Interesting Examples

- examples/wireless/wifi-simple-adhoc.cc
  - Simple ad hoc with two nodes (fixed RSS loss)

- examples/wireless/wifi-simple-infra.cc
  - Infrastructure mode (1 AP + 1 STA)

- examples/routing/manet-routing-compare.cc
  - Setting up different routing protocols
Future Work to Expect in 2014

- Wi-Fi Sleep Mode
- A-MPDU
- 802.11n
References and other interesting papers


