Appendix

In this talk

- Wireless simulation (Ad hoc)
- 802.11 MAC in brief

Wireless simulation (Ad hoc)

Two nodes example

- Similar to the example in Marc Greis' Tutorial
- But
  - No mobility
  - TCP throughput measure

% cat wireless.tcl

- Initiated simulator instance
  - *set ns [new Simulator]*
- Trace-all is required
  - *set tracefd [open trace.tr w]*
    $ns trace-all $tracefd

    - Running without trace file
      % ns wireless.tcl
      Warning: You have not defined you tracefile yet!
      ……

Nam trace

- Interface of nam is changed
  - *set namtracefd [open trace.nam w]*
  - *$ns namtrace-all-wireless $namtracefd 500 500*
    - *Last two parameters of 500 500 is the space for mobility*

Topography

- Set up topography object
  - *set topo [new Topography]*
    $topo load_flatgrid 500 500

    - *Keep track of movements of nodes*
    - *Last two parameters are the topological boundary*

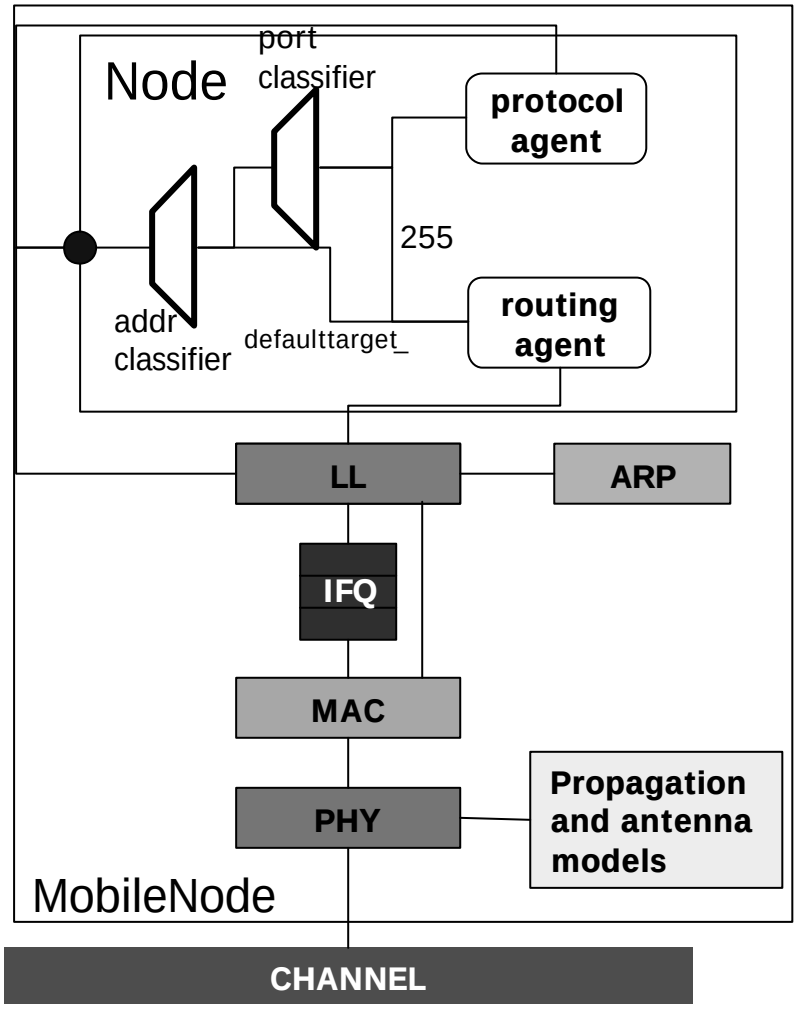
God and channel

- Create God object
  - *create-god 2*
    - General Operations Director
    - Store the total number of nodes and a table of shortest number of hops required to reach from one to another
- Create channel
  - *set chan_ [new Channel/WirelessChannel]*

Node configuration

- Configure before creating nodes
  - *$ns node-config*
    -adhocRouting DSDV
    -llType LL
    -macType Mac/802_11
    -ifqType Queue/DropTail/PriQueue
    -ifqLen 50
    -antType Antenna/OmniAntenna
    -propType Propagation/TwoRayGround
    *-channel $chan_*
    -topoInstance $topo
    -agentTrace ON
    -routerTrace OFF
    -macTrace ON
    -movementTrace OFF

Mobile node



Classifier: Forwarding

Agent: Protocol Entity

Node Entry

LL: Link layer object

IFQ: Interface queue

MAC: Mac object

PHY: Net interface

From Polly Hwang's Presentation

Create nodes

- Create nodes
  - *set node_(0) [$ns node]*
    $node_(0) random-motion 0
    set node_(1) [$ns node]
    $node_(1) random-motion 0

    - Random-motion set to 1 for random movement

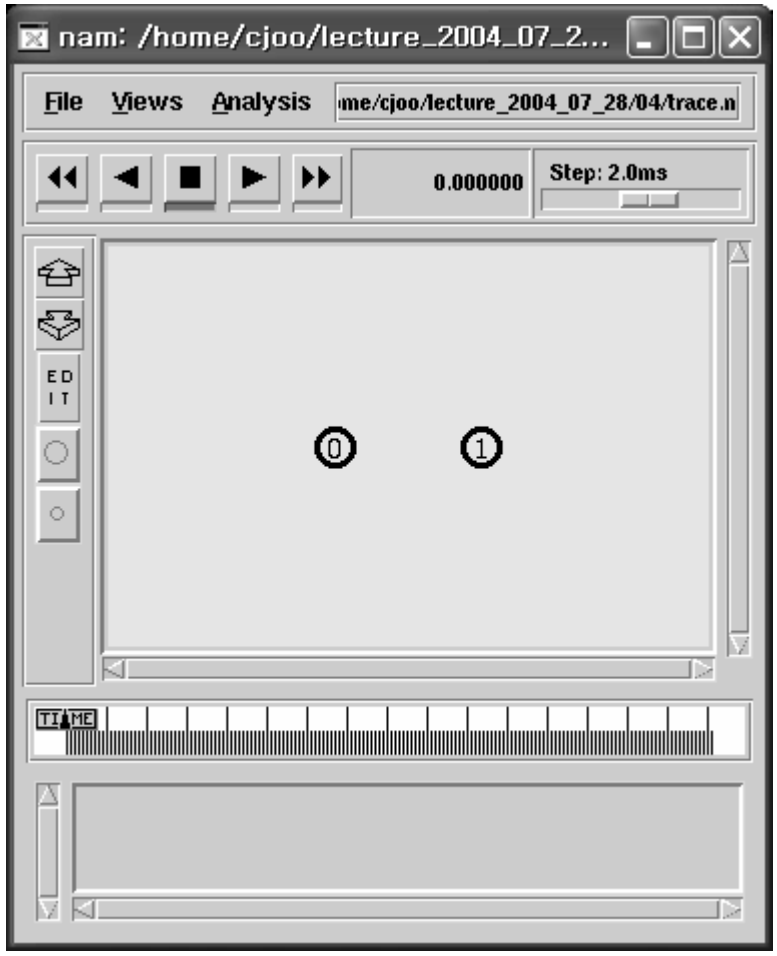
Set positions

- (X,Y,Z)
  - *$node_(0) set X_ 150.0*
    $node_(0) set Y_ 250.0
    $node_(0) set Z_ 0.0
    $node_(1) set X_ 350.0
    $node_(1) set Y_ 250.0
    $node_(1) set Z_ 0.0

    - Z-axis is not supported at this time

Node sizes in nam

- It is for node size in animation
  - *$ns initial_node_pos $node_(0) 50*
    $ns initial_node_pos $node_(0) 50
    - Not initial position, but node size in NAM
    - Nodes are shown in NAM, but packets are not at this time



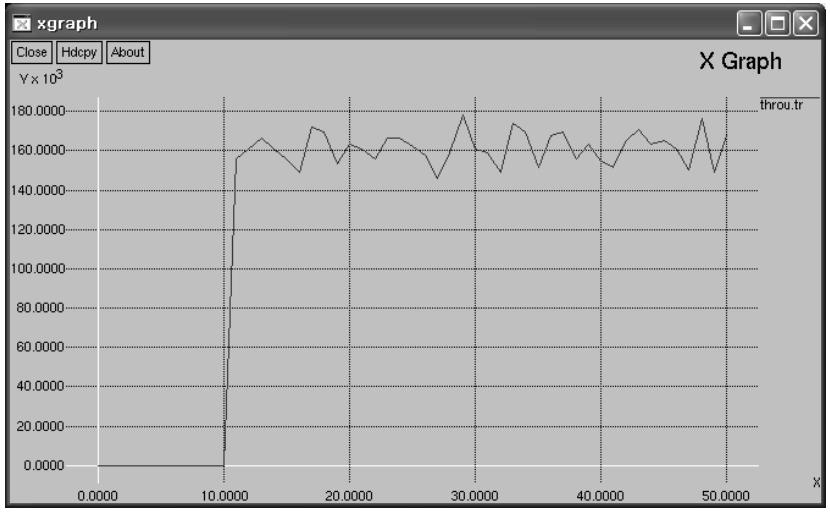
Remainder

- Remaining part is similar to wired simulation
  - Initiate agents and attach them to nodes
  - Initiate application and attach it to node
  - Define self-calling procedure for periodic polling
  - Run the simulation

Running example

- *% ns wireless.tcl*
  - *num_nodes is set 2*
    INITIALIZE THE LIST xListHead
    Starting Simulation…
    channel.cc:sendUp – Calc highestAntennaZ_...
    highestAntennaZ_ = 1.5, distCST_ = 550.0
    SORTING LISTS …DONE!
    NS EXITING…

Results



- *% xgraph throu.tr*
  - TCP throughput
  - Measured with interval of 1 second
- *% head trace.tr*
  s 0.029365548 _1_ MAC --- 0 message 84 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0] …

Wireless trace output format

NS – CMU MAC Trace Format

r 0.041142306 _1_ AGT --- 0 tcp 80 [13a 1 0 800] ------- [0:0 1:0 32 1] [0 0] 1 0

r : send, receive, drop, forwarding - s / r / D / f

0.041142306 : timestamp

_1_ : node ID for this node

AGT : name of object type tracing or trace level
(AGent Trace, Router Trace, MAC, and so on)

--- : reason for tracing

0 : packet identifier

tcp : packet type

80 : packet size (of mac?)

13a : expected time to send data in hexa

1 : MAC destination address

0 : MAC source address

800 : type – ARP (0x806) / IP (0x800)

----

0 : source IP address

0 : source port number

1 : destination IP address in decimal
(8.8.8 format – 36 implies 0.1.0)

0 : destination port number

32 : TTL

1 : next hop address

0 : TCP sequence number 0

0 : TCP ack number 0

1 : ?

0 : ?

802.11 MAC in brief

Feature of 802.11 MAC

- Contention based medium access

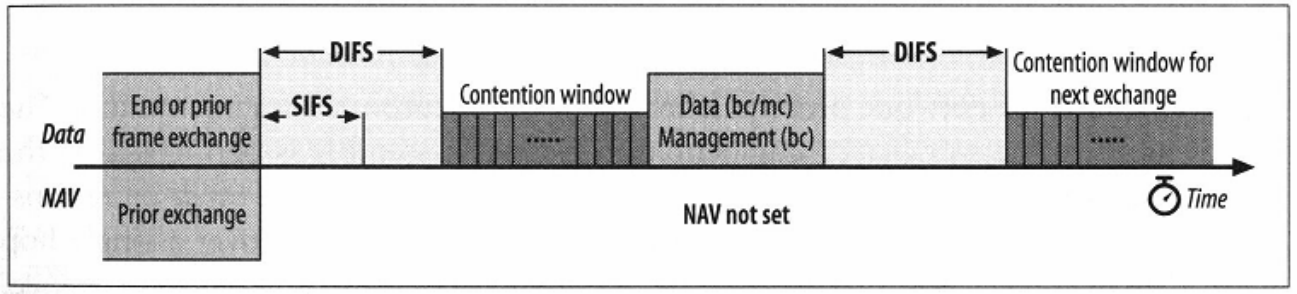


Figure from “802.11 wireless networks, The definite guide”

- RTS-CTS-DATA-ACK Exchange
  - For hidden terminal problem

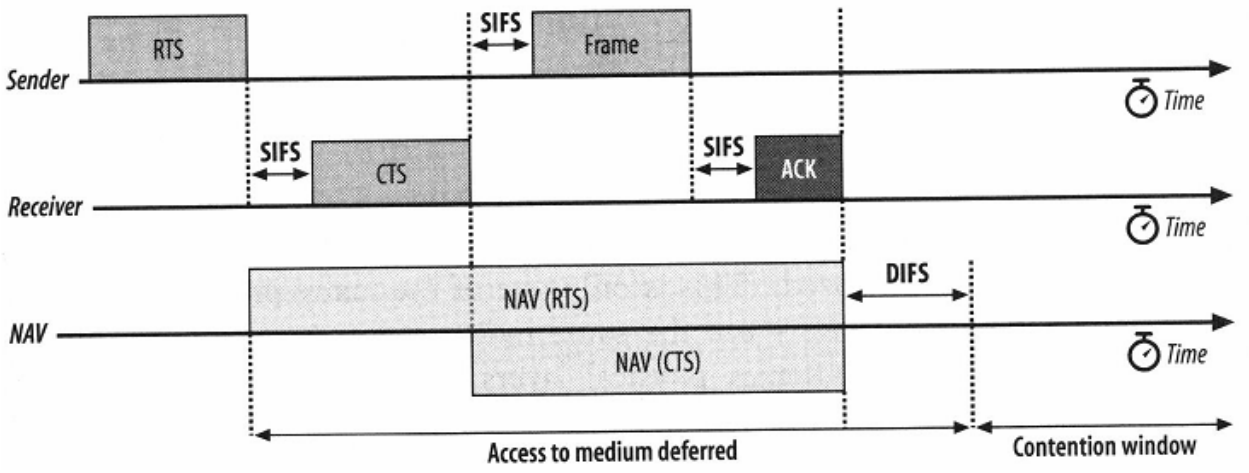


Figure from “802.11 wireless networks, The definite guide”

802.11 in NS-2

- Defined in *$ns/mac/mac-802_11.{h,cc}*
  - class Mac802_11 inherits class Mac defined in *$ns/mac/mac{h.cc}*
  - Timers are defined in *$ns/mac/mac-timers.{h.cc}*

Frame (packet) header

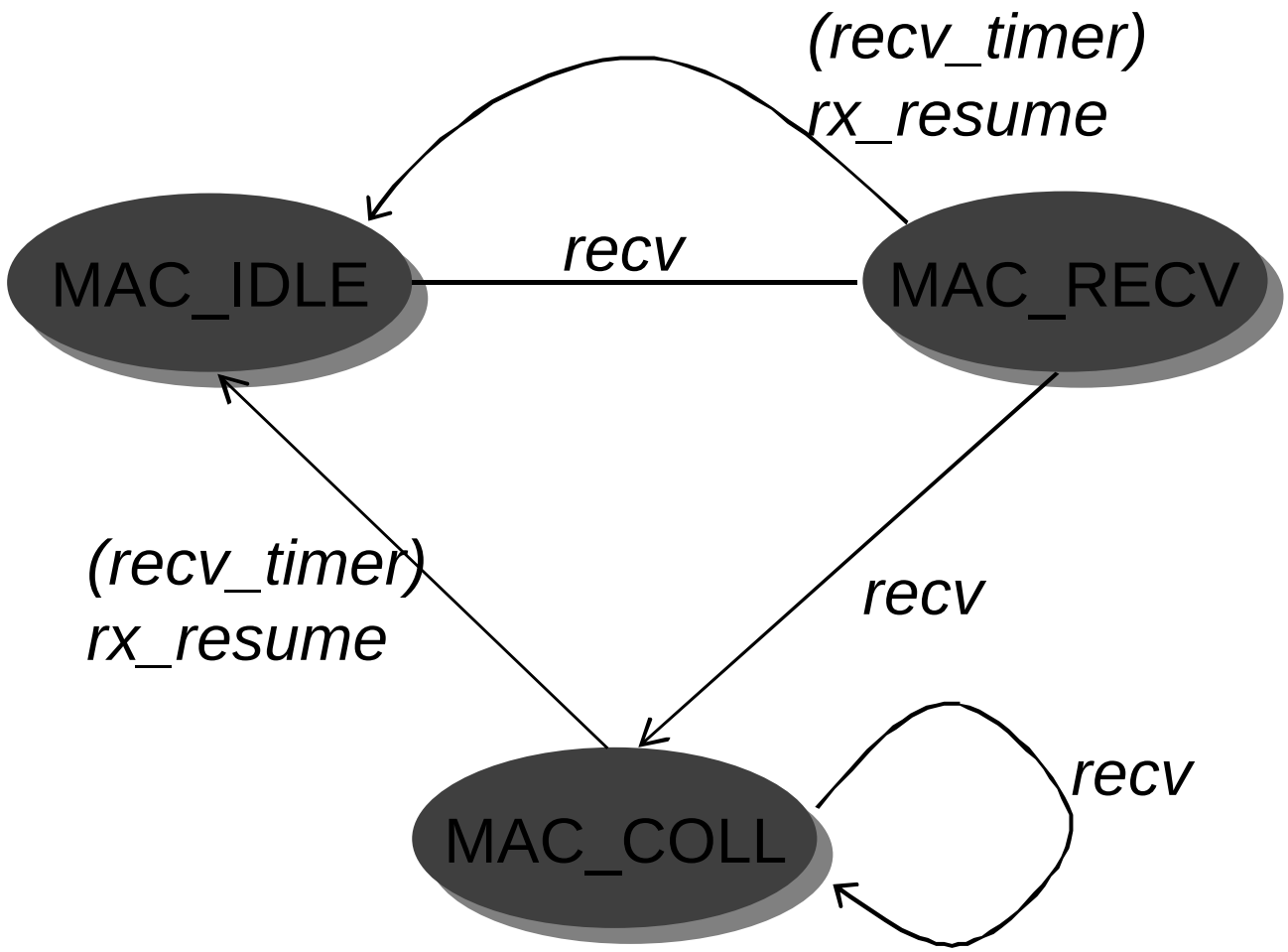
- Defined for each frame types
- Frame types of 802.11 MAC
  - Management
  - Control
    - RTS, CTS, ACK
  - Data

- Caution
  - Modification in frame may result in frame size
  - Implementation depends on *sizeof(struct)*

States

- RxState
  - MAC_IDLE
  - MAC_RECV
  - MAC_COLL
- TxState
  - MAC_IDLE
  - MAC_SEND
  - MAC_RTS
  - MAC_CTS
  - MAC_ACK

RxState


(recv_timer)
rx_resume
MAC_IDLE
recv
MAC_RECV
(recv_timer)
rx_resume
recv
MAC_COLL
recv

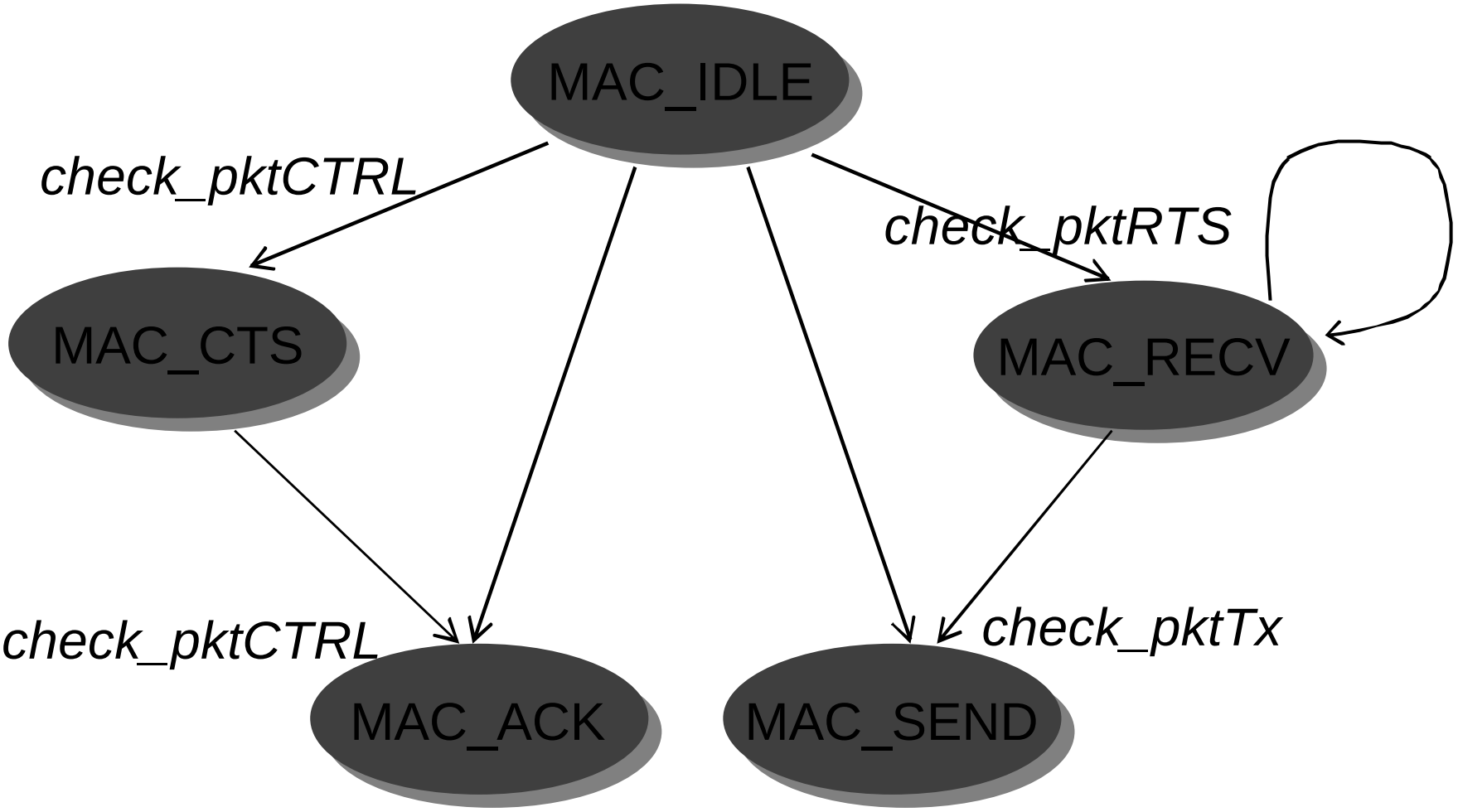
RxTimer

- Function *recv* is called when the first bit of a packet is received
- *recv* schedules *recv_timer*
  - *mhRecv_.start(txtime(p));*
    - *txtime* returns transmission time of packet *p*
  - RxTimer expires calling *recvHandler*
    - *mac-timer.cc*
  - *recvHandler* calls *recv_timer*

Function recv_timer()

- Called when the last bit of a packet is received
  - Check for collision and error
  - Set NAV
  - Address filtering
  - Calls functions based on types
    - *recvRTS*
    - *recvCTS*
    - *recvACK*
    - *recvDATA*
  - Call *rx_resume*

TxState


MAC_IDLE
check_pktCTRL
check_pktRTS
MAC_CTS
MAC_RECV
check_pktCTRL
check_pktTx
MAC_ACK
MAC_SEND

Function transmit()

- Called for the first bit of the transmitting packet to place on the medium
  - *tx_active = 1*
  - *If rx_state is not MAC_IDLE, set error of the receiving packet*
  - *send the transmitting packet*
  - *set TxTimer*
  - *set IFTimer*

TxTimer

- Used to schedule retransmission
- *transmit* schedules *send_timer*
  - *mhSend_.start(timeout);*
    - *timeout* includes transmission time of this and its response (CTS or ACK), and their maximum propagation delay
  - TxTimer expires calling *sendHandler*
    - *mac-timer.cc*
  - *sendHandler* calls *send_timer*

IFTimer

- Used to unset *tx_active*
- *transmit* schedules *txHandler*
  - *mhIF_.start(txtime(p));*
  - TxTimer expires calling *txHandler*
    - *mac-timer.cc*
  - *txHandler* unset *tx_active*

RTS/DATA procedure

- *recv* calls *send*
  - *recv* is called by both upper and lower layer
  - Packets are ready before backoff
    - *sendDATA, sendRTS*
  - Start backoff timer (*mhBackoff*)
  - When expires, calls *backoffHandler*
- Function *backoffHandler()*
  - If it has RTS, calls *check_pktRTS*
  - If it has data packet, calls *check_pktTx*

CTS/ACK procedure

- *recv_timer* calls *recvRTS* or *recvDATA*
- *recvRTS* or *recvDATA*
  - Prepare CTS and ACK
  - Calls *tx_resume*
- *tx_resume* sets DeferTimer for SIFS
  - When expires calls *deferHandler*
- Function *deferHandler()*
  - If it has CTS or ACK, calls *check_pktCTRL*

Timers

- Defined in *$ns/mac/mac-timers.{h,cc}*
- DeferTimer
  - To wait for SIFS/DIFS/EIFS
- BeaconTimer
- IFTimer
  - Unset *tx_active*
- NavTimer
  - To resume backoff timer
- RxTimer
  - To get when it receives the last bit of the packet
- TxTimer
  - For packet retransmission
- BackoffTimer
  - Implement 802.11 backoff
  - It can pause and resume

Get the next packet

- Who calls 802.11 MAC's recv ?
  - IFQ, but When?
- *callback_*
  - Set to IFQ by *send()*
  - *tx_resume* calls *callback_->handle()* when there is nothing to send
  - *tx_resume* is called by *recvRTS, recvCTS, recvDATA, recvACK* and by *send_timer*
  - *callback_->handle* calls *resume* of IFQ