

# EEC 687/787 Mobile Computing (Spring, 2009)

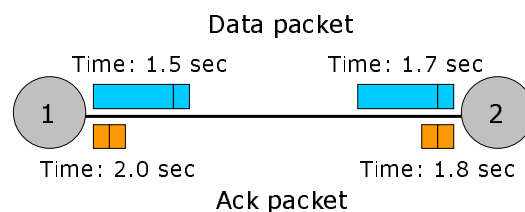
## Ns-2 Laboratory #6

**Prof. Chansu Yu**

<http://academic.csuohio.edu/yuc/>  
[c.yu91@csuohio.edu](mailto:c.yu91@csuohio.edu)

## ns-2: Discrete Event Simulator

- ❑ ns-2 is an discrete event driven simulation
  - Physical activities are translated to events
  - Events are queued and processed in the order of their scheduled occurrences
  - Time progresses as the events are processed



## Understanding ns-2 implementation

- ❑ 802.11 implementation in ns-2
  - Ns2 code overview
    - mac-802\_11.h
    - mac-802\_11.cc : recv() function
    - mac-802\_11.cc : send() function
  - Trace and monitoring support to obtain other outputs than standard trace output provides

3

*c.yu91@csuohio.edu*

## Overview

- ❑ Ns2 supports two MAC layer protocols for mobile networks
  - 802.11
  - TDMA
- ❑ The 802.11 is implemented in ~ns/mac/mac-802\_11.cc,  
~ns/mac/mac-802\_11.h
- ❑ The 802.11 MAC doesn't try to retransmit a broadcast packet in case there is a collision (the packet is dropped)

4

*c.yu91@csuohio.edu*

## Frame formats (mac-802\_11.h)

```
# define MAC_Type_Management
# define MAC_Type_Data

//Packet type
#define MAC_Subtype_RTS
#define MAC_Subtype_CTS
#define MAC_Subtype_ACK
#define MAC_Subtype_Data

// BSS type
#define BSS_Infrastructure
#define BSS_Adhoc
```

5

c.yu91@csuohio.edu

## Frame formats (mac-802\_11.h)

- For each of the subtypes, there are structures defined

```
struct rts_frame {
    struct frame_control rf_fc;
    u_int16_t          cf_duration;
    u_char             rf_ra[ETHER_ADDR_LEN];
    u_char             rf_ta[ETHER_ADDR_LEN];
    u_char             rf_fcs[ETHER_FCS_LEN];
};
```

6

c.yu91@csuohio.edu

## Internal MAC State (mac-802\_11.h)

- ❑ Defines Network allocation vector, incoming state, outgoing state and the transmission state

```
double nav_; //Network Allocation Vector
MacState rx_state; //incoming state

MacState tx_state; //outgoing state

int tx_active_; //Transmitter is active

u_int32_t cw_; //Contention Window

u_int16_t sta_seqno;
int cache_node_count_;
Host *cache_;
```

7

c.yu91@csuohio.edu

## 802.11 MAC class (mac-802\_11.h)

- ❑ Public methods and definitions

```
void rcv(Packet *p, Handler *h);
void trace_event(char *, Packet *);
EventTrace *et_;
inline int hdr_dst(char* hdr, int dst = -2);
inline int hdr_src(char* hdr, int src = -2);
inline int hdr_type(char* hdr, u_int16_t type = 0);

//To identify the BSS
inline int bss_id() {return bss_id; }
```

- ❑ Protected methods

```
void backoffHandler(void);
void deferHandler(void);
void beaconHandler(void);
void navHandler(void);
void rcvHandler(void);
void sendHandler(void);
void txHandler(void);
```

8

c.yu91@csuohio.edu

## 802.11 MAC class(mac-802\_11.h)

### Private methods

```

//Called by the timers
void recv_timer(void);
void send_timer(void);
int check_pktCTRL();
int check_pktRTS();
int check_pktTx();

//Packet Transmission Functions
void send(Packet *p, Handler *h);
void sendRTS(int dst);
void sendCTS(int dst, double duration);
void sendACK(int dst);
void sendData(Packet *p);
void RetransmitRTS();
void RetransmitDATA();

//Packet Transmission Functions
void recvRTS(Packet *p);
void recvCTS(Packet *p);
void recvACK(Packet *p);
void recvDATA(Packet *p);

```

9

c.yu91@csuohio.edu

## 802.11 MAC timers(mac-802\_11.h, mac-timers.h/cc)

Class	instance	method	functions
<b>DeferTimer</b>	<b>mhDefer_</b>	<b>start()</b>	<b>schedule()</b>
		<b>handle()</b>	<b>deferHandler()</b>
<b>BackoffTimer</b>	<b>mhBackoff_</b>	<b>start()</b>	<b>schedule()</b>
		<b>pause()</b>	<b>cancel()</b>
		<b>resume()</b>	<b>schedule()</b>
		<b>handle()</b>	<b>backoffHandler()</b>
<b>TxTimer</b>	<b>mhSend_</b>	<b>handle()</b>	<b>sendHandler()</b>
<b>IFTimer</b>	<b>mhIF_</b>	<b>handle()</b>	<b>txHandler()</b>
<b>RxTimer</b>	<b>mhRecv_</b>	<b>handle()</b>	<b>recvHandler()</b>
<b>NavTimer</b>	<b>mhNav_</b>	<b>handle()</b>	<b>navHandler()</b>

10

c.yu91@csuohio.edu

## 802.11 MAC timers(mac-802\_11.h, mac-timers.h/cc)

Class	instance	method	functions
<i>DeferTimer</i>	<i>mhDefer_</i>	<i>start()</i> <i>handle()</i>	<i>schedule()</i> <i>deferHandler()</i>

- (i) When is *mhDefer\_.start(...)* called?
- (ii) What is the time parameter?
- (iii) When is *mhDefer\_.handle(...)* called?
- (iv) What does *deferHandler()* do?

11

*c.yu91@csuohio.edu*

## 802.11 MAC timers(mac-802\_11.h, mac-timers.h/cc)

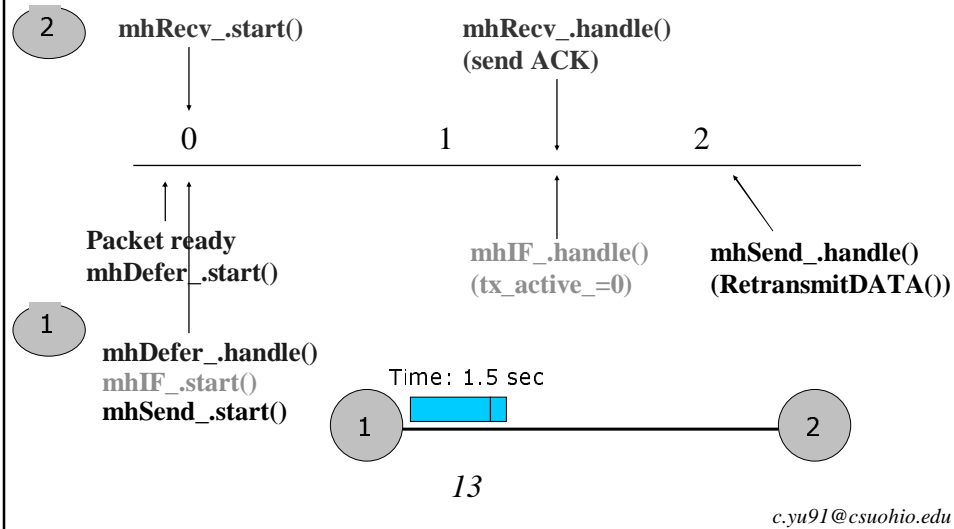
Class	instance	method	functions
<i>TxTimer</i>	<i>mhSend_</i>	<i>handle()</i>	<i>sendHandler()</i>

- (i) When is *mhSend\_.start()* called?
- (ii) What is the time parameter?
- (iii) When is *mhSend\_.handle()* called?
- (iv) What does *sendhandler()* do?

12

*c.yu91@csuohio.edu*

## ns-2: Discrete Event Simulator



## Understanding ns-2 implementation

- 802.11 implementation in ns-2
  - Ns2 code overview
    - mac-802\_11.h
    - mac-802\_11.cc : recv() function
    - mac-802\_11.cc : send() function
  - Trace and monitoring support to obtain other outputs than standard trace output provides

## recv()

- ❑ The packet to be sent is received by the recv() function.
- ❑ recv() also checks the direction field in the packet header.
- ❑ This function is called whenever a packet is received from either an upper or lower layer
  
- ❑ recv()(DOWN): The direction is DOWN, meaning the packet came from an upper layer, and is passed to send() function.
- ❑ recv()(UP): The direction is UP, meaning the packet came from a lower layer.

15

*c.yu91@csuohio.edu*

## There are four different paths the code can follow:

- ❑ Down
  - Transmitting a packet
  
- ❑ Up
  - Receiving a packet destined for itself
  - Overhearing a packet not destined for itself
  - Packets colliding (capture or collision)

16

*c.yu91@csuohio.edu*

## Packets Colliding

- ❑ capture()
  - This function is called when a second packet is received while the the MAC is currently receiving another packet, but the second packet is weak enough that it can be ignored
- ❑ collision()
  - The collision handler first checks the rx\_state\_variable and sets it to MAC\_COLL
  - The MAC calculates how much longer the new packet will last and how much longer the old packet will last.

17

c.yu91@csuohio.edu

```
Mac802_11::recv(Packet *p, Handler *h)
{
    struct hdr_cmn *hdr = HDR_CMN(p);

    /* Handle outgoing packets. */
    if(hdr->direction() == hdr_cmn::DOWN) {
        send(p, h);
        return;
    }

    /* Handle incoming packets.
     * We just received the 1st bit of a packet on the
     * network interface. */
    =>If medium idle, set timer (event queue)
    =>Otherwise, call collision() or capture()
}
```

c.yu91@csuohio.edu

## recv\_timer()

- ❑ The receive timer handler is called when mhRecv\_ expires.
- ❑ The expiration of receive timer means that a packet has been fully received and can now be acted upon.

19

*c.yu91@csuohio.edu*

## recv\_timer()

Depending on packet type,

- ❑ recvRTS() : call sendCTS() & tx\_resume()
- ❑ recvCTS() : call tx\_resume()
- ❑ recvDATA() : call sendACK() & tx\_resume()
- ❑ recvACK()
  
- ❑ rx\_resume(): This function is called after recv\_timer has completed. It sets the rx\_state\_ to idle and then invoke CHECK\_BACKOFF\_TIMER
- ❑ backoffHandler(): This function is called whenever the backoff timer expires.

20

*c.yu91@csuohio.edu*

## Understanding ns-2 implementation

### □ 802.11 implementation in ns-2

- Ns2 code overview
  - mac-802\_11.h
  - mac-802\_11.cc : recv() function
  - mac-802\_11.cc : send() function
- Trace and monitoring support to obtain other outputs than standard trace output provides

21

*c.yu91@csuohio.edu*

## tx\_resume()

- If there is CTS or ACK to send (pktCTRL\_)
  - Wait for SIFS by calling  
`mhDefer_.start(phymib_.getSIFS());`
- If there is RTS to send (pktRTS\_)
  - Wait for DIFS and random backoff time by calling  
`mhDefer_.start( phymib_.getDIFS() + rTime);`
  - where,  $rTime = (Random::random() \% cw\_)$  \* `phymib_.getSlotTime();`
- If there is DATA to send (pktTx\_)
  - Wait for either one of the above depending on packet size

22

*c.yu91@csuohio.edu*

## send()

- ❑ Simply call
  - sendDATA() and sendRTS()
  - They'll make pktRTS\_ and pktTx\_ non-null !!!
  
- ❑ As before, it needs to defer or backoff
  - If the channel (medium) is idle, the node will begin to defer.
  - If the medium is detected to be busy, then the node starts its backoff timer.

As of this point, the send() function has finished and control will resume when one of the timers expires, calling either deferHandler() or backoffHandler().

23

*c.yu91@csuohio.edu*

## Detailed Steps of send() Function

- ❑ The send() function first checks the energy model, dropping the packet if the node is currently in sleep mode.
  
- ❑ It then sets callback\_ to the handler passed along with the packet.
  
- ❑ Next, send() calls sendDATA() and sendRTS which build the MAC header for the data packet and the RTS packet to go along with the data packet – which are stored in pktTx\_ and pktRTS\_ respectively.
  
- ❑ The MAC header for the data packet is then assigned a unique sequence number (with respect to the node).

24

*c.yu91@csuohio.edu*

## TRANSMIT

- ❑ This macro takes 2 arguments
  - packet
  - a timeout value
- ❑ Sets a flag variable (tx\_active) to indicate that the MAC is currently transmitting a packet
- ❑ Finally 2 timers is started
  - *sendtimer()* with timeout value which will alert MAC about failure of transmission .
  - (*mhIF\_*) timer which when expires , Mac will know that Phy has completed transmission of the packet .

25

c.yu91@csuohio.edu

## The “real” Timer Handler Routines

```
Mac802_11::send_timer()
{
    switch(tx_state_) {
        /*
         * Sent a RTS, but did not receive a CTS.
         */
        case MAC_RTS:
            RetransmitRTS();
            break;
        /*
         * Sent a CTS, but did not receive a DATA packet.
         */
        case MAC_CTS:
            assert(pktCTRL_);
            Packet::free(pktCTRL_); pktCTRL_ = 0;
            break;
```

When is it called?

26

c.yu91@csuohio.edu

```

* Sent DATA, but did not receive an ACK packet.
*/
case MAC_SEND:
    RetransmitDATA();
    break;
/*
* Sent an ACK, and now ready to resume transmission.
*/
case MAC_ACK:
    assert(pktCTRL_);
    Packet::free(pktCTRL_); pktCTRL_ = 0;
    break;
case MAC_IDLE:
    break;
default:
    assert(0);
}tx_resume();

```

27

c.yu91@csuohio.edu

## send\_timer()

- This function is called at the expiration of the TxTimer, mhSend\_.
- The last packet sent was an RTS.
  - The expiration of the timer means a CTS wasn't received, presumably because the RTS collided or the receiving node is deferring.
  - The MAC responds by attempting to retransmit the RTS using `RetransmitRTS()`.
- If the last packet sent was a CTS packet.
  - The expiration of the timer means that no data packet was received.
  - This is an infrequent event occurring if the CTS packet collided or if the data packet was in error.
  - The MAC handles this by simply resetting itself to an idle state. This involves freeing the CTS packet stored in `pktCTRL_`.
- If the last packet sent was a data packet
  - the expiration of the timer means that an ACK was not received.
  - The MAC handles this situation by calling `RetransmitDATA()`.
- If the last packet sent was an ACK.
  - the expiration of the timer simply means that the ACK has been transmitted, as no response is expected from an ACK.
  - The MAC frees the ACK packet pointed to by `pktCTRL_`.

28

c.yu91@csuohio.edu

## send\_timer()

After each case has been handled and a packet has possibly been prepared for retransmission, the function `tx_resume()` is given control.

If a packet is going to be retransmitted, the backoff timer has already been started with an increased congestion window.

[http://www.ece.rice.edu/~jpr/ns/docs/802\\_11.html](http://www.ece.rice.edu/~jpr/ns/docs/802_11.html)  
`ns-2.30/mac/mac-timers.{cc, h}`

29

*c.yu91@csuohio.edu*

## Lab Report

- ❑ Investigate the lifetime of a packet.
  - `ns rate.tcl -rate 12e6 -rxthresh 1.0e-10.9 -cpthresh 1.0e0.903 -csthresh 1.0e-12.1`
  - `rcssim.tr` and `rcssim.nam`
  - Packet number “58689”
  - Node 72 sends the packet to node 10
  - The packet starts at 3.693663448 (node 72) and arrives at 6.313887321 (node 10)

30

*c.yu91@csuohio.edu*

# Lab Report

## Report

- Describe what happens to the packet during its lifetime (based on events)
- If there exists the time difference between two consecutive events regarding the packet 58689, explain why it takes that amount of time.
  - s 4.539931044 \_72\_ MAC --- 58689 cbr 1102 ...
  - r 4.540842210 \_82\_ MAC --- 58689 cbr 1044 ...
- Include events regarding ACK packets, which do not have packet number

## You may want to observe nam

31

*c.yu91@csuohio.edu*