

EEC 687 Mobile Computing (Spring, 2008)

Ns-2 Laboratory #6

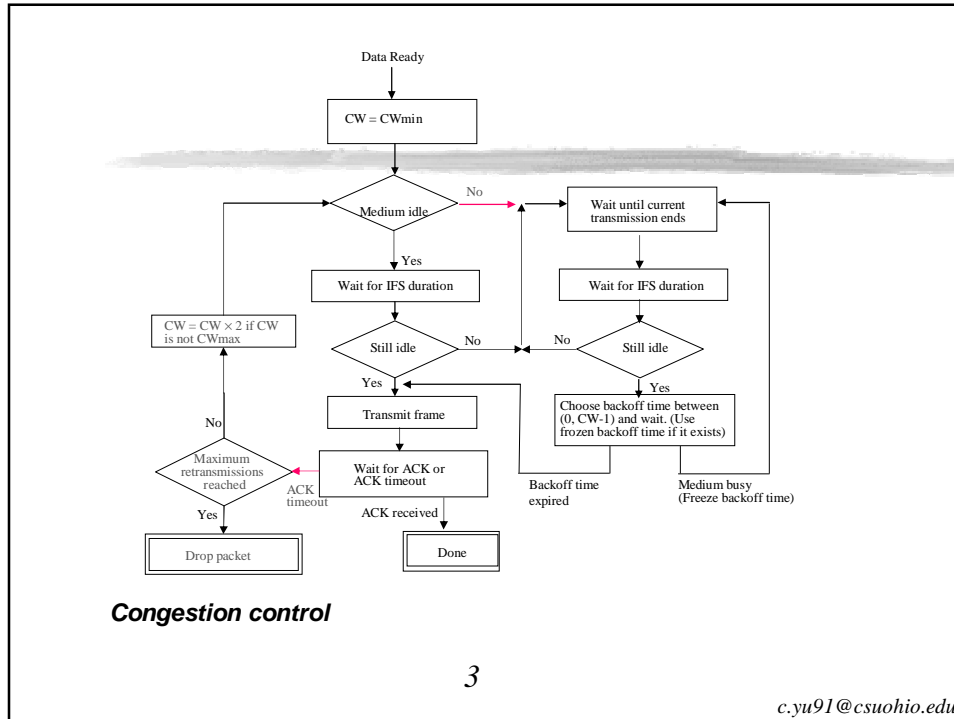
Chansu Yu

Cleveland State University

Effect of CW on Network Performance

- ❑ Binary Exponential Backoff (BEB) in 802.11 MAC
 - Since the number of nodes attempting to transmit simultaneously may change with time, some mechanism to manage congestion is needed

- ❑ IEEE 802.11 DCF: Congestion control achieved by dynamically choosing the contention window cw
 - When a node fails to receive CTS (ACK) in response to its RTS (Data), it increases the contention window: cw is doubled (up to an upper bound)
 - When a node successfully completes a data transfer, it restores cw to CW_{min}



Effect of CW on Network Performance (in-class lab)

- ❑ In this lab, we will consider a variant of 802.11 where the contention window size is fixed, i.e., $CW_{min}=CW_{max}=CW$.
- ❑ We will not dynamically change CW, but still use randomization.
- ❑ We will need a topology under which to study the effect of CW sizes. For this, let us choose a single hop network where all nodes are in range of each other. Specifically we consider a 150m X 150m area. All nodes are involved in two Constant Bit Rate (CBR) conversations: one as source, and one as destination.

Effect of CW on Network Performance (in-class lab)

1. Get the tcl script `cwfixed.tcl`.
2. In the script you will find the following lines that set the values of `CWMin` and `CWMax` to the desired value:

```
$val(mac) set CWMin_ 31  
$val(mac) set CWMax_ 31
```
3. Usage:

```
> ns cwfixed.tcl -rlen 2
```

It creates a topology of $rlen^2$ nodes arranged in a regular grid (150m x 150m grid area)

5

c.yu91@csuohio.edu

Effect of CW on Network Performance (in-class lab)

4. Run the script for `rlen=3, 4 and 5` (i.e. number of nodes=9, 16, 25), and vary `CWMin=CWMax` to take the following set of values: 2, 7, 15, 31, 63, 127 (A total of 18 combinations).
5. Obtain and plot the aggregate CBR throughput and Packet Delivery Ratio (PDR) vs CW size for different `rlen` values.
6. Use “grep” to calculate PDR. To get the throughput, since all flows are a CBR, with a constant packet size of 512 bytes, just multiply number of packets received with 512 x 8 and divide by total simulation time (25 seconds for this example).

6

c.yu91@csuohio.edu

MILD Algorithm

- ❑ When a node fails to receive CTS in response to its RTS, it multiplies cw by 1.5
 - Similar to 802.11, except that 802.11 multiplies by 2
- ❑ When a node successfully completes a transfer, it reduces cw by 1
 - Different from 802.11 where cw is restored to $Cwmin$
 - In 802.11, cw reduces much faster than it increases
 - MILD: cw reduces slower than it increases
Exponential Increase Linear Decrease
- ❑ MILD can avoid wild oscillations of cw when congestion is high

7

c.yu91@csuohio.edu

MILD Algorithm (Lab report)

- ❑ In this lab, we will consider the MILD algorithm where CW increases by 50% and decreases by one.
- ❑ How to implement MILD?
 - Modify ns-802_11.cc file

8

c.yu91@csuohio.edu

MILD Algorithm (Lab report)

- ❑ Use the same network environment as in the in-class lab.
 - A single hop network in a 150m X 150m area
 - Number of nodes=9, 16, 25
 - The same traffic as in cwfixed.tcl
- ❑ Obtain and plot the aggregate CBR throughput and PDR vs CW size for different number of nodes. Repeat the same set of simulations for the original 802.11 MAC.
- ❑ Discussion
 - What trends do you observe? Which one is better?
 - Can you predict what would happen if you try a heavier and a lighter condition?

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

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)

11

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
```

12

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];
}
```

13

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_;
```

14

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
    => (1) which states are there and what do they mean?
MacState tx_state; //outgoing state
    => (2) which states are there and what do they mean?
int tx_active_; //Transmitter is active
    => (3) what does that mean?
u_int32_t cw_; //Contention Window
    => (4) when does it change?

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

15

c.yu91@csuohio.edu

802.11 MAC class (mac-802_11.h)

- ❑ Public methods and definitions

```
void recv(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 recvHandler(void);
void sendHandler(void);
void txHandler(void);
```

16

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);
```

17

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

18

c.yu91@csuohio.edu

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.

19

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)

20

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.

21

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.

23

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.

24

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

25

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

26

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().

27

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).

28

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 .

29

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?

30

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();

```

31

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_`.

32

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.