# EEC 687/787 Mobile Computing (Spring, 2007)

Ns-2 Laboratory, Week 5 (February 12)

Chansu Yu

Cleveland State University

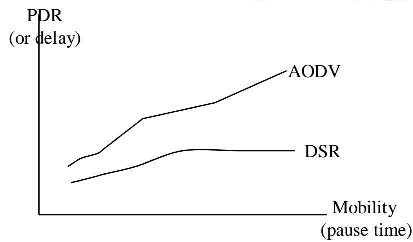
# Homework #2: Routing layer (due Monday, Feb. 26)

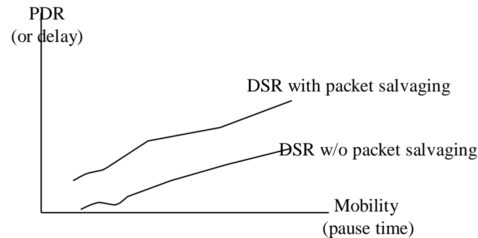
- Goal: Network protocol is the most difficult part in mobile networks as new routing paths must be discovered dynamically in the presence of node mobility. This homework is to compare two most popular routing layer protocols (DSR and AODV) to understand the dynamics of those protocols.
- Simulation input: 50-node in $300x1500m^2$ network, 802.11 MAC, TwoRayGround, 0~20m/second speed, 30 CBR sources, 4 512-byte packets/second (use cbrgen.tcl)
- Simulation variation: Pause time of 0, 20, 50, 100, 300, 500, 900 seconds
- Simulation output: PDR (packet delivery ratio) and delay
- Report: Includes xgraph (or excel) charts with explanation and discussion.
- Bonus: Consider to measure different metrics such as the number of route discoveries or RERR/RREQ/RREP messages generated in case of DSR/AODV. Consider also to enable/disable features of DSR/AODV to see their effects on performance. (See DSR/AODV source files to know the features- “selectors” in dsragen.cc and aodv.h.)

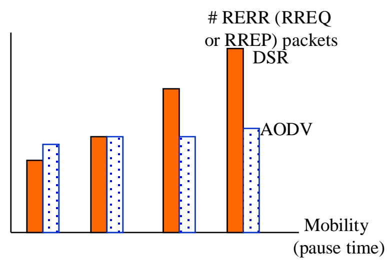
2

*c.yu91@csuohio.edu*

# Homework #2: Routing layer




References
(1) J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu and J. Jetcheva, "**Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols**," MobiCom'98, Oct. 1998.
(2) S. R. Das, C. E. Perkins, E. M. Royer, "**Performance Comparison of Two On-Demand Routing Protocols for Ad Hoc Networks**," INFOCOM 2000.



*c.yu91@csuohio.edu*

# dsragent.cc

```
/************** selectors *****************/
bool dsragent_snoop_forwarded_errors = true;          // give errors we forward to our cache?
bool dsragent_snoop_source_routes = true;             // should we snoop on any source routes we see?
bool dsragent_reply_only_to_first_rtreq = false;      // should we only respond to the first route request we receive from
                                                      // a host?
bool dsragent_propagate_last_error = true;            // should we take the data from the last route error msg sent to us
                                                      // and propagate it around on the next propagating route request we do?
                                                      // this is aka grat route error propagation
bool dsragent_send_grat_replies = true;               // should we send gratuitous replies to effect route shortening?
bool dsragent_salvage_with_cache = true;              // should we consult our cache for a route if we get a xmitfailure
                                                      // and salvage the packet using the route if possible
bool dsragent_use_tap = true;                         // should we listen to a promiscuous tap?
bool dsragent_reply_from_cache_on_propagating = true; // should we consult the route cache before propagating rt req's and
                                                      // answer if possible?
bool dsragent_ring_zero_search = true;                // should we send a non-propagating route request as the first action
                                                      // in each route discovery action?
                                                      // NOTE: to completely turn off replying from cache, you should
                                                      // set both dsragent_ring_zero_search and
                                                      // dsragent_reply_from_cache_on_propagating to false
bool dsragent_dont_salvage_bad_replies = true;        // if we have an xmit failure on a packet, and the packet contains a
                                                      // route reply, should we scan the reply to see if contains the dead link?
                                                      // if it does, we won't salvage the packet unless there's something aside
                                                      // from a reply in it (in which case we salvage, but cut out the rt reply)
bool dsragent_require_bi_routes = true;               // do we need to have bidirectional source routes?
                                                      // [XXX this flag doesn't control all the behaviors and code that assume
                                                      // bidirectional links -dam 5/14/98]
```



*c.yu91@csuohio.edu*

# aodv.h

```
#define AODV_LOCAL_REPAIR              //  Allows local repair of routes
#define AODV_LINK_LAYER_DETECTION                    // Allows AODV to use link-layer (802.11) feedback in determining
                                                     // when links are up/down.
#define AODV_USE_LL_METRIC             //  Causes AODV to apply a "smoothing" function to the link layer feedback
                                       //  that is generated by 802.11.  In essence, it requires that RT_MAX_ERROR
                                       //  errors occurs within a window of RT_MAX_ERROR_TIME before the link
                                       //  is considered bad.
//#define AODV_USE_GOD_FEEDBACK // Only applies if AODV_USE_LL_METRIC is defined.
                                       // Causes AODV to apply omniscient knowledge to the feedback received
                                       //  from 802.11.  This may be flawed, because it does not account for congestion.
*/#define MY_ROUTE_TIMEOUT        10  // 100 seconds

#define ACTIVE_ROUTE_TIMEOUT    10                    // 50 seconds
#define REV_ROUTE_LIFE          6                     // 5  seconds
#define BCAST_ID_SAVE           6                     // 3 seconds

// No. of times to do network-wide search before timing out for MAX_RREQ_TIMEOUT sec.
#define RREQ_RETRIES            3
// timeout after doing network-wide search RREQ_RETRIES times
#define MAX_RREQ_TIMEOUT               10.0 //sec

/* Various constants used for the expanding ring search */
#define TTL_START      5
#define TTL_THRESHOLD 7
#define TTL_INCREMENT 2
```

*5*

*c.yu91@csuohio.edu*

# Contents

- ❑ Understanding of simulation-based study
  - Discrete Event Simulation (DES) to better understand ns2 codes
  - Random number generation to understand “seed”
- ❑ 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

*6*

*c.yu91@csuohio.edu*

## Simulation Study

- Simulation study usually based on random numbers (seed)
  - Results vary depending on the selection of seed number
  - One single simulation run does not tell us the true performance metrics
- Methodology
  - Run multiple simulation runs with different seed numbers
  - Take the average
- Is it enough?
  - Observed packet delays are 2.9, 3.0 and 4.1 seconds
  - Observed packet delays are 0.0, 3.0 and 6.0 seconds
  - Averages are 3.0 seconds in both cases but we are more "confident" about this value in the first case.

7

c.yu91@csuohio.edu

## Confidence Interval & Level

- How much are we confident that the TRUE value lies within a certain value interval?
  - Confidence level
  - Confidence interval
- Example: Average packet delay is within [2.9, 3.1] seconds with probability 95%.
  - Confidence level: 95%
  - Confidence interval: [2.9, 3.1] seconds or ±3.3% error (=0.1/3.0)

8

c.yu91@csuohio.edu

## 95% Confidence Intervals

- Write as:

$$\mu = \bar{x} \pm 1.96\sigma_{\bar{x}}$$

- Method
  - Get the measurements
  - Calculate the mean and standard deviation
- If all the observed measurements are within the CI, we can say that "We are 95% confident that the true population mean is between..."



c.yu91@csuohio.edu

## 99% Confidence Intervals

- Write as:

$$\mu = \bar{x} \pm 1.96\sigma_{\bar{x}}$$

- Method
  - Get the measurements
  - Calculate the mean and standard deviation
- If all the observed measurements are within the CI, we can say that "We are 99% confident that the true population mean is between..."



c.yu91@csuohio.edu

## Using the Sample Std.Dev.

- Since we do not know our true population standard deviation to calculate the standard error, we must substitute the sample standard deviation in the standard error formula
- The provides us with an estimate of our standard error; thus, our confidence intervals can only be approximate
- Can be substituted because of the Central Limit Theorem, that a *n* greater than 30 will give us a nearly normal distribution



*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)



*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
```



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];
```



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_;
```

15

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_;
```

16

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);
```



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);
```



c.yu91@csuohio.edu

## Contents

- Understanding of simulation-based study
  - Discrete Event Simulation (DES) to better understand ns2 codes
  - Random number generation to understand “seed”
- 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

19

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.

20

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)



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.



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()

23

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.

24

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.



## Contents

- Understanding of simulation-based study
  - Discrete Event Simulation (DES) to better understand ns2 codes
  - Random number generation to understand "seed"

- 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

## 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

27

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 it's 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().

28

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).

*29*

*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 .*

*30*

*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?*



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();
```



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 occuring 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_.



*c.yu91@csuohio.edu*

# send_timer()

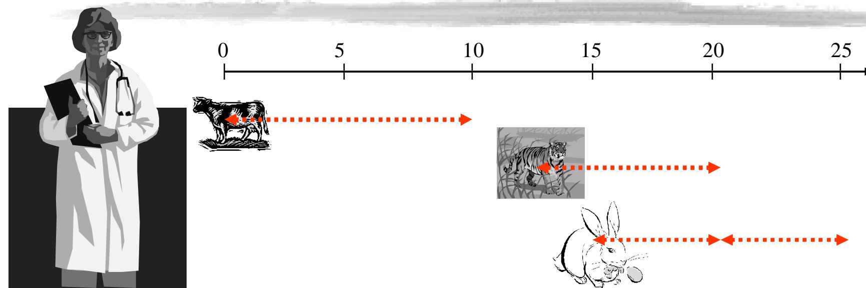
After each case has been handled and a packet has possibly been prepared for retransmission, the function tx_resume() is given control.

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



*c.yu91@csuohio.edu*

# Short Quiz




(1) Throughput =
(2) Delay =
(3) Service time =
(4) Queueing delay =
(5) Utilization =



c.yu91@csuohio.edu