University at Buffalo

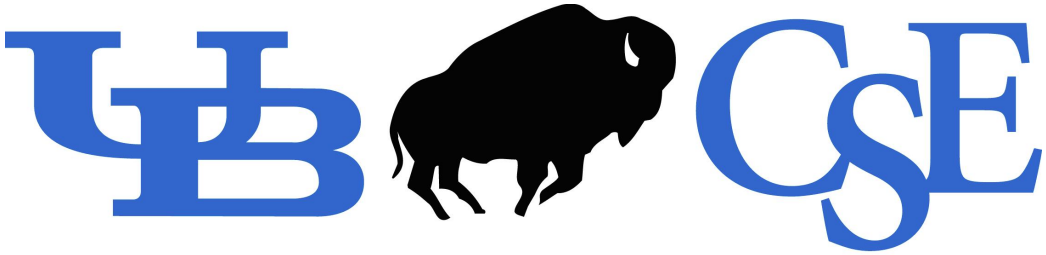
CSE 589 - Modern Networking Concepts, Fall 2016

Project 3: Evaluate MAC random transmission protocol using NS-2

Design document

Submitted by
Anurag Devulapalli (5020 8153)
Hema Madhav (5020 6563)

December 18, 2016



Department of Computer Science and Engineering
University at Buffalo *The State University of New York*

Contents

1 Objective

The objective of this project is to design and simulate a new random wireless MAC protocol and using Network Simulator (ns-v2.35),and analyze the results. In NS2 TCl scripting is used to define the topology of the network and C++ is used in designing the protocol.e.
In the following sections, we briefly describe the dataset used as Index, Preprocessing techniques, back end implementation and finally the front end features available to the end user.

2 Overview

In NS-2 environment, to simulate a scenario we need to first define the node configuration and other topology parameters in a Tcl (Tool Command Language) script. In the Tcl script, we define the new MAC layer protocol based on the project parameters to be used for the simulation.In the following sections we describe about the Tcl script, the changes we made for the new mac protocol and finally we analyse the trace file generated using a python script.

3 Tool command language (Tcl) Script

3.1 Project parameters

In the below lines we define the given project parameters, i.e the simulation time, interval size, the number of retransmissions and the packet size.

```
#Reference: http://www.isi.edu/nsnam/ns/tutorial/nsscript1.html
set duration 100;
set val(intervalVal) 0.02;
set val(packetSizeVal) 16; #128 bits
Mac/RandomMacNew set numRetransmit 10;
Mac/RandomMacNew set intervalDuration 0.02;
```

3.2 Node Parameters

In the below lines we define the simulation parameters as shown in the comments.

```
set val(chan)   Channel/WirelessChannel ;#Channel Type
```

```
set val(prop)   Propagation/TwoRayGround ;# radio-propagation
    model
set val(netif)  Phy/WirelessPhy          ;# network interface
    type
set val(mac)    Mac/RandomMacNew         ;# MAC type
set val(ifq)    Queue/DropTail/PriQueue  ;# interface queue type
set val(ll)     LL                       ;# link layer type
set val(ant)    Antenna/OmniAntenna      ;# antenna model
set val(ifqlen) 50                       ;# max packet in ifq
# routing protocol
set val(rp)            DumbAgent      ;
Mac/RandomMacNew set fullduplex_mode_ 0;
```

3.3 Creating ns object and configure node parameters

In the below lines, we create the ns object, the trace files and set the node configurations.

```
set ns [new Simulator]
set tracefd [open randommac.tr w]
$ns trace-all $tracefd
set topo      [new Topography]
$topo load_flatgrid      50 50
create-god 101

#Reference: http://www.isi.edu/nsnam/ns/doc/node46.html
$ns node-config \
-adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-channel [new $val(chan)] \
-agentTrace ON \
-routerTrace ON \
-macTrace ON \
-movementTrace OFF
```

3.4 Random placement of nodes

In the below code, we first generate some random numbers using rand() function and assign the node locations randomly between 0 and 50 , then we attach agents to the nodes and generate the CBR traffic.

```
#Reference: Generating the node positions
   http://www.cs.unc.edu/~clark/ns/rng/rng.pdf

#Generating random numbers
set randomGen [new RNG]
$randomGen seed 0

set randomX [new RandomVariable/Uniform]
$randomX use-rng $randomGen
$randomX set min_ 0
$randomX set max_ 50

set randomY [new RandomVariable/Uniform]
$randomY use-rng $randomGen
$randomY set min_ 0
$randomY set max_ 50

set randomTime [new RandomVariable/Uniform]
$randomTime use-rng $randomGen
$randomTime set min_ 0
$randomTime set max_ 0.02

#Creating the sink agent and connecting it to sink node
set sinkNode [$ns node]
$sinkNode random-motion 0
$sinkNode set X_ 25
$sinkNode set Y_ 25
$sinkNode set Z_ 0
$ns initial_node_pos $sinkNode 3

set null0 [new Agent/Null]
$ns attach-agent $sinkNode $null0

for {set i 0} {$i < 100} {incr i} {
#puts "i value: $i"
set node($i) [$ns node]
set random1 [$randomX value]
set random2 [$randomY value]
```

```
$node($i) set X_ $random1
$node($i) set Y_ $random2
$node($i) set Z_ 0
$ns initial_node_pos $node($i) 1

set udp($i) [new Agent/UDP]
$udp($i) set class_ $i
$ns attach-agent $node($i) $udp($i)

#Connecting te agents
$ns connect $udp($i) $null0

set cbr($i) [new Application/Traffic/CBR]
$cbr($i) set packetSize_ $val(packetSizeVal)
$cbr($i) set interval_ $val(intervalVal)
$cbr($i) attach-agent $udp($i)

set time1 $randomTime
$ns at $time1 "$cbr($i) start"
$ns at $duration "$cbr($i) stop"
}
```

3.5 Finish method and starting simulation

Finally we execute the finish method and start the simulation by running the ns object which we created earlier.

```
proc finish {} {
global ns tracefd
$ns flush-trace
close $tracefd
exit 0
}
puts "Starting simulation with random mac"
$ns run
```

4 newmac.cc

In the project we used the existing Mac/Simple protocol files included in the ns2 package and modified the protocol according to the requirement which is to send a packets "X" times in each time interval which is 0.02 seconds.Both these variables are passed to the protocol files by the Tcl script. We first create an array which stores some time intervals between 0 and T, and then schedule an event,which sends the same packet again for all these time intervals and later define a method to handle the function of forwarding the packets to the lower layer.

4.1 Defining the TclClass

As shown below, we need to set the Tcl class of our new random MAC protocol.

```
#include "ll.h"
#include "mac.h"
#include "newmac.h"
#include "random.h"

// Added by Sushmita to support event tracing
    (singal@nunki.usc.edu)
#include "agent.h"
#include "basetrace.h"
#include "cmu-trace.h"

static class RandomMacNewClass : public TclClass {
public:
// set the Tcl Class
RandomMacNewClass() : TclClass("Mac/RandomMacNew") {}
TclObject* create(int, const char*const*) {
return new RandomMacNew();
}
} class_RandomMacNew;
```

4.2 Binding the Tcl variables

Here, we bind the two interval duration and number of transmissions parameters which are set in the Tcl file.

```
//Accessing variables :
   http://www.slideshare.net/TBear76/ns2-binding-c-and-otcl-variables
RandomMacNew::RandomMacNew() : Mac() {
rx_state_ = tx_state_ = MAC_IDLE;
tx_active_ = 0;
waitTimer = new RandomMacNewWaitTimer(this);
sendTimer = new RandomMacNewSendTimer(this);
recvTimer = new RandomMacNewRecvTimer(this);
// Added by Sushmita to support event tracing
   (singal@nunki.usc.edu)
et_ = new EventTrace();
busy_ = 0;
bind("fullduplex_mode_", &fullduplex_mode_);
bind("intervalDuration", &intervalDuration);
bind("numRetransmit",&numRetransmit);
}
```

4.3 Retransmission changes

Here we first generate random time intervals in the given range and later schedule events at that randomly generated time intervals.We achieve this with the help of two while loops as below.

```
void RandomMacNew::send(Packet *p, Handler *h)
{
double* retransmissionIntervals = new double[numRetransmit];
hdr_cmn* ch = HDR_CMN(p);
/* store data tx time */
ch->txtime() = Mac::txtime(ch->size());

// Added by Sushmita to support event tracing
   (singal@nunki.usc.edu)
trace_event("SENSING_CARRIER",p);
Scheduler& retransmissionScheduler = Scheduler::instance();

//Random number generation
int count=0;
while(count<numRetransmit){
retransmissionIntervals[count] = intervalDuration *
   (rand()%40)/40.0 ;
count++;
}
```

```
//Retransmission.
int count1=0;
while(count1<numRetransmit){
//fprintf(stdout, "test\n");
//fprintf(stdout, "---\n");
retransmissionScheduler.schedule(this,
    (Event*)p->copy(),retransmissionIntervals[count1]);
count1++;
}

/* check whether we're idle */
if (tx_state_ != MAC_IDLE) {


Packet::free(p);
return;
}

pktTx_ = p;
txHandler_ = h;
// rather than sending packets out immediately, add in some
// jitter to reduce chance of unnecessary collisions
double jitter = Random::random()%40 * 100/bandwidth_;

if(rx_state_ != MAC_IDLE) {
trace_event("BACKING_OFF",p);
}

if (rx_state_ == MAC_IDLE ) {
// we're idle, so start sending now
waitTimer->restart(jitter);
sendTimer->restart(jitter + ch->txtime());
} else {
// we're currently receiving, so schedule it after
// we finish receiving
waitTimer->restart(jitter);
sendTimer->restart(jitter + ch->txtime()
+ HDR_CMN(pktRx_)->txtime());
}
}
```

4.4 Defining the handle method

This method forwards the packets to the lower layer when called.

```
void RandomMacNew::handle(Event *p)
{
downtarget_->recv((Packet*)p,txHandler_);
}
```

5 newmac.h

Here we declare the new function added and the two new variables - numRetransmit, intervalDuration as below.

```
class RandomMacNew : public Mac {
//Added by Sushmita to support backoff
friend class BackoffTimer;
public:
RandomMacNew();
void recv(Packet *p, Handler *h);
void send(Packet *p, Handler *h);
//Declared new function
void handle(Event *p);
void waitHandler(void);
void sendHandler(void);
void recvHandler(void);
double txtime(Packet *p);

// Added by Sushmita to support event tracing
    (singal@nunki.usc.edu)
void trace_event(char *, Packet *);
int command(int, const char*const*);
EventTrace *et_;

private:
Packet * pktRx_;
Packet * pktTx_;
MacState       rx_state_;     // incoming state (MAC_RECV or
    MAC_IDLE)
MacState       tx_state_;     // outgoing state
int            tx_active_;
int            fullduplex_mode_;
Handler *  txHandler_;
RandomMacNewWaitTimer *waitTimer;
RandomMacNewSendTimer *sendTimer;
RandomMacNewRecvTimer *recvTimer;
int busy_;
//Declared new variables
int numRetransmit;
double intervalDuration;
};
```

6 Make file changes

We need to add the new protocol in the makefile as below before running make.

```
mac/mac-802_3.o mac/mac.o mac/mac-tdma.o mac/smac.o \
mac/newmac.o \
```

7 Analysing the trace file

Once the simulation is performed, trace file is generated. We parse the trace file using the below python script to generate the simulation results and graph.

```
import sys
trace_file = open(sys.argv[1])
print ("Please wait for a moment......")
fields=[]
for line in iter(trace_file):
line=line.split(" ")
if (line[0] in ['r', 'D', 's'] and line[3] == 'MAC' and
    line[7] == 'cbr') :
fields.append({'node_id': line[2], 'Event_Type': line[0],
    'UPID': int(line[6])})
print ("Parsing Trace File:Done")
print ("Calculating Probability......")
received_Packet_len = len(set(trace['UPID'] for trace in
    fields if trace['Event_Type'] == 'r' and trace['node_id']
    == '_0_'))
Packets_sent = len(set(trace['UPID'] for trace in fields))
print("Probability: %.2f%%" %
    (float(received_Packet_len)/Packets_sent*100))
```

References

[1] http://www.isi.edu/nsnam/ns/tutorial/nsscript1.html

[2] http://www.isi.edu/nsnam/ns/doc/node46.html

[3] http://www.cs.unc.edu/ clark/ns/rng/rng.pdf

[4] mac-simple.cc and mac-simple.h in the default ns folder