Distributed Point Function based Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)

Assignment 1

SYSC 5104 - Methodologies of Discrete Event Modeling and Simulations

Fall 2012

Carleton University

DEVS Modeling and CD++ simulation of

Team members:

Mohammad El-Shabani 5123975(uOttawa)

Heli Amarasinghe 6152735(uOttawa)

# Overview

CSMA/CA (Carrier sense multiple access/ Collision avoidance) is a shared medium protocol that aims to allow multiple hosts to share a medium to communicate; where a collision is when multiple hosts try to transmit data using the shared medium in the same time. In wired mediums collision detection (CSMA/CD) is more popular. But in wireless communication, it is very difficult to detect collisions; thus, it is more popular to avoid collisions completely.

The case we are considering is when there is that of a homogenous network where there is not a coordinating node; thus, the method needs to be distributed; the basic DCF method is modeled here.

When a node wants to transmit a packet, it has to check if the medium is used; if it is not, it waits for a predefined DIFS period plus a period specified by an exponential back-off algorithm; if during that period no other nodes start transmitting, the node starts transmitting its packet; otherwise, it regenerates the back-off time, waits for when the medium is free again, waits the DIFS time plus the back-off time before sending its packet. Note that a third party node does knows the length of a packet that is being sent; this is because all nodes can overhear the header information of the packet. When a packet is received, the receiving node waits for SIFS period (shorter than DIFS) and transmits immediately.

The purpose of simulating our model is to assess the performance of CSMA/CA in terms of packet latency and medium throughput. The DCF CSMA/CA is a simplified version of the MAC protocol used in the IEEE 802.15.4 standard running in non-beacon mode. We are planning to simulate the case of 10 nodes.

Our model has a set of assumptions; to start with, we assume that all nodes are able to receive the signal of all other nodes; this dismisses the potential for the hidden node problem and the potential of collisions. Another assumption is that the carrier sensing technique the nodes utilize is able to recognize that the medium is free promptly. It is also able to know if another node has started transmitting promptly; the previous two assumptions basically assume that the inter-sampling time of the medium is zero. Lastly, a node is able to know the length of the message.

Brief Description of Components

1. Generator (atomic): The generator creates packet with random lengths in a predefined range with a predefined rate and outputs messages to the Buffer; each message has a unique ID consisting of the node ID and the message sequence number.
2. Buffer (atomic): The buffer performs queuing function for messages generated; it outputs messages on first in first out bases; it drops messages that cannot be buffered.
3. TX-RX controller (atomic): The TX-RX controller is the component that models the behavior of CSMA/CA.
4. MAC interface (coupled): The MAC interface is a coupled model consisting of the buffer and the TXRX controller.
5. Transducer (atomic): The transducer collects data regarding the time a packet is buffered, the time it was sent and produces latency times for output
6. Node (Coupled): A node is the coupled component consisting of the MAC interface, the generator and the transducer.
7. Medium (atomic): The medium is a logical entity defined to multiplex messages; the assumptions made about prompt recognition of free and busy medium by nodes is implemented by having the medium transmit an output message to nodes; we assume that the nodes continuously sense the medium.



Figure Atomic and Coupled model interaction diagram

# DEVS Formal Specifications

The CSMA/CA model consists of two main components; the wireless node and the medium. In reality, *N* number of heterogeneous wireless devices may inter-communicate over shared wireless communication medium. However this heterogeneity is not visible to CSMA/CA protocol since it was addressed by upper layer protocols of the network stack. Hence the system is modeled as *N* number of homogeneous wireless node system.

Wireless medium is modeled as a DEVS atomic model and wireless nodes are modeled as a DEVS coupled models. Each wireless node consist of; G*enerator (DEVS atomic)*, T*ransducer (DEVS atomic)* and *MACinterface (DEVS coupled)*. The coupled model, *MACinterface* is composed of *Buffer (DEVS atomic)* and a *TxRx Controller (DEVS atomic)* module. DEVS formal specifications for these atomic and coupled models can be defined as follows.

## DEVS Formal Specifications of coupled models

DEVS coupled model can be generally represented by;

Note that the following specifications describe a system with single wireless node (WN) and medium. We have executed different simulation rounds considering up to 10 wireless nodes in the system.

### Top model

X = ɸ;

Y = {throughput, latency};

D = { WN, Medium};

Md = { , };

EIC = ɸ;

EOC ⊆ {((WN, throughput), (self, throughput)); ((WN, latency), (self, latency))}

IC ⊆ { ((WN, send), (Medium, send)); ((Medium, receive), (WN, receive));

((Medium, free), (WN, free)); ((Medium, busy), (WN, busy))}

Select : ({ Medium, WN}) = Medium

### Wireless node

X = {receive, free, busy}

Y = {send};

D = {Generator, Transducer, MAC\_interface};

Md = {, , };

EIC = {((Self, receive), (MAC\_interface, receive)); ((Self, free), (MAC\_interface, free));

((Self, busy), (MAC\_interface, busy))};

EOC ⊆ {((MAC\_interface, send), (Self, send)); ((Transducer, throughput), (Self, throughput));

((Transducer, latency), (Self, latency))};

IC ⊆ { ((Generator, msg\_out), (MAC\_interface, msg\_in));

((Generator, msg\_gen), (Transducer, msg\_gen));

((MAC\_interface, msg\_sent), (Transducer, msg\_sent)) }

Select : ({ MAC\_interface, Generator, Transducer }) = MAC\_interface

({Generator, Transducer }) = Transducer

### MAC Interface

X = {msg\_in, receive, free, busy}

Y = {send, msg\_sent};

D = {Buffer, TxRxController};

Md = {, , };

EIC = {( (Self, receive), (TxRxControler, receive) ); ( (Self, free), (TxRxControler, free) );

((Self, busy), (TxRxControler, busy)); ((Self, msg\_in), (Buffer, msg\_in)) };

EOC ⊆ {( (TxRxControler, send), (Self, send) );

((TxRxControler, msg\_sent), (Self, msg\_sent))};

IC ⊆ { ((Buffer, job\_out), (TxRxControler, job\_in)); ((TxRxControler, done\_out), (Buffer, done\_in))}

Select = {TxRxControler, Buffer } = TxRxControler;

## DEVS Formal Specifications of atomic models

Formal specifications for atomic models can be defined as follows;

### Generator



Figure DEVS graph for the generator module

X = ɸ

Y = {msg\_out, msg\_gen}

S = {active}

δint : δint (active) = active

δext : unavailable

λ : λ(active){

send *message* to port *msg\_out* //int message = length + msg\_id\*1000+destination\*1000000

send *length* to port msg\_gen //sent to transducer

}

ta : ta(active) = delay //time delay between message generation

### Buffer



Figure DEVS graph for the buffer module

X = {msg\_in, done\_in}

Y = {job\_out}

S = {active, passive}

δint : δint (active) = passive

δext : δext (msg\_in, done\_in){

if X is from msg\_in {

elementsQ(tail) = message.value ;

tail++;

if elementsQ.size equals 1

phase = active;

}

if X is from done\_in){

remove elementsQ(front)

front - - ;

if elementsQ is not empty

phase = active; //if elementQ is empty, passivate by δint

}

}

λ : λ(active){

sendelementsQ(front) to port job\_out; //front element of the buffer is forwarded to Controller

}

### Transducer



Figure DEVS graph for the transducer module

X = {msg\_sent, msg\_gen}

Y = {latency, throughput}

S = {active, passive}

δint : δint (active) = passive

δext : δext (msg\_sent, msg\_gen){

if X is from port msg\_gen{

generatedQ(tail) = msg; //msg is an int created combining msg.val & msg.time

tail++;

passivate(); //passive for infinite time

}

Else if X is frm msg\_sent

Phase = active; //phase changed to active with ta = zero

λ : λ(active){

compute latency;

compute throughput;

send latency to port latency;

send throughput to port throughput;

}

### TxRxControler



Figure DEVS graph for the TxRxController module

X = {free, busy, receive, dataIn}

Y = {send, done, msgSent}

S = {idle, waitingForFreeChannel, waitingDIFS, waitForAck, receiving, rcvAck, transmit, sendAck,

waitSIFS} //states “idle” and “waitingForFreeChanel” in passive phase. All the other states are in active phase

δint : δint (waitingDIFS, transmit, rcvAck, receiving, waitSIFS, sendAck){

If current\_state is waitingDIFS{

current\_state = transmit;

sigma = msg\_length;

holdIn(active, sigma); //holdIn for the length of the message

}

Else if current\_state is transmit{

current\_state = waitForAck;

holdIn(passive, INFINITY);

}

Else if current\_state is rcvAck{

current\_state = idle;

holdIn(passive, INFINITY);

}

Else if current\_state is receiving{

current\_state = waitSIFS;

sigma = SIFS; //holdin SIFS time

holdIn(active, sigma);

}

Else if current\_state is waitSIFS{

current\_state = sendAck;

sigma = ack\_length;

holdIn(active, sigma); //holdIn for the length of the acknowledgement

}

Else if current\_state is sendAck{

if no more messages{ //if currMsg = NULL

current\_state = idle;

holdIn(passive, INFINITY);

}

else{ //there is a new-coming message to be sent

current\_state = waitingDIFS;

sigma = DIFS+BOT;

holdIn(active, sigma );

}

}

}

δext : δext (dataIn, receive, free, busy){

if X is from port dataIn {

if current\_state is idle{

if the medium is free{

current\_state = waitingDIFS;

sigma = DIFS + BOT; //BOT = random back-off time

holdIn( active, sigma);

}

else{

current\_state = waitingForFreeChannel;

holdIn( passive, INFINITY);

}

}

}

if X is from port receive{

if current\_state is idle OR waitingDIFS){

current\_state = receiving;

sigma = msg\_length; //msg\_length is the time length of the message

holdIn(active, sigma)

}

else{

if current\_state is waitForAck{

current\_state = rcvAck;

sigma = ack\_length;

holdIn(active, sigma);

}

else {

// error state

}

}

}

if X is from port free {

if current\_state is waitingForFreeChannel{

current\_state = waitingDIFS;

sigma = DIFS+BOT;

holdIn(active, sigma );

}

Else if current\_state is idle{

holdIn(passive, INFINITY); //set bool chanFree = true, and passivate

}

}

if X is from port busy {

chanFree = false;

if current\_state is waitingDIFS{

current\_state= waitingForFreeChannel;

holdIn( passive, INFINITY) ;

}

Else if (stat == idle){

holdIn(passive, INFINITY); //set bool chanFree = false, and passivate

}

}

}//end of δext

λ : λ(waitingDIFS, transmit, rcvAck, waitSIFS){

if current\_state is waitingDIFS{

send currentMessage to port send ;

}

else{

if current state is transmit{

send currentMessage to port msgSent;

currMsg = NULL; //clear variable

}

else{

if current\_state is rcvAck{

send value 1 to port done ;

}

else{

if current state is waitSIFS{

send ackpacket to port send;

}

}

}

}

}

### Medium



Figure DEVS graph for wireless medium model

X = {in1 , in2,..., in10 }

Y = { free1 , free2,..., free10 , busy1 , busy2,..., busy10 , forward1 , forward 2,..., forward 10 }

S = {free, t1, t2} //t1= sending\_start, t2 = sending\_done

δint : δint (t2, t2) {

if current\_status is t1{

current\_status = t2;

sigma = message\_length;

holdIn(active, sigma );

}

Else if current\_status is t2{

current\_status = free;

holdIn(passive, INFINITY);

}

}

δext : δext (free){

if X arrived from any of in1 , in2,..., in10 {

if current\_state is free{

current\_state = t1;

sigma=Time::Zero;

holdIn(active, sigma);

}

else{

//error state

}

}

}

λ : λ(t1, t1){

if current\_status is t1{

remove destination node number from of the message;

append senders node number to message;

send output to forward port, corresponding to destination node;

send value 1 to all busy ports;

}

Else if current\_state is t2{

Sent value 1 to all free ports;

}

}

# Simulation and Testing

We have evaluated the DEVS model of the CSMA/CA protocol using CD ++ simulation toolkit in eclipse IDE. In the simulator implementation, we have mapped the micro second values to milliseconds. This is because inbuilt CD++ Time( ) function defined in *time.h* defines time in the following format;

The average packet size is in the range of 100 bytes and 1000 bytes, which are equivalent to 15 μS and 150 μS message lengths (based on IEEE 802.11g average throughput data rate). To overcome this issue, we have simulated the time in the simulator supported format. However we have accounted it when calculating the throughput and latency at the transducer.

The generated message is at least a 7 digit integer, which is a multiplexed output of three integer values which represent; destination node, message ID and length. In this scenario, we have simulated the CSMA/CA protocol for maximum of 10 wireless nodes communicating over shared medium (frequency channel). If destination node number is given by *d*; and three digit message ID is given by and three digit message length is given by , the output message, is in the form of;

(3.1)

In order to verify the accuracy and integrity of the each functional component of the simulator, structural and black box testing were carried out on each atomic and coupled models. Test cases were created by injecting different combinations of inputs through event *(.ev)* files and analyzing the generated output *(.out)* and log *(.log)* files.

## Testing and output analysis of atomic models

### Generator

**Generator**

msg\_out

msg\_gen

MAC interface

Transducer

Figure 7 Interactions of generator with other modules

The generator atomic model is designed to generate messages in pre-defined time intervals. For this scenario, we have set a fixed generating interval; 1500 μS. The generator atomic module has two output ports; *msg\_out* and *msg\_gen*, which are connected to buffer (in MACinterface) and transducer modules respectively as shown in figure 7. Since the generator model does not have any input ports, we have performed standalone execution and obtained output results. Sample output obtained for 15 second simulation time is as follows;

00:00:00:000 out 6.00002e+06

00:00:00:000 msg\_gen 25

00:00:01:500 out 4.00112e+06

00:00:01:500 msg\_gen 116

00:00:03:000 out 1.00203e+06

00:00:03:000 msg\_gen 28

00:00:04:500 out 4.00306e+06

00:00:04:500 msg\_gen 65

00:00:06:000 out 1.0041e+06

00:00:06:000 msg\_gen 100

00:00:07:500 out 8.00503e+06

00:00:07:500 msg\_gen 32

00:00:09:000 out 1.00061e+07

00:00:09:000 msg\_gen 53

00:00:10:500 out 6.00714e+06

00:00:10:500 msg\_gen 137

00:00:12:000 out 1.00802e+06

00:00:12:000 msg\_gen 20

00:00:13:500 out 1.0009e+07

00:00:13:500 msg\_gen 34

00:00:15:000 out 5.01012e+06

00:00:15:000 msg\_gen 125

Since the message interval is kept at pre-defined value, 1500 mS, we can notice that at every 1500mS, two new outputs have been generated at *out* (same as *msg\_out*) port and *msg\_gen* port. This standalone testing was done at the wireless node 2 (WN2) for maximum of 10 wireless nodes. Therefore we can notice that destination node number has been randomly chosen such that since 2 is the own node ID in this scenario. The message ID value is starting from 0 and incremented by 1 at each set of messages. Also we can see the length of the message has been randomly generated as an integer value between 15 and 150. Also we can see that the multiplexed integer is sent to the buffer through the *out* port and only message length is sent to the transducer through the *msg\_gen* port. Based on these standalone test output, we can assert that the generator atomic module is functioning accurately.

### Buffer

The buffer atomic module is an internal component of the coupled model MACinterface. It stores messages generated by Generator and forwards to the TxRxControler in LIFO order.

Figure 8 Interaction of buffer with other modules

**Buffer**

TxRxControler

Generator

msg\_in

done\_in

job\_out

The message from the generator is in the form of the equation (3.1). Initially when the buffer is empty, the first message is directly forwarded to the TxRxController. After sending the message through the wireless medium, TxRxController is notifying the Buffer through the port done\_in. Then the buffer has to delete the top element in the LIFO queue and forward next job to the TxRxController through job\_out. All the messages received from msg\_in port during this time will be buffered in the LIFO queue.

In order to verify this proper functionality, we have carried out a test using an event file which contain following inputs;

00:00:00:001 msg\_in 5000150

00:00:00:152 msg\_in 8001015

00:00:00:200 done\_in 1

00:00:00:240 msg\_in 8002070

**00:00:00:240 msg\_in 1003090**

00:00:00:250 msg\_in 7004070

00:00:00:290 msg\_in 3005070

00:00:00:300 done\_in 1

00:00:00:400 done\_in 1

00:00:00:400 done\_in 1

The output obtained is as follows;

00:00:00:001 job\_out 5.00015e+06

00:00:00:200 job\_out 8.00102e+06

00:00:00:300 job\_out 8.00207e+06

00:00:00:400 job\_out 7.00407e+06

It can be noted that buffer has initially forwarded the first job with through the job\_out port as soon as it received it. But the second job was forwarded at 200mS, which is 48mS after its arrival. This is the expected results since buffer is holding the message until TxRxController notify the proper transmission through done\_in port. Also we can see no output is generated from the message with . This asserts the proper functionality because if two messages arrive at the same time, LIFO queue is discarding the latter message.

### Transducer

Transducer calculates throughput and latency of the simulated CSMA/CA protocol. Throughput is calculated in terms of average number of message sent over the medium per second and latency is calculated in terms of time delay between generation time and sent time.

Figure 9 Interactions of transducer with other modules

**Transducer**

TxRxControler

Generator

msg\_gen

msg\_sent

latency

throughput

Generator forwards the length of each message to the transducer and transducer multiplex this message length with message time in to a single integer and stores this value in its internal LIFO queue. When a frame has been sent through the wireless medium, TxRxController sends value of the message to the msg\_sent port of the transducer. If the total byes sent during the time interval between starting time and ending time is given by , the throughput during is given by;

(3.2)

and the latency is calculated by;

(3.3)

We have tested the accuracy of the transducer atomic model by performing standalone testing using an event file with following input combinations.

00:00:00:001 msg\_gen 40

00:00:00:090 msg\_gen 120

00:00:00:100 msg\_sent 1

00:00:00:130 msg\_gen 140

00:00:00:150 msg\_sent 1

00:00:00:200 msg\_sent 1

00:00:00:240 msg\_gen 70

00:00:00:290 msg\_gen 150

00:00:00:400 msg\_gen 80

00:00:00:450 msg\_sent 1

00:00:00:500 msg\_sent 1

00:00:00:600 msg\_sent 1

Transducer generated output for above input combinations are as follows;

00:00:00:100 latency 99

00:00:00:100 throughput 400000

00:00:00:150 latency 60

00:00:00:150 throughput 1.06667e+06

00:00:00:200 latency 70

00:00:00:200 throughput 1.5e+06

00:00:00:450 latency 210

00:00:00:450 throughput 822222

00:00:00:500 latency 210

00:00:00:500 throughput 1.04e+06

00:00:00:600 latency 200

00:00:00:600 throughput 1e+06

Transducer has generated an output each time when it receives msg\_sent input. It can be seen that the transducer has accurately calculated the throughput according to equation (3.2) and latency according to the equation (3.3).

### TxRxController

Figure 10 Interactions of TxRxController with other modules

**TxRxController**

Medium

Buffer

data\_in

free

send

msgSent

Transducer

done

receive

busy

MACinterface

WN

TxRxController is an atomic model inside the MACinterface of the wireless node. It continuously listens to the wireless medium and decides when to send frames over the medium guaranteeing collision-less data transfer. TxRxController is directly connected to buffer and indirectly connected to medium and transducer through external I/O ports of MACinterface and WM coupled models. The TxRxController continuously listen to the medium using its *free* and *busy* ports. Input from the buffer which is connected to port *dataIn* (or *job\_in*), delivers jobs to TxRxController. If the medium is free for certain time, MAC frames are transmitted over the medium through the *send* port. If the frame is successfully reached the destination, the acknowledgement sent by the destination node, will reach the TxRxController over the *receive* port. After successful transmission of data, buffer is notified through the port *done* and at the same time, transducer will be informed through the port *msgSent*.

We have tested the operation of the TxRxController by creating different test cases, changing input combinations in the event file.

Case 1:

*Input*

00:00:00:000 free 1

00:00:00:020 dataIn 5001150

00:00:00:300 receive 10

*Output*

00:00:00:080 send 5.00115e+06

00:00:00:230 msgsent 5.00115e+06

00:00:00:310 done 1

Initially the system is in idle (passive) state and the internal value of boolean variable, which is use to keep the medium state is set to *true* at the arrival of output message 1 over port *free*. When buffer forwarded job ‘5001150’ at 20mS through the port dataIn, TxRxController will change its state to waitingDIFS, to count-down DIFS time plus random back-off (BOT) time. Since no external events occur during count-down, TxRxController goes to transmit state and frame will be transmitted. We have set the DIFS value to 50mS and BOT is selected randomly. At 80mS, we can notice that the frame with the value ‘5001150’ has been started transmitting, to the medium through the output port *send*. Thus we can obtain the BOT generated for this case by;

Time equivalent to length of the message is required to upload all the bits to the medium. If the simulation time after sending all the message bits is given by

At time 230mS, we can notice output from the *msgsent* port, which will be sent to transducer. This output is generated after all the message bits have been successfully uploaded to the medium through the port *send*. This output verifies the accurate operation of TxRxControllerin this case. At 300mS, Controller stats receiving acknowledgement message from the receiver and after a time interval equivalent to *ack.length*, value 1 is sent to the buffer through port *done*, to update the job list.

Case 2:

*Input*

00:00:00:000 free 1

00:00:00:020 busy 1

00:00:00:030 dataIn 5001150

00:00:00:380 free 1

00:00:00:700 receive 10

*Output*

00:00:00:435 send 5.00115e+06

00:00:00:585 msgsent 5.00115e+06

00:00:00:710 done 1

In this case, although initially the medium is free, an input 1 is received at time 20mS over the port *busy*. Upon arrival of job at time 30mS, the Controller goes to state *waitingForFreeChannel*, which is a passive state until it receives 1 from port *free* again. At 380mS, it receives expected channel free message, initiating its DIFS plus BOT count-down. Since no event occur during this time, frame is forwarded through the *send* port at time 435ms (which equals to free.time + DIFS + BOT). Exactly after time interval equivalent to *message.length*, transducer is informed of successful message upload through the port msgSent and after receiving the acknowledgement, notified over the port *done*.

Case 3:

Input

00:00:00:000 free 1

00:00:00:020 dataIn 5001150

00:00:00:030 receive 2003120

00:00:00:800 receive 10

Output

00:00:00:170 send 2.00301e+06

00:00:00:230 send 5.00115e+06

00:00:00:380 msgsent 5.00115e+06

00:00:00:810 done 1

Input shows a scenario of arrival of message over the receive port during the DIFS plus BOT countdown before transmission. In this case the count-down is discarded and controller goes to receiving state. Receiving interval is equal to length of the message, which is 120mS in this case. After receiving time, sequence of internal transition functions takes controller to; waiting SIFS (ta = SIFS), sendAck (ta = ack.lenght) and if no more if jobs waiting idle state or else watingDIFS state. SIFS time delay is pre-selected as 10mS. Hence after SIFS time, acknowledgement is sent through the send port, which we can see at simulation time 170 in output. Transmission of acknowledgement takes 10mS and since the job received at simulation time 20mS is waiting in the buffer, controller goes back to waitingDIFS state to proceed with transmission.

### Medium

Medium is an atomic model which build communication links and which define basic rules associated with wireless communication channel. It contains 10 each from four types of ports namely; and .

When a particular wireless node wish to forward a frame over the wireless medium, the TxRxController inside the MACinterface of the WN first check the free and busy ports of the medium. Initially value of free port is 1 and busy port is undefined. In this case node recognizes the medium status as free and start transmitting after DIFS plus BOT count-down. When the wireless medium receives a frame over its input port *in* corresponding to that wireless node, its external transition function changes its state to t1 (sending\_start). Then the output function will forward the packet to the forward port connected to the destination node. The medium also interchange the target/destination digits if the frame allowing destination node to recognize the sender. This is a minor modification made for simulation purposes since in actual networks, frame header contains this information. At the same time, output function sends busy messages to all the nodes by sending value 1 through the busy ports of all of them. After sending the frame, medium will wait for time length of the message and then notify all the nodes that its free. Since SIFS time interval is always less than the DIFS time interval, the node which just received the message will be the first node to access the medium to send acknowledgement, after receiving the free message.

We have tested the operation of the medium by performing standalone testing for the medium by providing following input combinations from event file.

Case 1:

Input

00:00:00:100 in1 5001150

00:00:00:260 in5 1001010

00:00:00:380 in3 8001060

00:00:00:450 in8 3001010

Output

00:00:00:100 forward5 1.00115e+06

00:00:00:100 busy0 1

00:00:00:100 busy1 1

00:00:00:100 busy2 1

00:00:00:100 busy3 1

00:00:00:100 busy4 1

00:00:00:100 busy5 1

00:00:00:100 busy6 1

00:00:00:100 busy7 1

00:00:00:100 busy8 1

00:00:00:100 busy9 1

00:00:00:100 busy10 1

00:00:00:250 free0 1

00:00:00:250 free1 1

00:00:00:250 free2 1

00:00:00:250 free3 1

00:00:00:250 free4 1

00:00:00:250 free5 1

00:00:00:250 free6 1

00:00:00:250 free7 1

00:00:00:250 free8 1

00:00:00:250 free9 1

00:00:00:250 free10 1

00:00:00:260 forward1 5.00101e+06

00:00:00:260 busy0 1

00:00:00:260 busy1 1

00:00:00:260 busy2 1

00:00:00:260 busy3 1

00:00:00:260 busy4 1

00:00:00:260 busy5 1

00:00:00:260 busy6 1

00:00:00:260 busy7 1

00:00:00:260 busy8 1

00:00:00:260 busy9 1

00:00:00:260 busy10 1

00:00:00:270 free0 1

00:00:00:270 free1 1

00:00:00:270 free2 1

00:00:00:270 free3 1

00:00:00:270 free4 1

00:00:00:270 free5 1

00:00:00:270 free6 1

00:00:00:270 free7 1

00:00:00:270 free8 1

00:00:00:270 free9 1

00:00:00:270 free10 1

00:00:00:380 forward8 3.00106e+06

00:00:00:380 busy0 1

00:00:00:380 busy1 1

00:00:00:380 busy2 1

00:00:00:380 busy3 1

00:00:00:380 busy4 1

00:00:00:380 busy5 1

00:00:00:380 busy6 1

00:00:00:380 busy7 1

00:00:00:380 busy8 1

00:00:00:380 busy9 1

00:00:00:380 busy10 1

00:00:00:440 free0 1

00:00:00:440 free1 1

00:00:00:440 free2 1

00:00:00:440 free3 1

00:00:00:440 free4 1

00:00:00:440 free5 1

00:00:00:440 free6 1

00:00:00:440 free7 1

00:00:00:440 free8 1

00:00:00:440 free9 1

00:00:00:440 free10 1

00:00:00:450 forward3 8.00101e+06

00:00:00:450 busy0 1

00:00:00:450 busy1 1

00:00:00:450 busy2 1

00:00:00:450 busy3 1

00:00:00:450 busy4 1

00:00:00:450 busy5 1

00:00:00:450 busy6 1

00:00:00:450 busy7 1

00:00:00:450 busy8 1

00:00:00:450 busy9 1

00:00:00:450 busy10 1

00:00:00:460 free0 1

00:00:00:460 free1 1

00:00:00:460 free2 1

00:00:00:460 free3 1

00:00:00:460 free4 1

00:00:00:460 free5 1

00:00:00:460 free6 1

00:00:00:460 free7 1

00:00:00:460 free8 1

00:00:00:460 free9 1

00:00:00:460 free10 1

At simulation time 100mS, WN1 has sent frame with value ‘5001150’ to the *in1* port of the medium. The medium has forwarded the frame to the destination port *forward5* and informed all the nodes that its busy by sending value 1 from all the *busy* ports. After time length of the message, which is 150mS, medium has sent value 1 to all the *free* ports connected to nodes. Likewise for all four inputs, it can be seen that the medium has accurately generated output messages.

We have changed the input combination to observe the behavior of the medium if a node tries to send a frame before the medium is free. We have entered the following input combinations to the event file and observed the simulation output.

Case 2:

Input

00:00:00:100 in1 5001150

**00:00:00:120 in3 8001060**

00:00:00:260 in5 1001010

Output

00:00:00:100 forward5 1.00115e+06

00:00:00:100 busy0 1

00:00:00:100 busy1 1

00:00:00:100 busy2 1

00:00:00:100 busy3 1

00:00:00:100 busy4 1

00:00:00:100 busy5 1

00:00:00:100 busy6 1

00:00:00:100 busy7 1

00:00:00:100 busy8 1

00:00:00:100 busy9 1

00:00:00:100 busy10 1

00:00:00:250 free0 1

00:00:00:250 free1 1

00:00:00:250 free2 1

00:00:00:250 free3 1

00:00:00:250 free4 1

00:00:00:250 free5 1

00:00:00:250 free6 1

00:00:00:250 free7 1

00:00:00:250 free8 1

00:00:00:250 free9 1

00:00:00:250 free10 1

00:00:00:260 forward1 5.00101e+06

00:00:00:260 busy0 1

00:00:00:260 busy1 1

00:00:00:260 busy2 1

00:00:00:260 busy3 1

00:00:00:260 busy4 1

00:00:00:260 busy5 1

00:00:00:260 busy6 1

00:00:00:260 busy7 1

00:00:00:260 busy8 1

00:00:00:260 busy9 1

00:00:00:260 busy10 1

00:00:00:270 free0 1

00:00:00:270 free1 1

00:00:00:270 free2 1

00:00:00:270 free3 1

00:00:00:270 free4 1

00:00:00:270 free5 1

00:00:00:270 free6 1

00:00:00:270 free7 1

00:00:00:270 free8 1

00:00:00:270 free9 1

00:00:00:270 free10 1

From the generated output, it can be seen that the bolded frame has been discarded by the medium since it was sent for transmission before medium announce its status being free after transmitting message received at 100mS.

## Testing and output analysis of coupled models

We have carried out tests to verify correct operation of the MACinterface which is the coupled model which includes Buffer and TxRxController atomic models. MAC interface take inputs from generator, and medium (through WN). It has output ports connected to transducer and the medium.

In order to test the operation of the MACinterface, we have used an event file with following input combinations

Case 1:

Input

00:00:00:000 free 1

00:00:00:020 msg\_in 5001150

00:00:00:300 receive 10

00:00:00:320 msg\_in 2002035

00:00:00:550 receive 10

Ouput

00:00:00:080 send 5.00115e+06

00:00:00:230 msgsent 5.00115e+06

00:00:00:375 send 2.00204e+06

00:00:00:410 msgsent 2.00204e+06

Initially medium informs that it is free by sending value 1 to the *free* port of the MACinterface. At time 20mS, generator created message is simulated by entering value 5001150 through *msg\_in* port. MACinterface has started forwarding this message at 80mS through the *send* port after DIFS plus BOT countdown of 60mS. At 230mS, TxRxController inside the MACintercafe has sent message value to the transducer through the output port *msgsent* in the MACinterface. Similarly the message received at 320mS also forwarded through the send port at 375mS and updated the transducer at 410mS. We have tested the reaction of MACinterface to the busy message in following two cases.

Case 2:

Input

00:00:00:000 free 1

00:00:00:020 msg\_in 5001150

00:00:00:300 receive 10

**00:00:00:320 busy 1**

**00:00:00:330 msg\_in 2002035**

00:00:00:500 free 1

00:00:00:750 receive 10

In this case, after transmitting the first message as discussed in Case 1, MACinterface has given a busy input before the generator forwarding the second message. At this case, the received output is as follows;

Output

00:00:00:070 send 5.00115e+06

00:00:00:220 msgsent 5.00115e+06

**00:00:00:555 send 2.00204e+06**

00:00:00:590 msgsent 2.00204e+06

It can be seen that although the second message has been sent to the msg\_in port if the MACinterface at 330mS, message was not forwarded through its output port *send* until it sensed the availability of the medium at 500mS through its input port *free*. We have modified the input combinations so that the message arrives through the msg\_in port before the value 1 through the busy port in case 3 as follows;

Case 3:

Input

00:00:00:000 free 1

00:00:00:020 msg\_in 5001150

00:00:00:300 receive 10

**00:00:00:320 msg\_in 2002035**

**00:00:00:330 busy 1**

00:00:00:500 free 1

00:00:00:750 receive 10

In this case, the obtained output is as follows;

Output

00:00:00:070 send 5.00115e+06

00:00:00:220 msgsent 5.00115e+06

**00:00:00:555 send 2.00204e+06**

00:00:00:590 msgsent 2.00204e+06

As same as the case 2, the frame has not been forwarded through the send port until the MACinterface received value 1 from free port. From these analysis, it can be noted that the MACinterface coupled model is accurately perform its desired operations and atomic models inside the MACinterface has been correctly integrated.

Finally, we have executed the CSMA/CA (Top model), simulator by running simulation for 1000mS. The top model has two output ports namely; *throughput* and *latency*. Since the top model is not taking any inputs, we have created two additional output ports which notify the state of the medium; *free* and *busy*. To reduce the complexity, the case of three wireless nodes communication over the medium has been considered. The obtained simulation results are as follows;

Repeating pattern

Output

**00:00:00:050 busy 1**

msg

**00:00:00:115 free 1**

**00:00:00:115 latency1 115**

**00:00:00:115 throughput1 565217**

**00:00:00:135 busy 1**

ack

**00:00:00:145 free 1**

00:00:00:200 busy 1

00:00:00:257 free 1

00:00:00:257 latency2 257

00:00:00:257 throughput2 221789

00:00:00:277 busy 1

00:00:00:287 free 1

**00:00:00:347 busy 1**

**00:00:00:406 free 1**

**00:00:00:406 latency3 406**

**00:00:00:406 throughput3 145320**

**00:00:00:426 busy 1**

**00:00:00:436 free 1**

00:00:00:550 busy 1

00:00:00:643 free 1

00:00:00:643 latency3 143

00:00:00:643 throughput3 236391

00:00:00:663 busy 1

00:00:00:673 free 1

**00:00:00:728 busy 1**

**00:00:00:764 free 1**

**00:00:00:764 latency1 264**

**00:00:00:764 throughput1 132198**

**00:00:00:784 busy 1**

**00:00:00:794 free 1**

00:00:00:859 busy 1

00:00:00:941 free 1

00:00:00:941 latency2 441

00:00:00:941 throughput2 147715

00:00:00:961 busy 1

00:00:00:971 free 1

In the final output, as highlighted by the dashed rectangle, we can notice a repeating pattern, which has repeatedly occur six times during the simulation time of 100mS. Each of these repeating block is corresponding to successful transmission of message frame and arrival of acknowledgement frame to a node. First free-busy message pair indicates that the medium has been busy from 50mS to 115mS. We can assume that there has been a message transmitted through the medium with the length of 65mS (115mS – 50mS). The calculated latency and throughput values are delivered after sending the frame through the medium. Second free-busy pair in the repeating block indicates arrival of acknowledgement frame. We have mentioned earlier that we have fixed the length of our acknowledgement frame to 10mS. We can see this from the time difference between second free-busy pair, which is 10 mS (145mS – 135mS). From this analysis, we can proclaim that out designed CSMA/CA model and developed simulator are functioning accurately.