APPENDIX A

The following sections list the auto-generated diff text for the major files altered in the process of implementing the proposal in this thesis.

ns/mac/wireless-phy.h

```
51a52,54
> // Added this :) 9/3/2004
> typedef double Time;
>
55a59,71
> // Added this :) 21/3/2004
> struct nborinfo_struct {
>       int nnode_;          // Neighboring node record in table/array
>       int tablenum_;             //  This  neighbor  table's  node  (Added
16/3/2004)
>       float currtime_;     // Time when the record was changed
>       float oldtime_;            // Past value held in currtime_
>       float currdistance_;       // Distance between nodes at currtime_
>       float olddistance_; // Distance between nodes during oldtime_
>       double currrcvdpwr_;       // Last received Pr of packet at currtime_
>       double oldrcvdpwr_; // Last received Pr of packet at oldtime_
>       nborinfo_struct* next_; // Next!
>       };
>
70,71c86,87
<       void sendDown(Packet *p);
<       int sendUp(Packet *p);
---
>       virtual void sendDown(Packet *p);
>       virtual int sendUp(Packet *p);
81d96
<       //void setnode (MobileNode *node) { node_ = node; }
83a99,100
> //   double testcall();
>
105,107d121
<       // Why phy has a node_ and this guy has it all over again??
< //    MobileNode* node_;                // Mobile Node to which interface is
attached .
<
122a137,175
>
> class Wireless_802_11_Phy : public WirelessPhy{
> public:
>       Wireless_802_11_Phy();
>       virtual int command(int argc, const char*const* argv);
>       inline int getChannelNumber(){return channel_number_;}
>       virtual void sendDown(Packet *p);
>       virtual int sendUp(Packet *p);
>       // Added this :) 9/3/2004
>       //virtual int evalRetry();
```

```
>      virtual int evalRetry(Packet *p);
>      nborinfo_struct *neighbors_;
>
> protected:
>      /* the channel to which the phy is currently tuned to */
>      int channel_number_;
>
>      /* This function is used to get the frequency of the phy, when the
user sets the phy to a
>       * particular 802.11 channel
>       * Usually there are 11(14) overlapping channel, to which a 802.11
node can be tuned to
>       *
>       *      Siddhartha Saha [IIT Kanpur, INDIA, November 2002]
>       */
>
>      double getFreqFromChannelNumber(int channel);
>      /* If the packet is transmitted by a node tuned to txLambda with
power txPr, and if the
>       * receiving node is tuned to rxLambda, then the power with which the
receiving node will
>       * see the packet will *not* be txPr, but will be different depending
on txLambda and rxLambda
>       *
>       *      Siddhartha Saha [IIT Kanpur, INDIA, November 2002]
>       */
>
>      double calcChangedPr(double txPr,double txLambda,double rxLambda);
>
> };
>
```

ns/mac/wireless-phy.cc

```
97d94
<      //bind("bandwidth_", &bandwidth_);
103a101
>      //printf("WirelessPhy : Freq : %lf\n",freq_);
127c126,127
< }
---
>
>      }
153a154,156
>                      /*.Who would understand that -rxPower and -txPower set
in the nodeconfig
>                       * would result in this.
>                       */
199a203
>              printf("<DEBUG>Decreasing node's energy\n");
261c265
<      }
---
>      }//energy decreased
314,317c317,319
<                      printf("SM %f.9 _%d_ drop pkt from %d low POWER
%e/%e\n",
<                             Scheduler::instance().clock(), node()->index(),
<                             p->txinfo_.getNode()->index(),
<                             Pr,RXThresh);
---
>                      printf("SM %f.9 drop pkt low POWER %e/%e\n",
>                             Scheduler::instance().clock(),
>                             Pr,RXThresh_);
```

```
445a448,936
>
>
> /*double WirelessPhy::testcall()
> {
>       //printf("printf in WirelessPhy called\n");
>       return 4.0;
> }*/
>
> /* ==========================================================================
>  * Class Definitions of Wireless_802_11_Phy class
>  *
>  * ==========================================================================
>  */
>
> /* ==========================================================================
>    Wireless_802_11_Phy Interface
>    ========================================================================== */
*/
> static class Wireless_802_11_PhyClass: public TclClass {
> public:
>         Wireless_802_11_PhyClass() :
TclClass("Phy/WirelessPhy/Wireless_802_11_Phy") {}
>         TclObject* create(int, const char*const*) {
>                 return (new Wireless_802_11_Phy);
>         }
> } class_Wireless_802_11_Phy;
>
> /* The constructor */
>
> Wireless_802_11_Phy::Wireless_802_11_Phy() : WirelessPhy(){
>       /*change the lambda and freq to the one
>        * used for 802.11 channel 1., the default
>        * is set in ns-defaults.tcl
>        */
>
>       bind("channel_number_",&channel_number_);
>       freq_ = getFreqFromChannelNumber(channel_number_);
>       lambda_ = SPEED_OF_LIGHT / freq_ ;
>
>       // Added the following section 16/3/2004 :)
>       neighbors_=new nborinfo_struct;
>       neighbors_->tablenum_=-1;
>       //Can't initialise tablenum here cause no nodes yet!
>
>       //printf("<DEBUG>Wireless_802_11_Phy :, Freq : %lf\n",freq_);
>
>       /* word of cautiion, since freq_ and lambda_ are bound variable,
>        * if anybody changes these values from the script, then there will
>        * be a mismatch between the values of the channel and freq.
>        */
>
>
> }
>
> double Wireless_802_11_Phy::getFreqFromChannelNumber(int channel){
>       switch(channel){
>               case 1:
>                       //return 914e+6;
>                       return 2.412e+9;
>               case 2:
>                       return 2.417e+9;
>               case 3:
>                       return 2.422e+9;
>               case 4:
>                       return 2.427e+9;
>               case 5:
>                       return 2.432e+9;
```

```
>             case 6:
>                   return 2.437e+9;
>             case 7:
>                   return 2.442e+9;
>             case 8:
>                   return 2.447e+9;
>             case 9:
>                   return 2.452e+9;
>             case 10:
>                   return 2.457e+9;
>             case 11:
>                   return 2.462e+9;
>             case 12:
>                   return 2.467e+9;
>             case 13:
>                   return 2.472e+9;
>             case 14:
>                   return 2.483e+9;
>             default:
>                   printf("ERROR : Wrong channel number : %d\n",channel);
>                   assert(0);
>       }
>       .
> }
> int Wireless_802_11_Phy::command(int argc, const char*const* argv)
> {
>       TclObject *obj;
>
>       if(argc == 3) {
>             if (strcasecmp(argv[1], "setChannelNumber") == 0) {
>                   channel_number_ = atoi(argv[2]);
>                   freq_ = getFreqFromChannelNumber(channel_number_);
>                   lambda_ = SPEED_OF_LIGHT / freq_ ;
>                   return TCL_OK;
>             }
>       }
>       return WirelessPhy::command(argc,argv);
> }
>
>
>
> void Wireless_802_11_Phy::sendDown(Packet *p)
> {
>       /*
>        * Sanity Check
>        */
>       assert(initialized());
>
>       // Not using the energymodel now
>
>
>       /*
>        *  Stamp the packet with the interface arguments
>        */
>       p->txinfo_.stamp((MobileNode*)node(), ant_->copy(), Pt_, lambda_);
>
>       // Send the packet
>       channel_->recv(p, this);
> }
>
> int Wireless_802_11_Phy::sendUp(Packet *p)
> {
>       /*
>        * Sanity Check
>        */
>       assert(initialized());
>
>       PacketStamp s;
```

```
>       double Pr;
>       int pkt_recvd = 0;
>
>       Time now=Scheduler::instance().clock();
>       nborinfo_struct *ns;
>
>       s.stamp((MobileNode*)node(), ant_, 0, lambda_);
>
>       /* Initial way I calculated distance. Theoretically its wrong but the
result is exactly the same!
>       double Xt, Yt, Zt, Xr, Yr,Zr;
>       s.getNode()->getLoc(&Xt, &Yt, &Zt);
>       p->txinfo_.getNode()->getLoc(&Xr, &Yr, &Zr);
>
>       Xr += p->txinfo_.getAntenna()->getX();          // In real life, nodes
won't know their position coordinates
>       Yr += p->txinfo_.getAntenna()->getY();
>       Zr += p->txinfo_.getAntenna()->getZ();
>       Xt += s.getAntenna()->getX();                   // Neither would their
attenas!
>       Yt += s.getAntenna()->getY();
>       Zt += s.getAntenna()->getZ();
>       double dX = Xr - Xt;
>       double dY = Yr - Yt;
>       double dZ = Zr - Zt;                            // So all this would
be wrong wrong wrong :P
>       float d = sqrt(dX*dX+dY*dY);*/
>
>
>       if(neighbors_->tablenum_==-1) {
>               neighbors_->tablenum_=node()->nodeid();                 // Label
table as current nodes' table
>               neighbors_->nnode_=p->txinfo_.getNode()->nodeid();      // Insert
neighbor node id
>               neighbors_->currtime_=now;                      // Insert
current time
>               neighbors_->oldtime_=0.0;                       // Null time
>               neighbors_->next_=NULL;                                 // Point
to nowhere first :P
>               neighbors_->currdistance_=0.0;
>               neighbors_->olddistance_=0.0;
>               neighbors_->currrcvdpwr_=0.0;
>               neighbors_->oldrcvdpwr_=0.0;
>               //printf("[INITIAL ENTRY] Table: %2d NNode: %2d curr time: %6f
old time: %6f cdist: %f odist: %f\n",neighbors_->tablenum_,neighbors_-
>nnode_,neighbors_->currtime_,neighbors_->oldtime_,neighbors_-
>currdistance_,neighbors_->olddistance_);
>               }
>
>       if(propagation_) {
>               //s.stamp((MobileNode*)node(), ant_, 0, lambda_);
>
>               //printf("%g  ",p->txinfo_.getTxPr());
>
>               /* For now, just follow a wild hack. change the power of
>                * the trasmitting packetstamp depending on the difference in
>                * channel of the receiving and the incoming node
>                *     - Siddhartha
>                */
>               p->txinfo_.setTxPr(calcChangedPr(p->txinfo_.getTxPr(),p-
>txinfo_.getLambda(),lambda_));
>               Pr = propagation_->Pr(&p->txinfo_, &s, this);
>               //printf("%g\n",Pr);
>
>               if (Pr < CSThresh_) {
>                       pkt_recvd = 0;
>                       goto DONE;
>               }
```

```
>             if (Pr < RXThresh_) {
>                     /*
>                      * We can detect, but not successfully receive
>                      *
>                      * this packet.
>                      */
>                     hdr_cmn *hdr = HDR_CMN(p);
>                     hdr->error() = 1;
> #if DEBUG > 3
>                     printf("SM %f.9 drop pkt low POWER %e/%e\n",
>                            Scheduler::instance().clock(),
>                            Pr,RXThresh_);
> #endif
>             }
>     }
>     if(modulation_) {
>             hdr_cmn *hdr = HDR_CMN(p);
>             hdr->error() = modulation_->BitError(Pr);
>     }
>
>     // If it reaches here, that means Pr > CSThresh and RXThresh which
requires an update to the table
>     for (ns = neighbors_;ns != NULL;ns = ns->next_) {
>             if (ns->nnode_ == p->txinfo_.getNode()->nodeid()) {
>                     ns->oldtime_ = ns->currtime_;                  // Backup
previous time
>                     ns->currtime_ = now;                           // Replace
current time
>                     ns->oldrcvdpwr_ = ns->currrcvdpwr_;
>                     ns->currrcvdpwr_ = Pr;                         // Store rcvd
pkt pr
>                     ns->olddistance_ = ns->currdistance_;   // Backup
previous distance
>                     ns->currdistance_ = propagation_->CalcDistance(&p-
>txinfo_, &s, this, Pr); // Insert new distance
>                     //printf("#[UPDATE] Table[%2d] NNode[%2d] Now[%6f]
Past[%6f] cdist[%f] odist[%f] accel[%f] Pr[%g]\n",ns->tablenum_,ns-
>nnode_,ns->currtime_,ns->oldtime_,ns->currdistance_,ns-
>olddistance_,accelval,Pr);
>                     goto EVALUATE;
>             }
>     }
>
>     ns = new nborinfo_struct;
>     ns->tablenum_ = neighbors_->tablenum_;
>     ns->nnode_ = p->txinfo_.getNode()->nodeid();
>     ns->currtime_ = now;
>     ns->oldtime_ = 0.0;
>     ns->currdistance_ = propagation_->CalcDistance(&p->txinfo_, &s, this,
Pr);
>     ns->olddistance_ = ns->currdistance_;
>     ns->oldrcvdpwr_ = ns->currrcvdpwr_;
>     ns->currrcvdpwr_ = Pr;
>     ns->next_ = neighbors_;
>     neighbors_ = ns;
>     //printf("[NEW NODE] Table: %2d NNode: %2d curr time: %6f old time:
%6f cdist: %f odist: %f\n",neighbors_->tablenum_,neighbors_-
>nnode_,neighbors_->currtime_,neighbors_->oldtime_,neighbors_-
>currdistance_,neighbors_->olddistance_);
>
> EVALUATE:
>     /*
>      * The MAC layer must be notified of the packet reception
>      * now - ie; when the first bit has been detected - so that
>      * it can properly do Collision Avoidance / Detection.
>      */
>     pkt_recvd = 1;
>
```

```
> DONE:
>       p->txinfo_.getAntenna()->release();
>
>       /* WILD HACK: The following two variables are a wild hack.
>          They will go away in the next release...
>          They're used by the mac-802_11 object to determine
>          capture.  This will be moved into the net-if family of
>          objects in the future. */
>       p->txinfo_.RxPr = Pr;
>       p->txinfo_.CPThresh = CPThresh_;
>
>
>       return pkt_recvd;
> }
>
> double Wireless_802_11_Phy::calcChangedPr(double txPr,double
txLambda,double rxLambda){
>       /* Original code by Siddharta:
>       //int diff = abs(getChannelFromLambda(txLambda) -
getChannelFromLambda(rxLambda));
>       int diff = abs((int)((SPEED_OF_LIGHT/txLambda -
SPEED_OF_LIGHT/rxLambda)/5e+6));
>       //printf("SIDD_DEBUG Channel Diff : %d\n",diff);
>       if(diff > 2) return 0.0;
>       if(diff == 1)return txPr*2.0/3.0;
>       if(diff == 2)return txPr/3.0;
>       return txPr;
>       //My version of the code is as follows */
>
>       int diff = abs((int)((SPEED_OF_LIGHT/txLambda -
SPEED_OF_LIGHT/rxLambda)/5e+6)); // Obtain channel difference
>       if (diff == 1) return txPr*0.2728;
>       else if (diff == 2) return txPr*0.7286; // Untested values from
cirond
>       else if (diff == 3) return txPr*0.9625;
>       else if (diff == 4) return txPr*0.9946;
>       else if (diff == 5) return txPr*0.9992;
>       else if (diff == 6) return txPr*0.9998;
>       return txPr;
> }
>
> int Wireless_802_11_Phy::evalRetry(Packet *p) {
>
>       PacketStamp s;
>       s.stamp((MobileNode*)node(), ant_, 0, lambda_);
>       // Added following :) 27/04/2004 --- edited 09/05/2004
>
>       float myrange[2][7] = {
>                       {1.0,   0.0,   5.0, 55.0, 60.0,0.0,0.0},
>                       {1.0, 55.0, 60.0, 65.0,160.0,0.0,0.0},
>               };
>
>       double myaccel[2][6] = {
>                       {0.0,0.0,0.1,10.0,0.0,0.0},
>                       {-10.0,-1.0,0.0,0.0,0.0,0.0}
>               };
>
>
>       float myretry[4] = {3,5,1,6};
>
>       float preout1[4]={0};
>       float preout2[4]={0};
>       float foutputHeight[4]={0};
>       float finalOutput=0,totalY=0,totalH=0,speed_=0,estrange_;
>       int i,j;      // used to store index values when traversing array
>       nborinfo_struct *testvar,*ns2=neighbors_;       // used to identify
correct table entry for node
>       double avrgrcvdPr_,currpktPr_;
```

```
>
> /*   // Debugging neighbor table ...WARNING LARGE SCREEN DUMP IF MANY
NODES IN SIMULATION!
>       printf("#### STARTING NEIGHBOR TABLE DUMP!\n");
>       for (testvar=neighbors_;testvar!=NULL;testvar=testvar->next_)
>             printf("Table[%2d] NNode[%2d] Now[%6f] Past[%6f] cdist[%f]
odist[%f] accel[%f] Pr[%g]\n",testvar->tablenum_,testvar->nnode_,testvar-
>currtime_,testvar->oldtime_,testvar->currdistance_,testvar->olddistance_);
>       printf("#### DUMP COMPLETE!\n");
> */
>       for (ns2;ns2!=NULL;)
>       {       // for loop1
>             if(ns2->nnode_!=p->txinfo_.getNode()->nodeid())
>                   ns2=ns2->next_;             // locate node entry in table
>             if(ns2==NULL) {
>                   printf("\033[33mNeighbour entry not found\033[0m\t");
>                   return 0;             // if no entry found abort retry
>                   }
>             if(ns2->nnode_==p->txinfo_.getNode()->nodeid())
>             {
>
>       //printf("DEBUG!! Checking node table [%d] for RTS pkt [%d] with
currnode in table [%d]\n",ns2->tablenum_, p->txinfo_.getNode()-
>nodeid(),ns2->nnode_);
>
>       // Check if table info too old ...estimated at max 20m/s x 8 = 160m
which is Rx range
>       if (Scheduler::instance().clock() - ns2->currtime_ > 8.0) {
>             printf("\033[34mNeighbour info too old\033[0m\t\t");
>             return 0;
>             }
>
>       avrgrcvdPr_ = (ns2->currrcvdpwr_ + ns2->oldrcvdpwr_)/2;
>       currpktPr_ = propagation_->Pr(&p->txinfo_, &s, this);
>
>
>       speed_=(ns2->currdistance_-ns2->olddistance_)/(ns2->currtime_-ns2-
>oldtime_);
>
>       if (ns2->currdistance_ >= ns2->olddistance_)
>             estrange_=ns2->currdistance_+(ns2->currdistance_-ns2-
>olddistance_)/(ns2->currtime_-ns2-
>oldtime_)*(Scheduler::instance().clock()-ns2->currtime_);
>       else
>             estrange_=ns2->currdistance_-(ns2->olddistance_-ns2-
>currdistance_)/(ns2->currtime_-ns2-
>oldtime_)*(Scheduler::instance().clock()-ns2->currtime_);
>
>
>       printf("DEBUG!! Inputs: estrange_ [%f] avrgrcvdPr_ [%g] accel [%f]
\n", estrange_,avrgrcvdPr_,speed_);
>       if (estrange_>160.0) {
>             printf("\033[35mOut of range!\033[0m\t\t\t");
>             return 0;
>             }
>
>       //printf("DEBUG!! Actual range [%f]\n", propagation_-
>CalcDistance(&p->txinfo_, &s, this, currpktPr_));
>
>       // Fuzzify distance
>       for (i=0;i<2;i++)
>       {
>             if ((int)myrange[i][0]==1) // If 1, then shape is trapezium
with 4 points + 2 inf values
>             {
>                   if ((myrange[i][5] && estrange_<=myrange[i][1]) ||
(myrange[i][6] && estrange_>=myrange[i][4]))
>                         preout1[i]=1.00;
```

```
>                   else
>                   {
>                         if (estrange_ < myrange[i][1])
>                               preout1[i]=0.00;
>                         else if (estrange_ < myrange[i][2])
>                               preout1[i]=(estrange_-
myrange[i][1])/(myrange[i][2]-myrange[i][1]);
>                         else if (estrange_ < myrange[i][3])
>                               preout1[i]=1.00;
>                         else if (estrange_ < myrange[i][4])
>                               preout1[i]=(myrange[i][4]-
estrange_)/(myrange[i][4]-myrange[i][3]);
>                         else
>                               preout1[i]=0.00;     // Default case
>                   }
>             }
>
>             if ((int)myrange[i][0]==0) // If 0, shape is triangular and
m'ship functions change
>             {
>                   if ((estrange_ <= myrange[i][1]) || (estrange_ >
myrange[i][3]))
>                         preout1[i]=0.00;
>                   else if (estrange_ < myrange[i][2])
>                         preout1[i]=(estrange_-
myrange[i][1])/(myrange[i][2]-myrange[i][1]);
>                   else if (estrange_ == myrange[i][2])
>                         preout1[i]=1.00;
>                   else if (estrange_ <= myrange[i][3])
>                         preout1[i]=(myrange[i][3]-
estrange_)/(myrange[i][3]-myrange[i][2]);
>                   else
>                         preout1[i]=0.00;     // Default case
>             }
>       }// Closes i loop
>
>       // Test input for speed
>       for(j=0; j<2; j++)
>       {
>             if ((myaccel[j][4] && speed_<=myaccel[j][0]) || (myaccel[j][5]
&& speed_>=myaccel[j][3]) || (speed_==0.0))
>                   preout2[j]=1.00;
>             else
>             {
>                   if (speed_ <= myaccel[j][0])
>                         preout2[j]=0.00;
>                   else if (speed_ < myaccel[j][1])
>                         preout2[j]=(speed_-myaccel[j][0])/(myaccel[j][1]-
myaccel[j][0]);
>                   else if (speed_ < myaccel[j][2])
>                         preout2[j]=1.00;
>                   else if (speed_ < myaccel[j][3])
>                         preout2[j]=(myaccel[j][3]-speed_)/(myaccel[j][3]-
myaccel[j][2]);
>                   else
>                         preout2[j]=0.00;
>             }
>       } // Closes j loop
>
>       // Inferencing
> /*    // Dump results before inferencing commences
>       for (i=0;i<3;i++) {
>             for (j=0;j<2;j++) {
>                   printf("[%f]:",preout1[i]);
>                   printf("[%f]\n",preout2[j]);
>                   }
>             }*/
>
```

```
>       int index=0;
>       for (i=0;i<2;i++)
>       {
>               for (int j=0; j<2; j++)
>               {
>                       if (preout2[j] < preout1[i])      // AND
>                               foutputHeight[index]=preout2[j];
>                       else
>                               foutputHeight[index]=preout1[i];
>                       index++;      // Increments from 0 to 5
>               }
>       }
>
>       /*for (i=0;i<3;i++)
>       {
>               if (preout2[i] < preout1[i])
>                       foutputHeight[i]=preout2[i];
>               else
>                       foutputHeight[i]=preout1[i];
>       }*/
>
> /*    // Dump output of min
>       printf("Dump inf table:");
>       for (i=0;i<6;i++)
>               printf("%f||", foutputHeight[i]);
>       printf("\n");*/
>
>
>       // Defuzzification
>       for (i=0;i<4;i++)
>       {
>               totalY+=myretry[i]*foutputHeight[i];
>               //totalY+=myretry[i]*preout1[i];
>               totalH+=foutputHeight[i];
>               //totalH+=myretry[i];
>       }
>
>       if (totalH>0)
>               finalOutput=totalY/totalH; // Ensures no division by zero
operation to avoid crashing
>       else
>               finalOutput=0;
>
>       //printf("==> Final output %f/%f=%f\nType casting to int
%d\n",totalY,totalH,finalOutput,finalOutput);
>       return (int)(finalOutput);
>       }
>       } //Closes "for (ns-....." loop
> } // Closes evalRetry() function
```

ns/mac/mac-802_11.h

```
44a45,46
> // Added this 23/03/2004 :)
> #include "propagation.h"
146c147
< #define DSSS_SlotTime                   0.000020        // 20us
---
> //#define DSSS_SlotTime                 0.000020        // 20us
149c150
< #define DSSS_SIFSTime                   0.000010        // 10us
---
> //#define DSSS_SIFSTime                 0.000010        // 10us
153a155,160
```

```
> /* Added according to source by siddharta */
> #define DSSS_RxRFDelay                0.0000025
> #define DSSS_RxPLPCDelay        0.0000025
> #define DSSS_MACProcessingDelay       0
> #define DSSS_AirPropagationTime       0.0000010
>
158a166,167
> #define MAC_AIR_PROPAGATION_CONST     2
> #define TRACE_CW
167c176
<       double          SIFSTime;
---
>       //double            SIFSTime;
170a180,184
>
>       double          RxRFDelay;
>       double          RxPLPCDelay;
>       double          MACProcessingDelay;
>       double          AirPropagationTime;
223a238,239
> // Added this 23/03/2004 :)
> class Propagation;
248a265,266
>       // Added this 23/03/2004 :)
>       Propagation *propagation_;
371a390,394
>       // Added the following :) 27/04/2004
>       u_int32_t       retryflg_;    // Has retry been attempted?
>       int             fuzzyflg_;
>       //-----------------------------------
>
```

ns/mac/mac-802_11.cc

```
120,122c120,124
<       DSSS_CWMin, DSSS_CWMax, DSSS_SlotTime, DSSS_CCATime,
<       DSSS_RxTxTurnaroundTime, DSSS_SIFSTime, DSSS_PreambleLength,
<       DSSS_PLCPHeaderLength, DSSS_PLCPDataRate
---
>       DSSS_CWMin, DSSS_CWMax,
>       DSSS_CCATime + DSSS_RxTxTurnaroundTime + DSSS_AirPropagationTime +
DSSS_MACProcessingDelay,
>       DSSS_CCATime, DSSS_RxTxTurnaroundTime, DSSS_PreambleLength,
>       DSSS_PLCPHeaderLength, DSSS_PLCPDataRate, DSSS_RxRFDelay,
>       DSSS_RxPLPCDelay, DSSS_MACProcessingDelay, DSSS_AirPropagationTime
       // Changes according to Siddharta
130c132
<       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
---
>       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0       // Added extra 0?
148c150
< Mac802_11::Mac802_11(PHY_MIB *p, MAC_MIB *m) : Mac(), mhIF_(this),
mhNav_(this), mhRecv_(this), mhSend_(this), mhDefer_(this, p->SlotTime),
mhBackoff_(this, p->SlotTime)
---
> Mac802_11::Mac802_11(PHY_MIB *p, MAC_MIB *m) : Mac(), mhIF_(this),
mhNav_(this), mhRecv_(this), mhSend_(this), mhDefer_(this,p->CCATime + p-
>RxTxTurnaroundTime + p->AirPropagationTime + p->MACProcessingDelay ),
mhBackoff_(this,p->CCATime + p->RxTxTurnaroundTime + p->AirPropagationTime +
p->MACProcessingDelay )
152a155,157
>       /* Added according to Siddharta */
>       phymib_->SlotTime = phymib_->CCATime + phymib_->RxTxTurnaroundTime +
phymib_->AirPropagationTime + phymib_->MACProcessingDelay;
```

```
>
162c167
<       ssrc_ = slrc_ = 0;
---
>       ssrc_ = slrc_ = retryflg_ = 0;    // Added retryflg_ and fuzzyflg_ :)
29/04/2004
164c169,170
<       sifs_ = phymib_->SIFSTime;
---
>       /* sifs_ = phymib_->SIFSTime; Changed */
>       sifs_ = phymib_->RxRFDelay + phymib_->RxPLPCDelay + phymib_-
>MACProcessingDelay + phymib_->RxTxTurnaroundTime;
180a187
>       // Changed bandwidth to suit 802.11b rates
193a201,212
>       // Implement and remove fuzzy logic code here :)
>       tcl.evalf("Mac/802_11 set fuzzylogic_");
>       if (strcmp(tcl.result(), "0") != 0)
>       {
>             //printf("Implementing fuzzy logic code\n");
>             fuzzyflg_=1;
>       }
>       else
>       {
>             //printf("Removing fuzzy code\n");
>             fuzzyflg_=0;
>       }
618c637,639
<                   + DSSS_MaxPropagationDelay              // XXX
---
>                   //+ DSSS_MaxPropagationDelay                    // XXX
>                   // Changed to account for journey to/from
>                   + MAC_AIR_PROPAGATION_CONST*DSSS_AirPropagationTime
620c641,642
<                   + DSSS_MaxPropagationDelay              // XXX
---
>                   //+ DSSS_MaxPropagationDelay                    // XXX
>                   + MAC_AIR_PROPAGATION_CONST*DSSS_AirPropagationTime
666c688,689
<                   + DSSS_MaxPropagationDelay              // XXX
---
>                   //+ DSSS_MaxPropagationDelay                    // XXX
>                   + MAC_AIR_PROPAGATION_CONST*DSSS_AirPropagationTime
669c692,693
<                   + DSSS_MaxPropagationDelay;                   // XXX
---
>                   //+ DSSS_MaxPropagationDelay;                   // XXX
>                   + MAC_AIR_PROPAGATION_CONST*DSSS_AirPropagationTime;
702c726
<             if((u_int32_t)ETHER_ADDR(mh->dh_da) != MAC_BROADCAST)
---
>             if((u_int32_t)ETHER_ADDR(mh->dh_da) != MAC_BROADCAST) {
704c728,729
<                         + DSSS_MaxPropagationDelay        // XXX
---
>                         //+ DSSS_MaxPropagationDelay              // XXX
>                         +
MAC_AIR_PROPAGATION_CONST*DSSS_AirPropagationTime
707c732,734
<                         + DSSS_MaxPropagationDelay;               // XXX
---
>                         //+ DSSS_MaxPropagationDelay;             // XXX
>                         +
MAC_AIR_PROPAGATION_CONST*DSSS_AirPropagationTime;
>             }
783a811
>       //printf("sending RTS %x\n",pktRTS_);
827c855
```

```
<
---
>      //printf("sending CTS %x\n",pktCTRL_);
930a959,962
>      //printf("RetransmitRTS called %f
!!\n",Scheduler::instance().clock());
>      //double testcallvar=netif_->testcall();
> if (fuzzyflg_==0) // If 0 then no logic used...using normal code
> {
931a964,965
>            printf("\n*****\nLimit reached on retransmit RTS at %f\n",
Scheduler::instance().clock());
>            //printf("(%d)....discarding RTS:%x\n",index_,pktRTS_);
947d980
<            //printf("(%d)....discarding RTS:%x\n",index_,pktRTS_);
959a993,1066
> } else {
>      if(ssrc_ >= macmib_->ShortRetryLimit) {
>            printf("\n*****\nLimit reached on retransmit RTS at %f\n",
Scheduler::instance().clock());
>
>            if (retryflg_==1)
>            {
>                  printf("\033[36mFUZZY LOGIC RETX FAILED for
%x\033[0m\n",pktRTS_);
>                  //printf("(%d)....discarding RTS:%x\n",index_,pktRTS_);
>                  discard(pktRTS_, DROP_MAC_RETRY_COUNT_EXCFEDED);
>                  pktRTS_ = 0;
>                  /* tell the callback the send operation failed
>                  before discarding the packet */
>                  hdr_cmn *ch = HDR_CMN(pktTx_);
>                  if (ch->xmit_failure_) {
>                        ch->size() -= ETHER_HDR_LEN11;
>                        ch->xmit_reason_ = XMIT_REASON_RTS;
>                        ch->xmit_failure_(pktTx_->copy(),
>                              ch->xmit_failure_data_);
>                  }
>                  discard(pktTx_, DROP_MAC_RETRY_COUNT_EXCEEDED); pktTx_ =
0;
>                  ssrc_ = retryflg_ = 0;      // Reset counters and flags
>                  rst_cw();
>            }
>            else //if (retryflg_==0)
>            {           // No retry yet
>                  int retrytimes_ = netif_->evalRetry(pktRTS_); // Should
I retry? Function returns int value
>                  if (retrytimes_ > 0)
>                  {
>                        printf("\n\033[31mRETX USING FUZZY
LOGIC...retrying RTS %x\033[0m\n",pktRTS_);
>                        printf("\033[36m[RESET] ssrc_ %d ", ssrc_);
>                        if (retrytimes_ > 7)
>                              ssrc_ = ssrc_ - 6;
>                        else
>                              ssrc_ = ssrc_ - retrytimes_ ;
>                        printf("to %d for RTS:%x \033[0m\n",
ssrc_,pktRTS_);
>                        retryflg_ = 1;            //set flag to indicate
retried...to avoid constant loop
>
>                        struct rts_frame *rf;
>                        rf = (struct rts_frame*)pktRTS_-
>access(hdr_mac::offset_);
>                        rf->rf_fc.fc_retry = 1;
>
>                        //if(cw_ < phymib_->CWMax)
>                        inc_cw();
>                        mhBackoff_.start(cw_, is_idle());
```

```
>                       }
>                       if (retrytimes_ <= 0)
>                       {
>                               printf("\033[32mNO RETRY MADE\033[0m\n");
>                               //printf(" (%d)....discarding
RTS:%x\n",index_,pktRTS_);
>                               discard(pktRTS_, DROP_MAC_RETRY_COUNT_EXCEEDED);
>                               pktRTS_ = 0;
>                               hdr_cmn *ch = HDR_CMN(pktTx_);
>                               if (ch->xmit_failure_)
>                                       {
>                                       ch->size() -= ETHER_HDR_LEN11;
>                                       ch->xmit_reason_ = XMIT_REASON_RTS;
>                                       ch->xmit_failure_(pktTx_->copy(),
>                                       ch->xmit_failure_data_);
>                                       }
>                               discard(pktTx_, DROP_MAC_RETRY_COUNT_EXCEEDED);
pktTx_ = 0;
>                               ssrc_ = retryflg_ = 0;      // Reset counters and
flags
>                               rst_cw();
>                       }
>       }
>       } else {
>               //printf("(%d)...retxing RTS:%x\n",index_,pktRTS_);
>               struct rts_frame *rf;
>               rf = (struct rts_frame*)pktRTS_->access(hdr_mac::offset_);
>               rf->rf_fc.fc_retry = 1;
>
>               inc_cw();
>               mhBackoff_.start(cw_, is_idle());
>       }
>       }
976a1084,1087
>       //printf("RetransmitDATA called !!\n");
>       //double testcallvar=netif_->testcall();
>       //netif_->getMyInfo();
>
996c1107
<               rcount = &ssrc_;
---
>               rcount = (u_int32_t*)&ssrc_;                    // Edited :)
1000c1111
<               rcount = &slrc_;
---
>               rcount = (u_int32_t*)&slrc_;                    // Edited :)
1127c1238
<
---
>               //printf("Part 1 IDLE : %e, %f, \n", pktRx_->txinfo_.RxPr, p-
>txinfo_.CPThresh);
1138a1250
>               //printf("pktRx = %s; p = %s", pktRx_->txinfo_.RxPr,p-
>txinfo_.RxPr);
1139a1252
>               //printf("Part 2 BUSY : %e, %e, %e, %f Packet
captured!\n",pktRx_->txinfo_.RxPr, p->txinfo_.RxPr, pktRx_->txinfo_.RxPr /
p->txinfo_.RxPr, p->txinfo_.CPThresh);
1141a1255
>               //printf("Part 2 BUSY : %e, %e, %e, %f Collision!\n",pktRx_-
>txinfo_.RxPr, p->txinfo_.RxPr, pktRx_->txinfo_.RxPr / p->txinfo_.RxPr, p-
>txinfo_.CPThresh);
1280c1394
<
---
>       //printf("recvRTS %x\n",pktRx_);
1347a1462
>       //printf("recvCTS %x so discarding %x\n",pktRx_,pktRTS_);
```

```
1362a1478
>       retryflg_ = 0;
```

ns/mac/channel.h

```
87a88,94
> class Ch_80211 : public Channel {
>       public:
>               Ch_80211();
>       protected:
>               virtual double get_pdelay(Node* tnode, Node* rnode);
>               virtual void sendUp(Packet* p, Phy *txif);
> };
```

ns/mac/channel.cc

```
84a85,91
> static class Ch_80211Class : public TclClass {
> public:
>       Ch_80211Class() : TclClass("Channel/Ch_80211") {}
>       TclObject* create(int, const char*const*) {
>               return (new Ch_80211);
>       }
> }class_Ch_80211;
140a148
>       //printf("now in Channel::recv\n");
156a164
>       //printf("now in Channel::sendUp\n");
305c314,316
< WirelessChannel::WirelessChannel(void) : Channel() {}
---
> WirelessChannel::WirelessChannel(void) : Channel() {
>       printf("DEBUG : WirelessChannel opened\n");
> }
327a339,341
> Ch_80211::Ch_80211():Channel() {
>       printf("DEBUG : Channel 802.11b opened\n");
> }
328a343,425
> void Ch_80211::sendUp(Packet* p, Phy *tifp) {
>       Scheduler &s = Scheduler::instance();
>       Phy *rifp = ifhead_.lh_first;
>       Node *tnode = tifp->node();
>       Node *rnode = 0;
>       Packet *newp;
>       double propdelay = 0.0;
>       struct hdr_cmn *hdr = HDR_CMN(p);
>
>       //printf("now in Ch_80211::sendUp\n");
>       hdr->direction() = hdr_cmn::UP;
>
>          if (GridKeeper::instance()) {
>           int i;
>           GridKeeper* gk = GridKeeper::instance();
>           int size = gk->size_;
>
>           MobileNode **outlist = new MobileNode *[size];
>
>                 int out_index = gk->get_neighbors((MobileNode*)tnode,
>                                                   outlist);
>           for (i=0; i < out_index; i ++) {
>
```

```
>                 newp = p->copy();
>                 rnode = outlist[i];
>                 propdelay = get_pdelay(tnode, rnode);
>
>                 rifp = (rnode->ifhead()).lh_first;
>                 for(; rifp; rifp = rifp->nextnode()){
>                       if (rifp->channel() == this){
>                             s.schedule(rifp, newp, propdelay);
>                             break;
>                       }
>                 }
>           }
>           delete [] outlist;
>       } else {
>           for( ; rifp; rifp = rifp->nextchnl()) {
>              rnode = rifp->node();
>              if(rnode == tnode)
>                    continue;
>              /*
>               * Each node needs to get their own copy of this packet.
>               * Since collisions occur at the receiver, we can have
>               * two nodes canceling and freeing the *same* simulation
>               * event.
>               *
>               */
>              newp = p->copy();
>              propdelay = get_pdelay(tnode, rnode);
>
>              /*
>               * Each node on the channel receives a copy of the
>               * packet.  The propagation delay determines exactly
>               * when the receiver's interface detects the first
>               * bit of this packet.
>               */
>              s.schedule(rifp, newp, propdelay);
>           }
>       }
>       Packet::free(p);
> }
>
> double Ch_80211::get_pdelay(Node* tnode, Node* rnode)
> {
>       // Scheduler &s = Scheduler::instance();
>       MobileNode* tmnode = (MobileNode*)tnode;
>       MobileNode* rmnode = (MobileNode*)rnode;
>       double propdelay = 0;
>
>       propdelay = tmnode->propdelay(rmnode);
>
>       assert(propdelay >= 0.0);
>       if (propdelay == 0.0) {
>             /* if the propdelay is 0 b/c two nodes are on top of
>                each other, move them slightly apart -dam 7/28/98 */
>             propdelay = 2 * DBL_EPSILON;
>             //printf ("propdelay 0: %d->%d at %f\n",
>             //      tmnode->address(), rmnode->address(), s.clock());
>       }
>       return propdelay;
> }
```

ns/mobile/propagation.h

```
51a52,54
> #include <mac-802_11.h>
>
```

```
87a91,92
>    double CalcDistance(PacketStamp *t, PacketStamp *r, WirelessPhy *ifp,
double Pr);
>    double testcall();
```

ns/mobile/propogation.cc

```
89a90,101
> /*
> double
> Propagation::CalcDistance(PacketStamp *t, PacketStamp *r, WirelessPhy
*ifp, double Pr)
> {
>       fprintf(stderr,
>               "Propagation model %s not implemented for blablabla\n",
>               name);
>       abort();
>       return 0; // Make msvc happy
> }
> */
104a117,158
> double
> Propagation::CalcDistance(PacketStamp *t, PacketStamp *r, WirelessPhy
*ifp, double Pr)
> {
>       double L = ifp->getL();             // system loss
>       double lambda = ifp->getLambda();   // wavelength
>
>       double Xt, Yt, Zt;          // location of transmitter
>       double Xr, Yr, Zr;          // location of receiver
>
>       t->getNode()->getLoc(&Xt, &Yt, &Zt);
>       r->getNode()->getLoc(&Xr, &Yr, &Zr);
>
>       // Is antenna position relative to node position?
>       Xr += r->getAntenna()->getX();
>       Yr += r->getAntenna()->getY();
>       Zr += r->getAntenna()->getZ();
>       Xt += t->getAntenna()->getX();
>       Yt += t->getAntenna()->getY();
>       Zt += t->getAntenna()->getZ();
>
>       double dX = Xr - Xt;
>       double dY = Yr - Yt;
>       double dZ = Zr - Zt;
>
>       // get transmission power
>       double Pt = t->getTxPr();
>
>       // get antenna gain
>       double Gt = t->getAntenna()->getTxGain(dX, dY, dZ, lambda);
>       double Gr = r->getAntenna()->getRxGain(dX, dY, dZ, lambda);
>
>       return sqrt((Pt * Gt * Gr * lambda * lambda)/((4 * PI) * (4 * PI) * L
* Pr));
>
> }
>
> double
> Propagation::testcall()
> {
>       printf("arrgh in prop,testcall\n");
>       return 0;
> }
```

APPENDIX B

The main TCL script used to run the simulations with different traffic and scenarios is listed as follows.

```
#===================================================================
# Define options
#===================================================================
if {$argc !=3} {
     puts stderr "ERROR! Expected \"ns run.tcl cp sc fl\""
     exit 1
     } else {
     set opt(cp) [lindex $argv 0]
     set opt(sc) [lindex $argv 1]
     set opt(fl) [lindex $argv 2]
     Mac/802_11 set fuzzylogic_ [lindex $argv 2]      ;# 0 = no logic
     }

set opt(chan) Channel/Ch_80211
set opt(prop) Propagation/FreeSpace
set opt(netif)      Phy/WirelessPhy/Wireless_802_11_Phy
set opt(mac)        Mac/802_11
set opt(ifq)        Queue/DropTail/PriQueue
set opt(ll)         LL
set opt(ant)        Antenna/OmniAntenna
set opt(x)          1000
set opt(y)          500
set opt(ifqlen)     15
set opt(seed)       0.0
set opt(adhocRouting)    AODV
set opt(nn)         50
set opt(stop)       180.0

# ======================================================================
# Other default settings
# ======================================================================

LL set mindelay_        50us
LL set delay_       25us
LL set bandwidth_ 0    ;# not used

Agent/Null set sport_   0
Agent/Null set dport_   0

Agent/CBR set sport_        0
Agent/CBR set dport_        0

Agent/TCPSink set sport_    0
Agent/TCPSink set dport_    0

Agent/TCP set sport_        0
Agent/TCP set dport_        0
#Agent/TCP set packetSize_ 1000   ;#1460
#Agent/TCP set window_ 50
```

```
Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1

# unity gain, omni-directional antennas
# set up the antennas to be centered in node and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like Orinoco/Lucent 802.11b WLan adapter. Defaults given for
# the 914MHz Lucent WaveLAN DSSS radio interface which is built in.

Phy/WirelessPhy set CPThresh_ 10.0          ;# Default 10.0
Phy/WirelessPhy set CSThresh_ 5.011872e-12 ;# Default 1.559e-11
Phy/WirelessPhy set RXThresh_ 1.1662e-10    ;# Default 3.652e-10
Phy/WirelessPhy set Rb_ 11e6                ;# Default 2*1e6
Phy/WirelessPhy set bandwidth_ 11e6         ;# Default 2
Phy/WirelessPhy set Pt_ 0.031622777         ;# Default 0.079432824
#Phy/WirelessPhy set freq_ 2457e+6          ;# Not used
Phy/WirelessPhy set L_ 1.0                  ;# Default 1.0
Phy/WirelessPhy set debug_ false            ;# Default false

Phy/WirelessPhy/Wireless_802_11_Phy set channel_number_ 10

Mac/802_11 set bandwidth_ 11Mb
Mac/802_11 set dataRate_ 11Mb
Mac/802_11 set basicRate_ 2Mb

#===================================================================
# Main Program
#===================================================================

# Initialize Global Variables
set ns_        [new Simulator]

# set up topography object
set topo       [new Topography]
$topo load_flatgrid $opt(x) $opt(y)

# setup trace objects for ns and nam - filenames following scenarios
set tracefd   [open $opt(sc)_$opt(cp)_$opt(fl).tr w]
$ns_ trace-all     $tracefd
$ns_ use-newtrace

#set namtrace [open $opt(cp).nam w]
#$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)


# Create God
set god_ [create-god $opt(nn)]

# configure node properties
        $ns_ node-config -adhocRouting $opt(adhocRouting) \
             -llType $opt(ll) \
             -macType $opt(mac) \
             -ifqType $opt(ifq) \
             -ifqLen $opt(ifqlen) \
             -antType $opt(ant) \
             -propType $opt(prop) \
             -phyType $opt(netif) \
             -channel [new $opt(chan)] \
             #-channelType $opt(chan) \
             -topoInstance $topo \
             -agentTrace ON \
             -routerTrace ON \
             -macTrace ON \
```

```
            -movementTrace OFF

# create nodes and attach to channel
    for {set i 0} {$i < $opt(nn) } {incr i} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0      ;# disable random motion
    }


# Define node movement model
puts "Loading connection pattern..."
source $opt(cp)

# Define traffic model
puts "Loading scenario file..."
source $opt(sc)


# Define node initial position in nam
for {set i 0} {$i < $opt(nn)} {incr i} {
    # 20 defines the node size in nam
    # The function must be called after mobility model is defined
    $ns_ initial_node_pos $node_($i) 20
}

# Tell nodes when the simulation ends
for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).000000001 "$node_($i) reset";
}

# tell nam the simulation stop time
#$ns_ at  $opt(stop)    "$ns_ nam-end-wireless $opt(stop)"
$ns_ at  $opt(stop).000000001 "puts \"NS EXITING...\" ; $ns_ halt"

puts "Starting Simulation..."
$ns_ run
```

The BASH/AWK script used to parse and analyse the output trace files from the simulations is as follows.

```
#! /usr/bin/awk -f
#
# Parse a ns2 wireless trace file and generate the following stats:
# - number of flows (senders)
# - time of simulation run
# - number of packets sent (at the Application)
# - number of packets received (at the Application)
# - number of packets dropped (at the Application)
# - number of collisions (802.11)
# - average delay
# - average throughput
# - average traffic rate (measured)
#
# Updated for new wireless trace format

function average (array)
{
    sum = 0;
    items = 0;
```

```
    for (i in array) {
        sum += array[i];
        items++;
    }
#    printf("DEBUG                sum is %d, items is %d\n", sum, items);
    if (sum == 0 || items == 0)
        return 0;
    else
        return sum / items;
}

function max( array )
{
    for (i in array) {
        if (array[i] > largest)
            largest = array[i];
    }
    return largest;
}

function min(array)
{
    for (i in array) {
        if (0 == smallest)
            smallest = array[i];
        else if (array[i] < smallest)
            smallest = array[i];
    }
    return smallest;
}

BEGIN {
    total_packets_sent = 0;
    total_packets_received = 0;
    total_packets_dropped = 0;
    first_packet_sent = 0;
    last_packet_sent = 0;
    last_packet_received = 0;
}
{
    event = $1;
    time = $3;
    node = $9;
    type = $19;
    reason = $21;
    packetid = $41;

# strip leading and trailing _ from node
#    sub(/^_*/, "", node);
#    sub(/_*$/, "", node);

    if ( time < simulation_start || simulation_start == 0 )
        simulation_start = time;
    if ( time > simulation_end )
        simulation_end = time;

    if ( reason ~ /COL/ )
            total_collisions++;

    if ( type ~ /AGT/ ) {
        nodes[node] = node; # to count number of nodes
        if ( time < node_start_time[node] || node_start_time[node] == 0 )
            node_start_time[node] = time;

        if ( time > node_end_time[node] )
            node_end_time[node] = time;

        if ( event ~ /s/ && $35=="tcp") {
```

```
            flows[node] = node; # to count number of flows
            if ( time < first_packet_sent || first_packet_sent == 0 )
                first_packet_sent = time;
            if ( time > last_packet_sent )
                last_packet_sent = time;
            # rate
            packets_sent[node]++;
            total_packets_sent++;

            # delay
            pkt_start_time[packetid] = time;
        }
        else if ( event ~ /r/ && $35=="tcp") {
            if ( time > last_packet_received )
                last_packet_received = time;
            # throughput
            packets_received[node]++;
            total_packets_received++;

            # delay
            pkt_end_time[packetid] = time;
        }
        else if ( event ~ /d/ ) {
            total_packets_dropped++;
#           pkt_end_time[packetid] = time; # EXPERIMENTAL
        }
    }
    if ( event ~ /d/ && $35=="tcp") {
            total_packets_dropped++;
#           pkt_end_time[packetid] = time; # EXPERIMENTAL
        }
}
END {
    #print "" > "throughput.dat";
    #print "" > "rate.dat";
    number_flows = 0;
    for (i in flows)
        number_flows++;

    # find dropped packets
    if ( total_packets_sent != total_packets_received ) {
        printf("***OUCH*** Dropped Packets!\n\n");
        #for ( packetid in pkt_start_time ) {
          #   if ( 0 == pkt_end_time[packetid] ) {
            #       total_packets_dropped++;
#                   pkt_end_time[packetid] = simulation_end; # EXPERIMENTAL
            # }
        # }
    }

    for (i in nodes) {
        if ( packets_received[i] > 0 ) {
            end = node_end_time[i];
            #start = node_start_time[i - number_flows];
            start = node_start_time[i];
            runtime = end - start;
            if ( runtime > 0 ) {
                throughput[i] = packets_received[i]*8000 / runtime;
                #printf("%d %f %f %d %d\n", i, start, end, throughput[i],
packets_received[i]) >> "throughput.dat";
            }
        }
        # rate - not very accurate
        if ( packets_sent[i] > 2 ) {
            end = node_end_time[i];
            start = node_start_time[i];
            runtime = end - start;
            if ( runtime > 0 ) {
```

```
                rate[i] = (packets_sent[i]*8000) / runtime;
                #printf("%d %f %f %d\n", i, start, end, rate[i]) >>
"rate.dat";
            }
        }
    }

    # delay
    for ( pkt in pkt_end_time) {
        end = pkt_end_time[pkt];
        start = pkt_start_time[pkt];
        delta = end - start;
        if ( delta > 0 ) {
            delay[pkt] = delta;
            #printf("%d %f %f %f\n", pkt, start, end, delta) >> "delay.dat";
        }
    }

    # offered load
    total_runtime = last_packet_sent - first_packet_sent;
    if ( total_runtime > 0 && total_packets_sent > 0)
        load = ((total_packets_sent * 8000)/total_runtime) / 11000000; # n=o
overhead

    printf("<<RESULTS>>\nFlows : %5d \
            \nRuntime : %5.1f \
            \nOffered load : %7.4f \
            \nPkt sent : %8d \
            \nPkt rcvd : %8d \
            \nPkt drop : %8d \
            \nCollision : %10d \
            \nAvrg delay : %10.4f \
            \nMax delay : %10.4f \
            \nMin delay : %10.4f \
            \nAvrg t-put : %12d \
            \nAvrg traffic rate : %12d\n",
          number_flows,
          total_runtime,
          load,
          total_packets_sent,
          total_packets_received,
          total_packets_dropped,
          total_collisions,
          average(delay),
          max(delay),
          min(delay),
          average(throughput),
          average(rate));

}
```

The BASH batch script used to run multiple simulations consecutively is listed as follows

```
#!/bin/sh
#
# My own script to run ns simulations by batch
#
if [ $# -lt 1 ]
then
```

```
        echo "Expected inputs - CP (eg. 10nltcp)"
elif [ $# -gt 1 ]
then
        echo "Only 1 input expected"
else
        CP=$1
        test -e $CP
        if [ $? -eq 0 ]
        then
                test -e results.txt
                if [ $? -eq 1 ]
                then
                        echo -e "Results file generated ==> \c" > results.txt
                else
                        echo -e "Results file continued ==> \c" >> results.txt
                fi
                date >> results.txt

                for SC in 01 02 03 04 05 06 07 08 09 10
                        do
                        echo -e "\a\033[32mUsing scenario number $SC"
                        echo -e "\033[31mRUNNING WITHOUT F-LOGIC! \033[0m"
                        ns run.tcl $CP $SC 0

                        echo -e "\033[31mRUNNING WITH F-LOGIC \033[0m"
                        ns run.tcl $CP $SC 1

                        echo -e "\033[34mCALCULATING STATS TO FILE\033[0m"
                        echo "Scenario : $SC | Connection pattern : $CP | No
                                                        Logic" >>results.txt
                        stats $SC"_"$CP"_"0.tr >> results.txt
                        echo "Scenario : $SC | Connection pattern : $CP | With
                                                        Logic" >>results.txt
                        stats $SC"_"$CP"_"1.tr >> results.txt
                        echo "**********************************">>results.txt
                        echo -e "\033[34mREMOVING TRACE FILES\033[0m"
                        rm $SC"_"$CP"_"0.tr
                        rm $SC"_"$CP"_"1.tr
                        done

                echo -e "Batch simulation ended ==> \c" >> results.txt
                date >> results.txt
                exit 0
        else
                echo "File not found..."
        fi
fi
```