Appendix A

Cognitive Model Implemented in NS2

The cognitive model code with multi-interface support and single interface multi-channel support in NS2 can be downloaded from https://github.com/DeeptiSinghal/CR_NS2.git.

Adding Cognitive Model

To implement the cognitive model three main classes (PUModel class, SpectrumManager class and Repository class) are added to the NS2 code. The functionality of these classes are discussed in the following subsections.

PUModel.h,cc

This class describes the primary user activities on specific channels, including operating channel, physical location, and transmitting range. Primary user activity in each spectrum band is modeled as a sequence of ON and OFF periods over simulation time. All the information about primary users are contained in a log file, which is generated offline. The channel information, including their bandwidth, frequency and BER, is also contained in log file, which is also generated online. Names and use of functions in PUModel class are:

```c
// PUModel creator
PUModel();

// Method for receiving command from OTCL
int command(int argc, const char* const* argv);

// Receiving packet method (NOT used)
void recv(Packet*, Handler*);
```
// Return true if a primary user is transmitting in the same spectrum of the
cognitive user
bool is_PU_active(double timeNow, double ts, double x, double y, int channel);
bool scan_PU_activity(double timeNow, double ts, int node_id, int channel, double
    prob_niadetect);
// Write the statistics about interference on primary user receivers
void write_stat(int param);
// Check if the transmission of a cognitive user may cause interference to a primary
user receiver
void update_stat_pu_receiver(int id, double timeNow, double txtime, double x,
    double y, int channel, double TX_POWER);
// Set Repository
void setRepository(Repository* rep);
// Method to read data from primary user file and save them in the pu_activity data
structure
void read_data(char * dir);
// Method to check if a primary user is transmitting on a given spectrum at a given
time
bool check_active(double timeNow, double ts, int channel);

Listing A.1: PUModel class public functions

SpectrumManager.[h,cc]

This class performs the main functionality of the cognitive cycle. It coordinates for three main
functions of cognitive cycle: spectrum sensing, spectrum decision and the spectrum mobility. Spectrum
decision implementation supports random allocation, sequential allocation or allocation at upper layers
(like network layer). Names and use of functions in SpectrumManager class are:

// Initialize a new Spectrum Manager
SpectrumManager(Mac802_11 *mac, int id);
SpectrumManager(Mac802_11 *mac, int id, double sense_time, double transmit_time,
    bool ChDecisionMAC);
// Start method: cognitive user agent starts sensing activity on the current
channel
void start();
// Setup Functions
void setPUmodel(double prob, PUModel *p);
void setRepository(Repository* rep);
// Return true if cognitive user is NOT doing sensing and is NOT doing spectrum handoff
bool is_channel_available();
// Return true if cognitive user is NOT doing spectrum handoff
bool is_channel_switching();
// Return true if a primary user is active while receiving the packet, on the same channel
bool is_PU_interfering(Packet *p);
// Update primary user interference
bool update_pu_interference(int nodeid, double txpwr, double time_tx);
// Handler for sensing timer
void senseHandler();
// Handler for transmission timer: start sensing on the current channel
void stopTransmitting();
// Start handoff timer
void performHandoff();
// Handler for handoff management: start sensing on the new channel
void endHandoff();
// Decide whether to stay on the current channel or switch to a new channel
bool decideSwitch();
// Get the next channel to be used, based on the allocation policy
int decideSpectrum(int current_channel);
// Perform sensing and return true if primary user activity is detected on the current channel
bool sense(int id, double sense_time, double transmit_time, int channel);

Listing A.2: SpectrumManager class public functions

Repository.h, cc

This class acts as a repository for all the information related to the cognitive model. It enables information sharing among protocols at different layers. Names and use of functions in Repository class are:

Repository();
int command(int argc, const char* const* argv);
void recv(Packet*, Handler*);
Listing A.3: Repository class public functions

Modification to support cognitive model

Some of the changes in existing NS2 code are done to support the above discussed implementation of cognitive model. Changes are also done in wireless physical layer and MAC layer to support various carrier frequencies and to calculate transmission time for different bandwidth channels on the fly.

Modification in ns-lib.tcl

Function **node-CR-configure** is added so that cognitive model classes can be associated with the MAC layer of node.

```tcl
Node/MobileNode instproc node-CR-configure { puMap rep } {  
  $self instvar mac_  
  $self instvar ragent_  
  $mac_ set-pu-model $puMap
```
Listing A.4: Modification in ns-lib.tcl

Modification in packet.h

- New packet type PT_NOTIFICATION is added to send notification to upper layers about interface change and other updates from MAC layer code.

- ‘int channel.’ variable is added in struct hdr_cmn which is initialized as zero in ‘inline Packet* Packet::alloc(‘ function with ‘(HDR_CMN(p))->channel = 0’.

Modification in queue.[h,cc]

- Function ‘void resume(int channel);’ is added. This function is called by MAC Layer when a new packet has been sent to upper layers or a new packet has to be sent down.

- Function ‘Packet* deqPacket_from_channel(int channel);’ is added to deq packets which needs to sent on specific channel.

- Function ‘QueueHandler::handle’ is updated so that the correct resume function can be called. Similarly function ‘Queue::recv’ is updated so that ‘deqPacket_from_channel’ function is called in place of ‘deq’ function.

Modification in mac-802.11.[h,cc]

- Below are the functions added in mac-802.11 class.

```c
// Load the spectrum characteristics for the actual channel (bandwidth, PER etc.)
void load_spectrum(spectrum_entry_t spectrum);

// Notify the detection of a primary user on the actual channel
```
void notifyUpperLayer(int channel);

// Handler for SwitchQueueTimer
void switchqueueHandler();

// Handler for SwitchChannelTimer
void switchchannelHandler();

Listing A.5: Modification in mac-802.11 class

- **SpectrumManager** class is associated with its MAC layer using ‘sm_ = new SpectrumManager(this,index /MAX_RADIO,0.1,1.0)’ in its constructor.

- Below is the code added in command function of the MAC layer.

```c
// Set the actual cross-layer repository
else if (strcmp(argv[1], "setRepository") == 0) {
    repository_ = (Repository*) TclObject::lookup(argv[2]);
    if (index_SMAX_RADIO == RECEIVER_RADIO)
        sm_->setRepository(repository_);
    return TCL_OK;
}

// Setting the PUModel for the cognitive user
else if (strcasestr (argv[1], "set-pu-model") == 0) {
    if (index_SMAX_RADIO == RECEIVER_RADIO) {
        pumodel_ = (PUModel *) TclObject::lookup(argv[2]);
        sm_->setPUModel(0.1, pumodel_);
    }
    return TCL_OK;
}

// Start sensing activity
else if (strcasestr (argv[1], "sensing-start") == 0) {
    if (index_SMAX_RADIO == RECEIVER_RADIO)
        sm_->start();
    return TCL_OK;
}
```
Listing A.6: Modification in mac-802_11 class

- A channel number is set and stored in common header before scheduling packet for physical layer.

- Transmission time of a packet changes with the change in carrier frequency of band in multi-channel environment. Thus, transmission time calculations should be updated. Listing A.7 shows the time calculation for pktTx_ packet.

```c
timeout = ttime(pktTx_) + DSSS_MaxPropagationDelay + phymb_.getSIFS()
    + ttime(phymb_.getACKlen(), basicRate_)
    + DSSS_MaxPropagationDelay;

// changed to

double band ;

if (hdr_cmn::access(pktTx_) ->ChannelIndex() == -1)
    band = basicRate_;
else {
    MobileNode *Nnode = (MobileNode*)Node::get_node_by_address(addr());
    band = Nnode ->repository_table_spectrum_data[hdr_cmn::access(pktTx_) ->
        ChannelIndex()].bandwidth;
}

timeout = ttime(pktTx_) + DSSS_MaxPropagationDelay + phymb_.getSIFS()
    + ttime(phymb_.getACKlen(), band )
    + DSSS_MaxPropagationDelay;
```

Listing A.7: Modification in mac-802_11 class

Modification in mac-timers.[h,cc]

Timers for switching queue (SwitchQueueTimer) and channel (SwitchChannelTimer) are added which respectively call the switchqueueHandler and switchchannelHandler functions added in mac-802_11.

Modification in wireless-phy.cc

In the original wireless physical layer code in NS2, lamda_ is a fixed value. While in cognitive environment, value of lamda_ should be changed with the change in carrier frequency. Calculation of lamda_ is done as:
Listing A.8: Modification in wireless-phy.cc class

Modification in tworayground.cc

Here also lambda should be changed with the change in carrier frequency.

Listing A.9: Modification in tworayground.cc class

Modification in shadowing.cc

Here also lambda should be changed with the change in carrier frequency.

Listing A.10: Modification in tworayground.cc class

Changes on Scenario Script

Below code is used to configure nodes with cognitive functionality.
\$node\(\$i\) node-CR-configure \$pumap \$repository \$val\(\text{nCH}\)

Listing A.11: Creating a number of nodes with cognitive model

Modification in Makefile.in

Add below mentioned files for compilation in \texttt{Makefile}.

\begin{verbatim}
cognitive/SpectrumManager.o \n
cognitive/PUModel.o \n
cognitive/repository.o \n\end{verbatim}

Listing A.12: Modification in Makefile
Appendix B

Multi-interface Implementation in NS2

This appendix presents the details about implementing multi-interface support in NS2 as referred in chapter 2. In the multi-interface implementation, routing agent of mobile node is connected to multiple blocks containing link layer, interface queue, mac agent, network interface. More details about the topic can be found in [84].

TCL changes

Most of the changes are needed in the TCL code for implementation of multi-interface implementation.

Changes in ns-lib.tcl

To support multi-interface implementation, we need to create four new functions in ns-lib.tcl. The first function (Listing B.1) is called before creating the wireless node, and allows the user to specify a different number of interfaces per node. This function is called from the scenario script.

```tcl
Simulator instproc change-numifs (newnumifs) {
    $self instvar numifs_
    set numifs_ $newnumifs
}
```

Listing B.1: Function to change the number of interfaces

Second function (Listing B.2) is used to get the number of interfaces from other parts of the TCL coding.

```tcl
Simulator instproc get-numifs {} {
    $self instvar numifs_
}
```
if [info exists numifs_] {
    return $numifs_
} else {
    return ""
}

Listing B.2: Function to get the number of interfaces

Third function (Listing B.3) is used to add a channel on a node. This function is also called from scenario script before the node is created. Here ‘index’ is the number of node and ‘ch’ is the channel to be added.

Simulator instproc add-channel (index ch) {
    $self instvar chan
    puts "Adding Channel number $index";
    set chan($index) $ch
}

Listing B.3: Function to add an interface on a node

The last function (Listing B.4) is used to add the number of interfaces as an argument to the node-config command of the TCL script.

Simulator instproc ifNum (val) {
    $self set numifs_ $val
}

Listing B.4: Function to add multiple interfaces as an argument to node-config label

In addition to the above added functions, two of the already existing functions need to be modified. The first one is node-config. In this function, first we have to add the numifs_ variable to the list of arguments passed to the function as listed in listing B.5.

$self instvar addressType_ routingAgent_ propType_ macTrace_ \
    routerTrace_ agentTrace_ movementTrace_ channelType_ channel_ \
# Changed to
$self instvar addressType_ routingAgent_ propType_ macTrace_ \
    routerTrace_ agentTrace_ movementTrace_ channelType_ channel_ numifs_ \

Listing B.5: Changes in node-config function
In this function, originally \texttt{chan} variable is initialized as a single variable; this specification should be changed as shown in listing B.6.

```tcl
set chan $channel_
# Changed to
if { (info exists numifs_)} {
    set chan(0) $channel_
} else {
    set chan $channel_
}
```

Listing B.6: Changes in node-config function

Second function to be modified is \texttt{create-wireless-node}. First we have to add the \texttt{numifs} variable to the list of arguments passed to the function as listed in listing B.7.

```tcl
$self instvar routingAgent_ wiredRouting_ propInstance_ llType_ \
    macType_ ifqType_ ifqlen_ phyType_ chan antType_ \
    energyModel_ initialEnergy_ txPower_ rxPower_ \
    idlePower_ sleepPower_ sleepTime_ transitionPower_ transitionTime_ \
    topoInstance_ level1_ level2_ inerrProc_ outerrProc_ FECProc_ 
# Changed to
$self instvar routingAgent_ wiredRouting_ propInstance_ llType_ \
    macType_ ifqType_ ifqlen_ phyType_ chan antType_ \
    energyModel_ initialEnergy_ txPower_ rxPower_ \
    idlePower_ sleepPower_ sleepTime_ transitionPower_ transitionTime_ \
    topoInstance_ level1_ level2_ inerrProc_ outerrProc_ FECProc_ numifs_ 
```

Listing B.7: Changes in node-config function

When the multi-interface extension is being used, the \texttt{add-interface} function (defined in the \texttt{ns-mobilenode.tcl}) has to be called as many times as the number of interfaces added to the node as listed in B.8.

```tcl
$node add-interface $chan(0) $propInstance_ $llType_ $macType_ \
    $ifqType_ $ifqlen_ $phyType_ $antType_ $topoInstance_ \
    $inerrProc_ $outerrProc_ $FECProc_ 
# Changed to
if { (info exist numifs_)} {
    for {set i 0} {($i < $numifs_)} {incr i} {
        $node add-interface $chan($i) $propInstance_ $llType_ $macType_ \
        $ifqType_ $ifqlen_ $phyType_ $antType_ $topoInstance_ \
```

105
Listing B.8: Changes in node-config function

Listing B.9 defines the way to notify routing agent about number of interfaces used in simulation.

if ([info exist numCHs]) {
    $agent numChannel $numCHs
}

Listing B.9: Changes in node-config function

Changes in ns-mobilenode.tcl

The first function which was modified is add-target. In this function, first get-numifs function defined in ns-lib.tcl is called to get the number of interface added as listed in listing B.10.

set numIfsSimulator [ $ns get-numifs ]

Listing B.10: Changes in add-target of ns-mobilenode.tcl

Later, if-queue command of the routing agent is called as many times as the number of interfaces the node has as listed in listing B.11.

agent if-queue [ $self set ifq_0 ]
# Changed to
if ($numIfsSimulator != "") {
    for (set i 0) { ($i < [ $self set nifs_ ] ) { incr i } {
        agent if-queue $i [ $self set ifq_($i) ]
    }
}

Listing B.11: Changes in add-target of ns-mobilenode.tcl

The second function that needs to be modified is the add-target-rtagent. As we did before, we use the get-numifs function to get the number of interfaces that the node has.
Listing B.12: Changes in add-target-ragent of ns-mobilenoode.tcl

We later use this variable so as to link the routing agent with the corresponding link layer entities (LL).

```tcl
$agent target $sndT
$sndT target [$self set ll_(0)]
# Changed to
if ($numIfsSimulator != "") {
  for {set i 0} {$i < [$self set nifs_]} {incr i} {
    if ($newapi != "") {
      set sndT [$self mobility-trace Send "RTR"]
    } else {
      set sndT [cmu-trace Send "RTR" $self]
    }
    $agent target $i $sndT
    $sndT target [$self set ll_($i)]
  }
} else {
  $agent target $sndT
  $sndT target [$self set ll_(0)]
}
```

Listing B.13: Changes in add-target-ragent of ns-mobilenoode.tcl

The last function that needs to be modified is the add-interface. Originally one ARP table per node was created, now we need to add one ARP table per interface. This change is listed in listing B.14.

```tcl
if ( $arpTable_ == "" ) {
  set arpTable_ [new ARPTable $self $mac]
  # FOR backward compatibility sake, hack only
  if ($imepflag != "") {
    set drpT [$self mobility-trace Drop "IFQ"]
  } else {
    set drpT [cmu-trace Drop "IFQ" $self]
  }
  $arpTable_ drop-target $drpT
  if ( $namfp != "" ) {
    $drpT namattach $namfp
```
Listing B.14: Changes in add-interface of ns-mobilenode.tcl

This latter affects the way the MobileNode is created and reset as shown in listing B.15 and B.16.

Listing B.15: Changes in init of ns-mobilenode.tcl
$ifq_($i) reset
    if { [info exists opt(imep)] && $opt(imep) == "ON" } {
        $imep_($i) reset
    }
    if { $arp_table_($i) != "" } {
        $arp_table_($i) reset
    }
}

Listing B.16: Changes in reset of ns-mobilenode.tcl

C++ changes

Some of the changes in C++ code are also needed to adapt to multi-interface implementation. These changes are presented here.

Changes in mobilenode.[cc,h]

Originally the nodes are connected to a channel by means of two pointers, one to store the address of previous node and another to store the address of next node. To support multi-interface implementation these pointers are changed to the array of pointers as listed in B.17.

```
MobileNode* nextX_;
MobileNode* prevX_;  // Changed to
MobileNode* nextX_[MAX_CHANNELS];
MobileNode* prevX_[MAX_CHANNELS];
```

Listing B.17: Changes in mobilenode.h

The original inline `getLoc` function did not work properly with all the changes done to implement multi-interface. So, it is modified as listed in listing B.18.

```
inline void getLoc(double *x, double *y, double *z) {
    update_position();
    *x = X_;
    *y = Y_;
}
```
*z = z_;
}

// Changed to
// In mobilenode.h
void getLoc(double *x, double *y, double *z);

// In mobilenode.cc
void MobileNode::getLoc(double *x, double *y, double *z) {
    update_position();
    *x = X_;  
    *y = Y_; 
    *z = Z_; 
}

Listing B.18: Changes in mobilenode.h-cc

Changes in channel.cc

The two arrays mentioned in listing B.17 are used to manage the corresponding node lists in channel.cc. This is updated as shown in listing B.19.

# Change all nextX_ to nextX_[this->index()] and
# Change all prevX_ to prevX_[this->index()]

Listing B.19: Changes in channel.cc

Before scheduling packets, it is checked that the node’s interface is connected to the channel. If it is connected then only packet reception is scheduled. This change is listed in listing B.20.

s.schedule(rifp, newp, propdelay);
// Changed to
rifp = (rnode->ifhead()).lh_first;
}
for(; rifp; rifp = rifp->nextnode()){
    if (rifp->channel() > 0) {
        s.schedule(rifp, newp, propdelay);
    }
}

Listing B.20: Changes in channel.cc
Changes in mac-802_11.cc

The last change in the C++ code is needed to identify the interface through which a packet was received. This is done by registering the correct MAC receiving interface within the `recv` method of `mac-802_11.cc` as shown in listing B.21.

```c
if(tx_active_ && hdr->error() == 0) {
    hdr->error() = 1;
}
hdr-face() = addr(); //Added this line
if(rx_state_ == MAC_IDLE) {
```

Listing B.21: Changes in mac-802_11.cc

Changes in Routing Protocol

In this section we show how a routing agent adapts to the multi-interface structure.

Changes in routing agent implementation

Instead of using the single `ifqueue` and `target` in routing agent, we declare two arrays `targetlist` and `ifqueuelist` as shown in listing B.22. For this `MAX_IF` needs to be declared beforehand.

```c
NsObject  *targetlist[MAX_IF];
PriQueue  *ifqueuelist[MAX_IF];
```

Listing B.22: Changes in routingAgent.h

The next step would be to modify the `command` function of the routing agent class to initialize the values of the `ifqueuelist` and `targetlist` as shown in B.23.

```c
else if (argc == 4){
    if(strcmp(argv[1], "if-queue") == 0) {
        printf("[DS-ROUTE] ifqueuelist set for node %d :\n",index);
        ifqueue = (PriQueue*) TclObject::lookup(argv[3]);
        int temp_=atoi(argv[2]);
        if (temp_==nIfaces){
            nIfaces++;
        }
    }
```
ifqueue = ifqueue;
if (ifqueue) {
    return TCL_OK;
}
return TCL_ERROR;

if (strcmp(argv[1], "target") == 0) {
    int temp = atoi(argv[2]);
    if (temp == nIfaces) {
        nIfaces++;
    }
    targetlist[temp] = (NsObject *) TclObject::lookup(argv[3]);
    //fprintf(stdout, "sono %d sto attivando lo stato dell'interfaccia %d\n", index, temp);
    interface_state[temp] = 1;
    if (targetlist[temp]) {
        return TCL_OK;
    }
    return TCL_ERROR;
}

Listing B.23: Changes in command method of the routing agent class

Modification related to send a broadcast packet is listed in B.24.

Scheduler::instance().schedule(target, pkt, jitter);
//Changed to
if (nIfaces) {
    for (int i = 0; i < nIfaces; i++) {
        Packet *p_copy = p->copy();
        Scheduler::instance().schedule(targetlist[i], p_copy, jitter);
    }
} else {
    Scheduler::instance().schedule(target, pkt, jitter);
}

Listing B.24: To send a broadcast packet
Modification related to send a unicast packet on **Iface** interface is listed in B.25.

```cpp
Scheduler::instance().schedule(target_, pkt, jitter);
// Changed to
if(nIfaces) {
    Scheduler::instance().schedule(targetlist[Iface], p_copy, jitter);
}
else {
    Scheduler::instance().schedule(target_, pkt, jitter);
}
```

**Listing B.25: To send a unicast packet**

Before adding any route to routing packet, its receiving interface can be obtained with the code listed in B.26.

```cpp
if(nIfaces) {
    Iface= ch->iface->{(Mac *)ifqueuelist[0]->target())->addr();
}
else {
    Iface = -1;
}
```

**Listing B.26: Getting the interface index**

**Changes in the Route Table**

The interface (**u_int8_t rt_interface**;) is also added in to the routing table as only the next node information (**nextnode**) is not enough for forwarding the data in multi-interface environment. The corresponding function to add and/or update the routing table should be updated.

**Changes in Scenario Script**

At the beginning of the script we must specify the number of interfaces to be added as:

```plaintext
set val(ni) 3; # Number of interfaces
set val(nn) 10; # Number of users
```

**Listing B.27: Initialization of number of interfaces**

113
The creation of wireless channels should be put in the loop to create as many interfaces as specified in the above listing.

```plaintext
for {set i 0} {$i < $val(ni)} {incr i} {
    set chan_($i) [new $val(chan)]
}
```

Listing B.28: Creation of wireless channels

In order to ensure that an appropriate memory management is performed, the initialization of the god is updated as:

```plaintext
create-god [expr $val(nn)*$val(ni)]
```

Listing B.29: Initialization of the god

Adding **ifNum** option to the **node-config** function call to add multiple interfaces to node as shown below:

```plaintext
$ns node-config -adhocRouting $val(rp) \
    -llType $val(ll) \n    -macType $val(mac) \n    -ifqType $val(ifq) \n    -ifqLen $val(ifqlen) \n    -antType $val(ant) \n    -propInstance $prop \n    -phyType $val(netif) \n    -topoInstance $topo \n    -ifNum $val(ni)
```

Listing B.30: Initialization of the god

Listing B.31 shows a way to configure all nodes to use the same number of interfaces.

```plaintext
$ns change-nums $val(ni)
for {set i 0} {$i < $val(ni)} {incr i} {
    $ns add-channel $i $chan($i)
}
```

```plaintext
for {set i 0} {$i < $val(nn)} {incr i} {
    set node($i) [$ns node]
    $node($i) random-motion 0 ;# disable random motion
}
```
Listing B.31: Creating a number of nodes with same number of interfaces

Listing B.32 shows a way to create two nodes with different number of interfaces.

```
# Adding 2 interface at node 0
$ns change-numifs 2
$ns add-channel 0 $chan(0)
$ns add-channel 1 $chan(2)
set node(0) [$ns node]
$node(0) random-motion 0

# Adding 1 interface at node 1
$ns change-numifs 1
$ns add-channel 0 $chan(0)
set node(1) [$ns node]
$node(1) random-motion 0
```

Listing B.32: Creating two nodes with different number of interfaces
Appendix C

Single Interface Multiple Channel Implementation in NS2

This appendix presents the details about implementing single interface multi-channel support in NS2 as referred in chapter 3. In this implementation multiple channels are connected to node’s physical layer. Thus, most of the modifications are done in physical layer implementation and channel implementation.

TCL changes

Changes in ns-lib.tcl

To support single interface multi-channel implementation, we need to create four new functions in ns-lib.tcl. These functions and their usage is listed in listing C.1.

```tcl
# Procedure to specify different number of channels per node
Simulator instproc change-numCHs {newnumCHs} {
    $self instvar numCHs_
    puts "Number of Channels set to $newnumCHs";
    set numCHs_ $newnumCHs
}

# Procedure to get the number of channels from other parts of the TCL coding
Simulator instproc get-numCHs {} {
    $self instvar numCHs_
    if {[info exists numCHs_]} {
        return $numCHs_
    } else {
        return ""
    }
}
```
# Procedure to get the channel given the its index from other parts of the TCL coding

Simulator instproc get-channel {index} {  
$self instvar chan  
return $chan($index)
}

# Procedure to add multiple channels as an argument to node-config label
Simulator instproc ChannelNum {val} {  
$self set numCHs_ $val
}

Listing C.1: Functions to change the number of interfaces

In addition to the above added functions, two of the already existing functions need to be modified. The first one is node-config. In this function, first we have to add the numCHs_ variable to the list of arguments passed as listed in listing C.2.

$self instvar addressType_ routingAgent_ propType_ macTrace_ \  
      routerTrace_ agentTrace_ movementTrace_ channelType_ channel_ numifs_\  
# Changed to  
$self instvar addressType_ routingAgent_ propType_ macTrace_ \  
      routerTrace_ agentTrace_ movementTrace_ channelType_ channel_ numifs_ numCHs_\  

Listing C.2: Changes in node-config function

In this function, originally chan variable is initialized either as a single variable or based on number of interfaces; this specification should be changed to include the single interface multi-channel case as shown in listing C.3.

if { [info exists numifs_]} {  
    set chan(0) $channel_  
} else {  
    set chan $channel_  
}

# Changed to  
if { [info exists numifs_]} {  
    set chan(0) $channel_  
} elseif { [info exists numCHs_]} {
Listing C.3: Changes in node-config function

Second function to be modified is `create-wireless-node`. First, we have to add the `numCHs` variable to the list of arguments passed as listed in listing C.4.

```bash
$self instvar routingAgent_ wiredRouting_ propInstance_ llType_ \  macType_ ifqType_ ifqlen_ phyType_ chan antType_ \  energyModel_ initialEnergy_ txPower_ rxPower_ \  idlePower_ sleepPower_ sleepTime_ transitionPower_ transitionTime_ \  topoInstance_ levell_ level2_ inerrProc_ outerrProc_ FECProc_ numifs_  
# Changed to
$self instvar routingAgent_ wiredRouting_ propInstance_ llType_ \  macType_ ifqType_ ifqlen_ phyType_ chan antType_ \  energyModel_ initialEnergy_ txPower_ rxPower_ \  idlePower_ sleepPower_ sleepTime_ transitionPower_ transitionTime_ \  topoInstance_ levell_ level2_ inerrProc_ outerrProc_ FECProc_ numifs_ numCHs_  
```

Listing C.4: Changes in node-config function

In `create-wireless-node` function, the `add-interface` function call has to be updated as listed in C.5.

```bash
if ( [info exists numifs_] ) {  
    for {set i 0} ($i < $numifs_) {incr i} {  
        $node add-interface $chan($i) $propInstance_ $llType_ $macType_ \  $ifqType_ $ifqlen_ $phyType_ $antType_ $topoInstance_ \  $inerrProc_ $outerrProc_ $FECProc_  
    }  
} else {  
    $node add-interface $chan(0) $propInstance_ $llType_ $macType_ \  $ifqType_ $ifqlen_ $phyType_ $antType_ $topoInstance_ \  $inerrProc_ $outerrProc_ $FECProc_  
}  
```
Listing C.5: Changes in node-config function

Listing C.6 defines the way to notify routing agent about number of channels used in simulation. This code is also added in `create-wireless-node` function.

Listing C.6: Changes in node-config function

Changes in ns-mobilenode.tcl

In this file `add-interface` function is updated to support multi-channel implementation. Listing C.7 shows the modification needed and their usage.
if ($channelnum!="") {
    for { set j 0; j < $channelnum; incr j } {
        set channel_(j) [$ns get-channel $j]
    }
} else {
    set channel [$ns get-channel 0]
}
#
# To pass the number of channels to ARP table code
if ($channelnum!="") {
    $arptable numChannel $channelnum
}
#
# Below update pass the channel information to physical layer. Originally single
# channel information was passed. The update pass information about multiple
# channels in multi-channel case
if { [$info exists channel]} {
    $netif channel $channel
}
#changed to
if ($channelnum!="") { ;# for every single-interface, add multi-channel
    for { set i 0; i <$channelnum; incr i } {
        $netif mchannel $channel_(i)
    }
} else {
    if { [$info exists channel]} {
        $netif channel $channel
    }
}
#
# Below update shows how the multiple channels get connected through physical
# layer, mobile node and topology code in multi-channel case.
$channel addif $netif
$channel add-node $self
$stopo channel $channel
#changed to
if ($channelnum!="") { ;# for every single-interface, add multi-channel
    $stopo numChannel $channelnum
}
for ( set i 0) {($i<$channelnum) (incr i) {
    $channel_($i) addif $netif
    $channel_($i) add-node $self
    $stopo mchannel $i $channel_($i)
}
} else {
    $channel addif $netif
    $channel add-node $self
    $stopo channel $channel
}

Listing C.7: Changes in add-interface of ns-mobilenode.tcl

C++ changes

Some of the changes in C++ code are also needed to adapt to multi-channel implementation. These changes are presented here.

Changes in mobilenode.[cc,h]

Other than the changes mentioned in appendix B, mobile node should store channel information as show in listing C.8.

```cpp
// Spectrum Entry Information defined before MobileNode class
struct repository_spectrum_data {
    double bandwidth; // current bandwidth
    double frequency;
    double per; // Packet Error Rate (PER) value
};

// Defined inside the MobileNode class
repository_spectrum_data repository_table_spectrum_data[MAXCHANNELS];

// Initialization of defined spectrum data inside MobileNode::MobileNode(void) {}
for (int channel = 0; channel < MAXCHANNELS; channel++) {
    repository_table_spectrum_data[channel].bandwidth = 0.0;
    repository_table_spectrum_data[channel].frequency = 0.0;
    repository_table_spectrum_data[channel].per = 0.0;
```
Listing C.8: Changes in mobilenode.h

Changes in channel.h-cc

Below variables are defined inside `WirelessChannel` class to store channel parameters.

```c
// Defined inside WirelessChannel class as public variables
double bandwidth_;  // current bandwidth
double frequency_;  // central frequency
double per_;        // Packet Error Rate (PER) value
```

Listing C.9: Changes in channel.h

Listing C.10 shows the modifications done in `channel.cc`.

```c
// The channel index is stored in Channel::recv function before sendUp function
call
hdr_cmn::access(p)->localif_ = index_;
```

```c
// Channel parameters defined in channel.h are bind to tcl parameters. This code is
// added in WirelessChannel::WirelessChannel(void)
bind("bandwidth_", &bandwidth_);
bind("frequency_", &frequency_);
bind("per_", &per_);
```

```c
// First it is checked that the node’s interface is connected to the channel and if
// it is connected then only packet reception is scheduled.
rifp = (rnode->ifhead()).lh_first;
for(; rifp; rifp = rifp->nextnode()){
    if (rifp->channel() > 0) {
        s.schedule(rifp, newp, propdelay);
    }
}
// changed to
rifp = (rnode->ifhead()).lh_first;
for(; rifp; rifp = rifp->nextnode()){
```
if (rifp->getchannelnum() > 0) {
    if (rifp->getmultichannel(this->index()) == this) {
        s.schedule(rifp, newp, propdelay);
    }
}
else {
    if (rifp->channel() == this) {
        s.schedule(rifp, newp, propdelay);
    }
}

Listing C.10: Changes in channel.cc

Changes in phy.[h,cc]

Listing C.11 shows the modification done in phy.h. In this file three protected variables are defined: nchannel to store number of channels supported, ChannelIndex to store the current channel index, and multichannel to store pointers to channels. Also two functions, getmultichannel (return the channel given its index) and getchannelnum (return the total number of channels), are added.

Channel *multichannel[MAX_IF];
int nchannel;
int ChannelIndex;

virtual Channel* getmultichannel(int index) const {
    return multichannel[index];
}

inline int getchannelnum() {
    return nchannel;
}

Listing C.11: Changes in phy.h

Listing C.12 shows the modifications done in phy.cc.

// Initialization of added variables. This should be added in Phy::Phy()
nchannel=0;
for(int i=0;i<MAX_IF;i++) {
    multichannel[i]=0;
}

// Below code is added in command function to add channel pointer in multichannel array
if (strcmp(argv[1], "mchannel") == 0) {
    assert(multichannel[nchannel] == 0);
    multichannel[nchannel] = (Channel*) obj;
    nchannel++;
    downtarget_ = (NsObject*) obj;
    // LIST_INSERT_HEAD() is done by Channel
    return TCL_OK;
}

// Update of setchnl function to support multi-channel
void Phy::setchnl (Channel *chnl) {
    if (nchannel > 0) {
        multichannel[chnl->index()] = chnl;
    }
    else {
        channel_ = chnl;
    }
}

Listing C.12: Changes in phy.cc

Changes in topography.h,cc

In topography class also, nchannel and multichannel are defined to store number of channels supported and pointers to those channels. Definition and initialization of these variable are listed in listing C.13.

// Definition of added variables
int nchannels;
WirelessChannel *multichannel_[MAX_CHANNEL];
// Initialization of added variables. Below case is added in Topography() function
nchannels=0;

for(int i=0;i<MAX_CHANNEL;i++)
    multichannel_[i]=0;

Listing C.13: Changes in topography.h

In topography.cc, updateNodesList function is modified as show in listing C.14.

void Topography::updateNodesList(class MobileNode* mn, double oldX) {
    channel_->updateNodesList(mn, oldX);
}

// Changed to
void Topography::updateNodesList(class MobileNode* mn, double oldX) {
    if(nchannels==0) {
        channel_->updateNodesList(mn, oldX);
    } else {
        for(int i=0;i<nchannels;i++) {
            multichannel_[i]->updateNodesList(mn, oldX);
        }
    }
}

Listing C.14: Changes in topography.cc

Changes in Routing Protocol

In this section we show how a routing agent adapts to the single interface multi-channel structure. Modification related to send a broadcast packet is listed in C.15.

if(nIfaces) {
    for(int i=0;i<nIfaces;i++) {
        Packet *p_copy=p->copy();
        Scheduler::instance().schedule(targetlist[i], p_copy, jitter);
    }
} else {
    Scheduler::instance().schedule(target_, pkt, jitter);
}
Listing C.15: To send a broadcast packet

Modification related to send a unicast packet on CH channel is listed in C.16.

if(nIfaces) {
    Scheduler::instance().schedule(targetlist[Iface], p_copy, jitter);
}
else {
    Scheduler::instance().schedule(target_, pkt, jitter);
}
Scheduler::instance().schedule(target_, pkt, jitter);
}

Listing C.16: To send a unicast packet

Before adding any route to routing packet, its receiving channel can be obtained with the code listed in C.17.

if(nIfaces) {
    Interface = ch->iface()->(Mac *)ifqueuelist[0]->target()--addr();
}
else if (numCHs) {
    Interface = ch->ChannelIndex();
}
else {
    Interface = -1;
}

Listing C.17: Getting the interface index

Changes in Scenario Script

At the beginning of the script we must specify the number of channels to be added as:

```
set val(nCH) 3; #Number of channel
set val(nn) 10; #Number of users
```

Listing C.18: Initialization of number of channels

The creation of wireless channels should be put in the loop to create as many channels as specified in the above listing.

```
for {set i 0} ($i < $val(nCH) ) { incr i } {
    set chan_($i) [ new $val (chan) ]
}
```

Listing C.19: Creation of wireless channels

In order to ensure that an appropriate memory management is performed, the initialization of the god is updated as:
Listing C.20: Initialization of the god

Adding **ChannelNum** option to the **node-config** function call to add multiple channels to node as shown below:

```plaintext
$ns node-config -adhocRouting $val(rp) \n   -llType $val(ll) \n   -macType $val(mac) \n   -ifqType $val(ifq) \n   -ifqLen $val(ifqlen) \n   -antType $val(ant) \n   -propInstance $prop \n   -phyType $val(netif) \n   -topoInstance $topo \n   -ChannelNum $val (nCH)
```

Listing C.21: Adding multiple channels to node

Listing C.22 shows a way to configure all nodes to use the same number of channels.

```plaintext
$ns change-numCHs $val(ni)
for {set i 0} { ($i < $val(ni)) } { incr i } {
   $ns add-channel $i $chan($i)
}
for {set i 0} { ($i < $val(nn)) } { incr i } {
   set node($i) [ $ns node]
   for {set j 0} { ($j < $val(nCH)) } { incr j } {
      $node($i) ChannelBandwith $j $Bandwidth($j)
      $node($i) ChannelFrequency $j $Frequency($j)
      $node($i) ChannelPer $j $Per($j)
   }
   $node($i) random-motion 0 ;# disable random motion
}
```

Listing C.22: Creating a number of nodes with same number of channels

Listing C.23 shows a way to create two nodes with different number of channels.

```plaintext
# Adding 2 interface at node 0
```
$ns\ change\-num\mbox{CHs}\ 2
$ns\ add\-channel\ 0\ \$chan(0)
$ns\ add\-channel\ 1\ \$chan(2)
set\ node(0)\ \{\$ns\ node\}
$node(0)\ Channel\\ Bandwidth\ 0\ 900000
$node(0)\ Channel\\ Frequency\ 0\ 24045000000
$node(0)\ Channel\\ Per\ 0\ 0.0
$node(0)\ Channel\\ Bandwidth\ 1\ 100000
$node(0)\ Channel\\ Frequency\ 1\ 24096000000
$node(0)\ Channel\\ Per\ 1\ 0.0
$node(0)\ random\-motion\ 0
#\ Adding\ 1\ interface\ at\ node\ 1
$ns\ change\-num\mbox{CHs}\ 1
$ns\ add\-channel\ 0\ \$chan(0)
set\ node(1)\ \{\$ns\ node\}
$node(0)\ Channel\\ Bandwidth\ 0\ 900000
$node(0)\ Channel\\ Frequency\ 0\ 24045000000
$node(0)\ Channel\\ Per\ 0\ 0.0
$node(1)\ random\-motion\ 0

Listing\ C.23:\ Creating\ two\ nodes\ with\ different\ number\ of\ channels