Cognitive Radio Cognitive Network Simulator

Table of Contents:

1 INTRODUCTION to CRCN
   1.1 BACKGROUND of COGNITIVE RADIO
   1.2 MOTIVATION
   1.3 WHY BASED on NS2?
   1.4 CRCN SIMULATOR OVERVIEW
2 Functionality Overview
   2.1 Design OVERVIEW for CRCN
   2.2 CR routing
   2.3 CR MAC
3 CRCN USER GUIDE
   3.1 SOFTWARE and Installation
   3.2 Exemplary Demonstrations
   3.3 GUI USER GUIDE
4 MISCELLANEOUS
   4.1 REFERENCES
   4.2 FAQ
   4.3 Cognitive Radio Research Groups
   4.4 Contact information

1 Introduction to CRCN

1.1 Background of Cognitive Radio

Cognitive radio network is a new emerging research area recently. It enhances the existing software-defined radio [1], whose physical layer behavior is largely defined in software. Cognitive radio has the following characteristics [2]. First, it is aware of its environment and its capabilities. Second, it is able to independently alter its physical layer behavior based on its previous experience and its current environment. Finally, it is capable of performing the complex adaptation strategies according to the cognitive cycle shown in [3]. With these capabilities, when spectrum environment changes around cognitive user, it is capable of sensing these changes and independently changing its physical layer settings such as transmission power, channel selection and etc to meet some constraints or QoS requirements of the users.
Cognitive radio gains popularity in the research area because it enables the current fixed spectrum channel assigned by FCC to be utilized by the new users. For example, most of the spectrums assigned to TV channels are idle most of the time, while wireless network users share a small range of spectrum, 2.4 G Hz and 5G Hz. When there are many wireless users at a time, the network is congested because of the limited channel. With the spectrum opportunities [4] provided by the cognitive radio network, the wireless network users are able to share the idle spectrum for TV channel, on the condition that it does not interfere with the normal TV channel.

1.1 Motivation
As cognitive radio research is emerging, more and more researchers are looking forward to a simulator that is suitable for cognitive radio. However, there is no existing simulator that is suitable for the demand of cognitive radio simulations. Many researchers implemented their algorithms for cognitive radios on existing network simulator such as NS-2 [5], OPNET [8], QUALNET [11]. However, since these simulators are created for the ordinary wireless network, researchers can not easily implement their cognitive radio algorithms over those simulators. Hence, there is a demand to extend existing simulators to support cognitive radio simulators. We make use of existing NS-2 to extend it to support cognitive radio network simulation.

1.2 Why based on NS-2?
There are several reasons to base on NS-2 for the development of cognitive radio simulators. First, NS-2 is open source software. Any contributions to NS-2 are accessible by people around the world. Second, NS-2 provides an interface for users to configure different network protocol at each network layers, which is essential for simulation. Third, NS-2 already provides many radio models, such as 802.11, 802.16, 802.15.3, 802.15.4. Users can make use of these radio models for cognitive radio network simulations. Finally, NS-2 has incorporated with different topology and traffic generators, which enable users to create different simulation scenarios.

1.3 CRCN Simulator Overview
This cognitive radio cognitive network (CRCN) simulator is a software based network simulator for network-level simulations. It is based on open-source NS-2 (network simulator 2). CRCN simulator supports performance evaluations for the proposed dynamic spectrum resource allocation, power control algorithms, and the adaptive Cognitive Radio (CR) networking protocols including the CR MAC and the CR Routing protocols. This simulator uses NS-2 to generate realistic traffic and topology patterns. For
each node in this simulator, a reconfigurable multi-radio multi-channel PHY layer is available by customizing the spectrum parameters such as transmission power, propagation and etc.

![Figure 1 Architecture of CRCN Simulator](image)

The architecture block diagram for this simulator is illustrated in Figure 1. CRCN simulator enables the interface parameters transmission between different layers, as shown in arrows with blue color. Users just need to replace their own Routing and MAC algorithms according to NS-2 protocol design requirements with the existing one in the CRCN.

2 Functionality Overview

The functionality provided by this simulator is summarized as follows.

- **Support for CR Routing**
  - Multi-radio multi-channel support
- Single-radio multi-channel support
- Interface to select radio
- Interface to select channel
- Heterogeneous radio and spectrum environment
- Information needed during routing process

**Support for CR MAC**

- Single-radio multi-channel support
- Multi-radio multi-channel support
- Interface to select channel
- Information needed during dynamic spectrum access

**Support for CR PHY**

- Reconfigurable spectrum parameters and radio parameters
- Interference information
- SINR/SNR physical model

**CR Routing/CRMAC algorithms**

Several exemplary algorithms for CR Routing and CR MAC are given in the CRCN simulators. These algorithms illustrate how to use the functionality provided by this simulator. Also, these algorithms provide simulation environment for CR networking protocols. Users can replace these components with their own algorithms, and test the performance of their designs.

**Performance Evaluation for CR algorithms**

This simulator contains several evaluation metrics for performance evaluations of the algorithms at different layers. Currently, the evaluation metric include **summate**, interference.

**Graphical User Interface (GUI)**

A user-friendly GUI is provided to define simulation scenarios. Users can select the different network protocols, different topology and traffic model and etc. Also, users can start simulation and see the simulation result through this GUI.

Please note that users need to follow the rules of NS-2 software architecture to design their protocols to fit in this simulator.
2 Design Overview for CRCN

2.1 SYSTEM REQUIREMENTS

This simulator is based on NS-2 (version 2.31). All codes have been tested in ns-2.31 under Linux environment. To use the GUI provided by this simulator, it is required that your system is installed with java runtime environment (JRE) 6 or above. Also, gnuplot should be installed under Fedora. The GUI is tested under Fedora 4.

2.2 Design Overview

To support simulation in CR wireless network, several new frameworks and functions are added in NS-2 for the functionality provided in section 2. These frameworks and functions are summarized as follows.

- Design Overview for CR Routing
- Design Overview for CR MAC/PHY
- Interaction between the core of CRCN simulator and GUI

Design Overview for CR Routing

Users create the TCL script to configure the number of radios and channels needed for simulations. The basic idea is to create multiple radios or channels through the TCL library according to user's script, which invokes the creation of several copies of LL, queue, MAC, NetIf, and channels for each radio in C++ library. The design overviews are shown through the following figures. The blocks with darker color are modified by CRCN to provide a complete solution.

A. Equal number of Radios and channels

Figure 2(a) shows the design for routing when the radio and channel are equals. In this design, the channel is created for each radio.
**B. Single radio multi-channel**

Figure 2 (b) shows the design structure for single-radio, multi-channel structures. Multiple channel copies are created for this structure. Although routing algorithms can still use the same design as shown in figure 2 (a) for single-radio multi-channel by checking the radios being used, existing MAC protocols in NS-2 need to be modified to support this checking. On the contrast, by adopting the design shown in figure 2 (b), new introduced routing algorithms for multi-channel, single-radio can use the existing MACs in NS-2 protocol stack, which is convenient to users.
C. Unequal number of radios and channels
Figure 2 (c) shows the design structure for multi-radio, multi-channel design with unequal number of radios and channels.
Due to the limitation of NS-2 design structure, the interface for routing and MAC algorithms to use the above structures are slightly different. Users can refer to the next following sections for more details.

There are some existing works [6][7][9] to enhance the NS-2 to support multi-radio multi-channel based on the design as shown in figure 2. However, there are several disadvantages for this kind of design if we only adopt them for CR simulation, which are listed as follows.

- MAC layer is not aware of multiple channels created for each radio. Channel selection must be made in routing layer instead of MAC layer, which is conflict with most current MAC protocol designs. Thus, additional work is required in MAC to support multiple channels in MAC, which can be found in next section.
- MAC layer address confliction is introduced by the related works.
- Since channels are created in the same way, these radios and channels have the same radio and spectrum characteristic, which is not sufficient to support the simulation in heterogeneous CR network.
- The radio and channel number are equal. Some routing and MAC algorithms simulations such as common control channel based algorithms are not supported if we only adopt the previous contribution work alone.

**Design Overview for CRMACPHY**
The high level design for MAC share some similar design structures with routing. Since the multi-channel design as shown in Figure 2 (a) for routing is transparent to the MAC and PHY layer, to provide MAC and PHY algorithm a multi-channel environment, multi-channel is needed at each radio. The design includes single radio multi-channel and multi-radio multi-channel, which are visible to both MAC and PHY layers. In both designs, multiple channel objects are created through the TCL library. Nodes can be switched to different channel objects during the simulation process.

**A. Single radio multi-channel**
Figure 3 shows the high level software design for single radio multi-channel support for MAC and PHY layer. The components with blue color are modified in CRCN to support the functionality required.

![Design Overview for CR MAC](image)

**B. Multi-radio, Multi-channel**
Figure 4 shows the high level software design for multi-radio multi-channel support for
MAC and PHY layer. The components with blue color are modified in CRCN to support the functionality required. This design combines the design shown in Figure 2 (a) and Figure 3.

**Interaction between the core of CRCN simulator and GUI**

In the CRCN simulator, the interaction between the core of CRCN simulator and GUI is enabled through the multi-threading mechanisms. The GUI process creates threads needed for the core of CRCN simulator. And share data area is used for the data sharing between different threads and GUI.

**2.2 CR ROUTING**

This simulator provides simulation support for channel assignment and CR routing. The functionalities provided are listed as follow.

- Create CR multi-radio/single-radio multi-channel simulation environment
  
  It includes the creation of radios with homogenous and heterogeneous radio parameters, as well as spectrum with homogenous and heterogeneous propagation property. Here, the radio parameter includes predefined transmission power on each radio.

- Interface for radio and channel decision
- Interference information
• Traffic information
• Channel Utilization information

Also, a routing implementation example is shown to use the interfaces provided by this simulator. User can replace this example with their CR routing algorithms.

The supported functionality for CR routing in this simulator is shown in the diagram as follow in the darker color. Starting from the top, this figure is explained according to the interfaces parameters provided to the user. Details...

• **CRR(Channel decision/Radio selection/Route selection decision)**

  CR Routing includes channel decision, radio selection, and route selection decision.
  • Channel decision includes channel number and transmission power on the selected channel.
  • Radio selection means which radio to use.
  • Route decision means which path for packet forwarding.

  These decision need to be sent down to the lower layer through the multi-channel, multi-radio/single-radio support module provided by CRCN, which is included in the diagram with darker blue color.

• **STICR(Spectrum Utilization information, Traffic information and Interference information from the Channel and Radio selected by CR Routing)**

  The lower layer collects the information needed by CR Routing and sends the information to the upper layers. The information needed by CR Routing includes spectrum utilization information, traffic information, and interference information. The information is processed at the **Traffic estimation/Spectrum utilization estimation/Interference estimation** functional block provided by CRCN.
2.2.1 CREATE MULTI-RADIO/SINGLE RADIO MULTI-CHANNEL ENVIRONMENTS

A. Homogenous multi-radio multi-channel
The radios and channels created in this section are homogenous. The homogenous radios and channels are configured through wireless TCL simulation script. According to the relationship between radio number and channel number, CR-CN provides three kinds of structures to satisfy different requirements from users: (1) Equal number of radios and channels, (2) Single-radio multi-channel, (3) Unequal number of radios and channels.

(1) Equal number of radios and channels
The API to configure homogenous radios and channels in TCL script is shown in the following four components. These four components need to be added in the simulation script by the user. Please note that if GUI is used to generate the wireless simulation script, this section can be skipped.

Component 1. Define the radio number in TCL script
...
set val(ni) 2 ;# 2 interface
...

Component 2. New channel objects according to the interface number provided in 1

```tcl
for {set i 0} { $i < $val(ni)} {incr i} {
    set chan_(\$i) [new $val(chan)] ;# new channel objects
}
```

Component 3. Configure the radio and channel option through the API provided

```tcl
... $ns_node-config -adhocRouting $val(rp) \ ...
... -ifNum $val(ni) ;#configure the interface number for node
    -channel $chan_(0) ;# configure the first channel object
...
```

Component 4. Assign the channel objects to the simulator

```tcl
for {set i 0} { $i < $val(ni)} {incr i} {
    $ns_add-channel $i $chan_(\$i) ;# pass the channel objects into channel array
}
```

Through the script created as instructed in this section, the radio is associated with each channel. Only routing layer is aware of each radio/channel. Users can refer to section 2.2.2 Radio and Channel decision for the radio and channel decision.

(2) **Single-radio multi-channel**

For the creation of single-radio and multi-channel, user can refer to the details in CRMAC. The channel decision from the CR routing only need to be stored into the channelindex_, which is newly added in packet header. The lower layer has been modified to support this decision.

(3) **Unequal number of radios and channels**

If user wants to have unequal number of radios and channels, user can refer to the CR MAC detail 2.3.1 (Part B. multi-radio, multi-channel) about how to create the TCL script. The channel created as in 2.3.1 can be visible to both routing and MAC layer. Here is an example script. Users should refer to section 2.2.2 about how to use the interface provided for radio and channel decision.
B. Heterogeneous multi-radio and multi-channel

Similar to the creation of homogenous radios and channels, heterogeneous radios and channels are created through wireless simulation script. The following four components should be added into the simulation script by the user.

Component 1, 2 and 3 are the same in A.

Component 4. Assign different channel objects with different spectrum parameters to the simulator as follow.

```
# configure the ns channel...
$ns_add-channel-new-phy 0 $chan_(1) 0.4Propagation/FreeSpace
# update this information for node 1
set node_(1) [$ns_node]         #:apply the change on node
...
$ns_add-channel-new-phy 0 $chan_(0) 0.4 default
...
```

In the example above, `add-channel-new-phy` is the newly added API. \(\theta\) is the radio number. \(Schan_(1)\) is the channel object created in component 2. 0.4 is the transmission power of this channel. Propagation/FreeSpace is the propagation method of this channel, which is used to simulate the possible wireless environment. The transmission power and propagation method can be set to default if you want to use the default value provided by NS-2.

2.2.2 Radio and channel decision

A. Radio decision

In the routing modules, radio decision can be stored in the routing entry such as the rt-rt_if field in AODV routing packet header. When your routing protocol is going to send down the packet, the following code should be added. Example for modifying the routing module can be found in here.

```
Example for the radio or channel decision used in AODV protocol.
...
    Scheduler::instance().schedule(targetlist[rt-rt_if], p, 0.);
..  
```

Downtarget array targetlist is created when the node is configured in TCL script. rt-rt_if is the radio decision specified by the routing packet.
The way to handle radio decision is the same for all of the three structures defined in section 2.2.1.

**B. Channel decision**

**Equal number of radios and channels**
When the radio and channel are created as in 2.2.1 A, channel is tied with each radio, thus, the radio decision is equal to the channel decision in this case.

**Single radio multi-channel**
For single radio multi-channel or the channels that are only visible to MAC layer, user need to follow the method in Single-radio and multi-channel for the creation of TCL script. The channel decision from the CR routing only need to be stored into the channelindex, which is newly added in packet header. The low layer has been modified to support this decision.

**Unequal number of radios and channels**
The channel decision should be specified by routing algorithms through the channelindex of packet header. The low layer has been modified to support this decision. The design structure for single radio multi-channel and this part shares the common design at low layer.

**2.2.3 INFORMATION NEEDED BY CR ROUTINGANDCHANNEL ASSIGNMENT**

**A. Interference information**
The interference information is associated with each node over each channel. Several kinds of interference information are provided: (1) Channel that has minimum interference, (2) Current interference value over a specific channel, and (3) Historical interference information of a specific channel.

To access the above information, routing module should have a pointer which points to the current node object. The code highlighted in bold in components 1 to 4, which are illustrated through AODV routing protocol, should be added in your routing module.

```
Component 1. Declare a Mobilenode pointer in routing module
Class AODV{
    ...
protected:
```
Component 2. Initialize the MobileNode pointer through in the command method of your routing module.

```c
MobileNode *node_;  // pointer to mobilenode object

...

} // for the node agent

else if(strncmp(argv[1], "node") == 0) {
    node_ = (MobileNode*) TclObject::lookup(argv[2]);
    if (node_) {
        return TCL_OK;
    }
    return TCL_ERROR;
} ...
```
After adding the components as shown above, the interference information can be obtained as follows.

(1) **Channel that has minimum interferences**
Interface for obtaining channel with minimum interference around one node.

Module Mobilenode: use the following method to obtain channel number which has minimum interference.

```cpp
Class Mobilenode {
    public:
        ...
        // return the channel id that minimum interference
        int ChannelwithMinimumIf{ return chanwithminIf; }
    ...
}
```

(2) **Current interference value over a specific channel**
Use the following method to obtain interference value. Please note the interference value is sometimes too small and you may need to scale it by some factor to see the result. This

Module Mobilenode: obtaining interference

```cpp
Class PacketPr {
    ...
    //obtain the interference information for current node
    double getInterference(int channelno);  
    ...
}
```

(3) **Historical interference information**
Besides obtaining the channel with minimum interference, the interference information over each channel around can be obtained through the ITfile generated by the simulator. The format of the ITfile is as follow. User can obtain their information through ITfile according to this format.

<table>
<thead>
<tr>
<th>ITfile format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
</tr>
</tbody>
</table>

**B. Noise**
TCL command for setting noise for physical layer is as follow. This should be added in the simulation script if you want to define the noise.
TCL command: setting noise for physical layer
Phy/WirelessPhy noise_ 0.1  \# 0.1 can be replaced by any other noise value

With the noise information and interference information, user can change the physical layer of NS-2 to use SINR/SNR reception model according to their needs.

C. Traffic information
For CR routing, it is necessary to use the sensing traffic information to predict the future traffic information in the neighborhood, and derive the best user strategy. The interface this simulator providing are in the two formats (1) obtaining the current traffic information (2) obtaining the historical traffic information.

(1)Obtaining the current traffic information

Module Mobilenode: use the following function in bold to obtain traffic information

```java
class PacketPr {
    public:
        ...
        int getTrafficCount() {return trafficcount;}
        ...
}
```

User can access this function through the pointer to Mobilenode as in 1.4.3.A. For example, in the AODV, user can access traffic information like this way.

```java
Module Mobilenode: use the following function in bold to obtain traffic information

...  
int traffic= node_ ->PacketPr_ ->getTrafficCount();
...  
```

(2)Obtaining the historical traffic information
The historical traffic information is stored in the Trafficfile. User can obtain this information based on the format provided.

Trafficfile format

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>node id</th>
<th>traffic information</th>
</tr>
</thead>
</table>

D. Channel Utilization
For some CR routing algorithms, they may need the channel utilization for routing decision. For users who need channel utilization information, users need to add the following codes in the MAC layer they are using.
Step1: Add the timer definitions in the MAC you are using.

### Module Macxx.h: add timers for channel utilization

```cpp
// Timer for channel utilization
class ChanUtil_Timer : public TimerHandler {
public:
    ChanUtil_Timer(Macxx *a) : TimerHandler() { a_ = a; }
    void expire(Event *e);
protected:
    Macxx *a_;
};
```

### Module Macxx.h: add timers for calculate channel utilization

```cpp
// Timer for Calculate channel utilization
class CalChanUtil_Timer : public TimerHandler {
public:
    CalChanUtil_Timer(Macxx *a) : TimerHandler() { a_ = a; }
    void expire(Event *e);
protected:
    Macxx *a_;
};
```

Step2: Add the timer functions’ implementation in the MAC you are using.

### Add the Module Macxx.cc: add timer functions’ implementation

```cpp
// Timer for recording channel utilization
void ChanUtil_Timer::expire(Event *)
{
    a_->ChanUtil_Calculate();

    // Traffic timer expires for every node or the node specified
    resched(0.01);
}

// Timer for calculating channel utilization
void CalChanUtil_Timer::expire(Event *)
{
    a_->CalChanUtil_Calculate();

    // Traffic timer expires for every node or the node specified
```
resched(1);

}

Step3: Add the timer definitions in the MAC class you are using.
Module Macxx.h: add timers variable in Macxx class
Class Macxx {
    ...
    friend class ChanUtil_Timer;
    friend class CalChanUtil_Timer;
    ...
    public:
        FILE *fp;
        ChanUtil_Timer ChanUtil_Timer_;
        CalChanUtil_Timer CalChanUtil_Timer_;
        void ChanUtil_Calculate(); //function for recording channel utilization
        void CalChanUtil_Calculate(); //function for calculating channel utilization
        void increaseChanUtil() {chanutilcount++;}
        void resetChanUtil() {chanutilcount=0;}
        int getChanUtil() {return chanutilcount;}
    ...
    private:
        ...
        int chanutilcount;
        ...
}

Step4: Add the timer functions’ implementation in the MAC class you are using.
Module Macxx.cc: add the related function implementation as defined in Macxx class.
void
Macxx::ChanUtil_Calculate()
{
    //if mac is not idle
    if(!is_idle()){
        //increase channel utilization counter
        increaseChanUtil();
    }
}
void
Macxx::CalChanUtil_Calculate()
{
    if (((fp = fopen("Chanfile", "a")) != NULL))
    {
        fprintf(fp, "%d %d
", NOW, ((MobileNode*)(netif_->node()))->nodeid(), getChanUtil()/100);
        fclose(fp);
        resetChanUtil();
    } else {
        printf("fail to open file");
    }
}

After finishing the above steps, users are able to obtain two kinds of channel utilization information: (1) Current channel utilization information (2) Historical channel utilization information.

(1) Current channel utilization information
The current channel utilization information can be obtained through function `getChanUtil()` defined in class Macxx. User can obtain this information based on the format provided.

(2) Historical channel utilization information
The historical channel utilization information is stored in `Chanfile`. User can obtain this information based on the format provided.

<table>
<thead>
<tr>
<th>Chanfile format</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>node id</td>
</tr>
</tbody>
</table>

### 2.3 CRMAC

This simulator provides simulation support for CR MAC. The functionalities provided are listed as follows.

- CR multi-radio/single-radio multi-channel simulation environment
- Interface for channel decision
- Interface for transmission power decision
- Interference information
- Traffic information

Also, two example MAC algorithms are used to illustrate how to use the interfaces provided by this simulator. User can replace these examples with their CR MAC algorithms.

The supported functionalities for CR MAC in this simulator are shown in the following diagrams based on the types of the MAC algorithms. Details…

**Contention based MAC**

Figure 7 shows the high level designs for contention based MAC algorithms. Starting from the CR MAC, the interface parameters are as follows.

- **TC** (Transmission power and Channel selection)

After CR MAC makes the decision, which includes the transmission power and Channel selection, the simulator will send down the information through the multi-channel support functional block provided by CRCN. Before the multi-channel support functional block is introduced, the MAC layer is only aware of a single channel.

- **TICC** (Traffic information, Interference information over specific Channel, Communication information over common control channel)

The physical layer will provide the traffic information, interference information and communication information to the upper layer through the **Information Block** provided by CRCN simulator.
**Collision freeMAC**

Figure 8 shows the high level designs for contention based MAC algorithms. Starting from the DSA, the interface parameters are as follows.

- **TC** (Transmission power and Channel selection)

  After DSA makes the decision, which includes the transmission power and Channel selection, the simulator will send down the information according the **multi-channel support** functional block provided by CRCN. The DSA decision will be verified by user defined metric, which is entered by user from GUI. If the DSA decision does not satisfy the user defined metric, the user defined metric block will provide this information to DSA.

- **TIC** (Traffic information, Interference information over data Channel)

  The physical layer will provide the traffic information, interference information to the upper layer through the **Information Block** provided by CRCN simulator.
2.3.1 CREATE CR MULTI-CHANNEL ENVIRONMENTS

A. Single-radio multi-channel
The following four components should be added by user to create the multi-channel environments. The channels created in this part have identical spectrum parameters. Please note that if you are using GUI provided by this simulator, this section can be skipped.

<table>
<thead>
<tr>
<th>Component 1. Define the channel number in TCL script</th>
</tr>
</thead>
<tbody>
<tr>
<td>set val(channum) 2 ;# 2 channels are available in the network</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component 2. New channel objects according to the interface number provided in Component 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>for {set i 0} { $i &lt;$val(channum)} {incr i} {</td>
</tr>
<tr>
<td>set chan_($i) [new $val(chan)] ; #new channel objects</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Component 3. Configure the multiple channel option
... $ns\_node-config\,-adhocRouting\,val(rp) \\
... -ChannelNum\,val(channum) ; # configure channel number \\
  -channel\,chan\,(0) ; # configure the first channel object \\
...

Component 4. Assign the channel objects to the channel array of the simulator

... for \{set\,i\,0\} \{i < val(ni) \} \{incr\,i\} \{
  $ns\_add-channel\,i\,\,chan\,(i) ; # initialize the channel array with channel objects
  \}
...

B. Multi-radio multi-channel

The following four components should be added by user to create the multi-radio, multi-channel environments for MAC. The channels created in this part have identical spectrum parameters. Please note that if you are using GUI provided by this simulator, this section can be skipped.

Component 1. Define the radio and channel number in TCL script

set val(ni) 2 ; # number of radios
set val(channum) 2 ; # 2 channels are available in the network

Component 2. New channel objects according to the radio and channel number provided in Component 1.

for \{set\,i\,0\} \{i < \{expr $val(ni)*$val(channum)\}\} \{incr\,i\} \{
  set chan\,(i) \{new\,$val(chan)\}
  \}

Component 3. Configure the multiple radio and channel option

$ns\_node-config\,-ifNum\,val(ni)\,-channel\,chan\,(0)$
$ns\_node-config\,-ChannelNum\,val(channum)$

Component 4. Assign the channel objects to the channel array of the simulator

for \{set\,i\,0\} \{i < \{expr $val(ni)*$val(channum)\}\} \{incr\,i\} \{
  $ns\_add-channel\,i\,\,chan\,(i)$
  \}


2.3.2 CHANNEL/RADIO DECISION

The channel decision from CR MAC or DSA algorithm should be set through the interfaces provided as follow, which reflects the channel that the receiver is using to receive packet. A new field `channelindex_` is added into the packet header. After CR MAC or DSA make channel decision, this decision can be stored into this field.

```
Module packet.h:

struct hdr_cmn {
    ...
    int channelindex_;  
    ...
}
```

The code as follow should be added in the `sendDown` function for wirelessphy. In our design, multiple channel object pointer is used to index the channel object. Hence, it is appropriate to use channel index derived in CR MAC or DSA algorithms to differentiate channel, which helps to achieve conflict free or reduce interference among neighboring nodes. `channelindex_` defined in the packet header can carry this information from CR routing, CR MAC or DSA to the wireless physical layer.

```
Module WirelessPhy: Interface for channel decision in MAC

void WirelessPhy::sendDown(Packet *p) {
    ...
    // Send the packet
    //channel_)->recv(p, this);
    //send packet over the channel specified by channel index.
    multichannel[hdr_cmn::access(p)->channelindex_]->recv(p, this);
    ...
}
```

If user’s MAC is for single-radio and multi-channel network, the above code change is enough to differentiate different channels for MAC protocols. However, users still need to design their own collision avoidance mechanism or DSA according to the NS-2 protocol design structures.

If user’s MAC is for multi-radio and multi-channel network, codes related to the radio selection is needed besides the channel decision code as above. Depending on the usages
of each radio, users need to add code in the MAC to differentiate what current radio is for. For example, if each node has 2 radios: radio 1 is assigned to common control channel; radio 2 is assigned to data channel. In user’s code for MAC, user need to restrict the radio 1 only handle the control packets, and radio 2 only handle the data packets. To provide flexibility for this simulator, we do not add any restrictions for the multi-radio, multi-channel MAC. Users also need to choose appropriate routing protocol for their multi-radio, multi-channel MAC. To differentiate the current radio from one another, users can just use the index_, which is declared in Mac.

2.3.3 TXPOWER DECISION

Packet will be transmitted using the default transmission power if transmission power is not specified in simulation script. However, as CR MAC and DSA algorithms may control the txpower over each channel to control the interference to primary users or nearing neighbors, it is necessary to provide an interface, which can control the txpower during the simulation.

The DSA algorithm can be implemented in the following functions, which are in bold.

Module Mobileno: Interface for obtaining the transmission power from DSA algorithms.

```cpp
Class PacketPr{
public:
    ...
    // Obtain DSA decision according to different input parameters.
    double getDSAPr(int receivernode);
    double getDSAPr(int receivernode, int channelindex);

    ...
}
```

Then, add the following interface for controlling the maximum transmission power and obtaining DSA power decision. Please note that other physical layer can be modified in the similar way.

Module WirelessPhy: Interface for channel decision in MAC

```cpp
int WirelessPhy::sendUp(Packet *p) {
    
    ...
    double DSAPr = (MobileNode*)node()->PacketPr_.getDSAPr();
    // if the Pr is within the limit range of user, assign DSAPr to Pr.
    Pr = DSAPr;
```
To determine the txpower, the DSA algorithms need to know the identity of sender and receiver. In some DSA algorithms (such as game based), sender and receiver has negotiated the channel decision, and thus the power control algorithm compute the txpower over the negotiated channel.

### 2.3.4 INFORMATION NEEDED BY CRMAC

**A. Interference information**
The interference information is associated with each node over each channel. Several kinds of interference information are provided: (1) Channel that has minimum interference, (2) Current interference value over a specific channel, and (3) Historical interference information. Users can use the corresponding API into their algorithm to obtain the information.

**Channel that has minimum interferences**
Interface for obtaining channel number with minimum interference around one node. Use the following method to obtain channel number which has minimum interference.

```java
Module Mobilenode: obtain channel with minimum interference
Class PacketPr {
    ...
    //return the channel id with minimum interference.
    int ChannelWithMinimumIf() {
        return chanWithMinIf;
    }
    ...
}
```

**Current interference value over a specific channel**
Use the following method to obtain interference value. Please note the interference value is sometimes too small and you may need to scale it by some factor to see the result.

```java
Module Mobilenode: Interface for channel decision in MAC
Class PacketPr {
    ...
    //obtain the interference information for current node
    double getInterference(int channelno);
    ...
}
```
(3) Historical interference information

To obtain the historical information of node i, the following option to select whether to record interference on node i should be added in simulation script by the user.

<table>
<thead>
<tr>
<th>TCL script: add the interference option</th>
</tr>
</thead>
<tbody>
<tr>
<td>for {set i 0} {si &lt; $val(nn)} {incr i} {</td>
</tr>
<tr>
<td>set node_(si) [sns_node]</td>
</tr>
<tr>
<td>$node_(si) set SingleIfMultiChan 1 ;</td>
</tr>
<tr>
<td>$node_(si) set recordIfAll 1 ; # enable interference information</td>
</tr>
<tr>
<td>$node_(si) random-motion 0 ;# disable random motion</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

If this option is added, the interference information about node i is provided. If using GUI, the interference information for all nodes is provided by default.

Besides obtaining the channel with minimum interference, the interference information over each channel around can be obtained through the ITfile generated by the simulator. The format of the ITfile is as follow. User can obtain their information through ITfile according to this format.

<table>
<thead>
<tr>
<th>ITfile format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp     node_id interference channel_id</td>
</tr>
</tbody>
</table>

B. Noise

TCL command for setting noise for physical layer is as follow. This should be added in the simulation script if you want to define the noise.

<table>
<thead>
<tr>
<th>TCL command: setting noise for physical layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phy/WirelessPhy noise_ 0.1 ;# 0.1 can be replaced by any other noise value</td>
</tr>
</tbody>
</table>

C. Traffic information

For intelligent based MAC, it is necessary to use the sensing traffic information to predict the future traffic information in the neighborhood, and derive the best user strategy. The interface this simulator providing are in the two formats (1) obtaining the current traffic information (2) obtaining the historical traffic information.

(1) Obtaining the current traffic information
Use the following function in bold to obtain traffic information.
Module MobileNode: obtain traffic information

class PacketPr {
public:
    ...
    int getTrafficCount() {return trafficcount;}
    ...
}

Users can access this function through the pointer to mobilenode. For example, in the MAC for 802.11, users can access traffic information like this way.

Module MobileNode: obtain traffic information

    ... int traffic= ((MobileNode*)(netif_->node()))->PacketPr.getTrafficCount();
    ...

(2) Obtaining the historical traffic information
The historical traffic information is stored in the Trafficfile. User can obtain this information based on the format provided.

<table>
<thead>
<tr>
<th>Trafficfile format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp  node id  traffic information</td>
</tr>
</tbody>
</table>

D. Packet based primary user detection
As MAC/DSA algorithms may need to know the existence of primary user, we also introduce how to implement the packet based primary user detection. The primary user sends out a primary user packet to indicate its existence. Other users that receive this packet can distinguish the existence of primary user from this packet. The primary user packet is controlled by a broadcast timer. As the primary user packet can collide with normal data packet, thus the sending and controlling of this packet should be done in the MAC layer.

The primary user mechanism is implemented in the Macgenhanced module. User can add the similar code in other MAC protocol. Here is just an example. User need to define whether to send the packet according to movements of primary user or use timer to control.

The packet module is modified to contain information about primary user. A new broadcast timer primaryuser time is added in MAC to control the broadcast of this packet. The sendprimaryuserpacket is added to send primary user packet. The recv function is changed to handle the primary user packet. User can refer to Macgenhanced for more detail.
3 CRCN USER GUIDE

3.1 Installation

3.1.1 NS2 COMPONENTS

Step 1: Make sure your system meets the system requirements.
Step 2: Download crcn.zip. (update on 7/27/2009)
Step 3: Change your working directory to XXX/ns-allinone-2.31/ns-2.31/
Step 4: Unzip the crcn.zip. Copy the files under each subfolders in crcn into the corresponding folders in XXX/ns-allinone-2.31/ns-2.31/. For example, copy the files under crcn/mac/ into XXX/ns-allinone-2.31/ns-2.31/mac/. Please note that it will overwrite some of your ns file in the same directory. Thus, backup the ns directory before executing this command.
Step 5: Add the following three lines in the Makefile, which is under XXX/ns-allinone-2.31/ns-2.31/.

```
wcett/wcett_logs.o wcett/wcett.o 
wcett/wcett_rtable.o wcett/wcett_rqueue.o 
mac/macng.o mac/maccon.o\nmac/macengenhanced.o\n```
Step 6: Run the following commands under XXX/ns-allinone-2.31/ns-2.31/ directory:
```
make clean
make depend
make
```

Users can use the NS2 components without using the GUI.

3.1.2 GUI

Step 2: Install the gnuplot from http://www.gnuplot.info/.
Step 3: Put throughput.pl and throughputcbn.pl under your ns working directory XXX/ns-allinone-2.31/ns-2.31/.
Step 4: Run CRCN.jar (updated)
Step 5: Refer to CRCN GUI guide to run the GUI if you want to use this GUI.
3.2 EXEMPLARY DEMONSTRATIONS

3.2.1 CR ROUTING
The folder wcett[10] provides an implementation example for using the interface provided by this simulator. The implementation is based on [10] with some simplification. The script file testwcett.tcl (with random.tcl) provided with the patch is a demonstration about how to run the multi-channel routing protocol. This section illustrates how this routing protocol can be embedded into the simulator.

Details...

3.2.2 CRMAC
The files maccon and macng in the mac folder provide implementation examples for using the interface provided by this simulator. This section illustrates how these MAC protocols can be embedded into the simulator. Please note that these MAC protocols are only for demonstration and testing purpose. Users need to design their own collision avoidance scheme or DSA according to the NS-2 protocol design structures.

Details...

3.2.3 DESIGN TEST CASE STUDY
This section illustrates how we test the functionality provided by CRCN simulator.

Details...

Exemplary implementation for CR Routing

(1)TCL Library Modification
Module ns-lib.tcl: invoke create-wcett-agent in function create-wireless-node as other routing protocols

Simulator instproc create-wireless-node args {
    ...
    switch -exact $RoutingAgent {
        ...
        WCETT {
            set ragent [$self create-wcett-agent $node]
        }
    }
}
(2) C++ code change

Define the interface queue and targetlist for multiple interfaces.

```cpp
Module wcett: define the interface queue and targetlist for multiple interfaces

```class WCETT: public Agent {

    ...

Protected:
    int nIfaces; //total interfaces per node
    NsObject *targetlist[MAX_IF];
    PriQueue *ifqueuelist[MAX_IF];
    MobileNode *node_;
```
Next, when sending the packets over the specific interfaces, just using the interface number as an index to send the packet as follow.

```c
if(!nfaces) { //if multiple interfaces are defined.
    for(int i=0;i<nFaces;i++) {
        Packet *p_copy=p->copy();
        Scheduler::instance().schedule(targetlist[i], p_copy,
            0.01 * Random::uniform());
    } 
    else { //if only one interface is defined
        Scheduler::instance().schedule(target_, p,
            0.01 * Random::uniform());
    }
}
```
A. Contention based MAC

The file maccon.h and maccon.cc in folder mac is an example implementation for contention based MAC. In this Maccon, each node randomly selects a channel from a predefined channel list. Since each node is not aware of what channel being used by other node, collision is still existed for this random channel selection. The channel information selected by each node is passed to PHY layer through the packet header.

In this Maccon, the channel list array number is equal to the number of nodes in the simulation script test4maccon.tcl. Nodes randomly select the channel based on this channel array. In the topology and traffic file topo4.tcl, 3 pairs of nodes are transmitted simultaneously. Since the channel decision of this MAC depends on specific transmission scenario, users need to add change to make Maccon adapt to other scenarios.

Besides the code change in MAC, the following code should be added in the sendDown function of wirelessphy.

```c
Module wirelessphy: example about how to use interface for MAC
void WirelessPhy::sendDown(Packet *p){
    ...
    // Send the packet
    //channel_->recv(p, this);

    multichannel[hdr_cmn::access(p)->channelindex]->recv(p, this);

    ...
}
```

Here is the simulation result for Maccon.

B. Collision free MAC

The file macng.h and macng.cc in folder mac is an example implementation for collision frees MAC. In intelligent based and schedule based MAC, they both require negotiation about channel decision and transmission parameters. Most of the current intelligent based and schedule based MAC are negotiated over common control channel. In this example, we illustrate the how to negotiate through common control channel and make the channel decision.
The operation of Macng is divided into two phases. In the first phase, each node sends out strategy packet with preferred receiving channel. If there is an empty channel available, then it is selected as the preferred receiving channel. If there is no empty channel, node shares the channel with the node that is furthest away from it. In the second phase, node will use the channel selected during the first phase to send and receive data. The channel selection algorithms we are using is a simple rule, which is for the demonstration purpose, hence, collision still exists.

Besides the code change in MAC, the following code should be added in the sendDown function of wirelesssh.

```
Module wirelesssh: example about how to use interface for MAC

void WirelessPhy::sendDown(Packet *p){
    ...
    // Send the packet
    // channel_->_recv(p, this);

    multichannel[hdr_emn::access(p)->channelindex_j->recv(p, this);

    ...
}
```

The simulation script for this MAC can be found in test4macng.tcl. Here is the simulation result for Maccon and Macng.

The second example Macgenenhanced illustrates how to implement a DSA based MAC through multiple rounds of negotiation. This MAC example shows the power adaptation process of each node. Each node adjusts its transmission power according to the equation in [12].

In this example, each node runs a DSA algorithm to calculate its transmission power. The DSA algorithm is run on each node whenever it has packet to send. In order to have enough information for the DSA algorithms, each node needs to negotiate with each other about the transmission power they are adopting. During negotiation phase, each node broadcast a strategy packet to its neighbors. Each node calculates the SINR based on the information from the strategy packet, and stores the related SINR information of send. During the data transmission phase, each node looks through its database to find the related SINR information for the receiver and run the DSA algorithm. For simplicity, the equation from [12] is adopted for the transmission power calculation. From the simulation result, it is found that the performance is greatly affected by period of negotiation. If the broadcast period is short, the performance is not as good as when the period is longer. The reason is that control packets collide more frequently with the data packets. This problem
can be avoided if multiple channels are used. Users can implement similar DSA-driven MAC following this example. Please note that the purpose of this implementation is not towards the good performance, as good performance needs careful selection of broadcast period and etc, which is beyond the scope of this simulator design.

The implementation of Macenhanced is also based on Mac-simple. Thus, the packet sending and receiving model is still the same as Mac-simple. Function sendStrategy() is used during the negotiation process to send the strategy packet. getInterference() function provided by this simulator is used to obtain the SINR in the receive handler, which handles the receiving packet. As stated above, the transmission power is calculated before sending out the packet to the downtarget. Please note that this MAC only acts as an example for implementation. Performance is not the focus of this example.

The simulation script for this MAC can be found in test4macgenenhanced.tcl.

### 3.2.3 Design test case studies

To test and verify the framework provided by CRCN simulator, demos and debug code test are used together to test and verify the functionality. Section A compares the methods used for testing. Section B introduces the test process.

#### A. Comparisons between the demo method and debug code method

For the demos, they can provide direct performance comparisons to users, however, the performance of demo is hard to analyze for the following reasons.

- The performance of demos depends on the algorithm design.
- The performance analysis of demos involves analysis of topology, traffic, and interaction with other layers. Thus, it makes the analysis is complex and not so accurate.
- For each demo, it can only adapt to one kind of simulation setting, such as single radio, multi-channel, or multi-radio, multi-channel.

For the debug code test, it is the most accurate and direct way to test the framework. The only disadvantage is that it is hard to understand by people other than the designers and NS-2 users. However, the debug code is not included in the release code to avoid confusing to users.

#### B. Test process

**1) Test the multi-radio provided by design structure figure 2 (a)**

Specific topology as follow is designed to test whether multi-radio is worked under this design structure. Each node is forced to make the channel selection as shown in figure 1 in
the routing module. Every node has 2 radios, but only node 2 has more chance to use 2 radios at the same time. Then we can check at node 2 whether multi-radio is used through simulation.

![Figure 1. Test topology for multi-radio](image)

From the snapshot of a trace file about the radio usages in table 1, two MACs associated with 2 radios of node 2 are busy at the same time. Thus, the multi-radio functionality is proved. Users can also write their simple debug code to test the functionality.

Table 1. Radio usages at node 2

<table>
<thead>
<tr>
<th>Time</th>
<th>Node ID</th>
<th>Radio Index</th>
<th>MAC Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.010000</td>
<td>2</td>
<td>4</td>
<td>busy</td>
</tr>
<tr>
<td>0.010000</td>
<td>2</td>
<td>5</td>
<td>busy</td>
</tr>
<tr>
<td>0.020000</td>
<td>2</td>
<td>4</td>
<td>busy</td>
</tr>
<tr>
<td>0.020000</td>
<td>2</td>
<td>5</td>
<td>idle</td>
</tr>
<tr>
<td>0.030000</td>
<td>2</td>
<td>4</td>
<td>idle</td>
</tr>
<tr>
<td>0.030000</td>
<td>2</td>
<td>5</td>
<td>idle</td>
</tr>
<tr>
<td>0.040000</td>
<td>2</td>
<td>4</td>
<td>idle</td>
</tr>
<tr>
<td>0.040000</td>
<td>2</td>
<td>5</td>
<td>idle</td>
</tr>
<tr>
<td>0.050000</td>
<td>2</td>
<td>4</td>
<td>busy</td>
</tr>
<tr>
<td>0.050000</td>
<td>2</td>
<td>5</td>
<td>idle</td>
</tr>
<tr>
<td>0.060000</td>
<td>2</td>
<td>4</td>
<td>idle</td>
</tr>
<tr>
<td>0.060000</td>
<td>2</td>
<td>5</td>
<td>idle</td>
</tr>
<tr>
<td>0.070000</td>
<td>2</td>
<td>4</td>
<td>busy</td>
</tr>
<tr>
<td>0.070000</td>
<td>2</td>
<td>5</td>
<td>busy</td>
</tr>
<tr>
<td>0.080000</td>
<td>2</td>
<td>4</td>
<td>idle</td>
</tr>
<tr>
<td>0.080000</td>
<td>2</td>
<td>5</td>
<td>idle</td>
</tr>
<tr>
<td>0.090000</td>
<td>2</td>
<td>4</td>
<td>idle</td>
</tr>
<tr>
<td>0.090000</td>
<td>2</td>
<td>5</td>
<td>idle</td>
</tr>
<tr>
<td>0.100000</td>
<td>2</td>
<td>4</td>
<td>idle</td>
</tr>
<tr>
<td>0.100000</td>
<td>2</td>
<td>5</td>
<td>idle</td>
</tr>
</tbody>
</table>

(2) **Test the multi-radio multi-channel provided by design structure** figure 2 (a)

Since the multi-channel provided through this design is only visible to routing, WCETT is used to test the multi-channel provided by figure 2(a). However, this implementation is a
simplifier version. With the design structure in figure 2 (a), channel is tied with each radio, thus, the corresponding radio and channel relationship is fixed. For example, radio 1 is always on Channel 1; radio 2 is always on Channel 2.

Figure 2, figure 3 and figure 4 compare the performance of WCETT when the radios or channel numbers are different. The simulation settings, topology and traffic for these simulations are the same except the radio and channel number. The throughput is increased nearly by half when the radios and channels are increased from 1 to 2. But the performance is not increased significantly when the radios and channels are increased from 2 to 3. The reason is that not all the radios are being used all the time. Table 2 is the snapshot of a trace file that shows the radio usage at node 2 when it has 3 radios. Users can also write their simple debug code to test the functionality.

![Throughput over Time](image)

**Figure 2.** WCETT with number of radio=1 and number of channel=1
Figure 3. WCETT with number of radio=2 and number of channel=2

Table 2. Radio usages at node 2

```
format<time, node id, radio index, MAC status>
2.700000 2 6 busy
2.700000 2 7 busy
2.700000 2 8 idle
2.710000 2 6 busy
```
(3) **Test the single radio, multi-channel in figure 2 (b)**

Since the multi-channel in this design structure is only visible to MAC, we design some simple MAC algorithms to test this functionality. This simple mac are based on the existing mac-simple in NS-2. Routing layer can just pass the channel decision to lower layer when routing algorithm is responsible for making channel decision.

**Macon**

Based on mac-simple, Maccon has random channel selection to make use of the multi-channel provided. Maccon is designed for specific topology, since our purpose is just to test multi-channel. Figure 2 and figure ? compare the throughput under mac-simple and maccon with the same simulation settings except channels.

<table>
<thead>
<tr>
<th>Time</th>
<th>Channel Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.710000</td>
<td>2 7 busy</td>
</tr>
<tr>
<td>2.710000</td>
<td>2 8 idle</td>
</tr>
<tr>
<td>2.720000</td>
<td>2 6 busy</td>
</tr>
<tr>
<td>2.720000</td>
<td>2 7 busy</td>
</tr>
<tr>
<td>2.720000</td>
<td>2 8 idle</td>
</tr>
<tr>
<td>2.730000</td>
<td>2 6 busy</td>
</tr>
<tr>
<td>2.730000</td>
<td>2 7 busy</td>
</tr>
<tr>
<td>2.730000</td>
<td>2 8 idle</td>
</tr>
</tbody>
</table>
Figure 5. Throughput for mac-simple
Macng is used to test single radio, multi-channel functionality. The operation of Macng is divided into two phases. In the first phase, each node sends out strategy packet with preferred receiving channel. If there is an empty channel available, then it is selected as the preferred receiving channel. If there is no empty channel, node shares the channel with the node that is furthest away from it. In the second phase, node will use the channel selected during the first phase to send and receive data. From figure 7 and figure 8, when the numbers of channels increase from 2 to 3, the throughput also increases. However, as we increase the number of channels to 4 or 5, the throughput is not increased significantly. The reason is that only 3 pairs of nodes are used under this simulation.
Figure 7. Test for Macng and the number of available channel are 2
3 pairs of nodes; topology and traffic file: topo4.tcl
If we use another simulation setting, the throughput change for Macng can be observed. With more nodes and more simultaneous transmissions in the random topology and traffic scenario, more channels are used. Thus, the increase of throughput can be observed when the number of channels increases from 5 to 8. The drop of performance from 30 seconds in figure 9 is due to too many collisions. As Macng is just used to provide example for the user, collisions introduced in mac-simple still exist. In comparison, when channel is increased to 10, the throughput is increased. Thus, multi-channel functionality is correct here since increasing the channel will increase the throughput, since the same simulation setting is used.
Figure 9. Test for Macng with random topology and traffic channel number =5
In sum, the functionalities are tested through the debug code and demos. However, as explained in section A, to obtain an accurate result, the performance analysis of demos test need to involve the consideration of topology and traffic pattern, and the design of the algorithms.

(4) Test the multi-radio, multi-channel in figure 2 (c)
This structure is a combination of figure 2 (a) and figure (b). Because of this combination, the supported functionalities in figure 2 (a) and figure (b) are included in figure2 (c). But still, some tests are done to make sure the reliability of this structure. Users need to carefully design MAC and choose an appropriate multi-radio routing to use this structure.

The simulation settings are as follows. Random topology and traffic is used. Routing protocol is WCETT. 802_11 is used as the MAC protocol. Each node has 2 radios and 3 channels per radios. Since 802_11 is default to channel 0, thus only channel 0 will be used in this case. The Maccon, Macng won’t work since they are not designed for this structure.
The test result is shown as follow. This result is to make sure no other framework change is needed. It is protocol designer’s responsibility to implement their MACs. The simulation result shown in figure 11 demonstrates that the multi-radio, multi-channel structure is correct.

Figure 11. Test for the structure in figure 2 (c)