TUHH
Technische Universität Hamburg

ComNets
Institute of Communication Networks

Code Demonstration on IntAirNet Channel Model Implementation

Fatema Khan
Institute of Communication Networks

Content

1. Contents of ini file
2. Contents of Receiver Model
3. Contents of Error Model

Contents of ini file(1)

TraceBasedErrorModel.cc | *omnetpp.ini | UnitDiskReceiverCustomized.cc | IntAirNet_Channel_model.ned

```
[Config IntAirNet_Channel_model]
description = IntAirNet project:- Channel model
network = IntAirNet_Channel_model
repeat=10

#Recording statistics
*.host[1].radio.receiver.errorModel.packetErrorRate:vector.vector-recording = true
record-eventlog = false

#setting simulation time
sim-time-limit = 4011s

*.numHosts = 2

# ARP
*.host[*].ipv4.arp.typename = "GlobalArp"

# UDP app
#host[0] = Tx
#host[1] = Rx
*.host[*].numApps = 1
*.host[0].app[0].typename = "UdpBasicApp"
*.host[0].app[0].destAddresses = "host[1]"
*.host[0].app[0].destPort = 5000
*.host[0].app[0].messageLength = 100B
*.host[0].app[0].sendInterval = 4ms
```

Form | Source

Contents of ini file(2)

```
27 *.host[0].app[0].packetName = "UDPData"
28 *.host[1].app[0].typename = "UdpSink"
29 *.host[1].app[0].localPort = 5000
30 *.host[0].app[0].startTime=uniform(9s,10s)
31 *.host[0].app[0].stopTime=4010s
32
33 #Defining the radio and the mac protocol
34 *.host[*].wlan[0].typename = "WirelessInterface"
35 *.host[*].wlan[0].radio.typename = "UnitDiskRadioCustomized"
36 *.host[*].wlan[0].mac.typename = "AckingMac"
37
38 *.host[*].wlan[0].mac.useAck = false
39 *.host[*].wlan[0].mac.fullDuplex = false
40 *.host[*].wlan[0].mac.headerLength = 23B
41
42 *.host[*].**.bitrate = 10 Mbps
43
44 #setting the communication range ---------------------------------------
45 *.host[*].wlan[0].radio.transmitter.communicationRange = 1922070m
46 *.host[0].wlan[0].radio.displayCommunicationRange = true
47
48 #configuring movement of the receiver and Transmitter node-------------
49 *.*host[*].mobility.typename = "StationaryMobility"
50
51 *.host[*].mobility.initFromDisplayString = false
52
```

Contents of ini file(3)

```
52
53 *.host[0].mobility.initialX = ${45000m, 55000m, 70000m, 85000m, 110000m, 140000m, 175000, 220000m, 275000m, 360000m}
54 *.host[0].mobility.initialY = 0m
55 *.host[0].mobility.initialZ = 18000m
56 *.host[1].mobility.initialX = 0m
57 *.host[1].mobility.initialY = 0m
58 *.host[1].mobility.initialZ = 18000m
59
60 #adding interference in the receiver side ---------------------------------
61 *.host[*].wlan[0].radio.receiver.ignoreInterference = true
62 *.host[*].wlan[0].radio.transmitter.interferenceRange = 1932000m
63 *.host[0].wlan[0].radio.displayInterferenceRange = true
64
65 #Parameterizing
66 *.host[*].wlan[0].radio.receiver.Receiver_bandwidth_in_KHz=500 #in kHz
67 *.host[*].wlan[0].radio.receiver.Radio_horizon=922060 #in m
68 *.host[*].wlan[0].radio.receiver.Tx_power=42 #in dBm
69 *.host[*].wlan[0].radio.receiver.Tx_antenna_gain=3 #in dBi
70 *.host[*].wlan[0].radio.receiver.Rx_antenna_gain=3  #in dBi
71 *.host[*].wlan[0].radio.receiver.Tx_loss=4 #in dB
72 *.host[*].wlan[0].radio.receiver.Rx_loss=4 #in dB
73 *.host[*].wlan[0].radio.receiver.Noise_figure=6 #in dB
74 *.host[*].wlan[0].radio.receiver.Thermal_noise_density=-174 #in dBm/Hz
75 *.host[*].wlan[0].radio.receiver.frequency=960 #in MHz
76 *.host[*].wlan[0].radio.receiver.SNR_margin=10 #in dB
77
```

Contents of Receiver Model (1)

wireless.ned | TraceBasedErrorModel.cc | omnetpp.ini | UnitDiskReceiverCustomized.cc

```
79
80 bool UnitDiskReceiverCustomized::computeIsReceptionSuccessful(const IListening *listening, const IReception *reception, IRadioSignal::SignalPart part, const
81 {
82     double Receiver_bandwidth_in_Hz = Receiver_bandwidth_in_KHz*1000;//in Hz
83
84     //getting the position of the Rx
85     Coord Rx_position = check_and_cast<IMobility *>(getContainingNode(this)->getSubmodule("mobility"))->getCurrentPosition();
86
87     //getting the position of the Tx
88     Coord Tx_position = check_and_cast<const IReception *>(reception)->getTransmission()->getStartPosition();
89
90     //applying the formula sqrt((x_2-x_1)^2+(y_2-y_1)^2) to get the distance between Tx and Rx
91     double distance_in_meter = sqrt(pow((Rx_position.x-Tx_position.x),2)+ pow((Rx_position.y-Tx_position.y),2)+ pow((Rx_position.z-Tx_position.z),2));
92     EV << " distance between Tx and Rx is in meter: " << distance_in_meter << " \n";// printing to check if the results are correct
93
94     double Received_Power;
95     //comparing distance with Radio Horizon rh(h1, h2)|km = 922.06 to decide for path loss
96     if(distance_in_meter < Radio_horizon){
97
98         //calculating path loss using -> 20 log10(d|km · f |MHz) + 32.4478|dB, for d < rh(h1, h2)
99         // as the above distance is in meter we are converting it to kilometer
100        double path_loss =(20*log10((distance_in_meter/1000)*frequency))+32.4478;
101
102        //calculating Received_Power
103        Received_Power = Tx_power + Tx_antenna_gain - Tx_loss + Rx_antenna_gain - Rx_loss - path_loss;
104
105        //calculating Signal-to-Noise Ratio SNR(SNRmargin|dB)
106        double SNR = Received_Power -(Noise_figure + Thermal_noise_density + 10*log10(Receiver_bandwidth_in_Hz))-SNR_margin;
107        EV << "Signal-to-Noise Ratio, SNR= " << SNR << " \n";
108
109        //calling the error Model to get packet error rate for the calculated SNR
110         double PER = ((TraceBasedErrorModel*) errorModel)->snrToPacketErrorRateMapping(SNR);
111         EV << "packetErrorRate= " << PER << " \n";
112
113        //Deciding a packet is correctly received or not based on PER
```

Problems | Module Hierarchy | NED Parameters | NED Inheritance | Console | Event Log | Search | Call Hierarchy | Dataset | Output Vector

Contents of Error Model (1)

```
100 double TraceBasedErrorModel::snrToPacketErrorRateMapping(double SNR)
101 {
102     Enter_Method_Silent();
103
104     //Rounding to nearest neighbor
105     SNR = round(SNR);
106
107     //recording each SNR for analysis
108     recordScalar("#Signal-to-Noise_Ratio",SNR);
109
110     //Taking care of the edges(SNR)
111     if(SNR > 10)
112         SNR = 10;      //since 10 is the largest SNR value in .csv
113     else if(SNR <-4)
114         SNR = -4;     //since -4 is the smallest possible SNR value within distance < Radio horizon
115
116     //finding the corresponding PER of the current SNR
117     double PER;
118     for(int k=0; k<line_number;k++){
119         if(SNR_to_PER_array[k][0]== SNR){
120             PER = SNR_to_PER_array[k][1];
121             break;
122         }
123     }
124
125     return PER;
126 }
127
```

Contents of Trace File

-5,1,0.493378795
-4,1,0.486143304
-3,1,0.472159226
-2,0.9992,0.439466071
-1,0.988666667,0.371906548
0,0.922266667,0.270322024
1,0.7276,0.157394643
2,0.428,0.069978869
3,0.178866667,0.022660417
4,0.053972414,0.005765333
5,0.012457143,0.001158769
6,0.002214815,0.000181713
7,0.000309735,2.62E-05
8,3.28E-05,2.42E-06
9,6.37E-06,4.12E-07
10,7.80E-07,5.22E-08

Contents of Receiver Model (2)

wireless.ned | TraceBasedErrorModel.cc | omnetpp.ini | UnitDiskReceiverCustomized.cc

```
99          // as the above distance is in meter we are converting it to kilometer
100         double path_loss =(20*log10((distance_in_meter/1000)*frequency))+32.4478;
101
102         //calculating Received_Power
103         Received_Power = Tx_power + Tx_antenna_gain - Tx_loss + Rx_antenna_gain - Rx_loss - path_loss;
104
105         //calculating Signal-to-Noise Ratio SNR(SNRmargin|dB)
106         double SNR = Received_Power -(Noise_figure + Thermal_noise_density + 10*log10(Receiver_bandwidth_in_Hz))-SNR_margin;
107         EV << "Signal-to-Noise Ratio, SNR= " << SNR << " \n";
108
109         //calling the error Model to get packet error rate for the calculated SNR
110          double PER = ((TraceBasedErrorModel*) errorModel)->snrToPacketErrorRateMapping(SNR);
111          EV << "packetErrorRate= " << PER << " \n";
112
113         //Deciding a packet is correctly received or not based on PER
114         if (PER == 0.0)
115             return true;
116         else if (PER == 1.0)
117             return false;
118         else{
119             // if the PER is in between 0 and 1 we compare the PER with a random number to decide if the packet is correctly received or not
120             double random_value= uniform(0,1);
121
122             if (random_value < PER)
123                 return false;  //Packet is not successfully received
124             else
125                 return true;  //Packet is successfully received
126         }
127
128     }
129
130     //When the distance between the Tx and Rx is greater than the Radio Horizon, packet will be dropped
131     else
132         return false;
133 }
```

Thank You