Validation of ns-3 802.11b PHY model

May 2009

Guangyu Pei and Tom Henderson

Boeing Research and Technology
The Boeing Company
P.O.Box 3707, MC 7L-49
Seattle, WA 98124-7707

Executive Summary

This document describes the modelling of 802.11b physical layer in ns-3. The model is validated against published measurement results using CMU wireless channel emulator. Under clear channel condition, The ns-3 802.11b simulation results reassemble the measurement data in terms of relative order and spacing among four 802.11b data modes. It also matches with minimum received signal strength required for each data mode. These key characteristics are important since they are the basis for more complicated validation such as reception under interference with competing transmissions and under multi-path channel conditions.

Nevertheless, the results showed that simulation model can not exactly reproduce the measurement data from real hardware. In particular, the transition slope of packet error rate as function of received signal strength. Since ns-3 is a packet level simulator, it does not model bit-level details and some discrepancies are expected. Since the discrepancies are small, further studies are need to investigate whether it can cause noticeable impact for upper layer protocols.

This report is a working-in-progress document. We plan to update it as we make progress. The next steps include validation of 802.11 capture models, off-channel behaviour and 802.11g modes using CMU wireless test bed.

Contents

Chapter 1

Validation of Clear Channel

The chapter describes the validation of 802.11b Physical layer under the clear channel condition. Clear channel validation is important as it serves as the basis for the other validation results such as reception under interference with competing transmissions.

1.1 Test bed measurements from wireless emulator

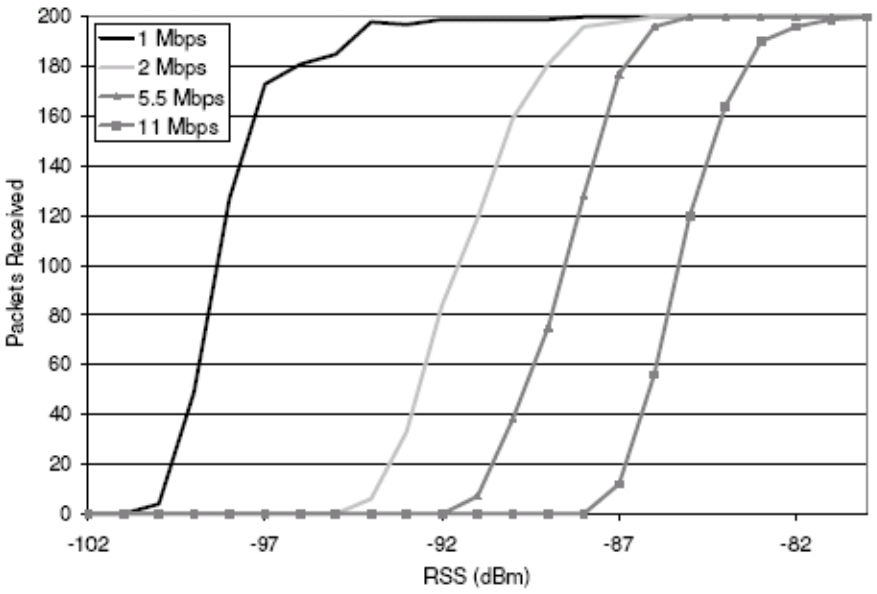

Figure 1.1: Figure 1 in [JS08]

Figure 1.1 is reported in [JS08]. The following is the list of parameter settings for the experiments:

- **Number of transceiver pairs:** 1
- **Total number of packets transmitted:** 200
- **Link level retries:** None (broadcast mode was used)
- **Noise floor:** approximately -99 dBm
- **Receiver Signal Strength:** Varies from -102 dBm to -80 dBm with 1 dB increments

- **Chipset:** Prism 2.5
- **Wireless Card:** Senao 2511CD
- **Packet size:** Not reported

1.2 Bit error rate (BER) model for 802.11b

In order to match the measurements in Figure 1.1, one has to model BER curves correctly in simulator. This is due to the following equation:

$$P_f = 1 - (1 - BER)^n \tag{1.1}$$

where P_f is the frame error rate and n is the number of bits in the frame.

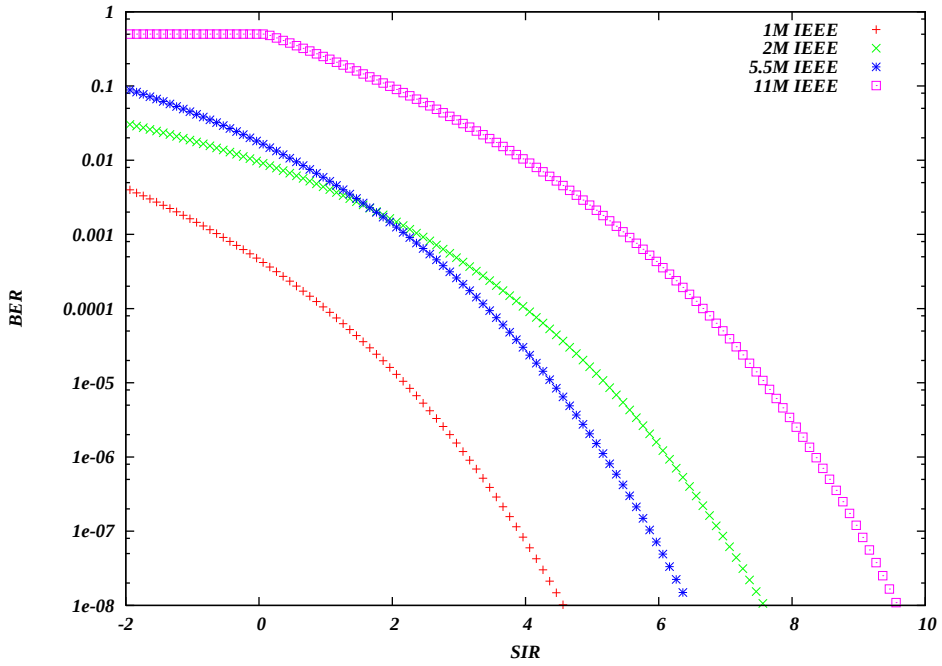

Figure 1.2: Reproduce Figure C.4 in [Com03]

One of the simple and popular model is described in [Com03]. This model was used in IEEE 802.15 working group as the simulation model to study the coexistence of 802.11b radios and 802.15 wireless personal area networks. The same model was also presented in book [Mor04].

Following is the BER calculation from [Com03]:

Let d be the minimum distance between any two points in the signal constellation and N_0 is the in-band noise power density at the receiver. The probability of BPSK is $P = Q(\sqrt{d^2/(2N_0)})$, where $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-(\frac{t^2}{2})} dt$. Since 802.11b 1 MBps uses differential BPSK (i.e., DBPSK), it approximate it with doubling effective noise power. So, $P_{DBPSK-Chip} = Q(\sqrt{d^2/(4N_c)})$, where N_c is the noise per chip. Since $d = 2\sqrt{E_c}$ for BPSK with E_c being the energy received per chip. Thus, $P_{DBPSK-Chip} = Q(\sqrt{E_c/N_c})$. Since each symbol is 11 chips for 1 Mbps mode, the symbol error rate (SER) is given by $P_{1MBPS-Symbol} = Q(\sqrt{11 \times SIR})$ where $SIR = E_c/N_c$ is the signal to noise ratio per chip. For 1 Mbps mode, because each symbol encodes a single bit, the BER is the same as SER. In case of 2 Mbps, the minimum distance between points in QPSK is reduced by factor of $\sqrt{2}$. The BER for 2 Mbps is $Q(\sqrt{5.5 \times SIR})$. For 5.5 Mbps and 11 Mbps, the model uses block coding to model BER. The BER for 5.5 Mbps is $BER_{5.5} \leqslant (\frac{2^{4-1}}{2^4-1})SER_{5.5} = \frac{8}{15}SER_{5.5}$, where $SER_{5.5} = \sum Q(\sqrt{2 \times SIR \times R_c \times W_m}) = 14 \times Q(\sqrt{8 \times SIR}) +$

$Q(\sqrt{16 \times SIR})$. Similarly, the BER for 11 Mbps is given by $BER_{11} \leqslant (\frac{2^{8-1}}{2^8-1})SER_{11} = \frac{128}{255}SER_{11}$ and SER_{11} is given by the following equation:

$$SER_{11} \leqslant 24 \times Q(\sqrt{4 \times SIR}) + 16 \times Q(\sqrt{6 \times SIR}) + 174 \times Q(\sqrt{8 \times SIR}) + 16 \times Q(\sqrt{10 \times SIR}) + 24 \times Q(\sqrt{12 \times SIR}) + Q(\sqrt{16 \times SIR})$$

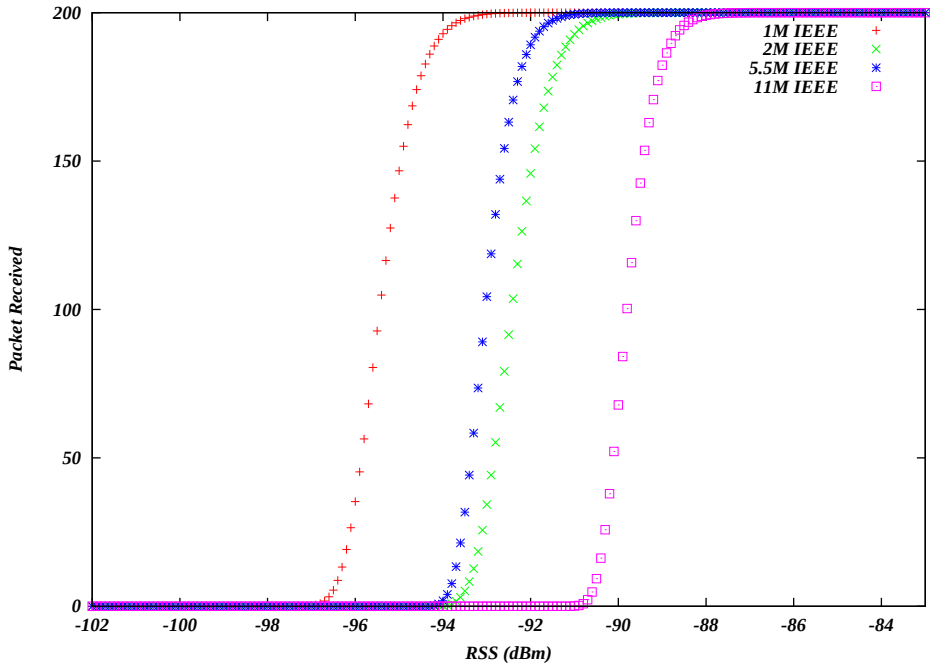

Figure 1.3: CMU clear channel experiment matching with IEEE BER model

To ensure the correctness of BER curve implementation, we first plotted the BER curves in Figure 1.2 based on aforementioned model. These curves matches exactly with Figure C.4 in [Com03] except for 11 Mbps when SIR is lower that 0 dB. Since the BER should be less or equal to 0.5, the authors believe there is an error in Figure C.4 in [Com03] for 11 Mbps as BER is more than 1 for SIR at - 2 dB. Thus, Figure 1.2 shows that the IEEE BER model for 802.11b was correctly implemented.

Now, we turn our attention to match the measurements shown in Figure 1.1 using equation 1.1. Figure 1.3 shows the results with the same parameters listed in section 1.1. As illustrated in this figure, the relative position of packet delivered curve for 2 Mbps and 5.5 Mbps is reversed. The relative spacings between different curves also do not match with those in Figure 1.1. Although IEEE model looks very reasonable and was used in IEEE 802.15 wireless coexistence simulation studies, it can not reproduce performance results that match with Figure 1.1. This leads us to look for more accurate models for 802.11b. The IEEE model showed the necessity to valid simulation model with real test bed results.

In the following sections, we will present BER models that are based on derivations from [Pro01] and [PR09].

1.2.1 1 Mbps BER

The modulation for each chip is DBPSK. From equation (5.2-69) in [Pro01], the chip error rate is:

$$P_{DBPSK-Chip} = \frac{1}{2}e^{-\frac{E_c}{N_0}}, \text{ where } E_c \text{ is the energy per chip} \tag{1.2}$$

Since each symbol has 11 chips and 1 bit per symbol, the BER is the same as SER and can be expressed by the following equation:

$$\begin{aligned} BER_{1MBPS} &= \frac{1}{2}e^{-\frac{11\times E_c}{N_0}} \\ &= \frac{1}{2}e^{-\frac{E_b}{N_0}}, \text{ where } E_b \text{ is the energy per bit} \end{aligned} \tag{1.3}$$

1.2.2 2 Mbps BER

2 Mbps mode uses DQPSK modulation. From equation (5.2-70) in [Pro01], the chip error rate is:

$$P_b = Q_1(a,b) - \frac{1}{2}I_0(a,b)e^{-\frac{1}{2}(a^2+b^2)} \tag{1.4}$$

where $Q_1(a,b) = e^{-(a^2+b^2)/2}\sum_{k=0}^{\infty}(\frac{a}{b})^k I_k(ab)$ is the Marcum Q-function. $I_k(a,b)$ is the k-th order modified Bessel function of the first kind. Since Marcum Q-Function has the semi-infinite integration, the numerical evaluation problem can be encountered. Thus, we adopted the following approximation (equation (8) from [FC04])

$$P_{DQPSK-chip} = \frac{\sqrt{2}+1}{\sqrt{8\pi\sqrt{2}}}\frac{1}{\sqrt{\frac{E_c}{N_0}}}e^{-(2-\sqrt{2})\frac{E_c}{N_0}} \tag{1.5}$$

For 2 Mbps mode, each symbol encodes two bits. Since the symbols are Gray coded, a decoding error between adjacent DQPSK constellation points yields only a single bit error. Thus, the SER is also the BER. The following expression provides the BER for 2 Mbps mode:

$$\begin{aligned} BER_{2MBPS} &= \frac{\sqrt{2}+1}{\sqrt{8\pi\sqrt{2}}}\frac{1}{\sqrt{\frac{11\cdot E_c}{2N_0}}}e^{-(2-\sqrt{2})\frac{11\cdot E_c}{2N_0}} \\ &= \frac{\sqrt{2}+1}{\sqrt{8\pi\sqrt{2}}}\frac{1}{\sqrt{\frac{E_b}{N_0}}}e^{-(2-\sqrt{2})\frac{E_b}{N_0}} \end{aligned} \tag{1.6}$$

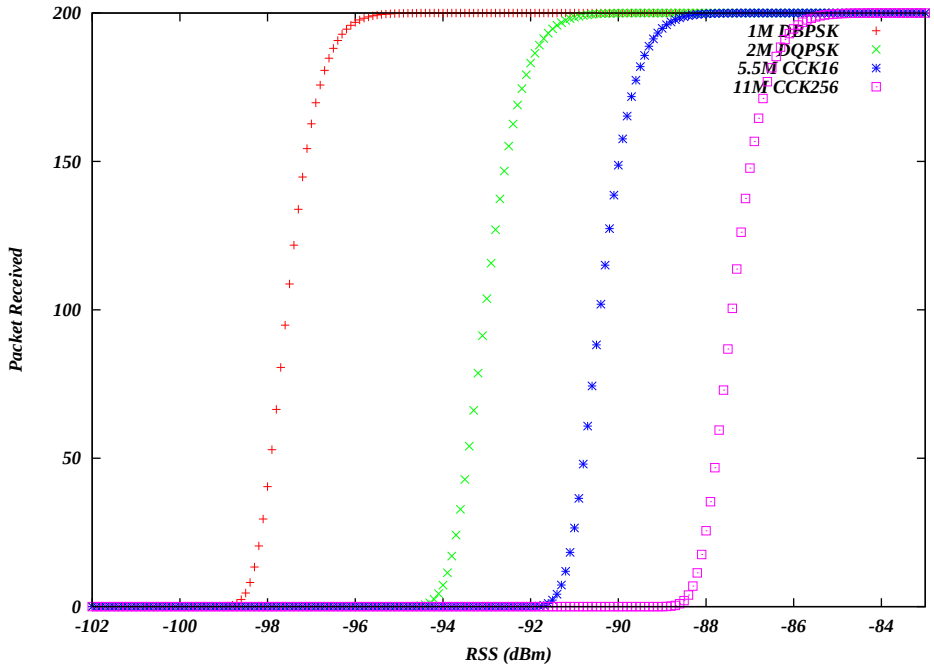

Figure 1.4: CMU clear channel experiment matching with ns-3 BER model

1.2.3 CCK modes: 5.5 Mbps and 11 Mbps

Complementary code keying (CCK) was adopted for in IEEE802.11b standard for 5.5 Mbps and 11 Mbps. Unlike 1 Mbps and 2Mbps, there is no analytical method available for IEEE 802.11b CCK modulation [PR09]. However, [PR09] showed that the analytical results for biorthogonal-key (BOK) modulation is an accurate approximation for the frame error rate for 11 Mbps mode and it is exact for 5.5 Mbps mode under AWGN. In particular, equations (18) and (20) in [PR09] are used for symbol error rate and frame error rate calculation for 5.5 Mbps respectively. Similarly, equations (17) and (21) are used for 11 Mbps. If SNR is the signal to noise ratio at the receiver, the ratio between energy per bit and noise density ϵ_b/N_0 is $4SNR$ for 5.5 Mbps and $2SNR$ for 11 Mbps. This is because the 802.11b bandwidth is 22 MHz and CCK modes use 8 chips per symbol which yields 1.375 MSPS. There are 4 bits per symbol for 5.5 Mbps and 8 bits per symbol for 11 Mbps. Thus, $\epsilon_2/N_0 = 16SNR$ should be used in equation (18) of [PR09] for 5.5 Mbps and $\epsilon_1/N_0 = 16SNR$ should be used in equation (17) of [PR09] for 11 Mbps. However, in order to take into account non-coherent demodulation, we have to the effective noise (i.e., β value is replaced with $\beta/\sqrt{2}$ in equation (18) of [PR09].

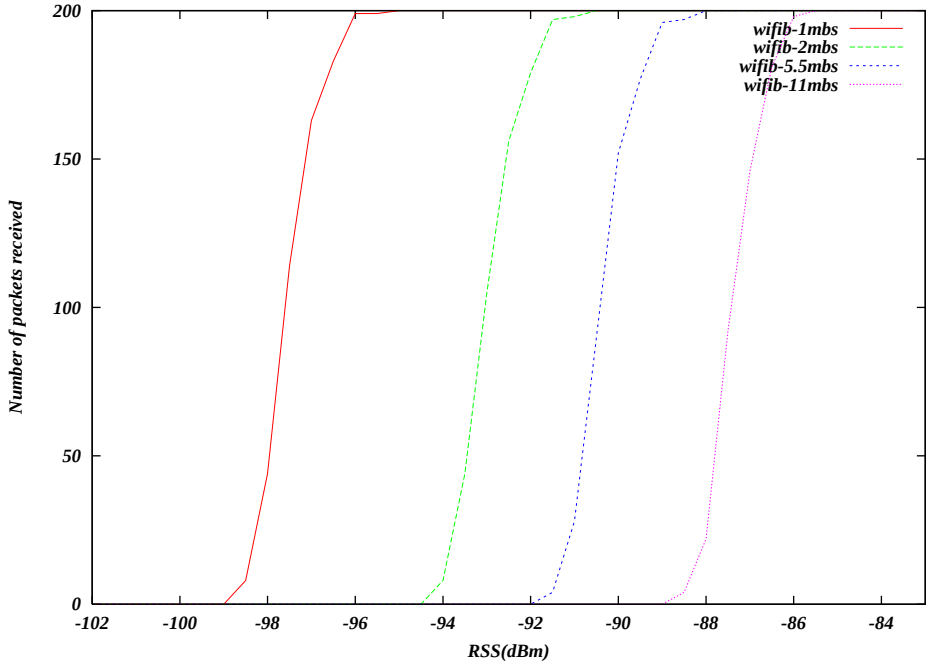

Figure 1.5: ns-3 clear channel results with broadcast

1.2.4 Clear Channel Matching Results

Figure 1.4 shows the clear channel results based on the above BER models. Appendix A provides the source code for reader to verify the results presented in this chapter. First, the order of each mode matches with Figure 1.1. Secondly, the spacing among 2Mpbs, 5.5 Mbps and 11 Mbps also matches with 1.1. Finally, the minimum RSS required for packet reception for 2 Mbps, 5.5 Mbps and 11 Mbps aligns well with Figure 1.1. The only mismatch is the position of 1 Mbps curve. In Figure 1.1, the receiver starts to receive packets around -100 dBm while it is around -98 dBm in Figure 1.4. In fact, the separation between 1 Mbps and 2 Mbps is more than 6 dB in Figure 1.1, which is not the theoretical expected difference between DBPSK and DQPSK. The authors wonder whether the noise floor of 1 Mbps is different than the other data rates.

As shown in Figure 1.4, the aforementioned BER models matched pretty well with the measurement data obtained in CMU test bed. Note that the results in Figure 1.4 are obtained from theoretical calculations. We ported these BER models in ns-3 simulation framework. Figure 1.5 shows the ns-3 simulation results using UDP broadcast traffic and Figure 1.6 shows the results with unicast UDP traffic generators. The broadcast results reassemble the theoretical results very closely. However, the unicast curves are steeper. This is due to IEEE802.11 MAC layer retransmissions.

As RSS increases, more packets can be recovered via retransmissions. Thus, the slope of unicast is steeper than that of broadcast.

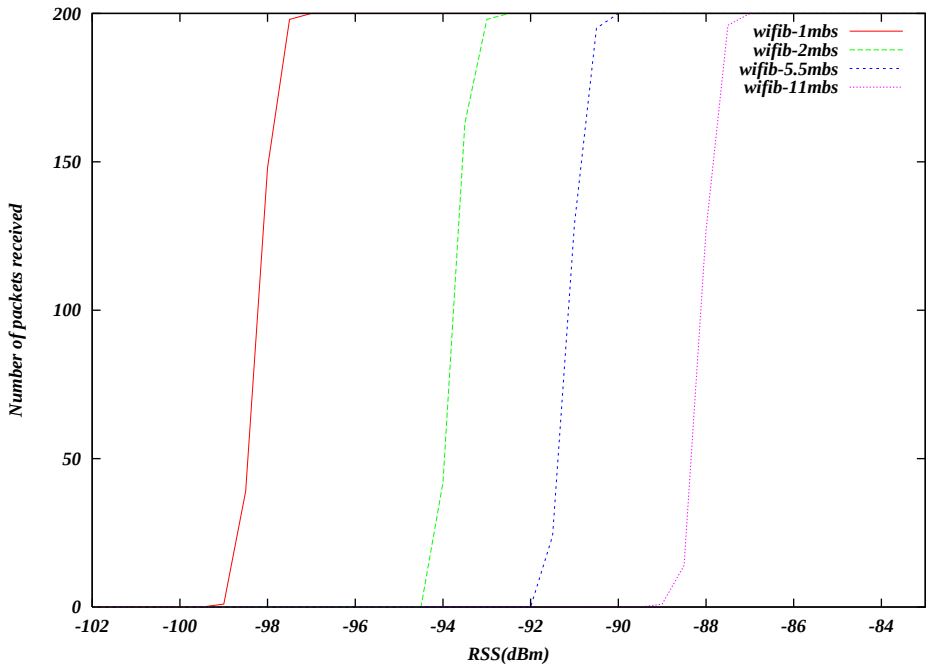

Figure 1.6: ns-3 clear channel results with unicast

Bibliography

[Com03] IEEE LAN/MAN Standards Committee. Part 15.2: Coexistence of wireless personal area networks with other wireless devices operating in unlicensed frequency bands, August 2003.

[FC04] G. Ferrari and G.E. Corazza. Tight bounds and accurate approximations for dqpsk transmission bit error rate. *ELECTRONICS LETTERS*, 40(20):1284–1285, September 2004.

[JS08] Glenn Judd and Peter Steenkiste. Characterizing 802.11 wireless link behavior. *Wireless Networks*, 2008.

[Mor04] Rober Morrow. *Wireless Network Coexistence*. McGraw-Hill, 2004.

[PR09] Michael B. Pursley and Thomas C. Royster. Properties and performance of the ieee 802.11b complementary-code-key signal sets. *IEEE TRANSACTIONS ON COMMUNICATIONS*, 57(2):440–449, February 2009.

[Pro01] John G. Proakis. *Digital Communications*. McGraw-Hill, 2001.

Appendix A

Reproducing Clear Channel

This appendix describes how to reproduce our results described in chapter 1.

A.1 802.11b BER curve models

The following program is used to model both IEEE model and ns-3 model. To calculate BER for 5.5 Mbps and 11 Mbps, the GNU Scientific Library (GSL) is used. GSL is available from web site: http://www.gnu.org/software/gsl/.

```
// Copyright 2009, The Boeing Company

#include "math.h"
#include "stdlib.h"
#include "stdio.h"

#include <gsl/gsl_math.h>
#include <gsl/gsl_integration.h>
#include <gsl/gsl_cdf.h>
#include <gsl/gsl_sf_bessel.h>

#define min(a,b) ((a)<(b) ? (a) : (b))
#define max(a,b) ((a)>(b) ? (a) : (b))
#define WLAN_SIR_perfect 10.0 // if SIR > 10dB, perfect reception
#define WLAN_SIR_impossible 0.1 // if SIR < -10dB, impossible to receive

typedef struct fn_parameter_t
{
 double beta;
 double n;
} fn_parameters;

double QFunction (double x)
{
 return 0.5 * erfc(x/sqrt(2.0));
}

double f(double x, void *params)
{
 double beta = ((fn_parameters *) params)->beta;
 double n = ((fn_parameters *) params)->n;
 double f = pow( 2*gsl_cdf_ugaussian_P (x+ beta) - 1, n-1)
```

```
            * exp (-x*x/2.0) / sqrt (2.0 * M_PI);
 return f;
}

double p_e2(double e2)
{
 double sep;
 double error;

 fn_parameters params;
 params.beta = sqrt (2.0*e2);
 params.n = 8.0;

 gsl_integration_workspace *
          w = gsl_integration_workspace_alloc (1000);

 gsl_function F;
 F.function = &f;
 F.params = &params;

 gsl_integration_qagiu(&F,
                       -params.beta,
                       0, 1e-7, 1000, w, &sep, &error);
 gsl_integration_workspace_free (w);
 if (error == 0.0) sep = 1.0;

 return 1.0 - sep;
}

double p_e1(double e1)
{
 return 1.0 - pow( 1.0 - p_e2 (e1/2.0), 2.0);
}


double DbToNoneDb (double x)
{
  return pow(10.0, x/10.0);
}

double NoneDbToDb (double x)
{
  return 10.0 * log10 (x) ;
}

double DQPSKFunction (double x)
{
 double pi = acos (-1.0);
 return ( (sqrt(2.0) + 1.0) / sqrt(8.0*pi*sqrt(2.0)))
        *(1.0/sqrt(x))
        *exp( - (2.0 - sqrt(2.0)) * x) ;
}

double Get80211bDsssDbpskBerIeee(double EcNc)
```

```
{
 double ber;
 if(EcNc > WLAN_SIR_perfect)
  ber = 0;
 else if(EcNc < WLAN_SIR_impossible)
  ber =  0.5;
 else
  ber = min(QFunction(sqrt(11.0*EcNc)),0.5);
 return ber;
}

double Get80211bDsssDbpskBer(double sinr)
{
 double EbN0 = sinr * 22000000.0 / 1000000.0;
 double ber = 0.5 * exp(-EbN0);
 return ber;
}

double Get80211bDsssDqpskBerIeee(double EcNc)
{
 double ber;
 if (EcNc > WLAN_SIR_perfect)
  ber = 0;
 else if(EcNc < WLAN_SIR_impossible)
  ber = 0.5;
 else
  ber = min(QFunction(sqrt(5.5*EcNc)),0.5);
 return ber;
}

double Get80211bDsssDqpskBer(double sinr)
{
 // 2 bits per symbol, 1 MSPS
 double EbN0 = sinr * 22000000.0 / 1000000.0 / 2.0;
 double ber = DQPSKFunction(EbN0);
 return ber;
}

double Get80211bDsssDqpskCCK5_5BerIeee(double EcNc)
{
 double ber;
 if(EcNc > WLAN_SIR_perfect)
  ber = 0.0 ;
 else if(EcNc < WLAN_SIR_impossible)
  ber = 0.5;
 else
 {
  double pew = 14.0*QFunction(sqrt(EcNc*8.0))
               + QFunction(sqrt(EcNc*16.0));
  pew = min(pew, 0.99999);
  ber = 8.0/15.0 * pew;
 }
 return ber;
}
```

```
double Get80211bDsssDqpskCCK11BerIeee(double EcNc)
{
 double ber;
 if(EcNc > WLAN_SIR_perfect)
  ber = 0.0 ;
 else if(EcNc < WLAN_SIR_impossible)
  ber = 0.5;
 else
 {
 double pew = 24.0*QFunction(sqrt(EcNc*4.0))  +
              16.0*QFunction(sqrt(EcNc*6.0))  +
              174.0*QFunction(sqrt(EcNc*8.0)) +
              16.0*QFunction(sqrt(EcNc*10.0)) +
              24.0*QFunction(sqrt(EcNc*12.0)) +
              QFunction(sqrt(EcNc*16.0));
  pew = min(pew, 0.99999);
  ber = 128.0/255.0 * pew;
 }
 return ber;
}

int main (int argc, char *argv[])
{
  double rss, sinr;
  double totalPkt = 200.0;
  //double noise = 1.552058;
  double noise = 7;
  double EcNc, EbN01, EbN02, EbN05, EbN011;
  double ieee1,ieee2,ieee5,ieee11;
  double numBits = (1024. + 40. + 14.) * 8.;
  double dbpsk,dqpsk,cck16,cck256,sepcck16,sepcck256;

  noise = DbToNoneDb(noise) * 1.3803e-23 * 290.0 * 22000000;
  for (rss=-102.0; rss <= -80.0; rss += 0.1)
  {
    sinr   = DbToNoneDb(rss)/1000.0/noise;
    EcNc   = sinr * 22000000.0 / 11000000.0; // IEEE sir
    EbN01  = sinr * 22000000.0 / 1000000.0;
    // 2 bits per symbol, 1 MSPS
    EbN02  = sinr * 22000000.0 / 1000000.0 / 2.0;
    EbN05  = sinr * 22000000.0 / 1375000.0 / 4.0;
    EbN011 = sinr * 22000000.0 / 1375000.0 / 8.0;
    // 1=rss, 2=EcNc, 3=EbN01, 4=EbN02, 5=EBN05, 6=EbN011
    printf("%g %g %g %g %g %g ", rss, NoneDbToDb(EcNc),
           NoneDbToDb(EbN01),NoneDbToDb(EbN02),
           NoneDbToDb(EbN05),NoneDbToDb(EbN011));

    ieee1  = Get80211bDsssDbpskBerIeee (EcNc);
    ieee2  = Get80211bDsssDqpskBerIeee (EcNc);
    ieee5  = Get80211bDsssDqpskCCK5_5BerIeee (EcNc);
    ieee11 = Get80211bDsssDqpskCCK11BerIeee (EcNc);
    // 7=ber_ieee1, 8=ber_ieee2, 9=ber_ieee5, 10=ber_ieee11
    printf(" %g %g %g %g ", ieee1, ieee2,ieee5,ieee11);
```

```
    ieee1  = totalPkt*pow(1-ieee1,  numBits);
    ieee2  = totalPkt*pow(1-ieee2,  numBits);
    ieee5  = totalPkt*pow(1-ieee5,  numBits);
    ieee11 = totalPkt*pow(1-ieee11, numBits);
    // 11=pkt_ieee1, 12=pkt_ieee2, 13=pkt_ieee5, 14=pkt_ieee11
    printf(" %g %g %g %g ", ieee1, ieee2,ieee5,ieee11);

    dbpsk  = Get80211bDsssDbpskBer (sinr);
    dqpsk  = Get80211bDsssDqpskBer (sinr);
    cck16  = max(0, 8.0/15.0*p_e2(4.0*EbN05/2.0));
    cck256 = max(0, 128.0/255.0*p_e1(8.0*EbN011/2.0));
    // 15=ber_dbpsk, 16=ber_dqpsk, 17=ber_cck16, 18=ber_cck256
    printf(" %g %g %g %g ", dbpsk, dqpsk,cck16,cck256);

    dbpsk     = totalPkt*pow(1-dbpsk,numBits);
    dqpsk     = totalPkt*pow(1-dqpsk,numBits);
    sepcck16  = p_e2(4.0*EbN05/2.0);
    sepcck256 = p_e1(8.0*EbN011/2.0);
    cck16     = totalPkt*pow(1.0-sepcck16,numBits/4.0);
    cck256    = totalPkt*pow(1.0-sepcck256,numBits/8.0);
    // 19=pkt_dbpsk, 20=pkt_dqpsk, 21=pkt_cck16, 22=pkt_cck256
    printf(" %g %g %g %g ", dbpsk, dqpsk,cck16,cck256);
    // 23=sinr
    printf(" %g \n",NoneDbToDb(sinr));
  }
  return 0;
}
```

To compile the above C program (80211b.c), the follow command is used:

```
gcc -Wall -lgsl -lgslcblas -lm -g 80211b.c -o 80211b
```

The following is the gnuplot script used to generate Figure 1.2 through Figure 1.4 assuming the output of the "80211b" program is redirected to a file "80211b.txt".

```
set term postscript eps color enh "Times-BoldItalic"
set output '80211b.ieee.pkt.eps'
set xlabel "RSS (dBm)"
set ylabel "Packet Received"
set yrange [0:200]
set xrange [-102:-83]
plot "80211b.txt" using 1:11 title '1M IEEE', \
     "80211b.txt" using 1:12 title '2M IEEE', \
     "80211b.txt" using 1:13 title '5.5M IEEE', \
     "80211b.txt" using 1:14 title '11M IEEE'
set term postscript eps color enh "Times-BoldItalic"
set output '80211b.ns3.pkt.eps'
set xlabel "RSS (dBm)"
set ylabel "Packet Received"
set yrange [0:200]
set xrange [-102:-83]
plot "80211b.txt" using 1:19 title '1M DBPSK', \
     "80211b.txt" using 1:20 title '2M DQPSK', \
     "80211b.txt" using 1:21 title '5.5M CCK16', \
```

```
     "80211b.txt" using 1:22 title '11M CCK256'
set term postscript eps color enh "Times-BoldItalic"
set output '80211b.ieee.sir.eps'
set xlabel "SIR"
set ylabel "BER"
set yrange [10e-9:1]
set xrange [-2:10]
set logscale y
plot "80211b.txt" using 2:7 title '1M IEEE', \
     "80211b.txt" using 2:8 title '2M IEEE', \
     "80211b.txt" using 2:9 title '5.5M IEEE', \
     "80211b.txt" using 2:10 title '11M IEEE'
```

A.2 ns-3 802.11b clear channel experiments

The following is the simulation script for ns-3 experiements that generates the Figure 1.5 and Figure 1.6. There are two patches needs to be applied before running the following program. The first patch is 802.11b PHY mode patch and second patch is to enable setting broadcast data mode to user specified rather than the basic data mode. These patches are available at TBD web site.

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 The Boeing Company
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Guangyu Pei <guangyu.pei@boeing.com>
 */

#include "ns3/core-module.h"
#include "ns3/common-module.h"
#include "ns3/node-module.h"
#include "ns3/helper-module.h"
#include "ns3/mobility-module.h"
#include "ns3/contrib-module.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>

NS_LOG_COMPONENT_DEFINE ("Main");
```

```
using namespace ns3;

class Experiment
{
public:
  Experiment ();
  Experiment (std::string name);
  uint32_t Run (const WifiHelper &wifi, const YansWifiPhyHelper &wifiPhy, const YansWifiCha
private:
  void ReceivePacket (Ptr<Socket> socket);
  void SetPosition (Ptr<Node> node, Vector position);
  Vector GetPosition (Ptr<Node> node);
  Ptr<Socket> SetupPacketReceive (Ptr<Node> node);
  void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
                        uint32_t pktCount, Time pktInterval );

  uint32_t m_pktsTotal;
  Gnuplot2dDataset m_output;
};

Experiment::Experiment ()
{}

Experiment::Experiment (std::string name)
  : m_output (name)
{
  m_output.SetStyle (Gnuplot2dDataset::LINES);
}

void
Experiment::SetPosition (Ptr<Node> node, Vector position)
{
  Ptr<MobilityModel> mobility = node->GetObject<MobilityModel> ();
  mobility->SetPosition (position);
}

Vector
Experiment::GetPosition (Ptr<Node> node)
{
  Ptr<MobilityModel> mobility = node->GetObject<MobilityModel> ();
  return mobility->GetPosition ();
}

void
Experiment::ReceivePacket (Ptr<Socket> socket)
{
  Ptr<Packet> packet;
  while (packet = socket->Recv ())
    {
      m_pktsTotal ++;
    }
}
```

```
Ptr<Socket>
Experiment::SetupPacketReceive (Ptr<Node> node)
{
  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> sink = Socket::CreateSocket (node, tid);
  InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);
  sink->Bind (local);
  sink->SetRecvCallback (MakeCallback (&Experiment::ReceivePacket, this));
  return sink;
}

void
Experiment::GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
                             uint32_t pktCount, Time pktInterval )
{
  if (pktCount > 0)
    {
      socket->Send (Create<Packet> (pktSize));
      Simulator::Schedule (pktInterval, &Experiment::GenerateTraffic, this,
                           socket, pktSize,pktCount-1, pktInterval);
    }
  else
    {
      socket->Close ();
    }
}

uint32_t
Experiment::Run (const WifiHelper &wifi, const YansWifiPhyHelper &wifiPhy, const YansWifiCl
{
  m_pktsTotal = 0;

  NodeContainer c;
  c.Create (2);

  InternetStackHelper internet;
  internet.Install (c);

  YansWifiPhyHelper phy = wifiPhy;
  phy.SetChannel (wifiChannel.Create ());
  NetDeviceContainer devices = wifi.Install (phy, c);

  MobilityHelper mobility;
  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
  positionAlloc->Add (Vector (0.0, 0.0, 0.0));
  positionAlloc->Add (Vector (5.0, 0.0, 0.0));
  mobility.SetPositionAllocator (positionAlloc);
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (c);

  Ipv4AddressHelper ipv4;
  NS_LOG_INFO ("Assign IP Addresses.");
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i = ipv4.Assign (devices);
```

```
  Ptr<Socket> recvSink = SetupPacketReceive (c.Get (0));

  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> source = Socket::CreateSocket (c.Get (1), tid);

  // Unicast experiment
  InetSocketAddress remote = InetSocketAddress (i.GetAddress (0), 80);

  // Broadcast experiment: comment the above line and uncomment the following line
  //InetSocketAddress remote = InetSocketAddress (Ipv4Address ("255.255.255.255"), 80);
  source->Connect (remote);
  uint32_t packetSize = 1014;
  //uint32_t maxPacketCount = 1;
  uint32_t maxPacketCount = 200;
  Time interPacketInterval = Seconds (1.);
  Simulator::Schedule (Seconds (1.0), &Experiment::GenerateTraffic,
                       this, source, packetSize, maxPacketCount,interPacketInterval);
  Simulator::Run ();

  Simulator::Destroy ();

  return m_pktsTotal;
}

int main (int argc, char *argv[])
{
  std::ofstream outfile("clear-channel.plt");
  std::vector <std::string> modes;

  modes.push_back("wifib-1mbs");
  modes.push_back("wifib-2mbs");
  modes.push_back("wifib-5.5mbs");
  modes.push_back("wifib-11mbs");
  // disable fragmentation
  Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold", StringValue
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue ("2200"

  CommandLine cmd;
  cmd.Parse (argc, argv);

  Gnuplot gnuplot = Gnuplot ("clear-channel.png");

  for(uint32_t i = 0; i < modes.size(); i++)
  {
   std::cout << modes[i] << std::endl;
   Gnuplot2dDataset dataset(modes[i]);

   for (double rss = -102.0; rss <= -80.0; rss += 0.5)
   {
     Experiment experiment;
     dataset.SetStyle (Gnuplot2dDataset::LINES);

     WifiHelper wifi;
```

```
    Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                        StringValue (modes[i]));
    wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                  "DataMode",StringValue(modes[i]),
                                  "ControlMode",StringValue(modes[i]));
    wifi.SetMac ("ns3::AdhocWifiMac");

    YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
    YansWifiChannelHelper wifiChannel ;
    wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
    wifiChannel.AddPropagationLoss ("ns3::FixedRSSLossModel","RSS",DoubleValue(rss));


    NS_LOG_DEBUG (modes[i]);
    experiment = Experiment (modes[i]);
    wifiPhy.Set ("Standard", StringValue ("802.11b") );
    wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue (-110.0) );
    wifiPhy.Set ("CcaMode1Threshold", DoubleValue (-110.0) );
    wifiPhy.Set ("TxPowerStart", DoubleValue (15.0) );
    wifiPhy.Set ("RxGain", DoubleValue (0) );
    wifiPhy.Set ("RxNoise", DoubleValue (7) );
    uint32_t pktsRecvd = experiment.Run (wifi, wifiPhy, wifiChannel);
    dataset.Add (rss, pktsRecvd);
   }

   gnuplot.AddDataset (dataset);
  }
  gnuplot.SetLegend ("RSS(dBm)", "Number of packets received");
  gnuplot.SetExtra  ("set xrange [-102:-83]");
  gnuplot.GenerateOutput (outfile);
  outfile.close();

  return 0;
}
```