ENSC 427: COMMUNICATION NETWORKS

SPRING 2019

FINAL REPORT

ANALYSIS OF PACKET LOSS RELATED TO TRANSMISSION DISTANCE

HTTPS://SCA185.WIXSITE.COM/ENSC427-TEAM08

SHUO CHEN (301242896, SCA185@SFU.CA)
HONGBIN LIN (301269972, HONGBINL@SFU.CA)
DANFENG SHENG (301251467, DANFENGS@SFU.CA)
TEAM 08

Abstract

In this report, we will demonstrate various models of propagation loss concerning the transmission distance between the antennas. We will use Ns-3 network simulator to model the transmission of the WIFI signals. The common models include FriisPropagationLossModel, RandomPropagationLossModel, JakesPropagationLossModel, LogDistancePropagationLossModel, NakagamiPropagationLossModel, and ItuR1411LosPropagationLossModelModel [1]. These models simulate the Rx signal power considering the Tx signal power and transmission distance.

Furthermore, we will construct a simulation scenario using Riverbed Modeler. We will utilize some of the conclusions from the analyzed propagation models to testify the data dropped along with the throughput. Through that analysis, we anticipate a consistency between theoretical models and actual simulation, and we also expect to possess a deeper understanding of the packet loss in the wireless network.

Table of Contents

Introduction

As the technologies of the Internet era permeate daily life, the expectations and needs of Internet applications explicitly increase. Instead of communicating using letter mails and phone calls, people now prefer video streaming. And the young generation most likely plays computer games. However, they sometimes complain about the Internet latencies [2] and slow network [3], which is mainly caused by packet loss. Packet loss is the loss of data during the process of digital communication transmission [4]. For a wireless network like WIFI, the packet loss is expected to increase as the increment of the distance [5]. High qualities of Internet communication, or in other words, stable and fast signal transmission, is significantly commanded to ensure a lower rate of packet loss. In this report, we will use the ns-3 network simulator to establish propagation models of signal transmissions. The propagation loss models will simulate the received power concerning the input power and transmission distance. Various models with different algorithms will be tested and compared. We expect to possess a deeper understanding of the reasons why packet loss occurs more frequently as the transmission distance generally increases.

Propagation Loss Models

In this section, we will introduce the common propagation loss models simulating the signal transmission in real life. The mechanisms, formulas and application areas will be demonstrated individually.

FriisPropagationLossModel

The Friis transmission formula was firstly introduced by Danish-American radio engineer Harald T. Friis in 1946 [6]. The formula associates the power at the receiving terminal with the power density of the wave and effective aperture [7]. The formula is illustrated as follows:

$$\frac{P_r}{P_t} = \frac{A_r A_t}{d^2 \lambda^2}$$

Equation 1. Friis Model

Considering the case of an isotropic antenna with no heat loss:

$$A_{iso} = \frac{\lambda^2}{4\pi}$$

Equation 2. Isotropic Antenna

We obtain the final formula utilized in the code:

$$P_r = \frac{P_t G_t G_r \lambda^2}{4\pi d^2 L}$$

Equation 3. Friis Final Equation

Where P_t stands for transmission power (W), P_r is reception power (W), G_t is transmission gain (unit-less), G_r is reception gain (unit-less), λ is wavelength (m), d is dthe istance (m), and L is system loss (unit-less) [1]. This model is considered valid only within a vacuum and a far-field region. For convenience, the model assumes $d > 3\lambda$ as the valid region. Also, the formula should be avoided when $d = 0$. To maintain the continuity of the formula, the model adopts a minimum threshold value called MinLoss. The returned value of power will keep approaching the threshold value as d

approaches 0 until the target value is reached. Via this convention, the input and output power are equal as d equals 0.

LogDistancePropagationLossModel

The log-distance path loss model is usually applied to the cases of compact space over distance. Considering no fading effect, the general formula is shown below:

$$L = L_0 + 10n \log \frac{d}{d_0}$$

Equation 4. Log Distance Model

Where L is the path loss(dB), L_0 is the path loss reference point(dB), n is the path loss distance exponent, d and d_0 are the actual and reference distances, respectively. In the office, the pass-loss exponent can vary from 2.4 to 3.0 due to different extents of partition. Furthermore, ThreeLogDistancePropagationLossModel is developed based on this concept. The difference is that it picks three reference distances. As long as the target distance is less than the reference distance, the individual path loss is added to the total path loss.

RandomPropagationLossModel

This model is included within the default ns-3 propagation loss model script. It takes a random variable which is used to pick a loss every time the Rx power is calculated. The random loss is then added to the previous value of Rx.

ItuR1411LosPropagationLossModelModel

The ItuR1411LosPropagationLossModel is designed for outdoor signal transmission within the frequency range of 300MHz to 100GHz, which includes the commonly used WIFI signal frequency. This model contains both the upper and lower bounds depending on the distance of the sender and receiver. [1]

$$L = L_{bp} + \begin{cases} 20log\frac{d}{R_{bp}} & for\ d \leq R_{bp} \\ 40log\frac{d}{R_{bp}} & for\ d \geq R_{bp} \end{cases}$$

Equation 5. Itu1411 Propagation Model

where

$$R_{bp} = \frac{4h_b h_m}{\lambda} \quad L_{bp} = \left|20log\frac{\lambda^2}{8\pi h_b h_m}\right|$$

Equation 6. Breakpoint Distance and Loss

Here, λ means wavelength. h_b and h_m mean eNB height and UE height, respectively, which are the z position of the two mobility models within the scripts. And the d represents the distance between the sender and receiver.

NakagamiPropagationLossModel

This model takes extra considerations on the variations of signal strength due to multipath fading [1]. However, this model does not account for the path loss caused by the signal travelling long distance. Thus, this model shall be combined with other models like ThreeLogDistancePropagationLossModel. The Nakagami-m distribution model is applied to the power level, which is formulated as:

$$\mathrm{P}(\mathrm{x},\mathrm{m},\omega) = \frac{2m^m}{\Gamma(\mathrm{m})\omega^m} x^{2m-1} e^{-\frac{m}{\omega}x^2}$$

Equation 7. Nakagami-m Distribution

Here, m is the fading depth parameter, ω is the average received power. The denominator contains Gamma random distribution, which can be switched by Erlang random distribution. But for m bigger than 0, these two random variables are equivalent.

Cost231PropagationLossModel

This model is commonly used for macro cells in urban areas. The COST Hata model is formulated as

$$L = 46.3 + 33.9 \log f - 13.82 \log h_b - a(h_R, f)(44.9 - 6.55 \log h_b) \log d + C$$

Equation 8. Pathloss COST231 OH Formula

$$a(h_R, f) = (1.1 \log f - 0.7)h_R - (1.56 \log f - 0.8)$$

Equation 9. Suburban or Rural Environments

Where f is the frequency, h_b is the base station antenna effective height, C is precisely defined as 0dB for small cities and 3dB for large towns, h_R is identified as mobile station antenna effective height, and d is the link distance between two mobility models.

Simulation Setup

We use Riverbed modeller to simulate a simple WLAN scenario. The setup is shown below:

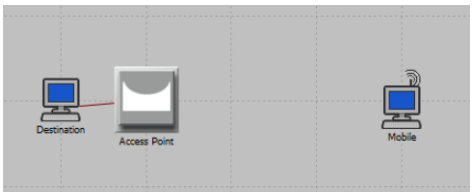

Figure 1. Riverbed Setup

Each of the function blocks is initialized as follows:

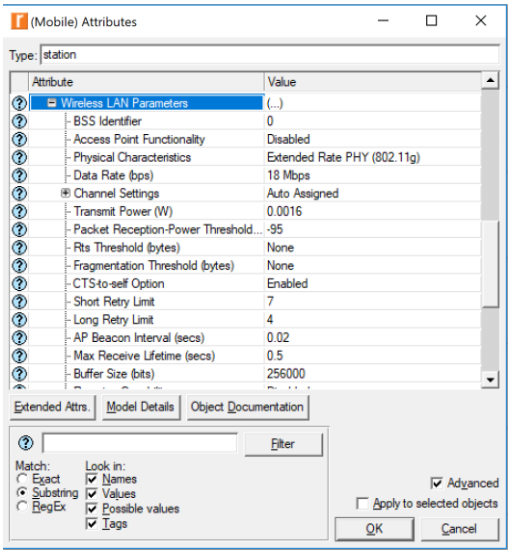

Figure 2. Mobile Setup

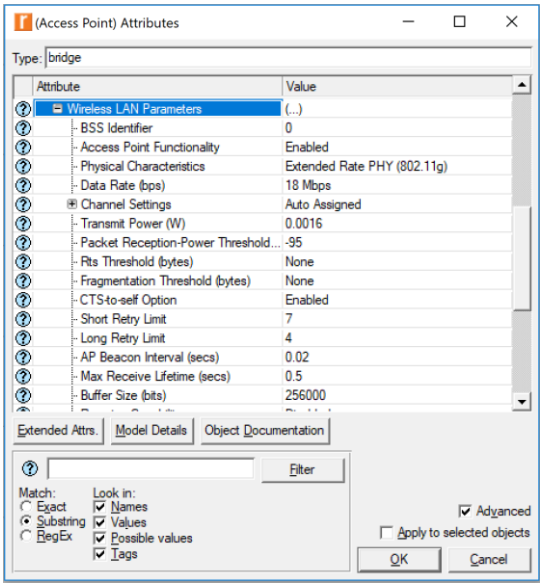

Figure 3. Access Point Setup

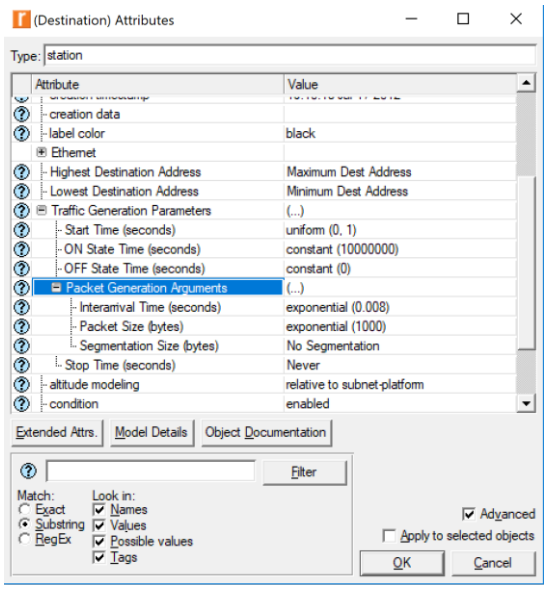

Figure 4. Destination Setup

Both Mobile client and Access point use the same physical characteristic and the data rate of 18 Mbps for 802.11g. Later, the data rate will change according to the variation of the physical characteristics. For the Destination block, we set the start time of the traffic randomly between 0-1 second, and the packets will be kept sending. Each packet will have an average interarrival time of 0.008 second, meaning on average of 125 packets will be generated every second. And the average packet size is 1000 bytes. We will ignore the segmentation option for simplicity.

We will investigate the following parameters:

- Data dropped due to the buffer overflow, meaning the higher layer buffer is full, or the size of higher layer packet is greater than the allowed value of 802.11 standard
- Data dropped due to retransmission, meaning the MAC address cannot receive the Acknowledgements for the retransmission of the packets, and the packets retry count is over the limit

- Throughput, meaning the total number of bits forwarded from WLAN to the client node

We will examine the parameters above by varying the following parameters:

- The distance between the mobile client and the access point
- The distance between the access point and the destination
- Physical characteristics of the mobile client

Simulation Results

In this section, we will first illustrate the simulation results based on the previously introduced propagation models. Then, we will investigate the simulation results in Riverbed found on the conclusions from the propagation models. We expect to observe a negative correlation between propagation distance and rxPower, and a positive relationship between packet loss and propagation distance.

To establish a reference point, we use the preamble CS (carrier-sense) Threshold value, which is sometimes referred to as SD (Signal Detect) Threshold value. [8] To simulate a WIFI channel, the CS Threshold value is set to -94dBm, which means the SD Threshold value is -74dBm. We assume the txPower to be 20dBm for all models, and we expect the rxPower to be less than 0 dBm.

Figure 5 shows the FriisPropagationLossModel. We observe an exponential drop of rxPower within the starting low-distance range. The rxPower keeps decreasing, and slowly approaches the CSThreshold value as the distance increases. When the distance reaches the 2500-meter mark, the rxPower starts falling below the CSThreshold value, which means the preamble signal gets harder to be detected.

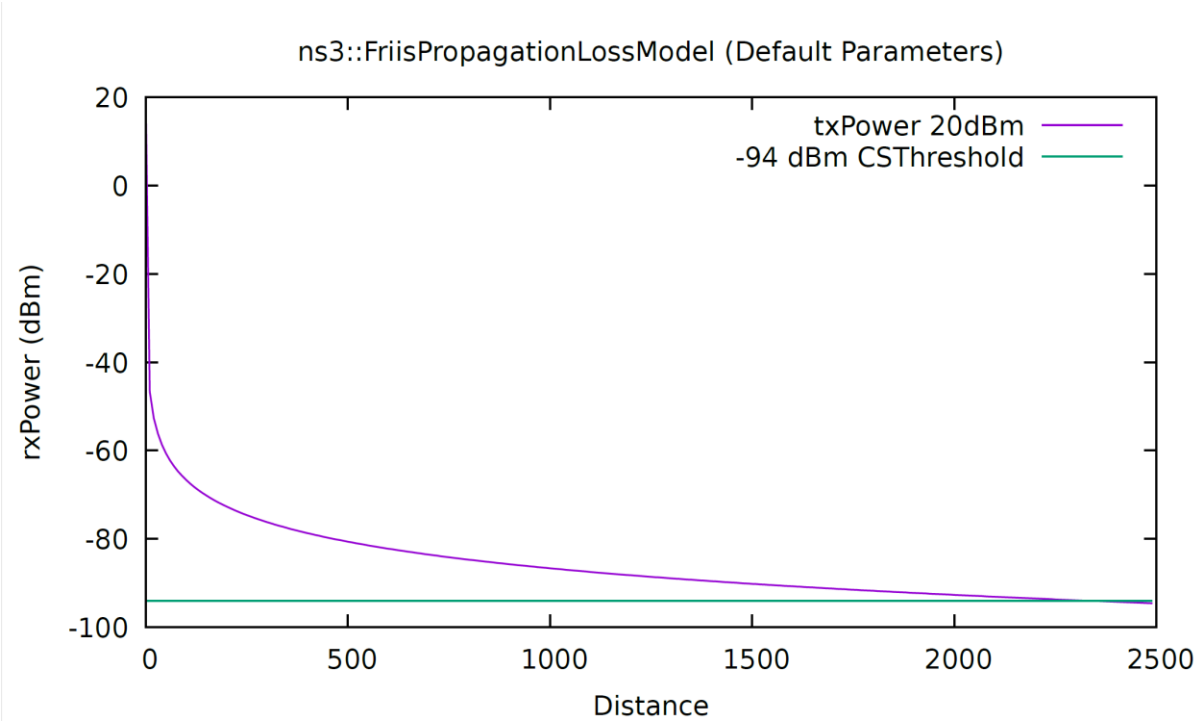

Figure 5.Friis Model

Figure 6 shows the LogDistancePropagationLossModel. The path loss distance exponent is set to 2.5 referred to Equation 4. We observe a faster exponential decrease of rxPower compared to Friis model, where the preamble signal is reached at a distance of 500 meters. In other words, this model inferred that the WIFI signal power is below the detectable level after 500 meters, where the number of packet loss will start to increase as the distance enlarges.

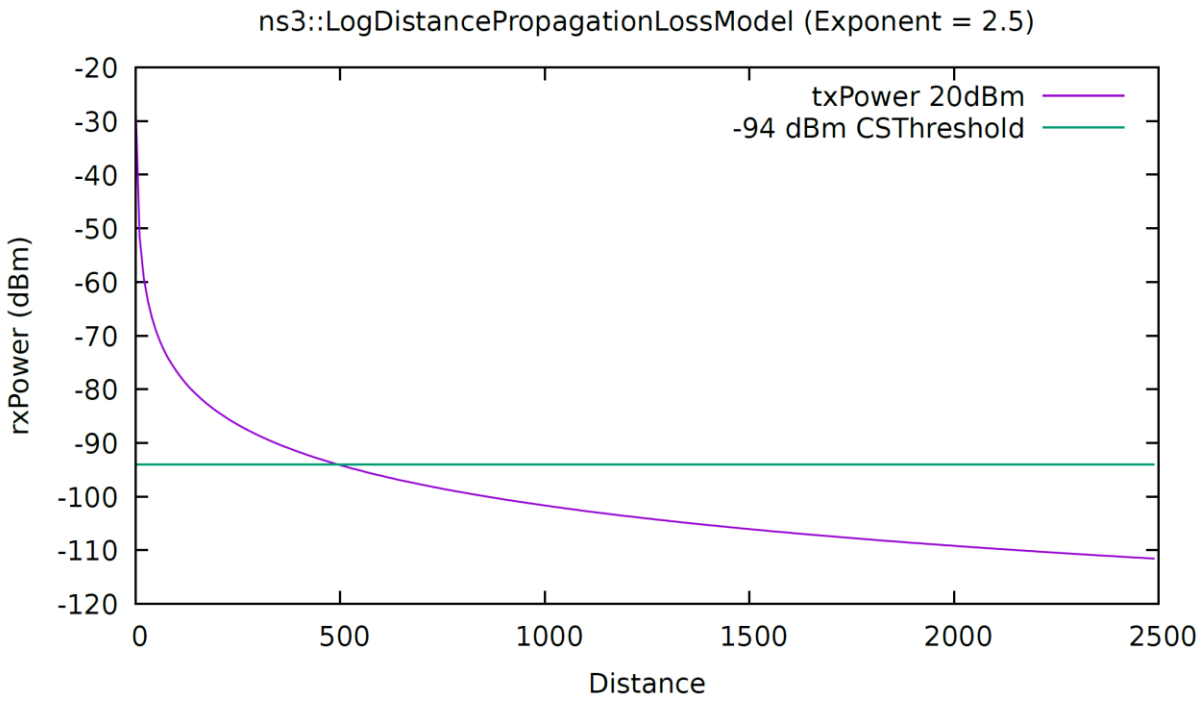

Figure 6.LogDistance Model Default

Figure 7 illustrates a particular but possibly more realistic scenario using RandomPropagationLossModel. As explained before, this model generates random losses even between two fixed nodes. Therefore, the rxPower fluctuates along the distance axis. We can see the rxPower is mostly above the CS Threshold value, where only a few pulses occur at random distance ticks showing poor rxPower levels. We cannot observe an explicit relationship between rxPower and distance in this case.

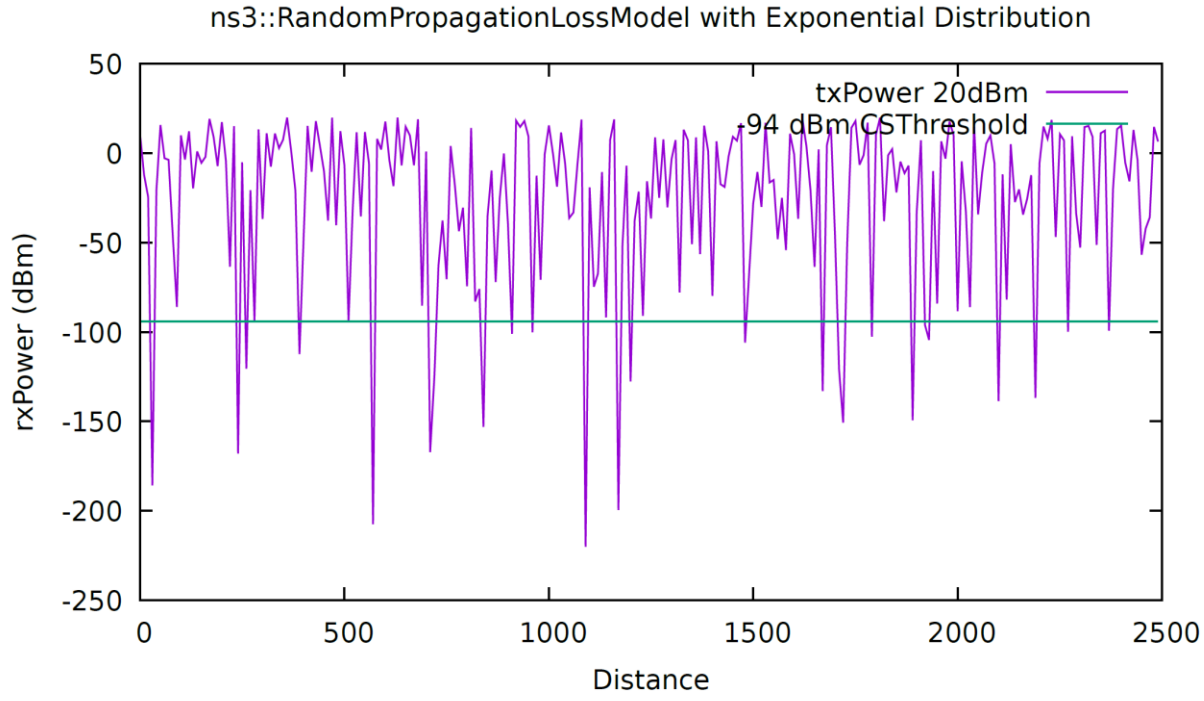

Figure 7.RandomPropagation Model

Figure 8 shows the Itu1411LosPropagationLossModel. According to Equation 5 and 6, and also the scripts implementation, the default parameters are set up as follow:

- Propagation frequency is by default 2.16 GHz
- eNB height and UE height, i.e. the z positions of the two constant mobility models, are set to 1 meter for simplicity

We can observe a steeper exponential drop than previous LogDistance model. The rxPower reaches the CS Threshold at around 300-400 meters mark. And the rxPower is expected to be entirely demolished at 2500 meters mark.

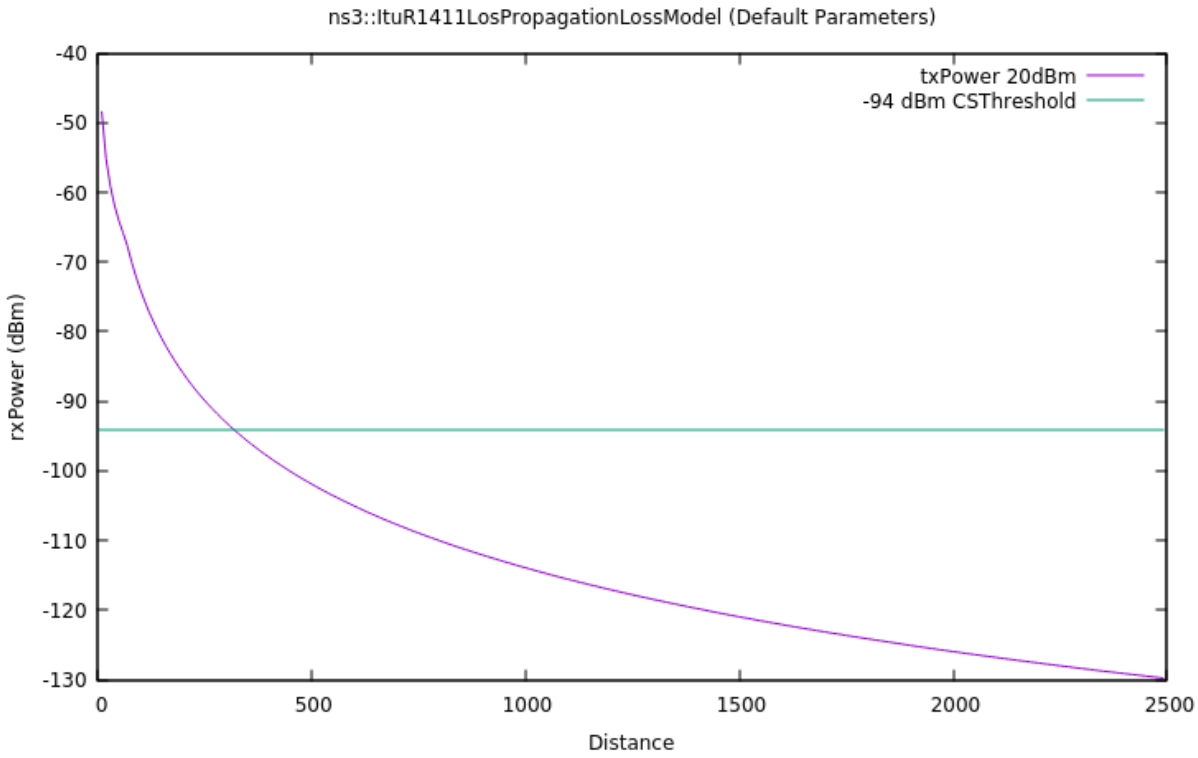

Figure 8.ItuR1411 Model

Figure 9 and 10 demonstrates ThreeLogDistancePropagationLossModel with different combinations of path loss exponent values. The path loss exponent is usually within the range of 2-4, where two can be considered as the free space, and 4 is regarded as a lossy environment. The tunnel with a value of less than 2 can be treated as a waveguide, which has a minimal loss of energy. The indoor environment or construction surroundings may have a value greater than 4.

Therefore, Figure 9 represents the tunnel combinations of two free spaces (or waveguides) followed by one lossy space. The distance where the preamble signal level is detected is approximately within 700-800 meters. Figure 10 represents the tunnel combinations of the perfect waveguide, slightly lossy space and extremely lossy environment. Due to the difference between the first space compared to Figure 9, the initial drop in Figure 10 is smaller than that in Figure 9. The extremely lossy environment in Figure 10 causes linear decrement while an exponential deaccelerating decay happens in Figure 9. In a word, the combination in Figure 10 causes lower signal power level under long-distance cases, but it has a larger value of the preamble signal level distance of around 1100 meters.

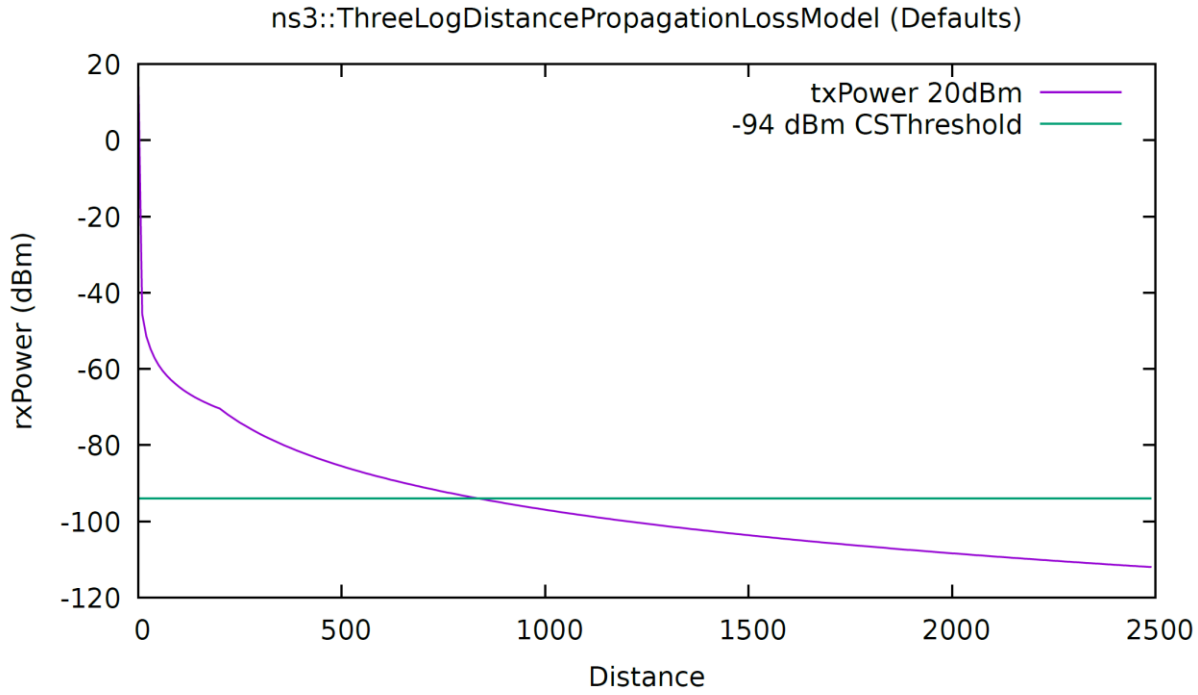

Figure 9.ThreeLog Model (exponent 1.9,1.9,3.8)

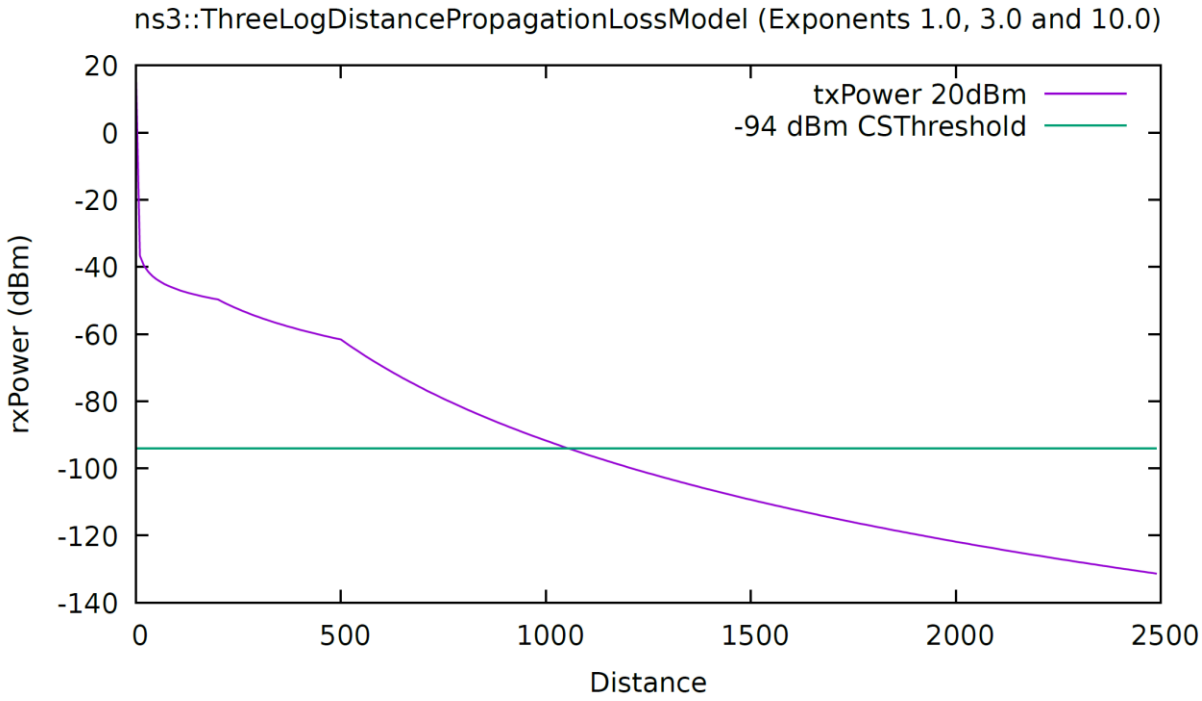

Figure 10.ThreeLog Model (exponent 1,3,10)

Figure 11 shows the COST231 model with default parameters. Based on Equation 7 and 8, and the scripts implementation, the default settings consist of:

- The wavelength is calculated based on 2.3 GHz frequency and speed of light (300 000 km/s)
- The base station antenna height is 50 meters

- The mobile station antenna height is 3 meters
- The distance under which the propagation model refuses to give results is 0.5 meters

In this model, we observe the steepest decay among the propagation models so far. The CS Threshold level reaches the preamble rxPower level at around 200-300 meters. But the rxPower is still not annihilated when transmission distance reaches 2500 meters.

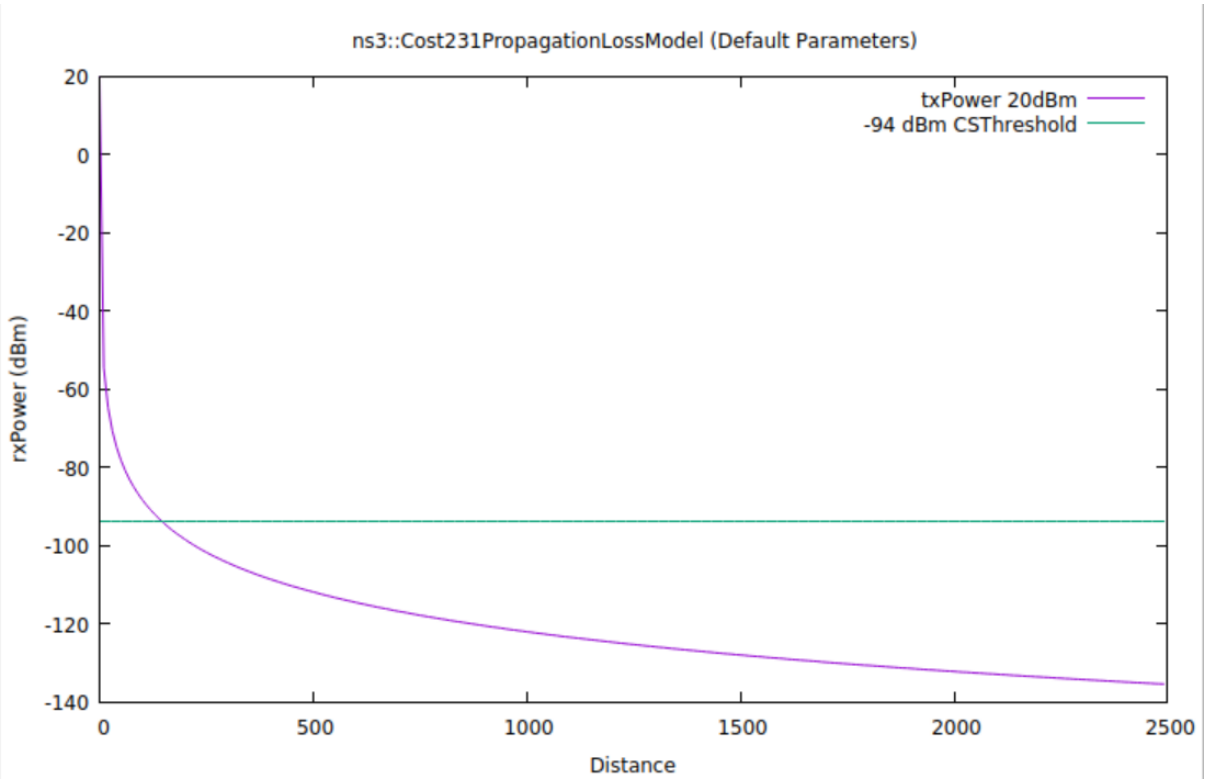

Figure 11.Cost231 Model

Figure 12 and Figure 13 illustrate the NakagamiPropagationLossModel. As discussed in the previous section, this model applies the Nakagami-m distribution model to the power function based on other propagation models. Figure 12 uses the default settings as follow:

- The beginning of the second distance field is 80 meters
- The beginning of the third distance field is 200 meters
- M0 for distance smaller than the second distance field is 1.5
- M1 for distance smaller than the third distance field is 0.75
- M2 for a distance greater than the second distance field is 0.75

Therefore, in Figure 13, the model uses the definition of fast fading for power levels and integrates onto the ThreeLogDistancePropagationLossModel. Only the model in Figure 12 uses the default formula mentioned in Equation 7.

From Figure 12, we can see that for a fixed distance, datasets with higher probability are constantly distributed around 10-20 dBm rxPower level. While in Figure 13, as the path loss factor is introduced, if we focus on only the highest possible points, the rxPower roughly experiences an exponential decrement. And the intersection point of txPower line and CS Threshold value is approximately within

the range of 500-1000 meters, which is consistent with the observations in ThreeLogDistancePropagationLossModel.

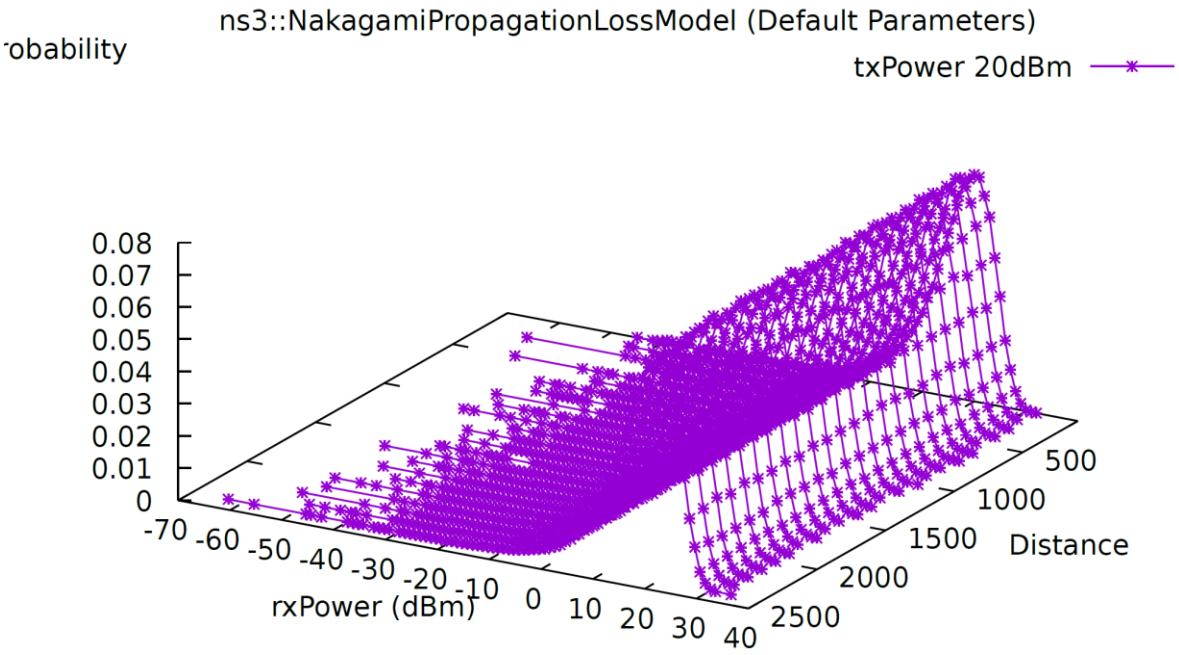

Figure 12.Nakagami Model Default

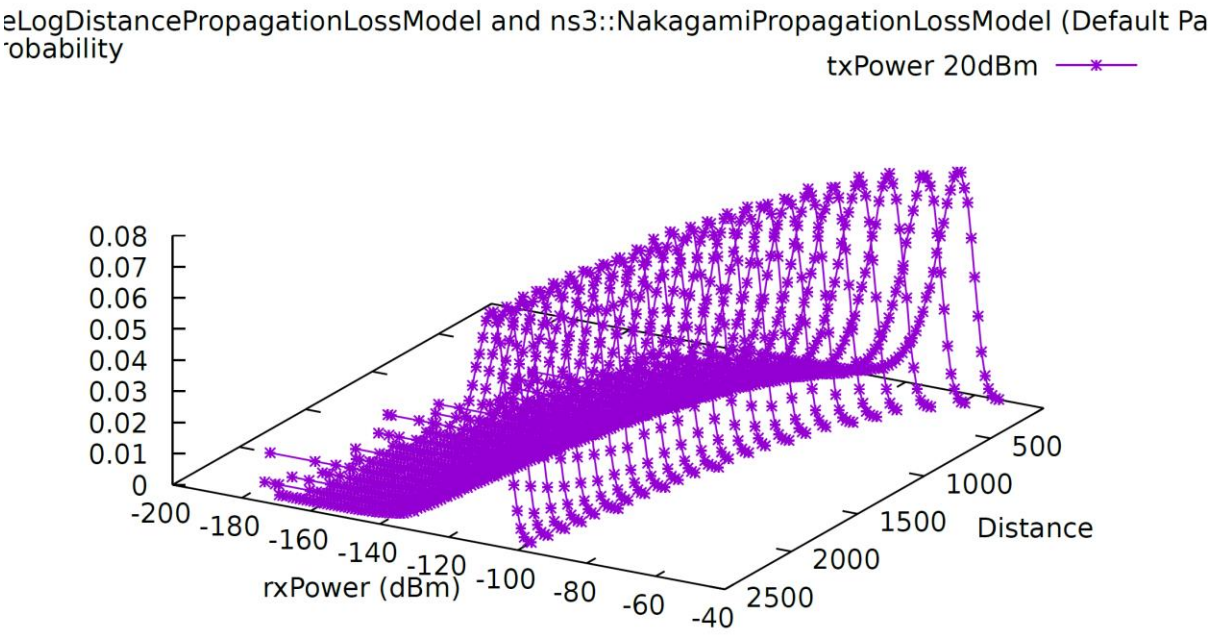

Figure 13.Nakagami Model + ThreeLog Model Default

From these models above, we can conclude two general inspections:

- The rxPower decays all the time
- The distance interval where steeper decrement and the intersection of CS Threshold and txPower line occur is around 500-1500 meters

Now, we start simulating the case in Riverbed model. We first vary the distance between the mobile client to the access point based on the obtained inspection from the propagation models.

Figure 14 to Figure 25 shows the process of varying distance while observing the data dropped versus the throughput. We can see that when the distance is less than 400 meters, the total throughput is 8 Mbps without data dropped. After that, the data dropped starts to emerge, and the data dropped due to buffer overflow starts to dominate. When d = 670 meters, the WLAN throughput is nearly demolished, which means the connection can be considered thoroughly corrupted.

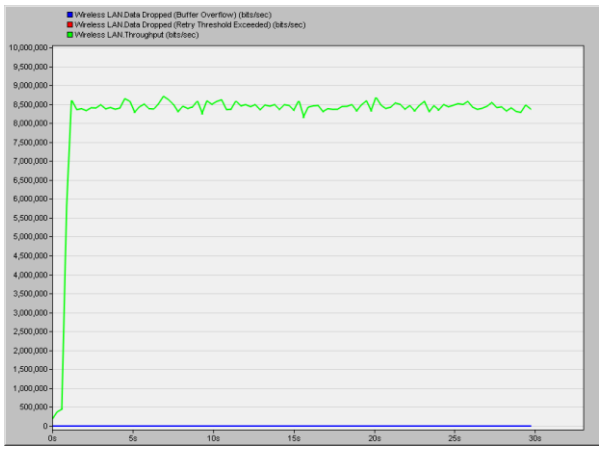

Figure 14. d <= 400 meters

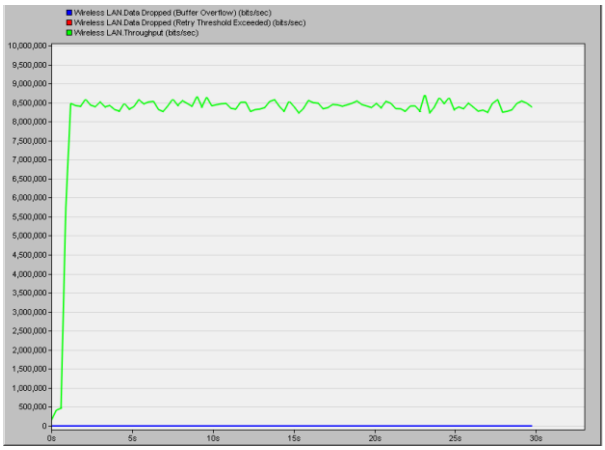

Figure 15. d = 500 meters

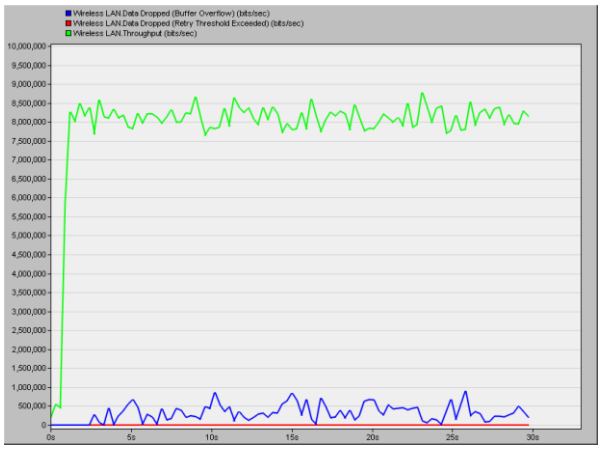

Figure 16. d = 560 meters

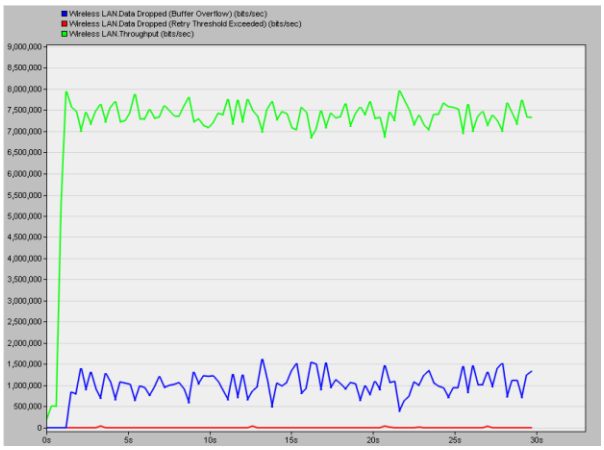

Figure 17.d = 570 meters

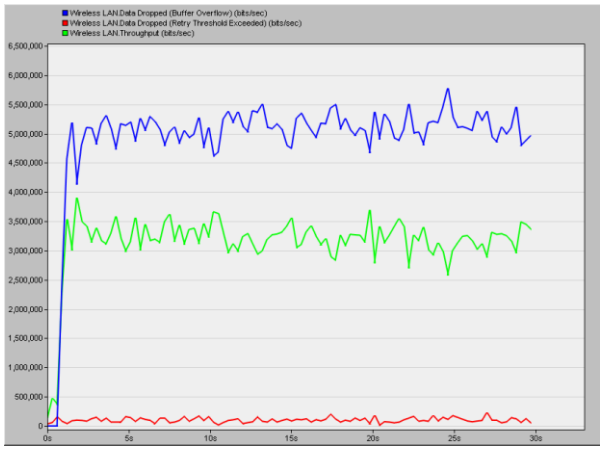

Figure 18. d = 600 meters

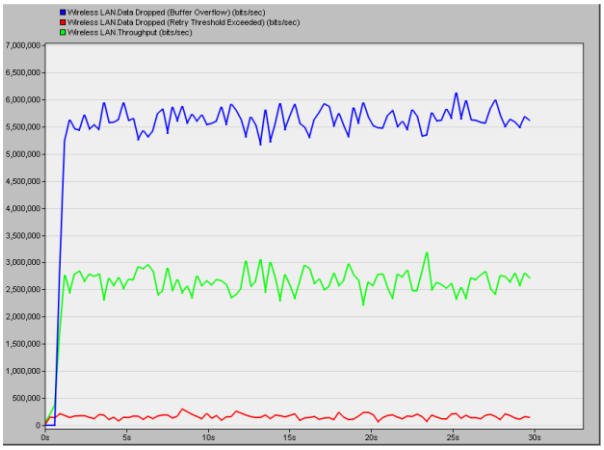

Figure 19. d = 605 meters

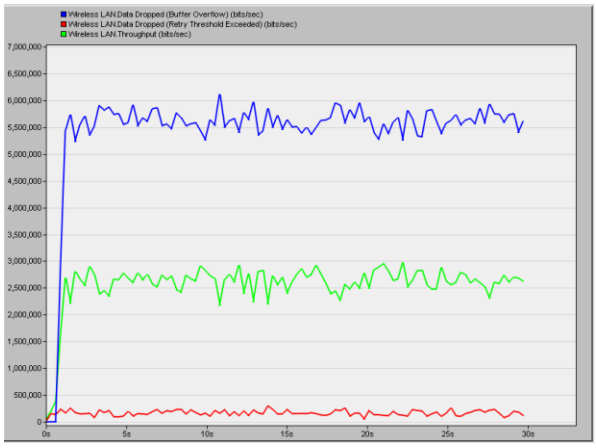

Figure 20. d = 610 meters

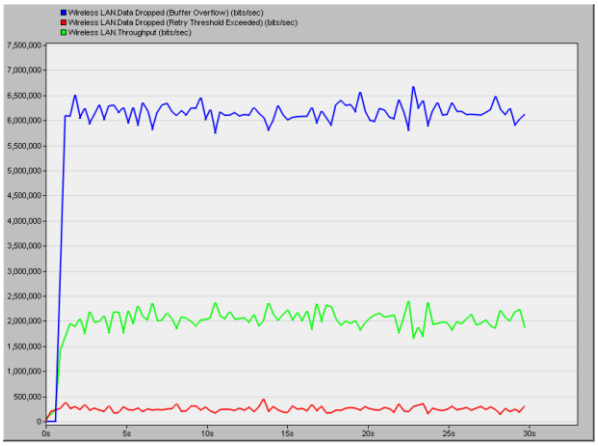

Figure 21. d = 615 meters

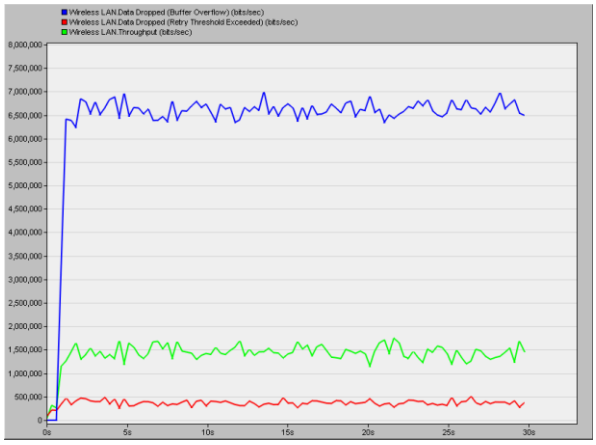

Figure 22. d = 620&625 meters

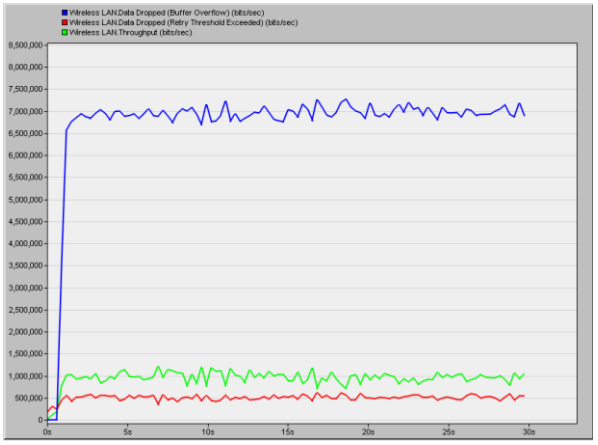

Figure 23. d = 630 meters

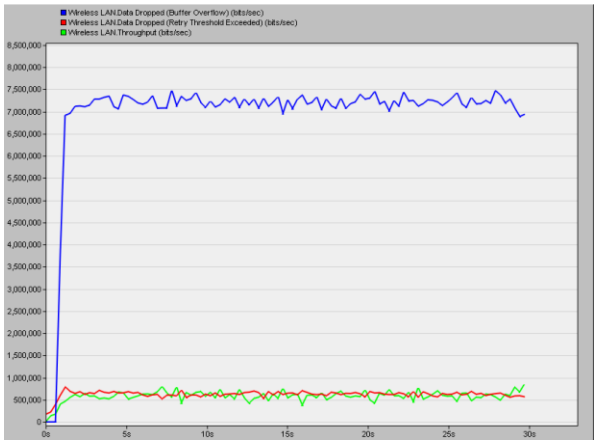

Figure 24. d = 635&640 meters

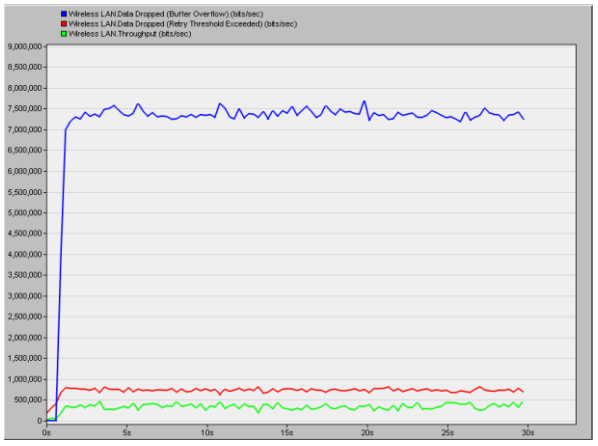

Figure 25. d = 645 meters

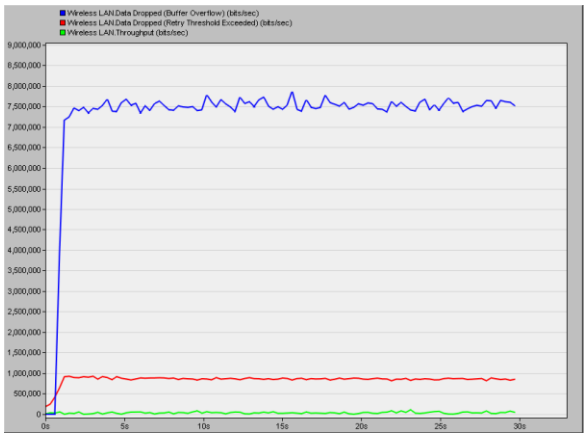

Figure 26. d = 650 - 665 meters

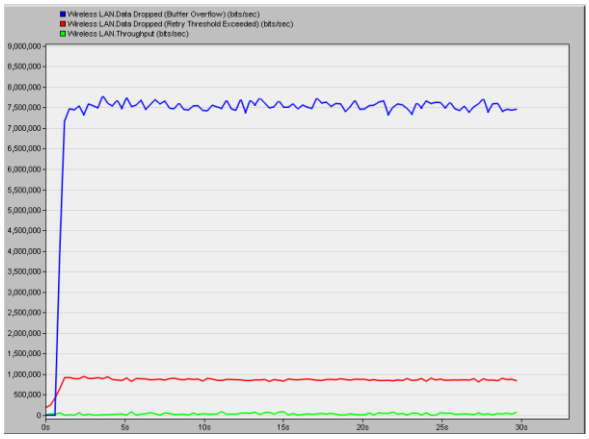

Figure 27. d = 670 meters

Now we fix the distance between the mobile client and access point at 300 meters (which guarantees the full throughput), and we try to vary the distance between the destination and access point.

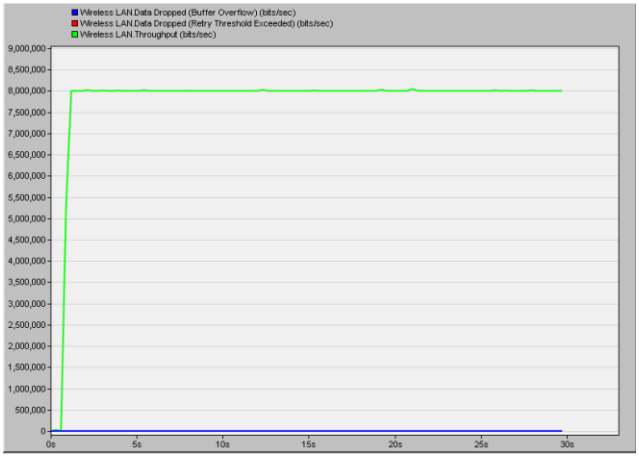

Figure 28. d = 3000 meters

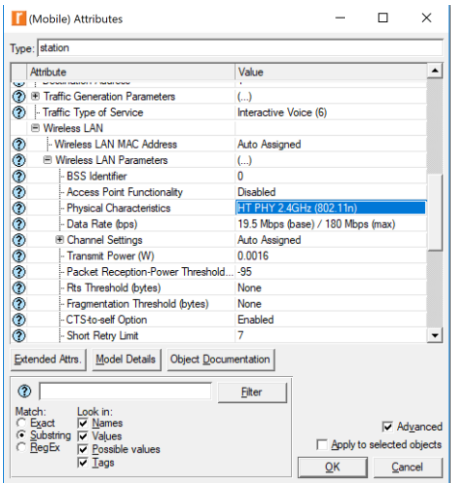

Figure 29. 802.11n settings

Figure 28 shows the case when d = 3000 meters. However, the WLAN throughput is still stabilized at maximum nearly without losses.

Lastly, we change the physical parameters of both the mobile client and access point to 2.4 GHz 802.11n, and the data rate changes to 19.5 Mbps as shown in Figure 29.

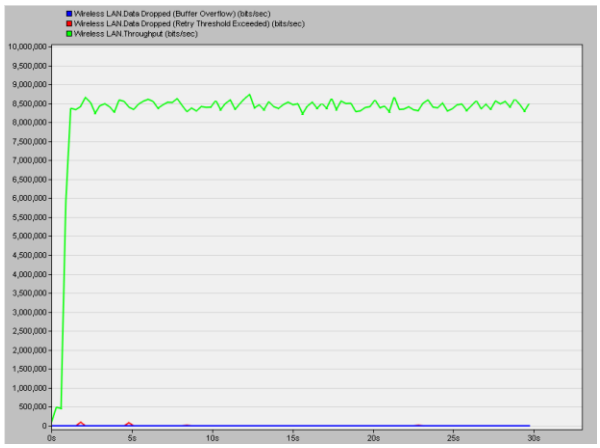

Figure 30. d <= 510 meters

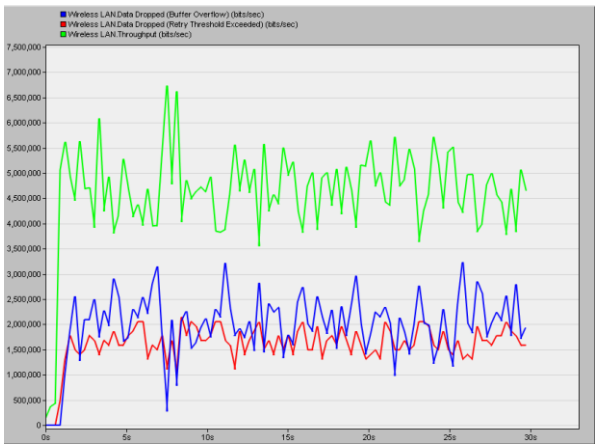

Figure 31. d = 520 meters

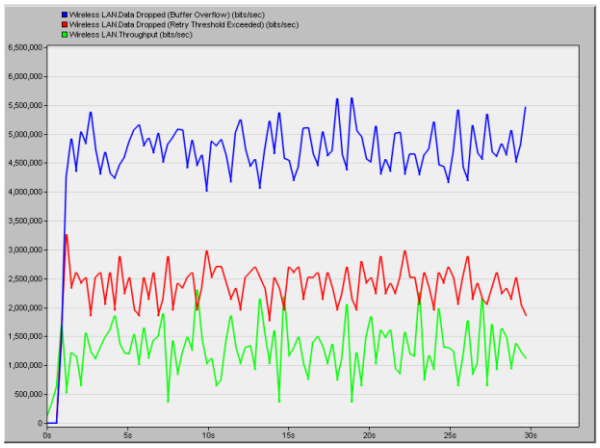

Figure 32. d = 530 meters

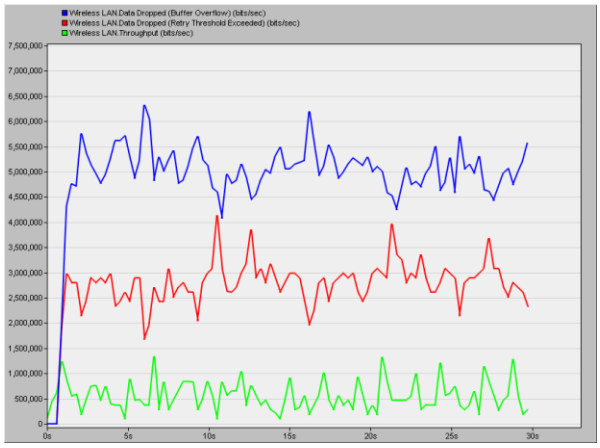

Figure 33. d = 540 meters

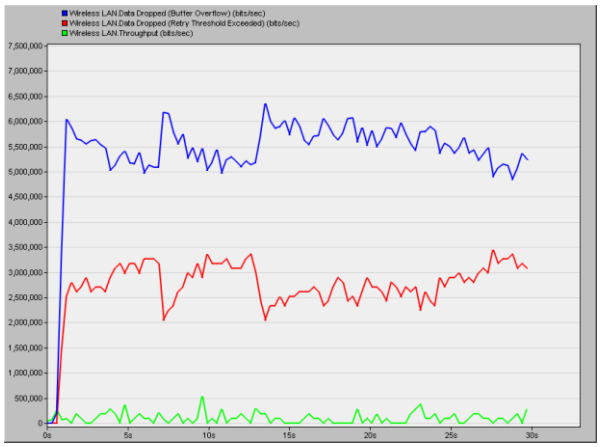

Figure 34. d = 550 meters

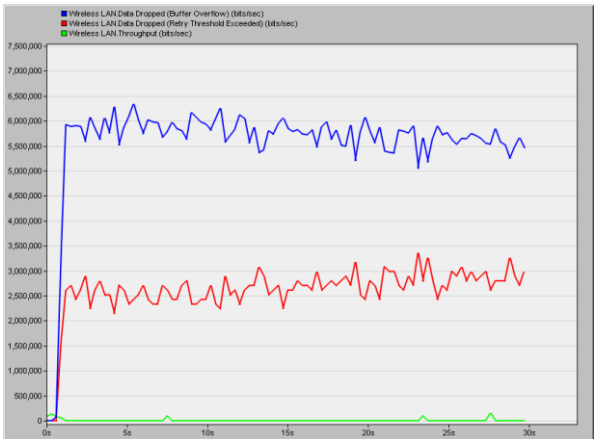

Figure 35. d = 560 meters

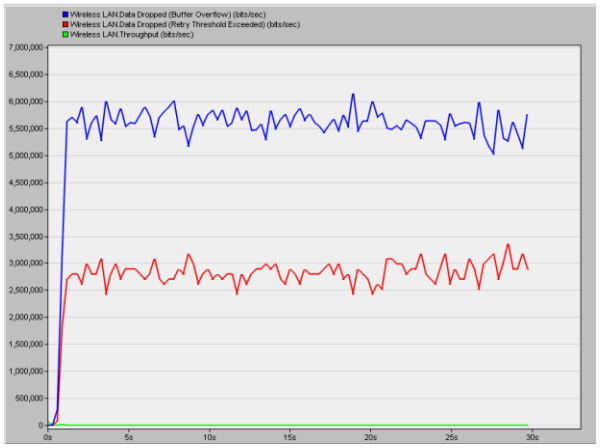

Figure 36. d = 570 meters

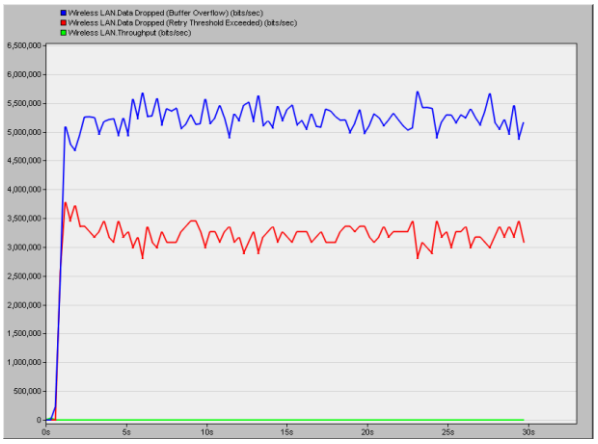

Figure 37. d = 580 meters

From Figure 30 to Figure 37, we can see that the data dropped bursts out at d = 520 meters, where the dropped data due to buffer overflow and retransmission are at the same level of quantity. When distance approaches 570-580 meters, the throughput can be regarded as annihilation, and the data dropped due to buffer overflow and retransmission are roughly stabilized at 4.3 Mbps and 3.7 Mbps.

Discussion and Conclusion

From the simulation results of the previous section, we have obtained some consistent observations:

- Both the theoretical models and actual simulation demonstrates a positive correlation between packet loss and transmission distance
- The distance range where data starts to drop until the CS Threshold value explicitly is reached consistent for both the theoretical models and actual simulation. Specifically, theoretical models indicate the range to be 500-1000 meters, while the two simulation results show the ranges of 400-670 meters and 510-580 meters, respectively.

Riverbed simulation results show that the distance between the access point and destination is not relevant to the packet loss. The reason is that these two blocks are connected via optical fibre 1000-BASE-X, which is widely used in Gigabit Ethernet. It can be considered as lossless with the rapid transmission rate.

For both 802.11g and 802.11n cases, the buffer overflow causes high data dropped rate as distance increases. The limitation of data transmission rate might cause this phenomenon. Specifically, long transmission distance will lead to more packets getting stuck in the buffer and waiting for a response. In 802.11n case, the retransmission rate is also significantly large for long distance, which might be caused by weaker rxPower level in the longer distance travelling. The propagation model in ns-3 modeller also indicates this relationship, where WIFI signal can be hardly detected when the rxPower level is below the CS Threshold value.

Furthermore, the distance range in the 802.11n case is narrower than the 802.11g case. It might be caused by the difference of bandwidth between two WIFI standards. 802.11g only supports bandwidth up to 54 Mbps while 802.11n supports up to 300Mbp by using MIMO technology [9]. Higher bandwidth may easily cause higher retransmission rate, which is illustrated in the simulation results.

In conclusion, the theoretical propagation loss models and simulation results indicate that longer transmission distance will cause a higher rate of packet loss in WIFI scenario.

References

[1] Ns-3, "Propogation," 04 March 2019. [Online]. Available: https://www.nsnam.org/docs/models/html/propagation.html. [Accessed 06 March 2019].

[2] J. Hartman, "This Is Why Your Live Stream Lags: Intro To Live Streaming Latency," Boxcast, 01 January 2019. [Online]. Available: https://www.boxcast.com/blog/live-stream-video-latency. [Accessed 7 February 2019].

[3] K. Daham, "Top 10 Most Common Causes for Poor Network Performance," Computer Works, 28 October 2016. [Online]. Available: http://www.cwims.com/performance/top-10-most-common-causes-for-poor-network-performance/. [Accessed 7 February 2019].

[4] L. Rivenes, "What are the Causes of Packet Loss?," Datapath.io, 8 March 2016. [Online]. Available: https://datapath.io/resources/blog/causes-of-packet-loss/. [Accessed 7 February 2019].

[5] C.-H. C. L.-F. L.-P. H. W.-T. H. G.-C. L. Yuan-Jen Chang, "Wireless Sensor Networks for Vital signs monitoring: Application in a Nursing Home," 25 December 2013. [Online]. Available: https://www.researchgate.net/publication/258386464. [Accessed 7 February 2019].

[6] R. Johnson, Antenna Engineering Handbook (2nd ed.), New York: NY: McGraw-Hill, Inc., 1984, pp. 1-12.

[7] H. Friis, in *A Note on a Simple Transmission Formula*, IRE Proc, 1946, pp. 254-256.

[8] D. COLEMAN, "What is a Clear Channel Assessment (CCA)?," 5 July 2018. [Online]. Available: https://blog.aerohive.com/what-is-clear-channel-assessment-cca/. [Accessed 23 March 2019].

[9] P. Romano, "WiFi Standards 802.11a/b/g/n vs. 802.11ac: Which is Best?," Symmetry Electronics, 16 July 2014. [Online]. Available: https://www.semiconductorstore.com/blog/2014/WiFi-standards-802-11a-b-g-n-vs-802-11ac-Which-is-Best/806/. [Accessed 29 March 2019].

APPENDIX:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
 /*
  * Copyright (c) 2008 Timo Bingmann
  *
  * 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: Timo Bingmann <timo.bingmann@student.kit.edu>
  * Modified by: Shuo Chen <sca185@sfu.ca>
  */

 #include "ns3/propagation-loss-model.h"
 #include "ns3/jakes-propagation-loss-model.h"
 #include "ns3/constant-position-mobility-model.h"
 #include "ns3/okumura-hata-propagation-loss-model.h"
 #include "ns3/cost231-propagation-loss-model.h"
 #include "ns3/itu-r-1411-los-propagation-loss-model.h"

 #include "ns3/config.h"
 #include "ns3/command-line.h"
 #include "ns3/string.h"
 #include "ns3/boolean.h"
 #include "ns3/double.h"
 #include "ns3/pointer.h"
 #include "ns3/gnuplot.h"
 #include "ns3/simulator.h"


 #include <map>

 using namespace ns3;

//The mathematical rounding function
 static double dround (double number, double precision)
 {
   number /= precision;
   if (number >= 0)
     {
       number = floor (number + 0.5);
     }
   else
     {
       number = ceil (number - 0.5);
     }
   number *= precision;
   return number;
 }

 static Gnuplot
//Generate plot based on the distance and Rx power
 TestDeterministic (Ptr<PropagationLossModel> model)
 {
//Set up constant position model which does not change once set
   Ptr<ConstantPositionMobilityModel> a =
CreateObject<ConstantPositionMobilityModel> ();
   Ptr<ConstantPositionMobilityModel> b =
CreateObject<ConstantPositionMobilityModel> ();

   Gnuplot plot;
//Add labels of distance, rxPower
```

```
   plot.AppendExtra ("set xlabel 'Distance'");
   plot.AppendExtra ("set ylabel 'rxPower (dBm)'");
   plot.AppendExtra ("set key top right");

   double txPowerDbm = +20; // dBm

   Gnuplot2dDataset dataset;

   dataset.SetStyle (Gnuplot2dDataset::LINES);

   {
//Set model a as the origin
     a->SetPosition (Vector (0.0, 0.0, 1.0));

     for (double distance = 0.0; distance < 2500.0; distance += 10.0)
       {
//Vary the position of b only on the distance axis
         b->SetPosition (Vector (distance, 0.0, 1.0));

         // CalcRxPower() returns dBm.
         double rxPowerDbm = model->CalcRxPower (txPowerDbm, a, b);

         dataset.Add (distance, rxPowerDbm);



         Simulator::Stop (Seconds (1.0));
         Simulator::Run ();
       }
   }
//output dataset
   std::ostringstream os;
   os << "txPower " << txPowerDbm << "dBm";
   dataset.SetTitle (os.str ());

   plot.AddDataset (dataset);

   plot.AddDataset ( Gnuplot2dFunction ("-94 dBm CSThreshold", "-94.0") );

   return plot;
 }

 static Gnuplot
//construct a 3d diagram of xyz axises representing distance, rxpower and
probablity
 TestProbabilistic (Ptr<PropagationLossModel> model, unsigned int samples =
100000)
 {
   Ptr<ConstantPositionMobilityModel> a =
CreateObject<ConstantPositionMobilityModel> ();
   Ptr<ConstantPositionMobilityModel> b =
CreateObject<ConstantPositionMobilityModel> ();

   Gnuplot plot;

   plot.AppendExtra ("set xlabel 'Distance'");
   plot.AppendExtra ("set ylabel 'rxPower (dBm)'");
   plot.AppendExtra ("set zlabel 'Probability' offset 0,+10");
   plot.AppendExtra ("set view 50, 120, 1.0, 1.0");
   plot.AppendExtra ("set key top right");
```

```
   plot.AppendExtra ("set ticslevel 0");
   plot.AppendExtra ("set xtics offset -0.5,0");
   plot.AppendExtra ("set ytics offset 0,-0.5");
   plot.AppendExtra ("set xrange [100:]");

   double txPowerDbm = +20; // dBm

   Gnuplot3dDataset dataset;

   dataset.SetStyle ("with linespoints");
   dataset.SetExtra ("pointtype 3 pointsize 0.5");

//map the floating value of rxpower to a index value
   typedef std::map<double, unsigned int> rxPowerMapType;

   // Take given number of samples from CalcRxPower() and show probability
   // density for discrete distances.
   {
     a->SetPosition (Vector (0.0, 0.0, 0.0));

     for (double distance = 100.0; distance < 2500.0; distance += 100.0)
       {
         b->SetPosition (Vector (distance, 0.0, 0.0));

         rxPowerMapType rxPowerMap;

         for (unsigned int samp = 0; samp < samples; ++samp)
           {
             // CalcRxPower() returns dBm.
             double rxPowerDbm = model->CalcRxPower (txPowerDbm, a, b);
//using predefined rounding function for mapping
             rxPowerDbm = dround (rxPowerDbm, 1.0);
//add rxpower to the mapping list
             rxPowerMap[ rxPowerDbm ]++;


             Simulator::Stop (Seconds (0.01));
             Simulator::Run ();
           }
//iterate the mapping list, calculate corresponding probablity
         for (rxPowerMapType::const_iterator i = rxPowerMap.begin ();
              i != rxPowerMap.end (); ++i)
           {
             dataset.Add (distance, i->first, (double)i->second /
(double)samples);
           }
         dataset.AddEmptyLine ();
       }
   }

   std::ostringstream os;
   os << "txPower " << txPowerDbm << "dBm";
   dataset.SetTitle (os.str ());

   plot.AddDataset (dataset);

   return plot;
 }

 static Gnuplot
//evaluate the relationship based on timestamp
```

```
 TestDeterministicByTime (Ptr<PropagationLossModel> model,
                          Time timeStep = Seconds (0.001),
                          Time timeTotal = Seconds (1.0),
                          double distance = 100.0)
 {
   Ptr<ConstantPositionMobilityModel> a =
CreateObject<ConstantPositionMobilityModel> ();
   Ptr<ConstantPositionMobilityModel> b =
CreateObject<ConstantPositionMobilityModel> ();

   Gnuplot plot;

   plot.AppendExtra ("set xlabel 'Time (s)'");
   plot.AppendExtra ("set ylabel 'rxPower (dBm)'");
   plot.AppendExtra ("set key center right");

   double txPowerDbm = +20; // dBm

   Gnuplot2dDataset dataset;

   dataset.SetStyle (Gnuplot2dDataset::LINES);

   {
     a->SetPosition (Vector (0.0, 0.0, 0.0));
     b->SetPosition (Vector (distance, 0.0, 0.0));
//using the embedded time counter to iterate dataset
     Time start = Simulator::Now ();
     while( Simulator::Now () < start + timeTotal )
       {
         // CalcRxPower() returns dBm.
         double rxPowerDbm = model->CalcRxPower (txPowerDbm, a, b);

         Time elapsed = Simulator::Now () - start;
         dataset.Add (elapsed.GetSeconds (), rxPowerDbm);


         Simulator::Stop (timeStep);
         Simulator::Run ();
       }
   }

   std::ostringstream os;
   os << "txPower " << txPowerDbm << "dBm";
   dataset.SetTitle (os.str ());

   plot.AddDataset (dataset);

   plot.AddDataset ( Gnuplot2dFunction ("-94 dBm CSThreshold", "-94.0") );

   return plot;
 }

 int main (int argc, char *argv[])
 {
   CommandLine cmd;
   cmd.Parse (argc, argv);

   GnuplotCollection gnuplots ("main-propagation-loss.pdf");

   {
```

```
     Ptr<FriisPropagationLossModel> friis =
CreateObject<FriisPropagationLossModel> ();
/*.AddAttribute ("SystemLoss", "The system loss",
                    DoubleValue (1.0),
                    MakeDoubleAccessor
(&FriisPropagationLossModel::m_systemLoss),
                    MakeDoubleChecker<double> ())*/

     Gnuplot plot = TestDeterministic (friis);
     plot.SetTitle ("ns3::FriisPropagationLossModel (Default Parameters)");
     gnuplots.AddPlot (plot);
   }

   {
     Ptr<LogDistancePropagationLossModel> log =
CreateObject<LogDistancePropagationLossModel> ();
     log->SetAttribute ("Exponent", DoubleValue (2.5));

     Gnuplot plot = TestDeterministic (log);
     plot.SetTitle ("ns3::LogDistancePropagationLossModel (Exponent =
2.5)");
     gnuplots.AddPlot (plot);
   }

   {
     Ptr<RandomPropagationLossModel> random =
CreateObject<RandomPropagationLossModel> ();
     Ptr<ExponentialRandomVariable> expVar =
CreateObjectWithAttributes<ExponentialRandomVariable> ("Mean", DoubleValue
(50.0));
     random->SetAttribute ("Variable", PointerValue (expVar));
/*.AddAttribute ("Variable", "The random variable used to pick a loss every
time CalcRxPower is invoked.",
                    StringValue
("ns3::ConstantRandomVariable[Constant=1.0]"),
                    MakePointerAccessor
(&RandomPropagationLossModel::m_variable),
                    MakePointerChecker<RandomVariableStream> ())*/

     Gnuplot plot = TestDeterministic (random);
     plot.SetTitle ("ns3::RandomPropagationLossModel with Exponential
Distribution");
     gnuplots.AddPlot (plot);
   }

   {
     Ptr<JakesPropagationLossModel> jakes =
CreateObject<JakesPropagationLossModel> ();

     // doppler frequency shift for 5.15 GHz at 100 km/h
     Config::SetDefault ("ns3::JakesProcess::DopplerFrequencyHz",
DoubleValue (477.9));
/*.AddAttribute ("DopplerFrequencyHz", "Corresponding doppler
frequency[Hz]",
                    DoubleValue (80),
                    MakeDoubleAccessor
(&JakesProcess::SetDopplerFrequencyHz),
                    MakeDoubleChecker<double> (0.0, 1e4))*/

     Gnuplot plot = TestDeterministicByTime (jakes, Seconds (0.001),
Seconds (1.0));
```

```
      plot.SetTitle ("ns3::JakesPropagationLossModel (with 477.9 Hz
shift)");
      gnuplots.AddPlot (plot);
    }

    {
      Ptr<JakesPropagationLossModel> jakes =
CreateObject<JakesPropagationLossModel> ();

      // doppler frequency shift for 5.15 GHz at 100 km/h
      Config::SetDefault ("ns3::JakesProcess::DopplerFrequencyHz",
DoubleValue (477.9));

      Gnuplot plot = TestDeterministicByTime (jakes, Seconds (0.0001),
Seconds (0.1));

      plot.SetTitle ("ns3::JakesPropagationLossModel (with 477.9 Hz
shift)");
      gnuplots.AddPlot (plot);
    }

    {
      Ptr<ThreeLogDistancePropagationLossModel> log3 =
CreateObject<ThreeLogDistancePropagationLossModel> ();

      Gnuplot plot = TestDeterministic (log3);
      plot.SetTitle ("ns3::ThreeLogDistancePropagationLossModel
(Defaults)");
      gnuplots.AddPlot (plot);
    }

    {
      Ptr<ThreeLogDistancePropagationLossModel> log3 =
CreateObject<ThreeLogDistancePropagationLossModel> ();
      // more prominent example values:
      log3->SetAttribute ("Exponent0", DoubleValue (1.0));
      log3->SetAttribute ("Exponent1", DoubleValue (3.0));
      log3->SetAttribute ("Exponent2", DoubleValue (10.0));
/*.AddAttribute ("Distance0",
                     "Beginning of the first (near) distance field",
                     DoubleValue (1.0),
                     MakeDoubleAccessor
(&ThreeLogDistancePropagationLossModel::m_distance0),
                     MakeDoubleChecker<double> ())*/

      Gnuplot plot = TestDeterministic (log3);
      plot.SetTitle ("ns3::ThreeLogDistancePropagationLossModel (Exponents
1.0, 3.0 and 10.0)");
      gnuplots.AddPlot (plot);
    }

    {
      Ptr<NakagamiPropagationLossModel> nak =
CreateObject<NakagamiPropagationLossModel> ();
/*.AddAttribute ("Distance1",
                     "Beginning of the second distance field. Default is
80m.",
                     DoubleValue (80.0),
                     MakeDoubleAccessor
(&NakagamiPropagationLossModel::m_distance1),
                     MakeDoubleChecker<double> ())*/
```

```
     Gnuplot plot = TestProbabilistic (nak);
     plot.SetTitle ("ns3::NakagamiPropagationLossModel (Default
Parameters)");
     gnuplots.AddPlot (plot);
   }


   {
     Ptr<Cost231PropagationLossModel> cos =
CreateObject<Cost231PropagationLossModel> ();

     Gnuplot plot = TestDeterministic (cos);
     plot.SetTitle ("ns3::Cost231PropagationLossModel (Default
Parameters)");
     gnuplots.AddPlot (plot);
   }

   {
     Ptr<ItuR1411LosPropagationLossModel> itu =
CreateObject<ItuR1411LosPropagationLossModel> ();

     Gnuplot plot = TestDeterministic (itu);
     plot.SetTitle ("ns3::ItuR1411LosPropagationLossModel (Default
Parameters)");
     gnuplots.AddPlot (plot);
   }

   {
     Ptr<ThreeLogDistancePropagationLossModel> log3 =
CreateObject<ThreeLogDistancePropagationLossModel> ();

     Ptr<NakagamiPropagationLossModel> nak =
CreateObject<NakagamiPropagationLossModel> ();
     log3->SetNext (nak);

     Gnuplot plot = TestProbabilistic (log3);
     plot.SetTitle ("ns3::ThreeLogDistancePropagationLossModel and
ns3::NakagamiPropagationLossModel (Default Parameters)");
     gnuplots.AddPlot (plot);
   }

   gnuplots.GenerateOutput (std::cout);

   // produce clean valgrind
   Simulator::Destroy ();
   return 0;
 }
```