ET4394

NS3 Assignment Report

Wireless Networking

April 30, 2016

Authors: Evelyn Rashmi Jeyachandra (Student Nr: 4469585)

Contents

1 Introduction

A wireless LAN (WLAN) is a data transmission system which is used to provide a location-independent network access between computing devices without any cable infrastructure. WLANs are part of a major connectivity solution for a wide range of users. The Institute of Electrical and Electronics Engineers (IEEE) consented to 802.112 as the wireless LAN standard in July 1999. The initial version provided a data rate of 1 to 2 Mbps and this resulted in a very limited throughput. In order to address this concern, IEEE then consented to make 802.11b as the official WLAN standard. This standard was able to achieve transmission rates of upto 11 Mbps. IEEE 802.11 operates in the 2.4 GHz band and uses the Direct Sequence Spread Spectrum (DSSS) method. IEEE 802.11b WLANs use dynamic rate shifting [1], which allows the data rates to be automatically adjusted in order to compensate for the changing nature of the radio channel. There are two modes of operation of 802.11 standards - Infrastructure mode and Ad-Hoc mode. In the infrastructure mode, many clients are connected to one access point. The clients use the resources of a traditional wired network through the access point. In Ad-Hoc mode, clients may communicate directly with each other, without the need for an access point. Four different data rates are available for the IEEE 802.11b standard - 1, 2, 5.5 and 11 Mbps. Normally, the speed available to the users is 11 Mbps. However, when there is an interference, the transmission will occur at lower speeds [2].

2 Objective

The aim of this assignment is to determine the performance of an IEEE 802.11b network under varying traffic conditions. For example, transmission of an audio file will generally be of shorter duration and will have less number of packets compared to a much larger video file. The traffic parameters that will be evaluated are the size of the packets (also called as payload) and the number of packets. Additional performance metrics such as the combined effect of the above two factors with the number of nodes and the data rate are also investigated.

3 Implementation

The tools used in this implementation are NS3 and gnuplot. NS3 is a free, open-source network simulator. Network simulation is defined as the process of verifying, understanding and testing the performance of a wireless communication network, by means of replication of interaction between network entities in a computer program [3]. NS3 is a follow up to NS2, but is not backward compatible with NS2. Though NS3 is the latest version, many completely developed modules that are available in NS2 and not present in NS3. Simulations are done using C++, and Python.

The implementation follows the infrastructure mode of operation - one access point and multiple clients or station nodes. The implementation steps are briefly described as follows: The access point and five station nodes are created through the `NodeContainer` class. Many helper classes are available that make common operations easier, so that the user does not have to dig too deep into the code. Propagation loss and delay models are set using `YansWifiChannelHelper`. There are two different control rate algorithms available - constant rate and Aarf wifi. The data rate used in the implementation is changed as per the analysis requirement. By default, the constant rate algorithm is used for this exercise. Subsequently, the media access control (MAC) parameters are defined through `NqosWifiMacHelper`. The mobility of the nodes are defined. There are several mobility models are available. The access point is modelled using `ConstantPositionMobilityModel` because, in real-time, the access point will remain at a fixed location. The nodes are modelled using `RandomWalk2dMobilityModel` as users will not remain at fixed spots, unlike the access point. Other notable mobility models are Random Direction 2D Mobility Model, and Gauss Markov model. The internet stack is configured and added to the nodes using `InternetStackHelper` and `Ipv4AddressHelper`. The former is used for IP, TCP and UDP aggregation purposes, while the latter is used for address configuration. UDP traffic is generated and received using `UdpServerHelper` and `UdpClientHelper`. Application is used to run the simulation for the specified duration of 10 seconds. Flow monitor is used to observe the performance measures such as throughput, delay, the number of

transmitted and received packets. Two important formulae used to determine the throughput and delay are

$$\text{Throughput} = \frac{\text{Number of bytes received} \times 8}{\text{Time between the last and the first received packet} \times 1024 \times \text{Number of nodes}}$$

$$\text{Delay} = \frac{\text{Total Delay}}{\text{Number of received packets}}$$

The following scenarios will be simulated for analysis:

1. Throughput vs packet size
2. Throughput vs number of packets
3. Throughput vs packet size (with varying number of nodes and data rates)
4. Throughput vs number of packets (with varying number of nodes and data rates)

4 Performance Analysis

The ns3 program is compiled using the waf compiler. Waf is a python-based compiler tool for configuring, compiling and installing applications. It is a good alternative to other autotools such as cmake. The simulation output is displayed in Figure .

```
Waf: Entering directory `/home/evelyn/workspace/ns-allinone-3.24.1/ns-3.24.1/build'
Waf: Leaving directory `/home/evelyn/workspace/ns-allinone-3.24.1/ns-3.24.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.777s)
Access point creation: Complete
Nodes creation: Complete
Wifi 802.11b Channel configuration: Complete
Control rate configuration: Complete
SSID, ApDevice & StaDevice configuration: Complete
Node mobility configuration: Complete
Internet Stack & IPv4 address configuration: Complete
UDP traffic generation: Complete
Flow 1 (10.1.1.1 -> 10.1.1.6)
 Tx Bytes: 6168000
 Rx Bytes: 1092764
 Average Throughput: 142.375 kbps
 Delay : +494479702.0ns
evelyn@evelyn-Inspiron-5558:~/workspace/ns-allinone-3.24.1/ns-3.24.1$
```

Figure 1: NS3 simulation output

4.1 Throughput vs Packet Size

One of the most distinguishing features of traffic is the size of the packet. The variable corresponding to this parameter, `packetSize`, is varied and the throughput results are documented. Figure 2 shows the variation of the throughput. The number of packets is equal to 10000 and the number of nodes is equal to 5.
The packet size is varied from 50 to 1500 bytes and the values of delay and throughput are measured. From the graph, it can be observed that a smaller packet size results in a lower throughput. When the packet size is small, the overall number of bytes transmitted will reduce. For example, within a two second window, the number of bytes involved in transmitting a 100-byte packet is more than a 10-byte packet. This subsequently lowers the number of bytes at the receiving end. As per the throughput formula, the throughput is directly proportional to the number of bytes received. Thus, the average throughput is reduced. One of the ways to improve the throughput is thus, by increasing the packet size. The packet size cannot be increased beyond 1500.

4.2 Throughput vs Number of Packets

Another important factor that distinguishes traffic is the number of packets. The simulation results for varying number of packets are shown in Figure 3. The values of packet size and number of nodes are

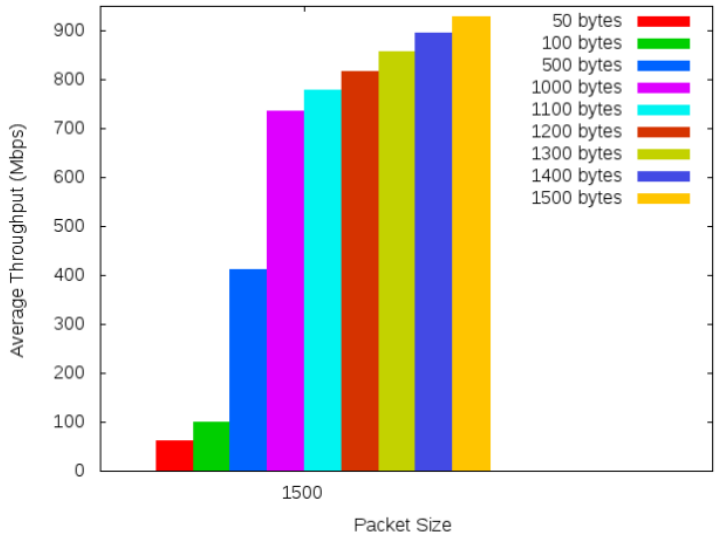


Figure 2: Throughput for varying packet sizes

constant and are equal to 1000 and 5, respectively. For these set values, it can be observed that the performance of the network is pretty reliable and that, the throughput does not falter in the event of a heavy load. The second half of the graph shows that with increasing number of packets, the throughput is more or less stable.

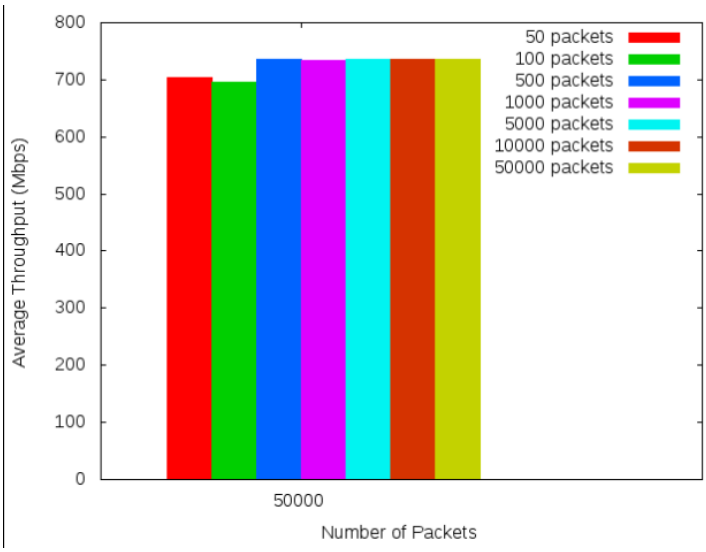
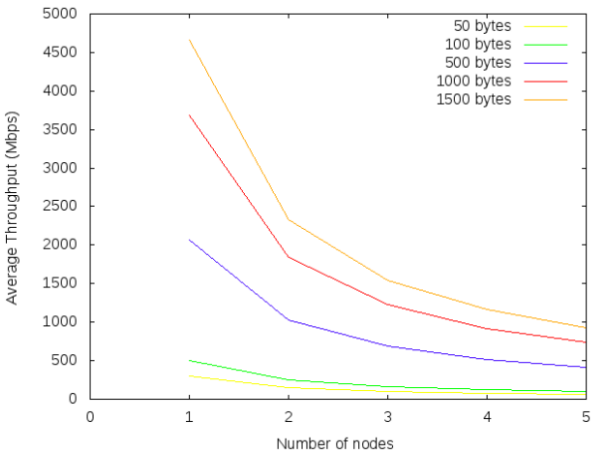


Figure 3: Throughput for different number of packets

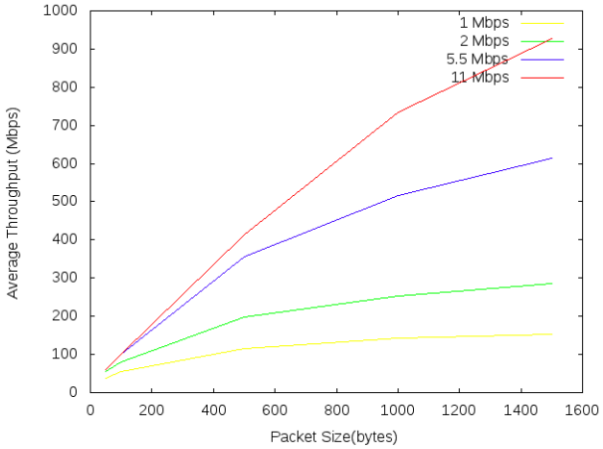
4.3 Throughput vs Packet Size (with varying number of nodes and data rates

As discussed in the introduction, there are four different data rates. In this particular simulation, the effect of packet size on the throughput is extended to include the number of nodes and the data rates too. The simulation is done for 1 to 5 nodes and for the four different data rates. The results are captured in Figure 4a and Figure 4b. Figure 4a shows that, as the number of nodes increases, the average throughput reduces. For any number of nodes, the highest average throughput is for the greatest value of packet size. As the number of nodes connected with the access point increases, the lesser the differences in average

throughput for different packet sizes.



(a) Different number of nodes



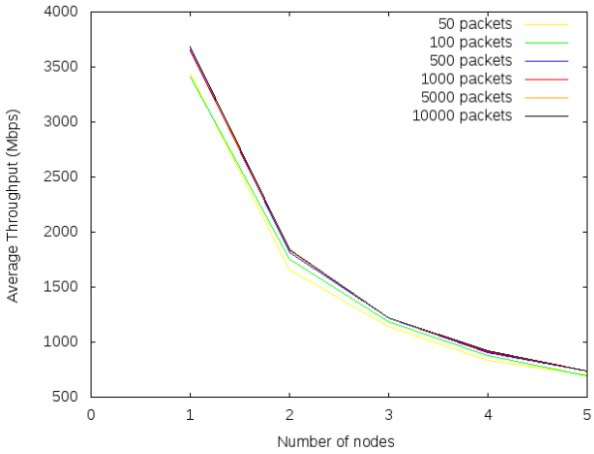
(b) Different data rates

Figure 4: Average throughput vs packet size

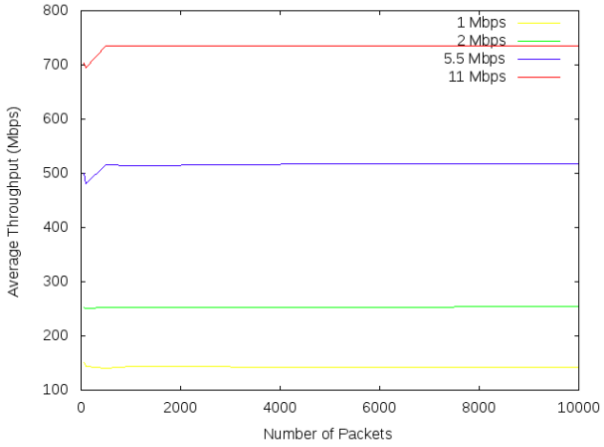
It was observed earlier that the average throughput increases with the packet size. From Figure 4b, it can be seen that for lower data rate values (1 or 2 Mbps), the average throughput is much lower than compared to 11 Mbps. This is evident because, with lower data rates, fewer number of bytes are transmitted per second. This results in a lower throughput.

4.4 Throughput vs number of packets (with varying number of nodes and data rates)

The same process done previously is repeated, but with varying number of packets. The variation of the average throughput with the number of packets for different number of nodes and data rates are shown in Figures 5a and 5b respectively. The fact that throughput remains almost stable for different number of packets is confirmed by the curves in Figures 5a and 5b. The curves in 5a, which represent different number of packets, are almost overlapping because the throughput is maintained uniformly throughout the number of nodes. In Figure 5b, there are vast differences in the throughput for a fixed number of packets. Data rate of 11 Mbps delivers the best throughput, followed by 5.5 Mbps, 2 Mbps and 1 Mbps. However, for a particular data rate, the throughput remains almost constant.



(a) Different number of nodes



(b) Different data rates

Figure 5: Average throughput vs number of packets

5 Conclusion

From the simulation results, it was evident that the average throughput reduced with an increase in the number of nodes. A data rate of 11 Mbps resulted in the best average throughput. This shows that higher the data rate, higher the throughput. Also, higher the packet size, higher the performance. The throughput did not fluctuate in the presence of a large amount of traffic. This indicates that the network performance is resilient to a sudden surge of data.

6 References

1. http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11b.php
2. http://www30.homepage.villanova.edu/phani.neelakantham/Comm%20Nets/Wireless%20Networking%20802.11.htm
3. https://blackboard.tudelft.nl/bbcswebdav/pid-2669891-dt-content-rid-9135470_2/courses/37651-151603/Slides_Wireless_Networking_P_Pawelczak_Network_Simulation_2016.pdf

Appendix

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * 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
 */
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/random-variable-stream.h"
#include "ns3/propagation-loss-model.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("Wifi");

int main (int argc, char *argv[])
{
// Number of nodes
uint32_t Nodes = 5;
// Size of packet
uint32_t packetSize = 1000;
// Number of packets
uint32_t numPackets = 10000;
// Execution time window
double StartTime = 0.0;
double StopTime = 10.0;
uint32_t rate = 0;
char rsm;

CommandLine cmd;
cmd.AddValue("numPackets","Number of packets",numPackets);
cmd.AddValue("packetSize","Packet size",packetSize);
cmd.AddValue("nodes","Nodes",nodes);
cmd.AddValue("rate","Rate",rate);
cmd.Parse (argc,argv);

// Set data rate
```

```
StringValue DataRate;
switch(rate)
{
case 1:
DataRate = StringValue("DsssRate1Mbps");
std::cout << 1 << '\t';
break;

case 2:
DataRate = StringValue("DsssRate2Mbps");
std::cout<< 2 << '\t';
break;

case 5:
        DataRate = StringValue("DsssRate5_5Mbps");
        std::cout<< 5 << '\t';
break;

case 11:
        DataRate = StringValue("DsssRate11Mbps");
        std::cout<< 11 << '\t';
break;

default:
DataRate = StringValue("DsssRate11Mbps");
std::cout<< "Default" << '\t';
}

// Create access point
NodeContainer wifiApNode;
wifiApNode.Create (1);
std::cout << "Access point creation: Complete" << '\n';

// Create station nodes
NodeContainer wifiStaNodes;
wifiStaNodes.Create (Nodes);
std::cout << "Nodes creation: Complete" << '\n';
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.Set ("RxGain", DoubleValue (0) );

// Set Propagation Delay and Loss Models
YansWifiChannelHelper channel;
channel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
channel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel","Exponent",DoubleValue(3.0));
phy.SetChannel (channel.Create ());
WifiHelper wifi = WifiHelper::Default ();
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
std::cout << "Wifi 802.11b Channel configuration: Complete" << '\n';
switch(rsm)
{
case 'c':
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager","DataMode", DataRate, "ControlMode", Da
break;

case 'a':
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");
```

```
break;

default:
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager","DataMode", DataRate, "ControlMode", Da
}

// Configure MAC parameter
NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
std::cout << "Control rate configuration: Complete" << '\n';

// Configure SSID
Ssid ssid = Ssid ("Wifi");
mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid), "ActiveProbing", BooleanValue (false));
NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);
mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));
NetDeviceContainer apDevice;
apDevice = wifi.Install (phy, mac, wifiApNode);
std::cout << "SSID, ApDevice & StaDevice configuration: Complete" << '\n';

// Configure nodes mobility
MobilityHelper mobility;

// Constant Mobility for Access Point
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);

// Random Walk Mobility Model for Station nodes
mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
"Bounds", RectangleValue (Rectangle (-500, 500, -500, 500)),
"Distance", ns3::DoubleValue (100.0));
mobility.Install (wifiStaNodes);
std::cout << "Node mobility configuration: Complete" << '\n';

// Set up Internet stack
InternetStackHelper stack;
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);

// Configure IPv4 address
Ipv4AddressHelper address;
Ipv4Address addr;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer staNodesInterface;
Ipv4InterfaceContainer apNodeInterface;
staNodesInterface = address.Assign (staDevices);
apNodeInterface = address.Assign (apDevice);
addr = apNodeInterface.GetAddress(0);
std::cout << "Internet Stack & IPv4 address configuration: Complete" << '\n';

// Create traffic (UDP)
ApplicationContainer serverApp;
UdpServerHelper myServer (8001); //port 8001
serverApp = myServer.Install (wifiStaNodes.Get (0));
serverApp.Start (Seconds(StartTime));
serverApp.Stop (Seconds(StopTime));
```

```
UdpClientHelper myClient (apNodeInterface.GetAddress (0), 8001);
myClient.SetAttribute ("MaxPackets", UintegerValue (numPackets));
myClient.SetAttribute ("Interval", TimeValue (Time ("0.002"))); //packets/s
myClient.SetAttribute ("PacketSize", UintegerValue (packetSize));
ApplicationContainer clientApp = myClient.Install (wifiStaNodes.Get(0));
clientApp.Start (Seconds(StartTime));
clientApp.Stop (Seconds(StopTime+5));
std::cout << "UDP traffic generation: Complete" << '\n';

// Calculate Throughput & Delay using Flowmonitor
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();
Simulator::Stop (Seconds(StopTime+2));
Simulator::Run ();
monitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end ();
{
Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
std::cout << "Flow " << i->first << " (" << t.sourceAddress << " -> " << t.destinationAddress << ")\
std::cout << " Tx Bytes: " << i->second.txBytes << "\n";
std::cout << " Rx Bytes: " << i->second.rxBytes << "\n";
std::cout << " Average Throughput: " << i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSec
std::cout << " Delay : " << i->second.delaySum / i->second.rxPackets << "\n";
}
Simulator::Destroy ();
return 0;
}
```