ET4394

Wireless Networking

NS-3 Network simulation project

April 30, 2016

Authors: Renukaprasad Manjappa (4518497): r.b.manjaapa@student.tudelft.nl

Delft University of Technology

Contents

Chapter 1

Introduction

The 802.11 specification as a standard for wireless LANS was approved by the Institute of Electrical and Electronics Engineers (IEEE) in the year 1997. Like all IEEE 802 standards, the 802.11 standards focus on the bottom two levels the ISO model, the physical layer and the link layer. Any LAN application, network operating system, protocol, including TCP/IP, will run on an 802.11-compliant WLAN as similar to how they run on the Ethernet. 802.11 has a variety of standards, each with a letter suffix. Table 1.1 provides an overview of difference among various standards.

Table 1.1: Overview of 802.11 variants

	802.11a	802.11b	802.11g	802.11n
Year of origin	July 1999	July 1999	June 2003	October 2009
Maximum Data rate(Mbps)	54	11	54	600
Modulation	OFDM	CCK or DSSS	CCK, DSSS or OFDM	CCK, DSSS or OFDM
RF band(GHz)	5	2.4	2.4	2.4 or 5
Number of spatial streams	1	1	1	1,2,3 or 4
Channel Width(MHz)	20	20	20	20 or 40

1.0.1 802.11 DCF

The basic mechanism to access the medium in 802.11 is the Distributed Coordination Function (DCF) which is based on the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) protocol. In this protocol, whenever a node has data to send, it first senses the medium to determine if another node is transmitting. If the medium was idle for duration larger than DIFS (DCF Inter-Frame Space), it can send its data. If the medium was busy, the node must defer until the channel is idle again.

A period called the contention window or backoff window follows the DIFS. The window is slotted, stations can pick a random slot and wait for that slot before accessing the medium again. Initially the value of contention window is set to CWmin. The maximum value of CW is CWmax. After each successful transmission the sender resets its CW size to CWmin. The value of contention window doubles after each collision and saturates at CWmax. The contention window stays at its maximum value until the packet is dropped after certain amount of retries or

its successfully transmitted in which case its value is reset to CWmin. The exponential increase of contention window with increasing number of transmissions is shown in figure 1.1.

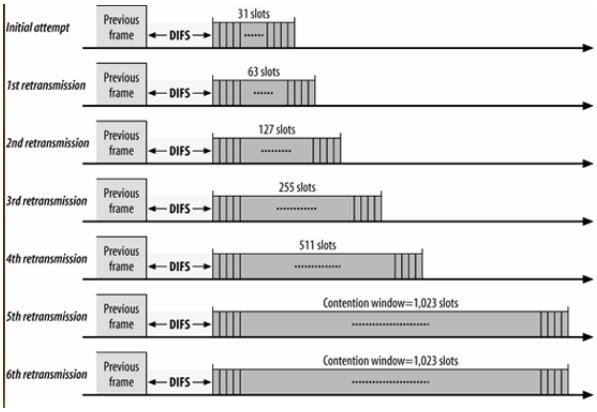

Figure 1.1: 802.11 contention window size

Chapter 2

Project description

2.0.1 Objectives

The DCF procedure is simple and used among all the different 802.11 variants. A lot of research has been done to analyze the performance of DCF and its reliability. Many research papers indicate the drawbacks of DCF and its dependency on parameters of the system under consideration. There have been numerous papers written on how the backoff procedure in DCF can be improved to provide increased performance.

This project attempts to recreate procedures mentioned in some of the research papers to simulate how the contention window of DCF affects the throughput in a wireless system containing a number of nodes. The simulations are done to incorporate methods suggested in papers. Simulations are also done to understand how different parameters such as packet loss percentage varies under different values of contention window.

2.0.2 Drawbacks of Binary exponential backoff (BEB)

In this section we highlight some of the drawbacks present in DCF as presented by the research papers such as [1]. We also discuss some of the methods proposed in [2] to improve the throughput and performance. The proposed methods are then simulated using NS3 and the results are compared.

The Binary exponential backoff scheme suffers from a fairness problem as some nodes can achieve significantly higher throughput than others. This fairness problem might occur due to the fact that scheme resets the contention window of a successful sender to CWmin after a successful transmission, while other nodes continue to maintain larger contention windows, thus reducing their chances of seizing the channel. The BEB does not take into account the previous collisions from the station. The size of the initial contention window is kept constant irrespective of the traffic activity and number of active stations present. Whereas it should be large when there are more number of active stations and small when there are less number of stations. The values of CWmin and CWmin can affect the performance of the system to a great extent.

Chapter 3

Implementation in NS-3

In this project the analysis of DCF is done using NS3 simulator. NS-3 is a discrete-event network simulator and is widely used by researchers and students to simulate various scenarios in networking domain. NS-3 is an open source software licensed under GPL and the users are provided with complete source which they are free to modify.

In this project simulations are done on an 802.11b network in infrastructure mode. It contains one access point and a variable number of nodes. In order to measure the effect of contention window on performance of the wifi network we need to be able to vary the contention window from the program. Upon doing some research in NS3 forums it was found that many people were having problems doing so. A solution was found in [3] that required certain small changes had to the NS-3 source code. In the file `src/wifi/model/dca-txop.cc` we add following additional attributes to `GetTypeId` method which will allow us to set the contention window from the program.

```
/* To set the Minimum value of contention window */
.AddAttribute ("CwMin",
                   "Sets the minimum contention window.",
                   UintegerValue (),
                   MakeUintegerAccessor (&DcaTxop::SetMinCw),
                   MakeUintegerChecker<uint32_t> ())

    /* To set the Maximum value of contention window */

     .AddAttribute ("CwMax",
                   "Sets the maximum contention window.",
                   UintegerValue (),
                   MakeUintegerAccessor (&DcaTxop::SetMaxCw),
                   MakeUintegerChecker<uint32_t> ())
```

Once these changes are implemented the value of contention window for the nodes can be set using

```
Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/
```

```
  $ns3::RegularWifiMac/DcaTxop/CwMin",UintegerValue(minCw));

Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/
$ns3::RegularWifiMac/DcaTxop/CwMax",UintegerValue(maxCw));
```

Following set of operations are done in the program to set up the wifi network in NS3 source code (wifi-cw.cc).

- We use `NodeContainer` class of NS-3 to create the nodes in our network for AP and STAs.
- wifi phy layer is setup using `YansWifiPhyHelper` class.
- To make simulations more realistic `RngSeedManager` is used to introduce randomness in the simulation based on time.
- `YansWifiChannelHelper` class is used to setup the wifi channel required for wifi communication.
- After setting up the channel its properties such as propagation delay and propagation loss models are set using `SetPropagationDelay()` and `AddPropagationLoss()` APIs
- `WifiHelper` is used to setup the required wifi standard on a large number of wifi net devices.
- The protocol version of the wifi is set using `SetStandard()` API and the appropriate rate control algorithm is setup using `SetRemoteStationManager` API.
- Wifi MAC layer is then configured using `NqosWifiMacHelper` class since we are using non-QoS (Quality of service) MAC layer.
- Mobility and positions of the different nodes in the network are then set using `MobilityHelper` class. There are several different types of mobility models available in NS-3 that can be used to describe how nodes move within the network.
- Internet Stack is installed on all nodes using `InternetStackHelper` class.
- IPV4 addresses are set for all the nodes in the network using `Ipv4AddressHelper` class.
- Applications that are used to generate traffic among nodes are installed in each of nodes. In this project we make use of `UdpServerHelper` and `UdpClientHelper` class to simulate UDP traffic between client and server nodes.
- Vital performance stats required to calculate throughput and delay from the simulation is gathered using `FlowMonitorHelper` class.

Following table summarizes the simulation parameters used in this project.

Wifi standard	802.11b
Propogation delay model	Constant speed propogation model
Propogation loss model	Friis Propagation Loss Model
Rate control algorithm	Arf
Mac Layer	Nqos mac
Mobility model	Random walk 2D model
Application	UDP client server application

Table 3.1: Simulation parameters used in the project

Chapter 4

Analysis of results

NS3 uses `waf` to compile and run the source code. `waf` is a build automation tool designed to assist in the automatic compilation and execution. It is written in Python. Shell scripts are written to simulate different scenarios of varying parameters and plot the obtained output using `gnuplot`. Gnuplot is a portable command-line driven graphing utility for Linux.
Simulation can be executed using following command.

```
./waf --run "scratch/wifi-cw --nNodes=5 --minCw=31 --maxCw=1023"
```

Here `nNodes` represents the number of nodes and `minCw` and `maxCw` represents minimum and maximum value of contention window respectively.

4.0.1 Effect of Contention window

We measure the effect of contention window on the throughput of the system. Simulations are performed for 5,10,20 and 30 stations by varying the value of CWmin and throughput is measured.

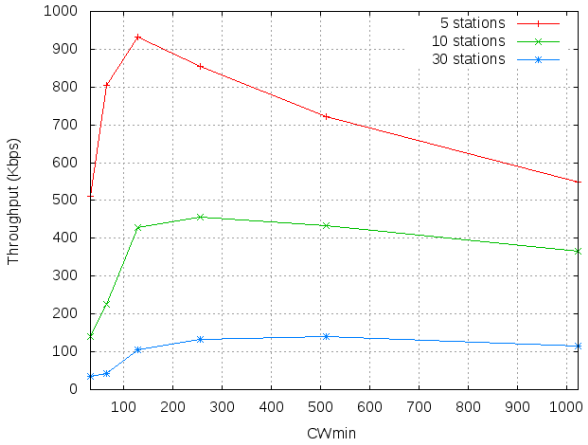

Figure 4.1: Throughput obtained for different number of stations for varying values of CWmin

The results obtained are shown in figure 4.1. Throughput obtained depends on the value of CWmin chosen and number of competing stations. It can be observed that throughput reduces when there are large number of competing stations. Throughput reaches a maximum value for certain CWmin and then gradually decreases or saturates. A low value of CWmin can be optimum for smaller amount of nodes but it might not be suitable for larger number of stations. At the same time a high value of Cwmin might be good for larger number of nodes but will decrease the performance if number of nodes is less. As shown in figure 4.2 for just 5 stations having small value of CWmin introduces lesser delay compared to having a large value of CWmin, same is not true for 30 stations which experience larger delay for smaller values of CWmin.

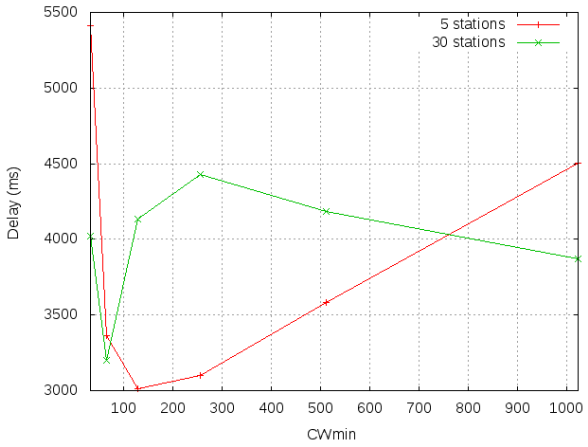

Figure 4.2: Delay obtained for different number of stations for varying values of CWmin

4.0.2 Optimal CW for 802.11

It has been observed that there exists an optimal value of CWmin for which maximum throughput is obtained. The number of collisions will increase if the value of CWmin less than the optimal one resulting in degraded throughput. Throughput will also decrease if the value of CWmin is increased above the optimal value due to long delays. This value depends on the number of active stations present in the system.

[2] [4] propose an adaptive contention window mechanism where the value of CWmin is dynamically changed based on number of nodes present in the system. [2] suggests that optimal contention window for a set of **N** nodes can be obtained by the equation 4.1.

$$CWmin = 8N - 6 \tag{4.1}$$

In order to validate adaptive contention window we compare throughput obtained by BEB for varying number of stations. It can be observed from the figure 4.3 that having adaptive contention window can improve the throughput significantly as compared to results obtained by BEB. Figure 4.4 shows that when adaptive contention window is used number of collisions decrease as a result packet loss percentage reduces which results in improved throughput. The paper [2] shows us that with adaptive contention window throughput remains constant irrespective of number of

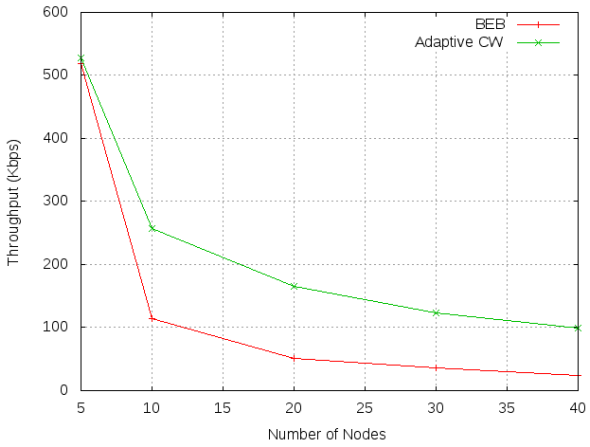

Figure 4.3: Comparison of throughput obtained for BEB and adaptive contention window

nodes in the system, this however was not observed in simulations. The throughput obtained still dropped when number of active stations increased.

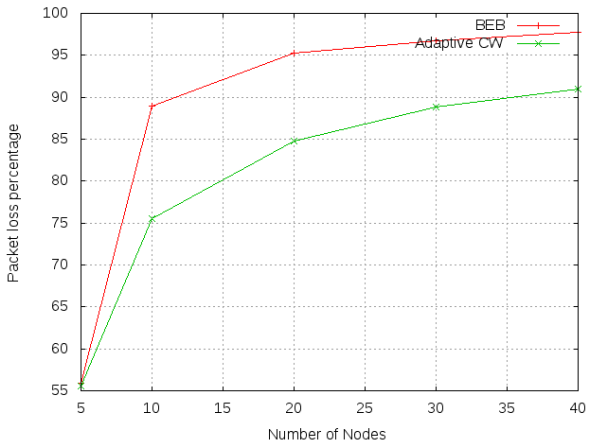

Figure 4.4: Comparison of packet loss percentage in BEB and adaptive contention window

4.0.3 Effect of rate control algorithms

Wireless medium is highly volatile in nature due to fading, attenuation, interference from other radiation sources. Mobility of nodes and moving objects also influence the throughput. In order to achieve a high performance under varying conditions, nodes need to dynamically adapt their transmission rates. Some of the popular rate control algorithms available are AAF, AARF, RBAR, CARA, RRAA. In this simulation we see the influence of arf, aarf in comparison with a constant data rate of 11Mbps.

From the figure 4.5 it can be seen that throughput obtained with constant rate is higher than

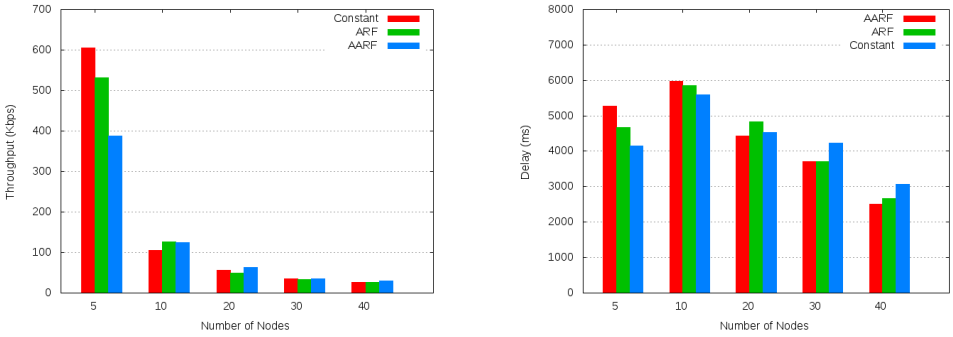

Figure 4.5: Comparison of throughput and delay for different rate control algorithms

the one obtained with rate control algorithms the delay experienced with use of constant rate is also higher. This is because of constant change in the rates in case of AARF and ARF where as rate remains constant at 11Mbps for the other one.

Chapter 5

Conclusion

Analysis of 802.11 DCF has shown many gray areas in the protocol. Especially the selection of contention window size and its effect on performance has been popular area of research.
There are several research papers out there that claim to outperform the traditional binary exponential backoff algorithm. In this project one such paper was selected and an attempt was made to recreate the procedure mentioned in the paper. When a comparison was made with the results obtained from the paper and the simulation results from NS-3, the results were not an exact match. Although there was some improvement in throughput it was no where near the claims made by the paper. This might raise questions about repeatability and reproducibility of the experiments conducted in research. This could also be because many parameters of the experiment were unknown and some of them very difficult to configure in NS-3.

Chapter 6

APPENDIX

```
/*
 # Wireless Networkng Assignment
 # Author Renukaprasad Manjappa
 */


#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/gnuplot.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("Adaptive contention window simulations");

/* Callbacks to trace number of collisions */

uint32_t MacTxDropCount, PhyTxDropCount, PhyRxDropCount;

void MacTxDrop(Ptr<const Packet> p)
{
    NS_LOG_INFO("Packet Drop");
    MacTxDropCount++;
}
```

```
void PrintDrop()
{
    //std::cout << Simulator::Now().GetSeconds() << "\t"
    //<< MacTxDropCount << "\t"<< PhyTxDropCount << "\t"
    //<< PhyRxDropCount << "\n";
    Simulator::Schedule(Seconds(5.0), &PrintDrop);
}

void PhyTxDrop(Ptr<const Packet> p)
{
    NS_LOG_INFO("Packet Drop");
    PhyTxDropCount++;
}

void PhyRxDrop(Ptr<const Packet> p)
{
    NS_LOG_INFO("Packet Drop");
    PhyRxDropCount++;
}


int main (int argc, char *argv[])
{
    double StartTime = 0.0;
    double StopTime = 15.0;
    int nNodes = 20; /*number of node*/
    int minCw = 0;
    int maxCw = 0;
    int arf = 1;
    int aarf = 0;
    int constant = 0;
    uint32_t payloadSize = 1472 /*payload 1472*/;
    uint32_t maxPacket = 1024 /*10000*/ ;
    int Enable_Adaptive_Cw = 0;


    StringValue DataRate;
    DataRate = StringValue("DsssRate11Mbps");
    // Create randomness based on time
    time_t timex;
    time(&timex);

    /* Introduce randomness in simulations */

    RngSeedManager::SetSeed(timex);
    RngSeedManager::SetRun(1);
    CommandLine cmd;
```

```
 cmd.AddValue ("nNodes", "number of nodes", nNodes);
 cmd.AddValue ("minCw", "Minimum Contention window size", minCw);
 cmd.AddValue ("maxCw", "Maximum contention window size", maxCw);
 cmd.AddValue ("arf", "arf rate control algorithm", arf);
 cmd.AddValue ("aarf", "aarf rate control algorithm", aarf);
 cmd.AddValue ("constant", "constant rate control ", constant);
 cmd.AddValue ("payloadSize", "offered load size", payloadSize);
 cmd.AddValue ("Enable_Adaptive_Cw", "Enable or disable Adaptive contention window",Enable_Ada
 cmd.Parse (argc,argv);

 // Create access point
 NodeContainer APNode;
 APNode.Create (1);
// std::cout << "Access point created.." << '\n';
 NS_LOG_INFO("Access point created..");
 // Create nodes
 NodeContainer StaNodes;
 StaNodes.Create (nNodes);
 NS_LOG_INFO("Nodes created..");
 YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
 phy.Set ("RxGain", DoubleValue (0) );
 YansWifiChannelHelper channel;
 channel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
 //channel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
 channel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel");
 phy.SetChannel (channel.Create ());
 WifiHelper wifi = WifiHelper::Default ();
 wifi.SetStandard (WIFI_PHY_STANDARD_80211b);


 NS_LOG_INFO("Wifi 802.11b PHY layer initilized");

 // configure MAC parameter

 if (constant) {
     wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager","DataMode", DataRate,
             "ControlMode", DataRate);
 }

 if (aarf) {
     wifi.SetRemoteStationManager ("ns3::AarfWifiManager");
 }
 if (arf) {
     wifi.SetRemoteStationManager ("ns3::ArfWifiManager");
 }
 NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
 NS_LOG_INFO("Control rate configured..");
```

```
    // configure SSID
    Ssid ssid = Ssid ("Wifi_AP");

    mac.SetType ("ns3::StaWifiMac",
            "Ssid", SsidValue (ssid),
            "ActiveProbing", BooleanValue (false));
    NetDeviceContainer staDevices;
    staDevices = wifi.Install (phy, mac, StaNodes);
    mac.SetType ("ns3::ApWifiMac",
            "Ssid", SsidValue (ssid));
    NetDeviceContainer apDevice;
    apDevice = wifi.Install (phy, mac, APNode);
    NS_LOG_INFO("SSID, ApDevice & StaDevice configured..");

    if(Enable_Adaptive_Cw) {

        if (!minCw && !maxCw ) {
        minCw = 8 * nNodes - 6;
        maxCw = 1024;
        }
        Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac/DcaTx
        Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac/DcaTx
    }
#if 0
    Config::Set ("/NodeList/*/DeviceList/*/Mac/DcaTxop/CwMin",UintegerValue (minCw));
    Config::Set ("/NodeList/*/DeviceList/*/Mac/DcaTxop/CwMax",UintegerValue (maxCw));
#endif



#if 0
    int minCw = 50;
    int maxCw = 1024;
    Ptr<DcaTxop> m_beaconDca = CreateObject<DcaTxop>();
    m_beaconDca->SetMinCw(minCw);
    m_beaconDca->SetMaxCw(maxCw);
    for (int i = 0 ; i < nNodes ; i++) {
        Ptr<Node> node = StaNodes.Get(i);
        Ptr<NetDevice> dev = node->GetDevice(0);
        Ptr<WifiNetDevice> wifi_dev = DynamicCast<WifiNetDevice>(dev);
        Ptr<WifiMac> mac = wifi_dev->GetMac();
        PointerValue ptr;
        mac->GetAttribute("DcaTxop", ptr);

        Ptr<DcaTxop> dca = ptr.Get<DcaTxop>();
        dca->SetMinCw(minCw);
        dca->SetMaxCw(maxCw);
```

```
    }

#endif
    // Configure nodes mobility
    MobilityHelper mobility;
    // Constant Position Node Mobility Model
    /* mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
       "MinX", DoubleValue (0.0),
       "MinY", DoubleValue (0.0),
       "DeltaX", DoubleValue (10.0),
       "DeltaY", DoubleValue (10.0),
       "GridWidth", UintegerValue (5),
       "LayoutType", StringValue ("RowFirst"));
    */
    // Constant Position Node Mobility Model
     //mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
    // Random Direction 2D Node Mobility Model
    #if 0
     mobility.SetMobilityModel ("ns3::RandomDirection2dMobilityModel",
       "Bounds", RectangleValue (Rectangle (-10, 10, -10, 10)),
       "Speed", StringValue ("ns3::ConstantRandomVariable[Constant=3]"),
       "Pause", StringValue ("ns3::ConstantRandomVariable[Constant=0.4]"));
#endif
    // Random Walk 2D Node Mobility Model
    mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
            "Bounds", RectangleValue (Rectangle (-1000, 1000, -1000, 1000)),
            "Distance", ns3::DoubleValue (300.0));
    mobility.Install (StaNodes);

    // Constant Mobility for Access Point
    mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
    mobility.Install (APNode);
    NS_LOG_INFO("Node mobility configured..");
    // Internet stack
    InternetStackHelper stack;
    stack.Install (APNode);
    stack.Install (StaNodes);
    // 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);
    for(int i = 0 ; i < nNodes; i++)
    {
        addr = staNodesInterface.GetAddress(i);
        //std::cout << " Node " << i+1 << "\t "<< "IP Address "<<addr << std::endl;
```

```
}
addr = apNodeInterface.GetAddress(0);

NS_LOG_INFO("Internet Stack & IPv4 address configured..");
// Create traffic generator (UDP)
ApplicationContainer serverApp;
UdpServerHelper myServer (4001); //port 4001
serverApp = myServer.Install (StaNodes.Get (0));
serverApp.Start (Seconds(StartTime));
serverApp.Stop (Seconds(StopTime));
UdpClientHelper myClient (apNodeInterface.GetAddress (0), 4001); //port 4001
myClient.SetAttribute ("MaxPackets", UintegerValue (maxPacket));
myClient.SetAttribute ("Interval", TimeValue (Time ("0.002"))); //packets/s
myClient.SetAttribute ("PacketSize", UintegerValue (payloadSize));

//ApplicationContainer clientApp = myClient.Install (StaNodes.Get (0));
ApplicationContainer clientApp = myClient.Install (StaNodes);

clientApp.Start (Seconds(StartTime));
clientApp.Stop (Seconds(StopTime+5));
// Calculate Throughput & Delay using Flowmonitor
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();




// Trace Collisions
Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/MacTxDrop", M
Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Phy/PhyRxDrop", M
Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Phy/PhyTxDrop", M

//Simulator::Schedule(Seconds(5.0), &PrintDrop);




Simulator::Stop (Seconds (100.0));

//Simulator::Stop (Seconds(StopTime+2));
Simulator::Run ();
monitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier (
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();


double TotalThroughput = 0.0;
double psent = 0.0;
double precv = 0.0;
```

```
    Time Delay = Seconds(0.0);
    int64_t Average_Delay = 0.0;
    for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.
    {
        //Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
        //std::cout << "Flow " << i->first << " (" << t.sourceAddress << " -> " << t.destinationA
        //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.timeLastRx
        //                                     i->second.timeFirstTxPacket.GetSeconds())/1024/
        double transmitTime = 0.0;
        transmitTime = (i->second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.Get
        if (transmitTime != 0)
            //TotalThroughput += i->second.rxBytes * 8.0 /transmitTime/1024/nNodes;
            TotalThroughput += i->second.rxBytes * 8.0 /transmitTime/1024;
        if (i->second.rxPackets != 0)
            Delay += (i->second.delaySum / i->second.rxPackets);
        psent +=  i->second.txPackets;
        precv += i->second.rxPackets;
    }

    Average_Delay = Delay.GetMilliSeconds();
    //std::cout << "Average Throughput obtained :" << TotalThroughput/nNodes << "\n";
    //std::cout << "Loss percentage : " << (100.0 * (psent - precv ))/psent << "\n";
    std::cout <<nNodes << "\t";
    std::cout <<TotalThroughput / nNodes << "\t";
    std::cout << (100.0 * (psent - precv )) / psent <<"\t";
    std::cout <<minCw<< "\t";
    std::cout <<Average_Delay/nNodes<< "\n";
    //std::cout << PhyRxDropCount << "\n";
    Simulator::Destroy ();
    return 0;
}
```

Bibliography

[1] D. P. A. Yunli Chen, "Effect of contention window on the performance of ieee 802.11 wlans."

[2] M. Elwathig Elhag, "Adaptive contention window schemes for wlans."

[3] "Ns-3 google groups." [Online]. Available: https://groups.google.com/forum/#!topic/ns-3-users/zhw7jbXNLUQ

[4] M. O. Giuseppe Bianchi, Luigi Fsatta, "Performance evaluation and enhancement of the csma/ca mac protocol for 802.11."