TU Delft Delft University of Technology

Simple Wifi Simulation Using ns-3

4503104

Bjarki Johannsson

b.johannsson@student.tudelft.nl

ET4394 Wireless Networking

Instructor: Przemyslaw Pawelczak

April 30, 2016

Contents

1 Introduction

The subject of this project is to use the *ns-3* [2] network simulator to explore aspects of the *IEEE 802.11b* wifi standard [1]. *ns-3* offers a wide range of simulation tools implemented in *C++* and is an open source project.
The source code for this project is available on GitHub at `https://github.com/bjohannsson/ET4394-ns3`. A bash shell script is provided in the repository to run the simulations and plot the results. Note that the project is written using ns-3 version 3.24 and the plots are generated with Python 2.7, matplotlib and numpy.
Section 2 describes the *distributed coordination function*(DCF) used by IEEE 802.11b. Section 3 describes the simulation where the DCF slot time is varied. Section 4 describes the simulation where the packet size is varied. Conlcusions are provided in section 5.

2 Distributed Coordination Function

IEEE 802.11b uses the DCF protocol to handle the scheduling of packets from possibly colliding sources. Figure 1 depicts how DCF works (with no RTS/CTS).

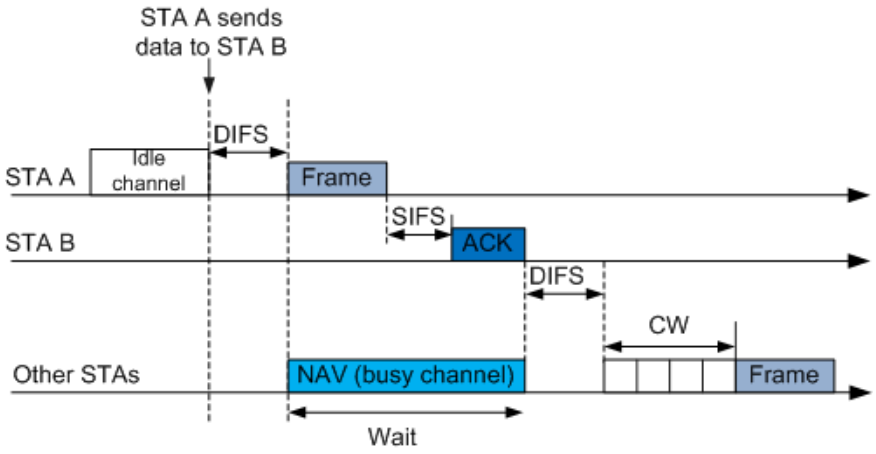

Figure 1: Illustration of the DCF behavior (source: support.huawei.com).

Before sending a package, a station must detect an idle channel for a duration of a DIFS. It can then proceed to send the package, and receives a ACK response upon successful reception of the package. In case two or more channels try to send a package at the same time after the channel is idle for a DIFS, no ACK response is sent back as the packages collide. The stations are now forced to back off for a period defined by the contention window (CF). The back-off time is slotted, and counts down only when the channel is idle, reducing the probability of further collision for packages.
If RTS/CTS is used, then stations that wish to send a packet must first send a *request-to-send*(RTS) message, and cannot proceed to send the data package until they receive a

clear-to-send(CTS) message. The RTS and CTS messages are relatively short, and RTS collision thus causes less delay than that of typical packages.

3 Simulation 1 - DCF Slot Time

3.1 Setup

A WLAN was setup using one access point and a number of wireless nodes(ranging from 1-10). The nodes each generate packets and send to the access point. In this simulation the slot time of the DCF is varied along with the number of nodes, and the throughput is gathered from the access point.
The relative DCF values for the IEEE 802.11b standard are shown in table 1.

Parameter	**Duration[μs]**
Slot time	20
SIFS time	10

Table 1: DCF values for IEEE 802.11b.

The values for the slot time were chosen arbitrarily as 12, 20, 25, 30 and 50 microseconds. A simulation was run for each of the slot times for each number of nodes, for a total of 50 runs. The results are shown in section 3.3.
This section shows the values and setup of the ns-3 script. Table 2 shows the ns-3 models used to simulate the network.

Item	**Value/Type**
Wifi channel	YansWifiChannel
PHY layer	YansWifiPhy
Wifi standard	802.11b
Data rate	11 Mbps
Remote station manager	ConstantRateWifiManager
MAC helper	QosWifiMacHelper
Mobility model	ConstantPositionMobilityModel
Server helper	UdpServerHelper
Client helper	UdpClientHelper

Table 2: Models and model parameter values used in the ns-3 script.

Table 3 shows the traffic generating parameters used for the UDP client/server model.

Item	Description	Value/Type
Packet size	Size of data packet	1472 bits
Max packets	Number of packets for each station	40000
Interval	Time between packets	15 μs

Table 3: Parameter values used in the ns-3 script.

The packet interval was assigned a low value in order to obtain constant trafficking of packages.
One access point was used as a server, which received packages from the wifi client nodes. The number of client nodes was varied from 1-10, and five simulations were run for each number of nodes. In the five runs the DCF slot time was set as 12, 20, 25, 30 and 50 microseconds. Each simulation was run for one second, as this allowed for plenty of traffic due to the short package interval.

3.2 Results

Figure 2 shows the throughput gathered from the access point against the varying values of the slot time and the number of nodes.

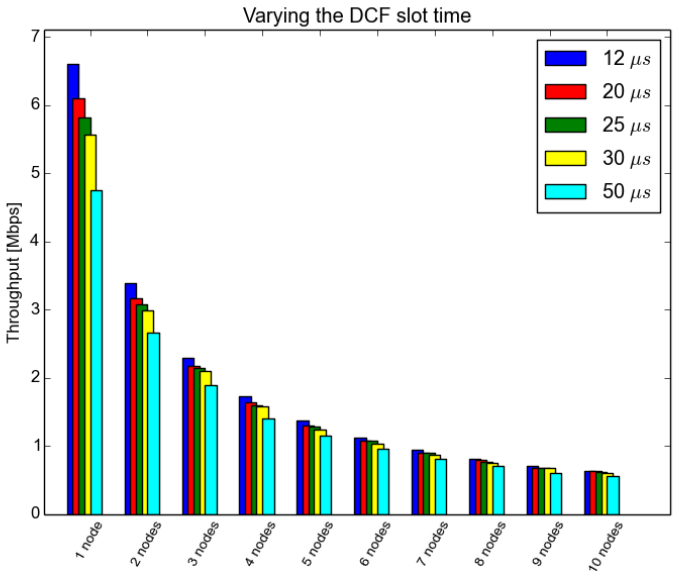

Figure 2: The throughput diminishes with increasing slot time and number of nodes.

This simulation was for observation only as the IEEE 802.11b slot time is fixed at 20 microsecond. This value is related to the time it takes to sense an ongoing transmission in the channel. The results from Figure 2 indicate however that it would be feasible to reduce the slot time if technological improvements allow.

The simulations do not incorporate any random factor. This was observed during development by performing repeated simulations using the same values.

4 Simulation 2 - Packet Size

4.1 Setup

In this simulation the payload was varied between values of 200, 400, 600, 800 and 1000 bytes per package. The number of nodes was varied from 1-10, for a total of 50 runs.
Apart from this the setup was the same as in simulation 1, with the exception that the number of packets was increased to 120,000 to allow saturated throughput with the smaller payload size.

4.2 Results

Figure 3 shows the measured throughput while varying the payload size.

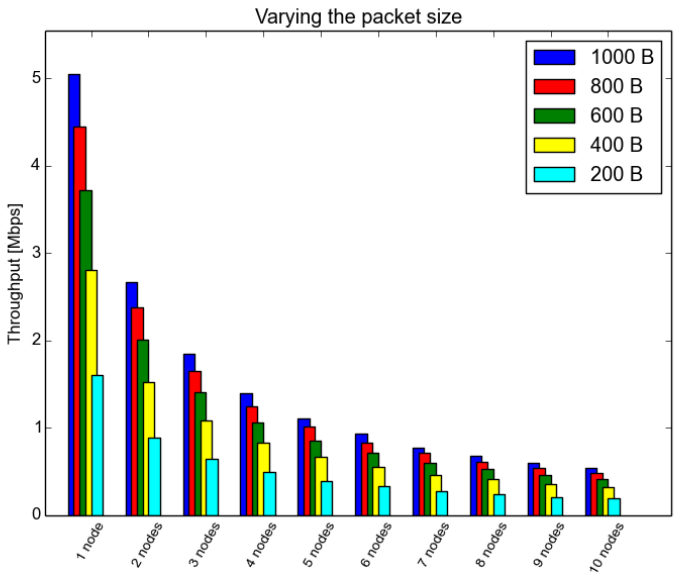

Figure 3: The throughput goes down as the payload is reduced and number of nodes increases.

It is evident from the graph that larger data packages result in higher throughput. This can be explained by the fact that for larger packages, more time is spent transferring data in the channel as the waiting times introduced by the DCF occur less frequently, i.e. the effects of the DCF overhead are reduced.

5 Conclusions

In the author's opinion the results from the simulations are not of significant value, although they provide some insight of the DCF behavior of the 802.11b standard. The value of this exercise comes from the skills learned while writing the scripts, and gathering the necessary knowledge to implement the simulations. The ns-3 simulator has a steep learning curve which is not surprising considering the vast functionality provided by it's models.

For curiosity's sake a small simulation was run with one node, where the slot time and payload size were varied using the same values as before. The throughput is shown in Figure 4. The figure clearly shows that the throughput benefits from short slot time and increased payload.

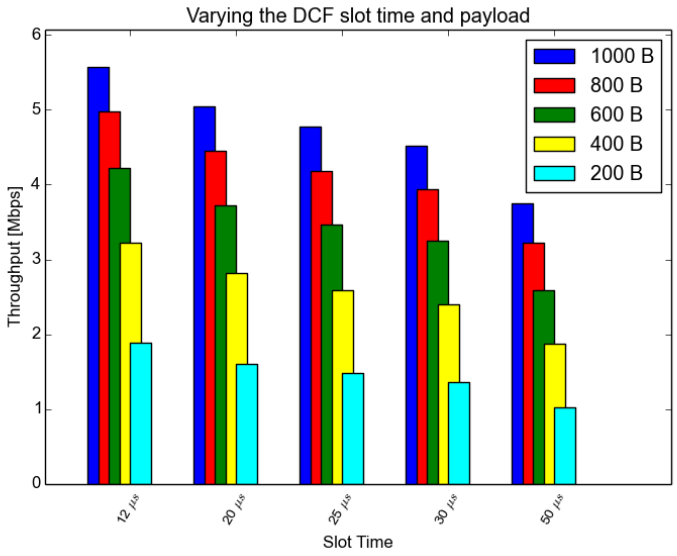

Figure 4: Slot time and payload varied.

References

[1] IEEE Std 802.11b-1999, http://standards.ieee.org/getieee802/download/802.11b-1999.pdf, accessed on 25-04-2016.

[2] NS-3 webpage, https://www.nsnam.org/.

A ns-3 Source Code

```
#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/object.h"
#include "ns3/uinteger.h"
#include "ns3/traced-value.h"
#include "ns3/trace-source-accessor.h"
#include "ns3/trace-helper.h"
#include "ns3/yans-wifi-helper.h"

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

using namespace std;
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("sp_trace");

int main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  // Number of nodes
  int nNodes = 1;
  uint32_t slot = 20; //slot time in microseconds
  uint32_t sifs = 10; //SIFS duration in microseconds

  uint32_t packetSize = 1472;
  uint32_t maxPacket = 120000;

  CommandLine cmd;
  cmd.AddValue ("n", "Number of nodes", nNodes);
  cmd.AddValue ("slot", "DCF slot time", slot);
  cmd.AddValue ("ps", "Packet size", packetSize);
  cmd.Parse (argc,argv);
```

```
  int32_t pifs = slot+sifs;
46 StringValue DataRate = StringValue("DsssRate11Mbps");

  // Set reference loss for 2.4 GHz
  Config::SetDefault ("ns3::LogDistancePropagationLossModel::ReferenceLoss",
   DoubleValue (40.046));

51 // Create AP
  NodeContainer wifiApNode;
  wifiApNode.Create(1);

  // Create nodes
56 NodeContainer wifiNodes;
  wifiNodes.Create(nNodes);

  // Create channel and phy
  YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
61 YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
  phy.SetChannel (wifiChannel.Create ());

  // Set 802.11b
  WifiHelper wifi = WifiHelper::Default ();
66 wifi.SetStandard ( WIFI_PHY_STANDARD_80211b);

  // Remote station
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode",
   DataRate, "ControlMode", DataRate);
  QosWifiMacHelper mac = QosWifiMacHelper::Default ();
71
  // Create wifi devices and set mac
  Ssid ssid = Ssid ( "bjarkiSsid");
  mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid), "ActiveProbing",
   BooleanValue (false));
  NetDeviceContainer wifiDevices = wifi.Install(phy, mac, wifiNodes);
76
  // Create AP device and set mac
  mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));
  NetDeviceContainer apDevice = wifi.Install (phy, mac, wifiApNode);

81 // Set DCF values
  Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/Slot",
   TimeValue (MicroSeconds (slot)));
  Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/Sifs",
   TimeValue (MicroSeconds (sifs)));
  Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/Pifs",
   TimeValue (MicroSeconds (pifs)));
```

```
86  // Mobility and position
    MobilityHelper mob;
    Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator>
      ();
    positionAlloc->Add (Vector (0.0, 0.0, 0.0));
    positionAlloc->Add (Vector (1.0, 0.0, 0.0));
91  positionAlloc->Add (Vector (0.0, 1.0, 0.0));
    positionAlloc->Add (Vector (1.0, 1.0, 0.0));
    positionAlloc->Add (Vector (2.0, 0.0, 0.0));
    positionAlloc->Add (Vector (0.0, 2.0, 0.0));
    positionAlloc->Add (Vector (2.0, 1.0, 0.0));
96  positionAlloc->Add (Vector (1.0, 2.0, 0.0));
    positionAlloc->Add (Vector (2.0, 2.0, 0.0));
    positionAlloc->Add (Vector (3.0, 1.0, 0.0));
    positionAlloc->Add (Vector (1.0, 3.0, 0.0));
    mob.SetPositionAllocator (positionAlloc);
101 mob.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

    mob.Install (wifiApNode);
    mob.Install (wifiNodes);

106 // Internet stack
    InternetStackHelper stack;
    stack.Install (wifiApNode);
    stack.Install (wifiNodes);

111 // IPv4
    Ipv4AddressHelper address;
    Ipv4Address addr;
    address.SetBase ("10.1.1.0", "255.255.255.0");
    Ipv4InterfaceContainer wifiNodesInterface;
116 Ipv4InterfaceContainer apNodeInterface;

    apNodeInterface = address.Assign (apDevice);
    wifiNodesInterface = address.Assign (wifiDevices);

121 // Confirm addresses
    // for (int i=0; i<nNodes; i++) {
    //   addr = wifiNodesInterface.GetAddress(i);
    //   cout << " Node " << i << " address: " << addr << endl;
    // }
126 // addr = apNodeInterface.GetAddress(0);
    // cout << " AP address: "<< addr << endl;

    // Simulation time
    double StartTime = 0.0;
131 double StopTime = 1;
```

```
LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);

// Server
UdpServerHelper myServer(4001);
ApplicationContainer serverApp = myServer.Install (wifiApNode.Get (0));;
serverApp.Start (Seconds(StartTime));
serverApp.Stop (Seconds(StopTime));

// Client
UdpClientHelper myClient (apNodeInterface.GetAddress (0), 4001);
myClient.SetAttribute ("MaxPackets", UintegerValue (maxPacket));
myClient.SetAttribute ("Interval", TimeValue (Time ("0.000015")));
myClient.SetAttribute ("PacketSize", UintegerValue (packetSize));

ApplicationContainer clientApp = myClient.Install (wifiNodes.Get (0));
for (int i = 1; i<nNodes; i++) {
  myClient.Install (wifiNodes.Get (i));
}

clientApp.Start (Seconds(StartTime));
clientApp.Stop (Seconds(StopTime));

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

// Flow monitor
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();


// Tracing
// AsciiTraceHelper ascii;
// phy.EnableAsciiAll(ascii.CreateFileStream("phy.tr"));

// Run simulation
cout << "C " << nNodes << " " << slot << " " << sifs << endl;
cout << "Starting simulation..." << endl << endl;

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


int tp = 0;
monitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.
  GetClassifier ());
map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
```

```
  for (map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i
    != stats.end (); ++i)
  {
    if (i->first > 0) {
      Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
      cout << i->first << " " << t.sourceAddress << " " << t.destinationAddress
   << " ";
      cout << i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSeconds()
    - i->second.timeFirstTxPacket.GetSeconds())/1024/nNodes << "kbps\n";
      tp = tp + i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSeconds
   () - i->second.timeFirstTxPacket.GetSeconds())/1024/nNodes;
    }
  }
  cout << "Total troughput: " << tp << " kbps" << endl;
  cout << "AVG: " << tp/nNodes << " kbps" << endl;

  cout << "Terminating simulation!" << endl;
  Simulator::Destroy ();


  // Get throughput from server
  uint32_t totalPackets = DynamicCast<UdpServer>(serverApp.Get (0))->GetReceived
    ();
  cout << totalPackets << endl;
  double throughput = totalPackets * packetSize * 8/((StopTime) * 1000000.0);
  cout << endl;
  cout << "V " << throughput/nNodes << " Mbit/s" << endl;

  return 0;
}
```

ET4394_ns3_Johannsson.cc