EE597
Project

Group:
Darshan Patil
USC-ID: 9575227834
patild@usc.edu

Vishal Guruprasad
USC-ID: 6388548976
vgurupra@usc.edu

Description:

Let τ be the probability that a station transmits in a generic slot time. As any transmission occurs when the backoff window is equal to zero, regardless of the backoff stage, it is

$$\tau = \sum_{i=0}^{m} b_{i,0} = \frac{b_{0,0}}{1-p} = \frac{2(1-2p)}{(1-2p)(W+1)+pW(1-(2p)^m)}$$

To finally compute the probability p that a transmitted packet collides, note that p is the probability that, in a time slot, at least one of the n-1 remaining stations transmits,
$p = 1-(1-\tau)^{n-1}$

Once τ is known, the probability P_{tr} that in a slot time there is at least one transmission, given n active stations, and the probability that a transmission is successful, are readily obtained as

$$P_{\mathrm{tr}} = 1-(1-\tau)^n$$

$$P_s = \frac{n\tau(1-\tau)^{n-1}}{P_{\mathrm{tr}}} = \frac{n\tau(1-\tau)^{n-1}}{1-(1-\tau)^n}$$

$$E[\Psi] = \frac{1}{P_{\mathrm{tr}}} - 1$$

We are finally in the condition to determine the normalized system throughput S, defined as the fraction of time the channel is used to successfully transmit payload bits. As the instants of time right after the end of a transmission are renewal points, it is sufficient to analyze a single renewal interval between two consecutive transmissions, and express as the ratio

$$S = \frac{E[\text{timeused for successful transm. in interval}]}{E[\text{length of a renewal interval}]} = \frac{P_s E[P]}{E[\Psi] + P_s T_s + (1-P_s)T_c}$$

$$\begin{cases} T_s^{\mathrm{bas}} = & H + E[P] + \mathrm{SIFS} + \delta + \mathrm{ACK} + \mathrm{DIFS} + \delta \\ T_c^{\mathrm{bas}} = & H + E[P^*] + \mathrm{DIFS} + \delta \end{cases}$$

Implementation on NS-3

To create a WifiNetDevice, users need to follow these steps:

1. Decide on which physical layer framework, the SpectrumWifiPhy or YansWifiPhy, to use. This will affect which Channel and Phy type to use. Here we use YansWifiPhy.

2. Configure the Channel: Channel takes care of getting signal from one device to other devices on the same Wi-Fi channel. The main configurations of WifiChannel are propagation loss model and propagation delay model.

3. Configure the WifiPhy: WifiPhy takes care of actually sending and receiving wireless signal from Channel. Here, WifiPhy decides whether each frame will be successfully decoded or not depending on the received signal strength and noise. Thus, the main configuration of WifiPhy is the error rate model, which is the one that actually calculates the probability of successfully decoding the frame based on the signal.

4. Configure WifiMac: this step is more on related to the architecture and device level. The users configure the wifi architecture (i.e. ad-hoc or ap-sta) and whether QoS (802.11e), HT (802.11n) and/or VHT (802.11ac) and/or HE (802.11ax) features are supported or not.
Create WifiDevice: at this step, users configure the desired wifi standard (e.g. 802.11b, 802.11g, 802.11a, 802.11n, 802.11ac or 802.11ax) and rate control algorithm. Create 'nSta' wireless stations and one AP, Create a channel helper and phy helper, and then create the channel with Default configuration inour case it is IEEE 802.11b

5. Input the value of cxmin: the minimum contention window to be used in our case, this describes the initial backoff window size by Bianchi minus 1. Default to be 31. This can be achieved using the command ./waf —-run “scratch/ns3-project —-cwmin=127”. Input the value of cxmax: the maximum contention window, it is computed by 2^m*W-1 by Bianchi. Default to be 255 (W = 32, m = 3). This can be achieved using the command ./waf —-run “scratch/ns3-project —-cwmax=1023”. Input the value of trafficRate: traffic rate(String) for each node with packet size. Default to be 1Mb/s. For example, type in ./waf —-run “scratch/ns3-project —-trafficRate=800kb/s”.

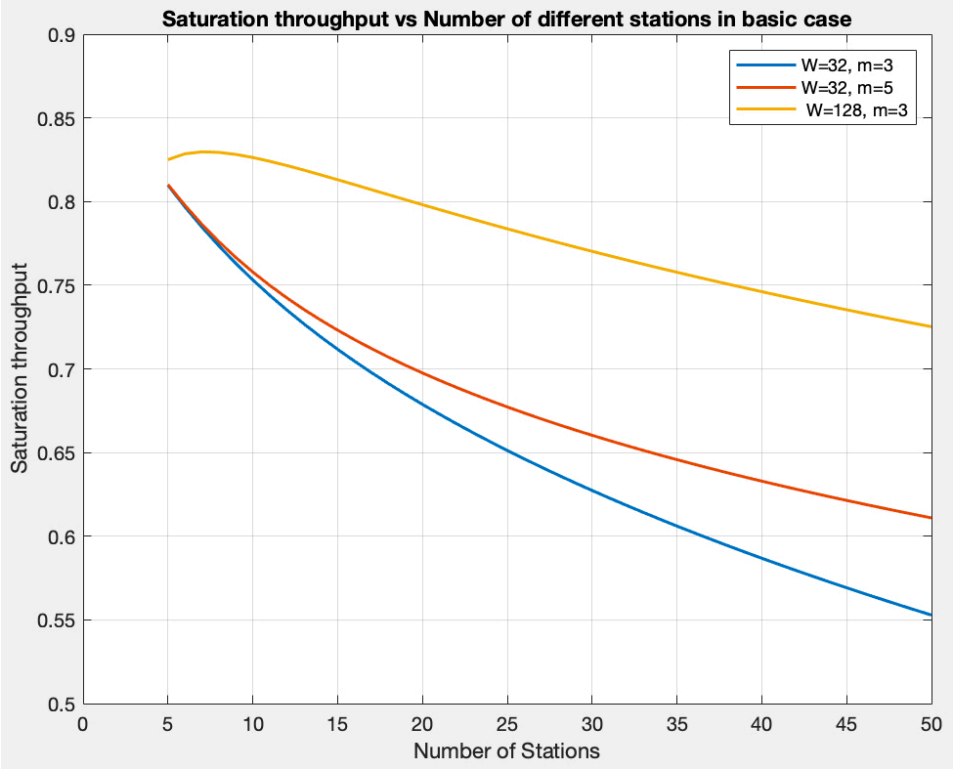
5. Configure mobility: finally, mobility model is (usually) required before WifiNetDevice can be used.

6. Set the default Simulation Time, Transmission Start Time, Transmission Stop Time. Then, Supervise the throughput to PacketSink each 0.1 sec till the end of Simulation Time and obtain the average actual throughput into ap.

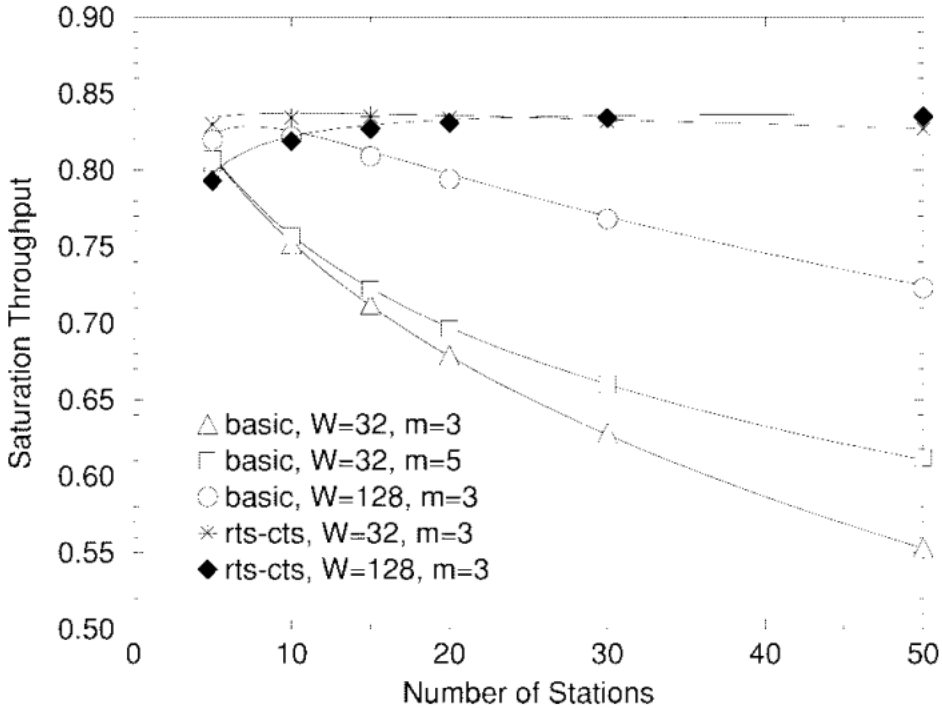
7. When we compare the output of MATLAB with that of ns-3 the values are matching closely which indicates that the simulation from Bianchi's Algorithm is closely matching the expected value of ns-3 computation which simulates the actual environment in software with all the necessary parameter.

Results:

Simulation Result



Bianchi's Algorithm Saturation Throughput



4. Saturation throughput: Analysis versus simulation.

Discussion of Result:

We observe that, the output of MATLAB with that of ns-3 the values are matching closely which indicates that the simulation from Bianchi's Algorithm is closely matching the expected value of ns-3 computation which simulates the actual environment in software with all the necessary parameter.

Matlab Code

```
% Calculations based on Bianchi's paper
clear; clc; close all;
% Set default parameters from the paper
SIFS = 28; DIFS = 128; slot_time = 50; prop_delay = 1; % in us (microseconds)
PayLoad = 8184; MAC_header = 272; PHY_header = 128; ack = 112; %in bits
H = MAC_header + PayLoad + PHY_header; %in bits
ACK = 112 + PHY_header;  %in bits
Ts = (H + SIFS + ACK + DIFS + 2 * prop_delay)/slot_time;
Tc = (H + DIFS + prop_delay)/slot_time;

% Set the window size and stage number
W=32;m=3;
% Compute
throughput=[];
itr=50;
for n=5:1:itr % Start from 5 stations condition
    fn = @(p)(p-1+(1-2*(1-2*p)/((1-2*p)*(W+1)+p*W*(1-(2*p)^m)))^(n-1));
    % P is the probability that transmitted packet collide
    P = fzero(fn,[0,1]);
    % tau is probability that a station transmits in a generic slot time
    tau = 2*(1-2*P)/((1-2*P)*(W+1)+P*W*(1-(2*P)^m));
    % Ptr is that in a slot time there is at least one transmission
    Ptr = 1-(1 - tau)^n;
    % Ps is the probability that a transmission is successful
    Ps = n*tau*(1-tau)^(n-1)/Ptr;
    % ETX is the number of consecutive idle slots between two consecutive
    transmissions on the channel
    E_Idle=1/Ptr-1;
    % Throughput = Ps * E[P] / (ETX + Ps * Ts + (1 - Ps) * Tc)
    S=Ps*(PayLoad/slot_time)/(E_Idle+Ps*Ts+(1-Ps)*Tc);
    throughput=[throughput,S];
end
plot(5:1:itr,throughput,'LineWidth',1.5);
hold on;
axis([0 itr 0.5 0.9]);
xlabel('Number of Stations');
ylabel('Saturation throughput');
title('Saturation throughput vs Number of different stations in basic case');
grid on;

%W=32 m=5
W=32;
m=5;
throughput = [];
for n = 5:1:itr
    fn = @(p)(p-1+(1-2*(1-2*p)/((1-2*p)*(W+1)+p*W*(1-(2*p)^m)))^(n-1));
    P = fzero(fn,[0,1]);
    tau = 2*(1-2*P)/((1-2*P)*(W+1)+P*W*(1-(2*P)^m));
    Ptr = 1-(1 - tau)^n;
```

```
    Ps = n*tau*(1-tau)^(n-1)/Ptr;
    E_Idle=1/Ptr-1;
    S=Ps*(PayLoad/slot_time)/(E_Idle+Ps*Ts+(1-Ps)*Tc);
    throughput=[throughput,S];
end
plot(5:1:itr,throughput,'LineWidth',1.5);
hold on;

%W=128, m=3
W=128;
m=3;
throughput = [];
for n = 5:1:itr
    fn = @(p)(p-1+(1-2*(1-2*p)/((1-2*p)*(W+1)+p*W*(1-(2*p)^m)))^(n-1));
    P = fzero(fn,[0,1]);
    tau = 2*(1-2*P)/((1-2*P)*(W+1)+P*W*(1-(2*P)^m));
    Ptr = 1-(1 - tau)^n;
    Ps = n*tau*(1-tau)^(n-1)/Ptr;
    E_Idle=1/Ptr-1;
    S=Ps*(PayLoad/slot_time)/(E_Idle+Ps*Ts+(1-Ps)*Tc);
    throughput=[throughput,S];
end
plot(5:1:itr,throughput,'LineWidth',1.5);
legend('W=32, m=3','W=32, m=5',' W=128, m=3');
hold off;
```

NS-3 Code

```
//Define the necessary Libraries for the project
#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/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include <cmath>
#define PI 3.14159265
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("Project1-NS3");//To define the Logging component for the file
double SimTime = 15.0;//Simulation Time
double txStartTime = 0.1;//Transmission Start Time
double txStopTime = 9.0;//Transmission Stop Time
void
get_throughput(Ptr<PacketSink> sinkApp)
{
// Supervising the the throughput to PacketSink each 0.1 sec
Time t = Simulator::Now();//Start of Simulation
std::cout << "current time: \t" << t.GetSeconds() << std::endl;
double bytesTotal = sinkApp->GetTotalRx();
float throughput = (bytesTotal*8.0)/1000/t.GetSeconds();
std::cout << "Throughput: " << throughput << std::endl;
//reschedule per 0.1 second
```

```
if(t.GetSeconds() < SimTime)
{
Simulator::Schedule(Seconds(0.1), &get_throughput, sinkApp);
}}
int
main(int argc, char *argv[])
{
  uint32_t nSta = 5 ;
  uint32_t cwmin = 1;
  uint32_t cwmax = 1023;
  bool trace = false;

  //add command line parameters for enabling or disabling logging components
  //also for changing number of devices
  CommandLine cmd;
  cmd.AddValue ("nSta", "Number of wifi STA devices", nSta);
  cmd.AddValue ("cwmin", "Minimum contention window size", cwmin);
  cmd.AddValue ("cwmax", "Maximum contention window size", cwmax);
  cmd.AddValue ("trace", "supervising the throughput each 0.1 second", trace);
  cmd.Parse (argc, argv);

  Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold", StringValue ("2200"));
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue ("2200"));
  Config::SetDefault("ns3::DcaTxop::MinCw", UintegerValue (cwmin));
  Config::SetDefault("ns3::DcaTxop::MaxCw", UintegerValue (cwmax));
  //Config::Set("/NodeList/0/DeviceList/0/Mac/DcaTxop/MinCw", UintegerValue (cwmin));

  //----------------------------------------------------------------------------------------------------------------//
  //create 'nSta' wireless stations and one AP
  NodeContainer wifiStaNodes;
  wifiStaNodes.Create (nSta);//create nSta node objects
  NodeContainer wifiApNode;
  wifiApNode.Create (1);//create 1 access point node objects
 //----------------------------------------------------------------------------------------------------------------//

// Create a channel helper and phy helper, and then create the channel
  YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
  //Speed : 500000 m/s
  //channel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel","Speed",StringValue("500000"));
  YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
  //make sure all PHY layer objects created by Yan, complete Phy helper configuration
  phy.SetChannel (channel.Create ());
 //----------------------------------------------------------------------------------------------------------------//
// Create a WifiHelper, which will use the above helpers to create
// and install Wifi devices.  Configure a Wifi standard to use, which
// will align various parameters in the Phy and Mac to standard defaults.
  WifiHelper wifi;
  //wifi.EnableLogComponents();
  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);//Setting standard according to IEEE 802.11b
  StringValue phymode = StringValue ("DsssRate1Mbps");//adding data rate to PHY Layer
                        wifi.SetRemoteStationManager                        ("ns3::ConstantRateWifiManager","DataMode",
phymode,"ControlMode",phymode);
```

```
//-----------------------------------------------------------------------------------------------------------------//
//Next, configure type of MAC, SSID of infrastructure network
NqosWifiMacHelper macSta, macAp;
Ssid ssid = Ssid ("project-1");
macSta.SetType ("ns3::StaWifiMac","Ssid", SsidValue (ssid),
                "Sifs", TimeValue(MicroSeconds(28)),
                "Slot", TimeValue(MicroSeconds(50)));
//After configuring MAC and PHY, now invoke Install to create wifi devices
NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, macSta, wifiStaNodes);

//configure AP node, changing default Attributes of NqosWifiMacHelper
macAp.SetType ("ns3::ApWifiMac","Ssid", SsidValue (ssid),
                "Sifs", TimeValue(MicroSeconds(28)),
                "Slot", TimeValue(MicroSeconds(50)));
NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, macAp, wifiApNode);
Config::SetDefault ("ns3::WifiRemoteStationManager::MaxSsrc", StringValue("10000"));
Config::SetDefault ("ns3::WifiRemoteStationManager::MaxSlrc", StringValue("10000"));
//Set constant position of stations & ap
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
//locate AP at the center with position (0, 0, 0)
positionAlloc->Add (Vector (0,0,0));
//locate N stations as a circle with a center at AP
//for N stations, set rho as radius and theta
//then calculate position of each station
float rho = 0.5;
for (uint32_t i = 0; i < nSta; i++)
{
  double theta = i * 2 * PI / nSta;
  positionAlloc->Add (Vector (rho * cos(theta), rho * sin(theta), 0.0));
}
mobility.SetPositionAllocator (positionAlloc);
//tell MobilityHelper to install the mobility models on STA nodes
mobility.Install (wifiApNode);
mobility.Install (wifiStaNodes);
//intall network protocol stacks
InternetStackHelper stack;
stack.Install(wifiApNode);
stack.Install(wifiStaNodes);
//assign Ipv4 address, use network "10.1.1.0"
Ipv4AddressHelper address;
address.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer ApInterface = address.Assign(apDevices);
Ipv4InterfaceContainer StaInterface = address.Assign(staDevices);
//station application, seting UDPSocketFactory as protocol, DataRate 5Mb/s, packtSize 512 bytes
//these stations begin transmitting data
for (uint32_t i = 0; i < nSta; i++)
{
OnOffHelper onoff("ns3::UdpSocketFactory", Address(InetSocketAddress(ApInterface.GetAddress(0), 9)));
onoff.SetConstantRate(DataRate ("800kb/s"), uint32_t(512));
```

```
 ApplicationContainer temp = onoff.Install(wifiStaNodes.Get (i));
 temp.Start(Seconds(txStartTime));
 temp.Stop(Seconds(txStopTime));
}
 //similarly, set AP(receiver) application speed, with its protocol, IP address and Port Number
 //and begin receiving data
 PacketSinkHelper sink ("ns3::UdpSocketFactory",
  Address(InetSocketAddress(ApInterface.GetAddress(0), 9)));
 ApplicationContainer Serverapp = sink.Install(wifiApNode.Get (0));
 Serverapp.Start(Seconds (0.0));
 //enable network routing
 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
 //run simulator and give throughput
 Ptr<PacketSink> sinkApp = DynamicCast<PacketSink> (Serverapp.Get (0));
 FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll ();
 //configure tracing for each 0.1 seconds
 if (trace == true)
 {
  Simulator::Schedule(Seconds(0.1), &get_throughput, sinkApp);
 }
 Simulator::Stop(Seconds (SimTime));
 Simulator::Run();

 //monitor->CheckForLostPackets ();
 //Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier());
 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
 double lastRxTime = txStopTime;
 double firstRxTime = SimTime;

 for(std::map<FlowId, FlowMonitor::FlowStats>::const_iterator set = stats.begin(); set != stats.end(); set++)
 {
  if(lastRxTime < set->second.timeLastRxPacket.GetSeconds())
  {
   lastRxTime = set->second.timeLastRxPacket.GetSeconds();
  }
  if(firstRxTime > set->second.timeFirstRxPacket.GetSeconds())
  {
   firstRxTime = set->second.timeFirstRxPacket.GetSeconds();
  }
 }
 double totalBytes = sinkApp->GetTotalRx();
 float throughput = totalBytes * 8.0/1000/(lastRxTime - firstRxTime);
 std::cout << "cwmin: " << cwmin << ", cwmax: " << cwmax << ", nSta: " << nSta << std::endl;
 std::cout << "firstRxTime: " << firstRxTime << "sec,\t lastRxTime: " << lastRxTime << "sec" << std::endl;
 std::cout << "throughput:\t" << throughput << "kbps" << std::endl;

 Simulator::Destroy ();
}
```