Ns3 Project

Martin Borek & Miguel Gordo

March 14, 2016

This document is our report with the solutions to the Ns3 project for the course Advanced Internetworking II.

This project has been coded in Python, and before continuing we want to make a small explanation on one of the hypothesis for this project. In the project statement, it was specified that workstations are always backlogged and have traffic to send. In C++, this is achieved with the class `OnoffHelper`, which accepts as attributes two RandomVariable objects, one for ”OnTime” and the other for ”OffTime”. These random variables are set via `RandomVariableAttribute` objects, which are not currently supported in Python after extensive searches in the documentation and in the libraries via the `help` statement in Python.

Therefore we make sure the stations always have traffic to send by setting the time interval between packets to 0.1 s with the help of `UdpEchoServer` and `UpdEchoClient` classes. Also, we used the data rate 1 sMbps and the packet size 2000Ḃ. The `UdpEchoServer` start sending packets at the simulation time 1 s and the `UpdEchoClient` start receiving packets at the simulation time 2 s.

1 Excercise 1

1.1 Problem 1a

For the first problem, we set 4 stations and their locations according to the diagram depicted in the assignment. In order for the stations to be able to sense only traffic as in the diagram, we changed the transmission range of the WiFi channel.

With the current settings for our script, a overheard that b sent 1980 ACK frames to B in total. That is 10 frames each second. Even though all traffic sent from b was overheard, a cannot know how many data frames are sent by B since b does not need to reply to each received frame (even though it does in our case). Moreover, B could be sending frames also to other nodes that are not part of the presented topology (in a real world situation).

1.2 Problem 1b

Again, with our settings, B and A send the same number of data frames. Each sends 1980 frames, that is 10 frames per second. No interference is observed.

1.3 Problem 1c

In this problem B sends only 990 data frames; 5 per second. That is only a half of its intended frames (990 down from 1980). A instead sends 1975 frames, that is almost 10 data frames per second. The main cause of this phenomenon is that a cannot sense B, and therefore it will sense the channel idle more often than not. Host b will sense the channel busy, be it because of host a or because B is transmitting. As B wants to send a packet to b, it send a RTS. Since b at the same time senses traffic directed to a, it cannot send a CTS. Thus, B keeps sending RTS and the time between RTS keeps increasing. With every packet that B wants to send the initial contention window is approximately 0.0015 s and it increases up to 0.02 s before b sends the CTS. For the packets sent from a to A, the CTS packets are sent immediately after the first RTS. Thus, we could not measure the contention window. However, it should be approximately 0.0015 s as that is the value for the first RTS for the station B.

2 Problem 2

B sends in the simulation only 63 packets to b. A sends 5573 packets whereas C sends 5542.

The main cause of this phenomenon is that A and C cannot hear each other. If all nodes could sense each other, the competition for airtime would be fair. As they do not sense the further neighbor, they keep sending more packets, resulting in starving the B neighbor. B will never or almost never have the chance to send packets since A and C never backoff as long as they sense the channel idle.

The main difference from B's point of view in figures 4 and 6 is that in Figure 4 he will see his connection effectively halved, whereas in Figure 6 his connection will be non-existent because A and C cannot hear each other, and this in turn starves B completely. With only one neighbor B cannot be completely starved.

A Code for Problem 1C

```python
import ns.core
import ns.network
import ns.point_to_point
import ns.applications
import ns.wifi
import ns.mobility
import ns.csma
import ns.internet
import sys

# Enabling rts/cts
ns.core.Config.SetDefault("ns3::WifiRemoteStationManager::
    RtsCtsThreshold", ns.core.StringValue("0"))

# Setting logging level for echos
ns.core.LogComponentEnable("UdpEchoClientApplication", ns.core.
    LOG_LEVEL_INFO)
ns.core.LogComponentEnable("UdpEchoServerApplication", ns.core.
    LOG_LEVEL_INFO)

# Created stations
wifiNodes = ns.network.NodeContainer()
wifiNodes.Create(4)
apA = wifiNodes.Get(0)
ha = wifiNodes.Get(1)
apB = wifiNodes.Get(3)
hb = wifiNodes.Get(2)

# One WiFi channel that is used for all stations so that we can
    observe collissions
channelA = ns.wifi.YansWifiChannelHelper.Default()
channelA.SetPropagationDelay("ns3::
    ConstantSpeedPropagationDelayModel")
# Ranges are set so that stations can hear each other only as
    depicted in the topology in the assignment
channelA.AddPropagationLoss("ns3::RangePropagationLossModel", "
    MaxRange", ns.core.DoubleValue (3.0))

# Creating a phy for the wifi
phyA = ns.wifi.YansWifiPhyHelper.Default()
phyA.Set("ChannelNumber", ns.core.StringValue('1'))
phyA.SetChannel(channelA.Create())

# Creating a wifi with the previously created phy
wifiA = ns.wifi.WifiHelper.Default()
wifiA.SetStandard(ns.wifi.WIFI_PHY_STANDARD_80211b)
# The bandwidth set to 1Mbps
wifiA.SetRemoteStationManager("ns3::ConstantRateWifiManager", "
    DataMode", ns.core.StringValue("DsssRate1Mbps"), "ControlMode",
    ns.core.StringValue("DsssRate1Mbps"))


# Creating default mac and together with phy and wifi applying it
    to all wifi Stations
macA = ns.wifi.NqosWifiMacHelper.Default()
wifiDevices = wifiA.Install(phyA, macA, wifiNodes)

# Setting locations of all stations so that they correspond to the
    depicted topology.
# This is necessary together with the transmission ranges set
```

```
    earlier so that not all stations can hear each other
mobility = ns.mobility.MobilityHelper()
mobility.SetPositionAllocator ("ns3::GridPositionAllocator", "MinX"
    , ns.core.DoubleValue(0.0), "MinY", ns.core.DoubleValue (0.0),
    "DeltaX", ns.core.DoubleValue(2.0), "DeltaY", ns.core.
    DoubleValue(0.0), "GridWidth", ns.core.UintegerValue(4), "
    LayoutType", ns.core.StringValue("RowFirst"))

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel")
mobility.Install(wifiNodes)


# set internet stack for all stations
stack = ns.internet.InternetStackHelper()
stack.Install(wifiNodes)


# assign addresses to all stations
# addresses would be 10.1.3.1 for AP A, 10.1.3.2 for host a,
    10.1.3.3 for host b and 10.1.3.4 for AP B
address = ns.internet.Ipv4AddressHelper()
address.SetBase(ns.network.Ipv4Address("10.1.3.0"), ns.network.
    Ipv4Mask("255.255.255.0"))
wifiInterfaces = address.Assign(wifiDevices)

# configuring echo server and client to start sending / receiving
    at the second 1 / 2 of the simulation time,
# the interval between messages is 0.1s, packet size 2000B and the
    amount of packets is "unlimited"
# echo server is host b
# echo client is Access Point B
echoServer = ns.applications.UdpEchoServerHelper(9)
serverApps = echoServer.Install(hb)
serverApps.Start(ns.core.Seconds(1.0))
serverApps.Stop(ns.core.Seconds(200.0))

echoClient = ns.applications.UdpEchoClientHelper(wifiInterfaces.
    GetAddress(2), 9)
echoClient.SetAttribute("MaxPackets", ns.core.UintegerValue
    (99999999))
echoClient.SetAttribute("Interval", ns.core.TimeValue(ns.core.
    Seconds(0.1)))
echoClient.SetAttribute("PacketSize", ns.core.UintegerValue(2000))

clientApps = echoClient.Install(apB)
clientApps.Start(ns.core.Seconds(2.0))
clientApps.Stop(ns.core.Seconds(200.0))

# configuring echo server and client to start sending / receiving
    at the second 1 / 2 of the simulation time,
# the interval between messages is 0.1s, packet size 2000B and the
    amount of packets is "unlimited"
# echo server is Access Point A
# echo client is host a
echoServer = ns.applications.UdpEchoServerHelper(9)
serverApps = echoServer.Install(apA)
serverApps.Start(ns.core.Seconds(1.0))
serverApps.Stop(ns.core.Seconds(200.0))

echoClient = ns.applications.UdpEchoClientHelper(wifiInterfaces.
    GetAddress(0), 9)
echoClient.SetAttribute("MaxPackets", ns.core.UintegerValue
    (99999999))
echoClient.SetAttribute("Interval", ns.core.TimeValue(ns.core.
    Seconds(0.1)))
echoClient.SetAttribute("PacketSize", ns.core.UintegerValue(2000))

clientApps = echoClient.Install(ha)
clientApps.Start(ns.core.Seconds(2.0))
clientApps.Stop(ns.core.Seconds(200.0))



ns.internet.Ipv4GlobalRoutingHelper.PopulateRoutingTables()

ns.core.Simulator.Stop(ns.core.Seconds(200.0))

# enabling pcap traces
phyA.EnablePcapAll("problem1c")

# actual simulation
ns.core.Simulator.Run()
ns.core.Simulator.Destroy()
```