# ENSC 427: COMMUNICATION NETWORKS
Spring 2019
Project Report
Project title: WiFi Network Design for Public Transportation System
URL: http://www.sfu.ca/~wongyauw/

Group #11 members:
Wong Yau Shing, 301186946 wongyauw@sfu.ca
Junchen (Steven) Wang, 301270404 junchenw@sfu.ca
Wang Xi, 301253334 axi@sfu.ca

Last Update:
14th April, 2019

## Abstract

The purpose of this project is providing Wireless Network service at public transport station. Citizens can be able to use wifi when waiting for the public transport through the public hotspot. In order to spread up the cover area of the hotspot service, setting up multiple APs in the terminal station is necessary. In our project, we are going to simulate the performance of hotspot network service, interacting with Access Points, packets switching through routers. And it will also include the PoC and the data transfer simulation capturing to check the data loss, throughput etc.

# Table of Content

## List of Figures

## List of Tables

# 1. Introduction

## 1.1 Background

In the modern day and age, finding a good quality hotspot is as simple as doing a web search. Some Internet provider may also provide hotspots in public area, like ShawOpen and Telus hotspot. A WiFi hotspot aims to provide a physical address to users so that they can access to the Internet with their devices outside their home. In the past 10 years, these hotspots services have become very popular in different fields, such as coffee shops, mall, hotel and airport. In some cities, people can also use hotspots even in public station and parks. This makes wireless access becomes very convenient, as long as people can find a security hotspot that gives them a strong enough connection signal. This will be a great innovation for people cutting their free time when waiting the bus.

There are mainly two ways to get a WiFi access from a hotspot in public. The first one is a free WiFi provided by merchants, such as mall, restaurants or hotel wireless service for their guests. Of course, some of them are not completely free. These vendors may limit the bandwidth of each user, which will slow down the visiting speed. The second way is the Portable WiFi hotspot. If people want to use a hotspot anywhere they expect, they can bring a portable WiFi hotspot. A portable WiFi hotspot allows people to access to the Internet through a build-in mobile router which can be used to connect several devices at the same time without downloading any additional software. However, people need to carry an extra device that needs to be recharged which reducing the convenience. Still, portable WiFi will ensure that people can get a network connection where public hotspots are not available.

## 1.2 Scope

This document entails introduces our ideas on how to implement this project. It will include general overview, simulation topology and simulation result.

General overview: there are our sample use case, project assumptions and project constraints.

Simulation topology: it describes the basic parameters and setting about the simulation of the hotspot network.

Simulation result: there are basically three parts. The first part uses a simple wifi network with two access points as a proof of concept. For second part, we increase the number of users to test for the hotspot network performance. For third part, it introduce some methods and solution to improve packet loss issue.

# 2. Main Sections

## 2.1 General Overview & Design Guidelines

The whole project will study the performance of the bus hotspot using multiple routers. Firstly, we should prove the single router can work and measure the performance within certain distance. Then, we will build more routers within this distance. The performance of packet transferring will be evaluated as well as routers numbers increased within this distance. The routers will put in the shape of the line. After this, we will fix the number of routers and study how the distance between locations of the routers can affect the wifi performance within this certain area.

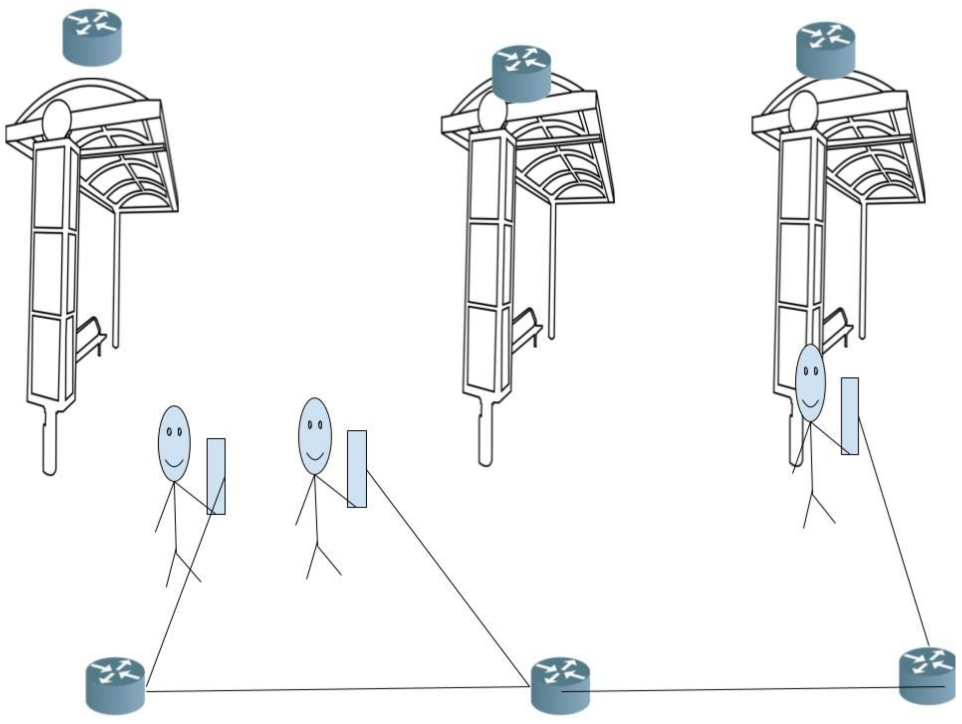

*Figure 2.1 System Overview*

## 2.2 Use Case

Figure 2.2 depicts a typical use case of and describes the flow of information between the user mobile devices and the underlying internals of the network.

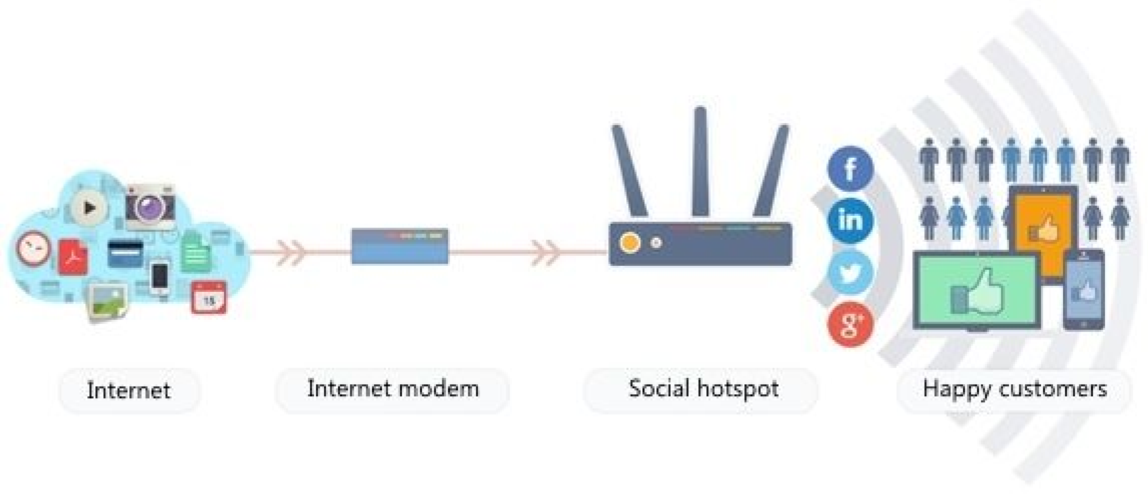

*Figure 2.2: Scenario of user browsing social media through hotspot*

## 2.3 Assumptions

The following are all the assumptions of the simulations.

- The routers are placed in an open area and there is no disruption between user’s mobile devices and the routers.
- The signal of the hotspot is evenly distributed.
- The routers will be put in the center line,with same distance when the number of routers is added.
- A pedestrian will be walking away from the center of the bus station with a constant speed.

## 2.4 Constraints

The following are all the constraints of the simulations.

- The effect of temperatures of the router can be simulated.

# 3. Simulation Topology

The main idea of this project is to simulate the performance of bus station hotshot system within a given area network topology based on the muti-ap scenario. The scenario is composed by a router with AccessPoints, for which the number of AP would depend on the category of the bus station (station or terminal). The AP would provide data transferring service through nodes and which is where users are able to access it. The entire system is connecting to the city network service.

In the simulations below the nodes represents internet users.

There are three simulations in total. The first simulation gives a brief idea of the bus station network with several nodes, representing users and access points. The first simulation aims at showing the basic idea works.

The second simulation is to study the general performance of the network design system as more users going to join the system. In this part of the simulation, tcp ip is specialized. The data rate is set to be constant as 5.5 Mbps and users are in the position, limited 60 meters from the stational router. The main data focused in this simulation is packet loss and throughput.

The third simulation aims at improve the design. Thus, it is going to simulate the system serve for 4 people.  As well the users are in the specific range of areas positioned in a 60 radius circle. Since the station is fixed, range of people cannot be adjusted to improve the performance. The main data focused in this simulation is packet loss and throughput improvement.The possible change that may enhance the performance are

- Adding an access point to serve people
- Improving the data rate from 5.5mbps to 11 mbps
- Enable RTS and CTS to solve collisions

# 4. Simulation Result

## 4.1 Proof of concept

We are using the ns-3 tutorial third.cc to be the basic test case of the simulation. Then we added nodes connected to the station to transport data. The result is shown below

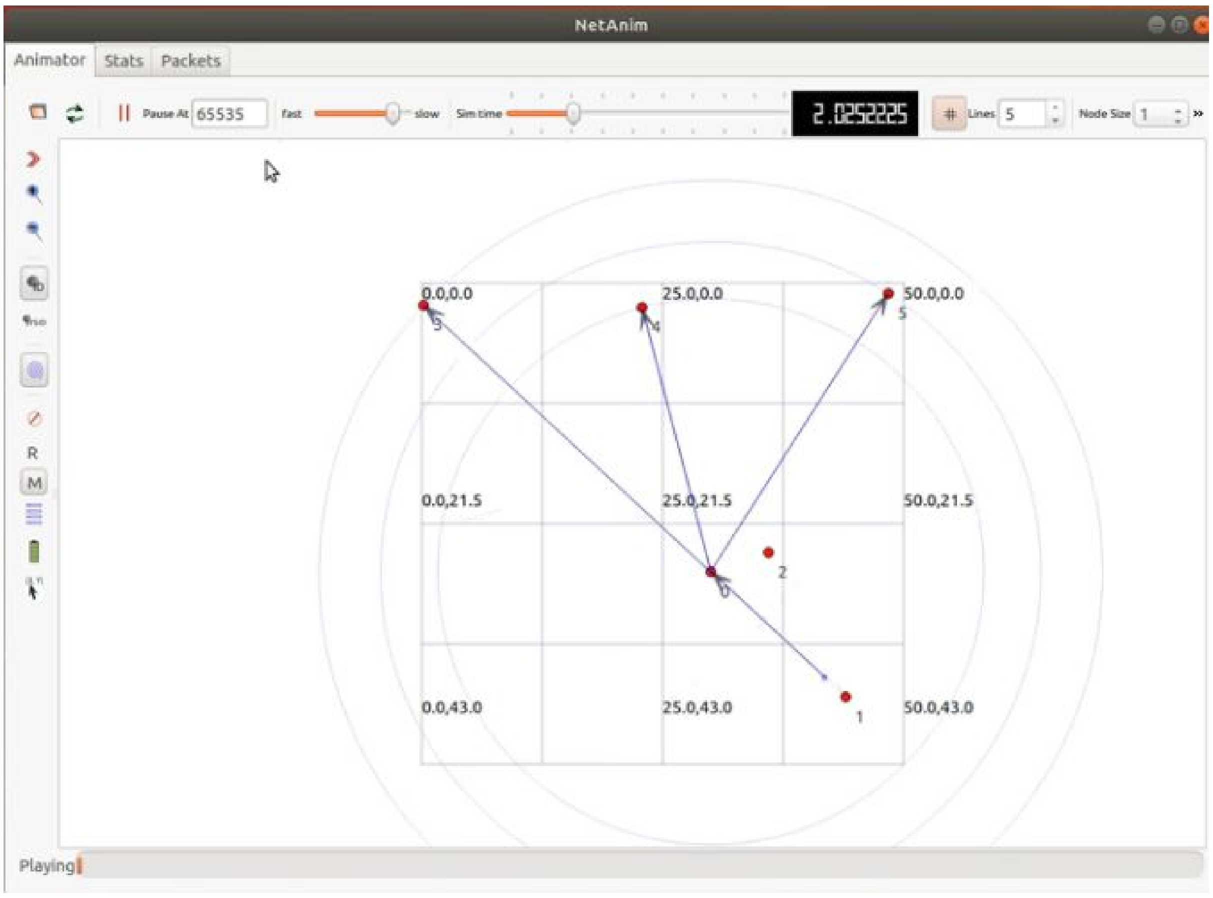

*Figure 4.1: Simulation Result 1*

In this case, node 3,4,5 are acted as users and node 0 is an access point and node 1 is station point. Packets are transferred from node 1 to node 0. Then it is transferred from node 0 to user's device.

Then we build the multiple access point system for the bus station. Three users,node 4 5 6, are added and node 2 is a new access point. Figure below displays the captured behavior.

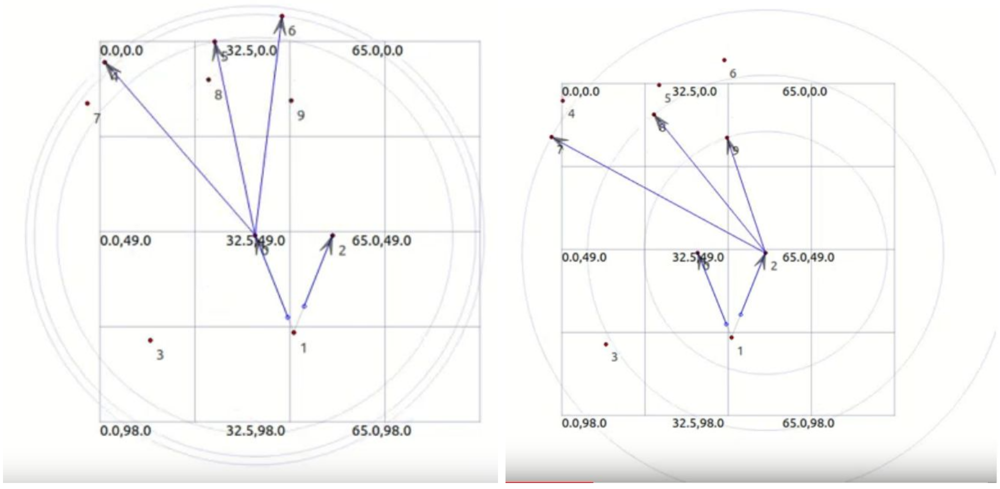

*Figure 4.2: Simulation Result 2*

From here, packets can be transferred from station nodes to access nodes through different access points. These results can prove the basic ideas of bus hotstation design.

## 4.2 General Performance of bus hotshot wifi with increasing numbers of user

The proof of this design is confirmed by the captured area. Thus, we tried to verify the performance of the design with increasing numbers of users. The scenario contains:

- One station node
- One access point
- 5.5MBPS data rate
- Each user requires ten 1024MB packets

Within these constant setting, we

| node number | Average throughput | packet loss ratio |
|---|---|---|
| 2 | 1.0939 | 0.4924905356 |
| 3 | 0.7286433333 | 0.6619569715 |
| 4 | 0.544 | 0.7476184566 |
| 5 | 0.436 | 0.7972395988 |

*Table 4.1: Data Collection*

The overall throughput is almost the same as the node increases. However, for each different flow between access points and users, there will be different throughput. Hence, we choose the average throughput for all users to indicate the performance of the throughput.

The average throughput vs node number can be shown below:

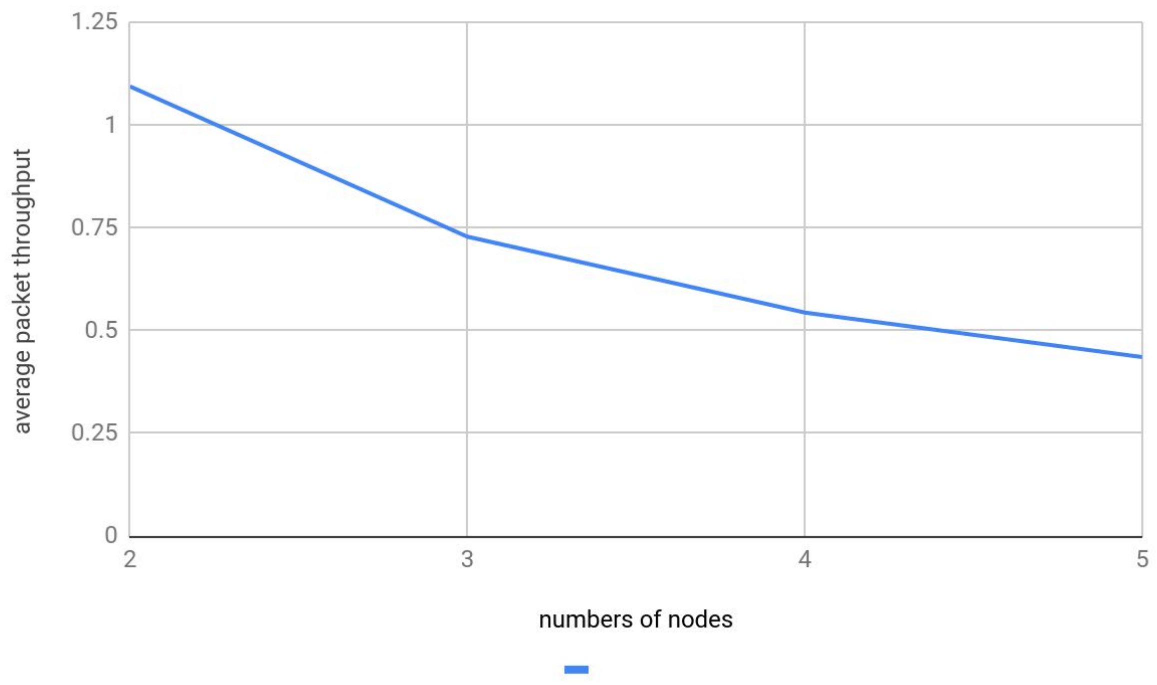

*Figure 4.3: Average throughput vs node number*

Average throughput and node number verses almost an inverse function. It can be understood since the total throughput is not changing while the node number is increasing.

The packet loss ratio is the ratio between lost pack and total packets transferred to all nodes.  Increasing users requires increasing packets leads to more packet loss. Along with more packets loss, the packet loss ratio is also increased. Since the packet loss ratio raises up to 75%, and the trend is still goes up, the system cannot get used in the real situation and requires improvement to reduce packet loss.

## 4.3 Improvement of the design

The packet loss is an issue. This session focuses three approaches trying to reduce the packet loss through the transferring. As well, average throughput should also be observed. Before adjusting the parameters, we make the assumptions:

- Users are limited within 60 meters circle of the bus station.
- The bus station wifi hotspot is going to serve 4 people(4 nodes).
- Each user requires 10 packets with size 1024MB.
- Packets can be sent from host to station nodes without obstacles

### Adding an access point to serve people

The idea is adding an access point. There are four nodes connected to one access point. The added access point will serve half of the users. By fewer users share one access point, the packet loss is going to be reduced ideally.

We keep the following conditions unchanged:

- Data rate is still 5.5Mbps

Below shows the table of the performance between one access point and two access point.

| AP | node | RANGE | Data rate | throughput | Lost packets | total packets | Lost ratio | average throughput |
|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 60 | 5.5 | 2.1949 | 48406 | 64454 | 0.745 | 0.548 |
| 2 | 4 | 60 | 5.5 | 2.2653 | 47520 | 64454 | 0.737 | 0.566 |

*Table 4.2: Adding an access point to serve people*

It can be seen that the packet loss ratio is reduced and average throughput is also enhanced. However, they are still far from expectation.

### Improving the data rate from 5.5mbps to 11 mbps

This time we widen the channel width from 5.5 mbps to 11 mbps. With a larger channel, the throughput and data loss should be improved ideally.

Before simulation, we keep

- Only one access point to serve user

Below shows the result:

| AP | node | RANGE | Data rate | throughput | Lost packets | total packets | Lost ratio | average throughput |
|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 60 | 5.5 | 2.1949 | 48406 | 64454 | 0.745 | 0.548 |
| 1 | 4 | 60 | 11 | 3.04112 | 41720 | 64454 | 0.647 | 0.760 |

*Table 4.3: Improving the data rate from 5.5mbps to 11 mbps*

It can be seen the lost ratio is decreased by 10% and average throughput is increased rapidly because total throughput added. Thus it is concluded that improve channel is a good effective way to improve performance.

**Enable RTS and CTS to solve collisions**

**"RTS/CTS** (**Request to Send / Clear to Send**), work as reducing frame collisions, which is introduced by the hidden node problem as well. [9] This is the optional mechanism used by the 802.11 wireless networking protocol. The difference is that modern RTS/CTS includes ACKs and does not solve the exposed node problem.[9] "RTS/CTS is a way to deal with the collisions happened during the transmission. We keep:

- Data rate=5.5 Mbps
- 1 access point

The result of enabled RTS and CTS is displayed below:

| AP | node | RTS/CTS | RANGE | Data rate | throughput | Lost packets | total packets | Lost ratio | average throughput |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | Disabled | 60 | 5.5 | 2.1949 | 48046 | 64454 | 0.74543084 | 0.548725 |
| 1 | 4 | Enabled | 60 | 5.5 | 1.74958 | 174422 | 187500 | 0.93025066 | 0.437395 |

*Table 4.4: Enable RTS and CTS to solve collisions*

However, in this case packet loss ratio is increased and average throughput is limited. To recover from collisions, more duplicated packets need to be sent, leading to more packets lost. As well, the throughput is wasted for duplicated packets. Thus, we decide to abandon this idea to improve the performance.

## 5. Discussion

This simulation firstly prove the design and capture the workflow of the bus station hotshot with bus station nodes and access nodes. The packets can be transmitted from the bus station host to the access points and then transmitted to users.

Then the simulation verifies the performance of the simple design. We assume the bus station is a circle with the radius of 60 meters. As more nodes (users) joining the wifi, the packet loss ratio goes up extremely high and throughput for each node is becoming smaller. As a result, the application of this design is rejected because of packet loss and uninsured throughput.

The third part of the simulation focuses on reducing the packet loss ratio. There are three approaches. Adding an access point can enhance the performance slightly while increasing the data rate can reduce a large amount of packet loss and guarantee a more throughput. Adding RTS and CTS cannot reduce packet loss and it will add barriers between transmitting.

## 6. Conclusion

This project designs a basic bus station hotshot. The hotshot is proved to work but with bad performance. The main problem is the packet loss ratio. Different approaches are tried but with limited enhancement.

The future work of this project is to make the design eligible with a high quality, containing low packet loss and guaranteed throughput. More complex approaches and impact on these two issues can be researched.

# 5. References

[1] David, Bertrand & Chalon, Rene. (2010). Hotspot Based Mobile Web Communication and Cooperation: ABRI+ Bus Shelter as a Hotspot for Mobile Contextual Transportation and Social Collaboration. 184-189.

[2] R. P. Folkes and L. A. Visser, "PROTOCOL COMMUNICATION AND OTHER PUBLICATIONS TRANSIT PACKET FORWARDING ROUTED BETWEEN MULTIPLE VIRTUAL ROUTERS WITHNA SINGLE PHYSICAL ROUTER", 24-Apr-2007.

[3] P. J. Pepperell, A. D. Cambridge, C. M. Spencer, R. F. Concord, and P. S. Commerford, "SYSTEM AND METHOD FOR ASSISTING IN CONTROLLING REAL-TIME TRANSPORT PROTOCOL FLOW THROUGH MULTIPLE NETWORKS VLAUSE OF ACLUSTER OF SESSION ROUTERS", 21-Feb-2006.

[4] ns-3: ns3::MinstrelHtWifiManager Class Reference. [Online]. Available: https://www.nsnam.org/doxygen/wifi-trans-example_8cc.html [Accessed: 17-Feb-2019].

[5] ns-3: ns3::Building a Wireless Network Topology. [Online]. Available: https://www.nsnam.org/docs/release/3.7/tutorial/tutorial_27.html [Accessed: 17-Feb-2019].

[6] ns-3 Tutorial 1, Configuring NetAnim, The Motivated Engineer [Online]. Available:

https://www.youtube.com/watch?v=1FBSAghbjNM&list=PLmcMMZCV897qkb1fv177Y69hX8YhcmVpS [Accessed: 28-Mar-2019].

[7] 3 node connection in ns3, UTKARSH VERMA [Online]. Available: https://www.youtube.com/watch?v=yW-_BSielS4 [Accessed: 28-Mar-2019].

[8] ns-3: WIFI-TCP, IMDEA Networks Institute, 2015 [Online]. Available: https://www.nsnam.org/doxygen/wifi-tcp_8cc_source.html [Accessed: 28-Mar-2019].

[9] Wikipedia: IEEE 802.11 RTS/CTS: Phil Karn, “MACA - A New Channel Access Method for Packet Radio” [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.11_RTS/CTS [Accessed: 28-Mar-2019].

# Appendix

## 1. CODE LISTING

```
#include "ns3/core-module.h"
#include "ns3/propagation-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/wifi-module.h"

#include <cmath>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <vector>
NS_LOG_COMPONENT_DEFINE ("Main");

using namespace ns3;

#define Downlink true
#define Uplink false
#define PI 3.14159265
#define PI_e5 314158

class Experiment
{
public:
  Experiment(bool downlinkUplink, std::string in_modes);
  void SetRtsCts(bool enableCtsRts);
  void CreateNode(size_t in_ap, size_t in_nodeNumber, double radius);
  void CreateNode(size_t in_ap, size_t in_nodeNumber);
  void InitialExperiment();
  void InstallApplication(size_t in_packetSize, size_t in_dataRate);
  void Run(size_t in_simTime);
  void PhyRxErrorTrace (std::string context, Ptr<const Packet> packet,
double snr);
  void PhyRxOkTrace (std::string context, Ptr<const Packet> packet,
                     double snr, WifiMode mode, enum WifiPreamble
preamble);
  void PhyTxTrace (std::string context, Ptr<const Packet> packet,
WifiMode mode,
              WifiPreamble preamble, uint8_t txPower);
  void ShowNodeInformation(NodeContainer in_c, size_t in_numOfNode);

private:
  void SetWifiChannel();
  void InstallDevices();
  void InstallIp();

  bool m_enableCtsRts;
  bool m_downlinkUplink;
  size_t m_apNumber;
  size_t m_nodeNumber;
  double m_radius;
  size_t m_rxOkCount;
  size_t m_rxErrorCount;
  size_t m_txOkCount;
  std::string m_modes;
  std::vector<std::vector<double> > m_readChannelGain;
  std::vector<int> m_serveBy;
  NodeContainer m_nodes;
  MobilityHelper m_mobility;
  Ptr<ListPositionAllocator> m_apPosAlloc;
  Ptr<ListPositionAllocator> m_nodePosAlloc;
  YansWifiChannelHelper m_wifiChannel;
  WifiHelper m_wifi;
  YansWifiPhyHelper m_wifiPhy;
  NqosWifiMacHelper m_wifiMac;
  NetDeviceContainer m_devices;
  InternetStackHelper m_internet;
  Ipv4AddressHelper m_ipv4;
  ApplicationContainer m_cbrApps;
  ApplicationContainer m_pingApps;
};

Experiment::Experiment(bool in_downlinkUplink, std::string in_modes):
  m_downlinkUplink(in_downlinkUplink), m_modes(in_modes)
{
  m_rxOkCount = 0;
  m_rxErrorCount = 0;
  m_txOkCount = 0;
}

void
Experiment::InitialExperiment()
{
  SetWifiChannel();
  InstallDevices();
  InstallIp();
```

```
}

void
Experiment::SetRtsCts(bool in_enableCtsRts)
{
  m_enableCtsRts = in_enableCtsRts;
  UintegerValue ctsThr = (m_enableCtsRts ? UintegerValue (10) :
UintegerValue (22000));
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
ctsThr);
}


void
Experiment::CreateNode(size_t in_ap, size_t in_nodeNumber, double
in_radius)
{
  m_apNumber = in_ap;
  m_nodeNumber = in_nodeNumber;
  m_radius = in_radius;

  m_nodes.Create(m_apNumber+m_nodeNumber);

  m_mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
  m_mobility.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
  m_apPosAlloc = CreateObject<ListPositionAllocator> ();
  m_nodePosAlloc = CreateObject<ListPositionAllocator> ();

  for(size_t i=0; i<m_apNumber; ++i){
    m_apPosAlloc->Add(Vector(m_radius*std::cos(i*2*PI/m_apNumber),
                     m_radius*std::sin(i*2*PI/m_apNumber), 1));
    //m_apPosAlloc->Add(Vector(m_radius, 0, 1));
  }
  m_mobility.SetPositionAllocator(m_apPosAlloc);
  for(size_t i=0; i<m_apNumber; ++i){
    m_mobility.Install(m_nodes.Get(i));
  }

  for(size_t i=0; i<m_nodeNumber; ++i){
   size_t inAp = i/(m_nodeNumber/m_apNumber);
   double nodeRadius = rand()%120+(rand()%1000)/1000;
   m_nodePosAlloc->Add(Vector(m_radius*std::cos(inAp*2*PI/m_apNumber)+
                      nodeRadius*std::cos((rand()%(2*PI_e5))/pow(10,
5)),
                      m_radius*std::sin(inAp*2*PI/m_apNumber)+
                      nodeRadius*std::sin((rand()%(2*PI_e5))/pow(10,
5)),
                      1));
   //m_nodePosAlloc->Add(Vector(0, 0, 1));
  }
  m_mobility.SetPositionAllocator(m_nodePosAlloc);
  for(size_t i=0; i<m_nodeNumber; ++i){
    m_mobility.Install(m_nodes.Get(m_apNumber+i));
  }
}

void
Experiment::SetWifiChannel()
{
  m_wifiChannel.SetPropagationDelay
("ns3::ConstantSpeedPropagationDelayModel");
  m_wifiChannel.AddPropagationLoss
("ns3::TwoRayGroundPropagationLossModel",
                                  "Frequency", DoubleValue(2.400e9));
}

void
Experiment::InstallDevices()
{
  m_wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
  //Config::SetDefault
("ns3::WifiRemoteStationManager::NonUnicastMode",
  //                              StringValue ("DsssRate2Mbps"));
  m_wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (m_modes),
                                "ControlMode",StringValue (m_modes));


  m_wifiPhy =  YansWifiPhyHelper::Default ();
  m_wifiPhy.SetChannel (m_wifiChannel.Create());
  m_wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue (-95.0) );
  m_wifiPhy.Set ("CcaMode1Threshold", DoubleValue (-95.0) );
  m_wifiPhy.Set ("TxPowerStart", DoubleValue (23.0) );
  m_wifiPhy.Set ("TxPowerEnd", DoubleValue (23.0) );
  m_wifiPhy.Set ("ChannelNumber", UintegerValue (1) );
  m_wifiPhy.Set ("RxGain", DoubleValue (-25.0));
  NqosWifiMacHelper m_wifiMac;
  m_wifiMac.SetType ("ns3::AdhocWifiMac"); // use ad-hoc MAC
  m_devices = m_wifi.Install (m_wifiPhy, m_wifiMac, m_nodes);
}

void
Experiment::InstallIp()
{
  m_internet.Install (m_nodes);
  m_ipv4.SetBase ("10.0.0.0", "255.0.0.0");
  m_ipv4.Assign (m_devices);
}

void
Experiment::PhyRxErrorTrace (std::string context, Ptr<const Packet>
packet, double snr)
{
  Ptr<Packet> m_currentPacket;
  WifiMacHeader hdr;
  m_currentPacket = packet->Copy();
  m_currentPacket->RemoveHeader (hdr);
  if(hdr.IsData()){
    m_rxErrorCount++;
  }
}

void
Experiment::PhyRxOkTrace (std::string context, Ptr<const Packet>
packet,
        double snr, WifiMode mode, enum WifiPreamble preamble)
{
  Ptr<Packet> m_currentPacket;
  WifiMacHeader hdr;

  m_currentPacket = packet->Copy();
  m_currentPacket->RemoveHeader (hdr);
  if(hdr.IsData()){
    m_rxOkCount++;
  }
}

void
Experiment::PhyTxTrace (std::string context, Ptr<const Packet> packet,
                  WifiMode mode, WifiPreamble preamble, uint8_t
txPower)
{
  Ptr<Packet> m_currentPacket;
  WifiMacHeader hdr;
  m_currentPacket = packet->Copy();
  m_currentPacket->RemoveHeader (hdr);
  if(hdr.IsData()){
    m_txOkCount++;
  }
}



void
```

```
Experiment::InstallApplication(size_t in_packetSize, size_t
in_dataRate)
{
  uint16_t cbrPort = 12345;
  for(size_t j=1; j<=m_apNumber; ++j){
    for(size_t i=m_apNumber+m_nodeNumber/m_apNumber*(j-1);
        i<m_apNumber+m_nodeNumber/m_apNumber*j ; ++i){
      std::string s;
      std::stringstream ss(s);
      if(m_downlinkUplink){
         ss << i+1;
      }else
      {
        ss << j;
      }
      s = "10.0.0."+ss.str();
      OnOffHelper onOffHelper ("ns3::UdpSocketFactory",
               InetSocketAddress (Ipv4Address (s.c_str()), cbrPort));
      onOffHelper.SetAttribute ("PacketSize", UintegerValue
(in_packetSize));
//onOffHelper.SetAttribute ("OnTime",  StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
//onOffHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
      std::string s2;
      std::stringstream ss2(s2);
      if(m_downlinkUplink){
        ss2 << in_dataRate+i*100;
      }else
      {
        ss2 << in_dataRate+i*100;
      }
      s2 = ss2.str() + "bps";
      onOffHelper.SetAttribute ("DataRate", StringValue (s2));
      if(m_downlinkUplink){
        onOffHelper.SetAttribute ("StartTime", TimeValue (Seconds
(1.00+static_cast<double>(i)/100)));
        onOffHelper.SetAttribute ("StopTime", TimeValue (Seconds
(50.000+static_cast<double>(i)/100)));
        m_cbrApps.Add (onOffHelper.Install (m_nodes.Get (j-1)));
      }else
      {
        onOffHelper.SetAttribute ("StartTime", TimeValue (Seconds
(1.00)));
        onOffHelper.SetAttribute ("StopTime", TimeValue (Seconds
(50.000+static_cast<double>(j)/100)));
        m_cbrApps.Add (onOffHelper.Install (m_nodes.Get (i)));
      }
    }
  }
  uint16_t  echoPort = 9;
  // again using different start times to workaround Bug 388 and Bug
912
  for(size_t j=1; j<=m_apNumber; ++j){
    for(size_t i=m_apNumber+m_nodeNumber/m_apNumber*(j-1);
        i<m_apNumber+m_nodeNumber/m_apNumber*j ; ++i){
      std::string s;
      std::stringstream ss(s);
      if(m_downlinkUplink){
         ss << i+1;
      }else
      {
        ss << j;
      }
      s = "10.0.0."+ss.str();
      UdpEchoClientHelper echoClientHelper (Ipv4Address (s.c_str()),
echoPort);
      echoClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
      echoClientHelper.SetAttribute ("Interval", TimeValue (Seconds
(0.1)));
      echoClientHelper.SetAttribute ("PacketSize", UintegerValue (10));
      if(m_downlinkUplink){
        echoClientHelper.SetAttribute ("StartTime", TimeValue (Seconds
(0.001)));
        echoClientHelper.SetAttribute ("StopTime", TimeValue (Seconds
(50.001)));
        m_pingApps.Add (echoClientHelper.Install (m_nodes.Get (j-1)));
      }else
      {
        echoClientHelper.SetAttribute ("StartTime", TimeValue (Seconds
(0.001)));
        echoClientHelper.SetAttribute ("StopTime", TimeValue (Seconds
(50.001)));
        m_pingApps.Add (echoClientHelper.Install (m_nodes.Get (i)));
      }
    }
  }
}


void
Experiment::ShowNodeInformation(NodeContainer in_c, size_t
in_numOfNode)
{
  for(size_t i=0; i<in_numOfNode; ++i){
    Ptr<MobilityModel> mobility = in_c.Get(i)->GetObject<MobilityModel>
();
    Vector nodePos = mobility->GetPosition ();
    // Get Ipv4 instance of the node
    Ptr<Ipv4> ipv4 = in_c.Get(i)->GetObject<Ipv4> ();
    // Get Ipv4 instance of the node
    Ptr<MacLow> mac48 = in_c.Get(i)->GetObject<MacLow> ();
    // Get Ipv4InterfaceAddress of xth interface.
    Ipv4Address addr = ipv4->GetAddress (1, 0).GetLocal ();
    //Mac48Address macAddr = mac48->GetAddress();
    std::cout << in_c.Get(i)->GetId() << " " << addr << " (" <<
nodePos.x << ", " <<
               nodePos.y << ")" << std::endl;
  }
}

void
Experiment::Run(size_t in_simTime)
{
  // 8. Install FlowMonitor on all nodes
  ShowNodeInformation(m_nodes, m_apNumber+m_nodeNumber);

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

  // 9. Run simulation
  Simulator::Stop (Seconds (in_simTime));
  Config::Connect ("/NodeList/*/DeviceList/*/Phy/State/RxError",
            MakeCallback (&Experiment::PhyRxErrorTrace, this));
  Config::Connect ("/NodeList/*/DeviceList/*/Phy/State/RxOk",
            MakeCallback (&Experiment::PhyRxOkTrace, this));
  Config::Connect ("/NodeList/*/DeviceList/*/Phy/State/Tx",
            MakeCallback (&Experiment::PhyTxTrace, this));
  Simulator::Run ();

  // 10. Print per flow statistics
  monitor->CheckForLostPackets ();
  Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
  std::map<FlowId, FlowMonitor::FlowStats> stats =
monitor->GetFlowStats ();
  double accumulatedThroughput = 0;
  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator
i=stats.begin();
        i!=stats.end(); ++i)
  {
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
    std::cout << "Flow " << i->first<< " (" << t.sourceAddress << " ->
" << t.destinationAddress << ")\n";
    std::cout << "  Tx Bytes:   " << i->second.txBytes << "\n";
    std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";
    std::cout << "  Tx Packets: " << i->second.txPackets << "\n";
    std::cout << "  Rx Packets: " << i->second.rxPackets << "\n";
    std::cout << "  Lost Packets: " << i->second.lostPackets << "\n";
    std::cout << "  Pkt Lost Ratio: " <<
((double)i->second.txPackets-(double)i->second.rxPackets)/(double)i->se
cond.txPackets << "\n";
    std::cout << "  Throughput: " << i->second.rxBytes * 8.0 /
in_simTime / 1024 / 1024  << " Mbps\n";

accumulatedThroughput+=(i->second.rxBytes*8.0/in_simTime/1024/1024);
  }
  std::cout << "apNumber=" <<m_apNumber << " nodeNumber=" <<
m_nodeNumber << "\n" << std::flush;
  std::cout << "throughput=" << accumulatedThroughput << "\n" <<
std::flush;
  std::cout << "tx=" << m_txOkCount << " RXerror=" <<m_rxErrorCount <<
               " Rxok=" << m_rxOkCount << "\n" << std::flush;
  std::cout << "===========================\n" << std::flush;
  // 11. Cleanup
  Simulator::Destroy ();
}

int main (int argc, char **argv)
{

  size_t numOfAp[6] = {1, 2, 3, 4, 5, 6};
  double range[4] = {60, 120, 180, 240};
  std::vector <std::string> modes;
  modes.push_back ("DsssRate1Mbps");
  modes.push_back ("DsssRate2Mbps");
  modes.push_back ("DsssRate5_5Mbps");
  modes.push_back ("DsssRate11Mbps");
  std::cout << "Hidden station experiment with RTS/CTS disabled:\n" <<
std::flush;
  for(size_t i=0; i<1; ++i){
    for(size_t j=0; j<1; ++j){
      for(size_t k=2; k<3; ++k){
        std::cout << "Range=" << range[j] << ", Mode=" << modes[k] <<
"\n";
        Experiment exp(Downlink, modes[k]);
        exp.SetRtsCts(false);
        exp.CreateNode(numOfAp[i], 2, range[j]);
        exp.InitialExperiment();
        exp.InstallApplication(1024, 5500000);
        exp.Run(60);
      }
    }
  }
  /*
  std::cout << "Hidden station experiment with RTS/CTS enable:\n" <<
std::flush;
  for(size_t i=0; i<6; ++i){
    for(size_t j=0; j<4; ++j){
      for(size_t k=0; k<modes.size(); ++k){
        std::cout << "Range=" << range[j] << "Mode=" << modes[k] <<
"\n";
        Experiment exp(Downlink, modes[k]);
        exp.SetRtsCts(true);
        exp.CreateNode(numOfAp[i], 60, range[j]);
        exp.InitialExperiment();
        exp.InstallApplication(1024, 16000000);
        exp.Run(60);
      }
    }
  }
  */
  return 0;
}
```