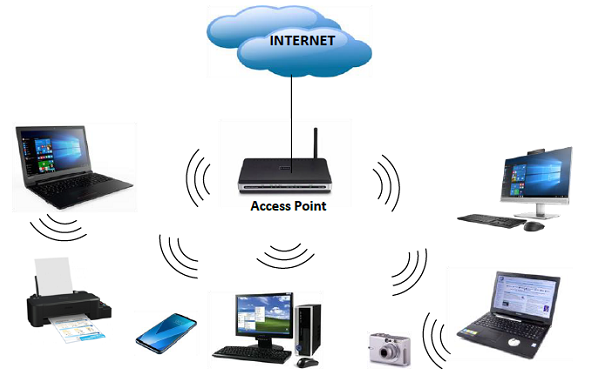
**Practical No 17**

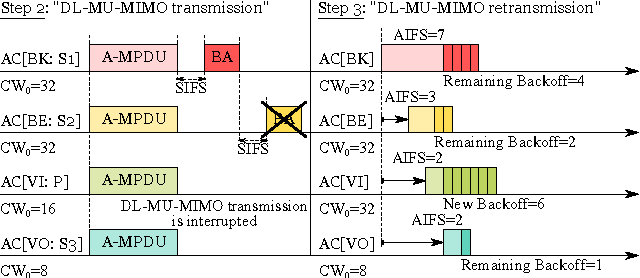
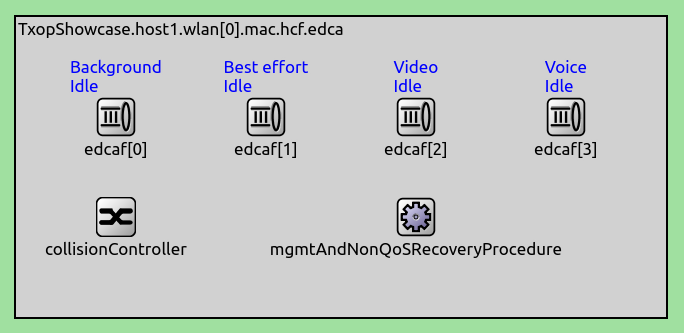
**Aim: -** Write a program to simulate wireless network wifi-80211e-txop.

**Objective: -**

* Understanding of wireless network
* Understanding and implementation of wireless network with transaction opportunity(txop)

**Theory: -**

Computer networks that are not connected by cables are called wireless networks. They generally use radio waves for communication between the network nodes. They allow devices to be connected to the network while roaming around within the network coverage.

TXOP is available in QoS mode as part of EDCA (Enhanced Distributed Channel Access), and it is a limited time period of contention-free channel access available to the channel-owning station. During such a period the station can send multiple frames that belong to a particular access category. The benefit of TXOP is that it increases throughput and reduces delay of QoS data frames via eliminating contention periods between transmissions. TXOP can be used in combination with aggregation and block acknowledgement to further increase throughput.

**Program: -**

#include "ns3/command-line.h"

#include "ns3/config.h"

#include "ns3/uinteger.h"

#include "ns3/boolean.h"

#include "ns3/double.h"

#include "ns3/string.h"

#include "ns3/log.h"

#include "ns3/yans-wifi-helper.h"

#include "ns3/ssid.h"

#include "ns3/mobility-helper.h"

#include "ns3/internet-stack-helper.h"

#include "ns3/ipv4-address-helper.h"

#include "ns3/udp-client-server-helper.h"

#include "ns3/packet-sink-helper.h"

#include "ns3/on-off-helper.h"

#include "ns3/ipv4-global-routing-helper.h"

#include "ns3/packet-sink.h"

#include "ns3/yans-wifi-channel.h"

// This is a simple example in order to show how to configure an IEEE 802.11ax Wi-Fi network.

//

// It outputs the UDP or TCP goodput for every HE MCS value, which depends on the MCS value (0 to 11),

// the channel width (20, 40, 80 or 160 MHz) and the guard interval (800ns, 1600ns or 3200ns).

// The PHY bitrate is constant over all the simulation run. The user can also specify the distance between

// the access point and the station: the larger the distance the smaller the goodput.

//

// The simulation assumes a single station in an infrastructure network:

//

// STA AP

// \* \*

// | |

// n1 n2

//

//Packets in this simulation belong to BestEffort Access Class (AC\_BE).

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("he-wifi-network");

int main (int argc, char \*argv[])

{

bool udp = true;

bool useRts = false;

bool useExtendedBlockAck = false;

double simulationTime = 10; //seconds

double distance = 1.0; //meters

double frequency = 5; //whether 2.4, 5 or 6 GHz

int mcs = -1; // -1 indicates an unset value

double minExpectedThroughput = 0;

double maxExpectedThroughput = 0;

CommandLine cmd (\_\_FILE\_\_);

cmd.AddValue ("frequency", "Whether working in the 2.4, 5 or 6 GHz band (other values gets rejected)", frequency);

cmd.AddValue ("distance", "Distance in meters between the station and the access point", distance);

cmd.AddValue ("simulationTime", "Simulation time in seconds", simulationTime);

cmd.AddValue ("udp", "UDP if set to 1, TCP otherwise", udp);

cmd.AddValue ("useRts", "Enable/disable RTS/CTS", useRts);

cmd.AddValue ("useExtendedBlockAck", "Enable/disable use of extended BACK", useExtendedBlockAck);

cmd.AddValue ("mcs", "if set, limit testing to a specific MCS (0-11)", mcs);

cmd.AddValue ("minExpectedThroughput", "if set, simulation fails if the lowest throughput is below this value", minExpectedThroughput);

cmd.AddValue ("maxExpectedThroughput", "if set, simulation fails if the highest throughput is above this value", maxExpectedThroughput);

cmd.Parse (argc,argv);

if (useRts)

{

Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue ("0"));

}

double prevThroughput [12];

for (uint32\_t l = 0; l < 12; l++)

{

prevThroughput[l] = 0;

}

std::cout << "MCS value" << "\t\t" << "Channel width" << "\t\t" << "GI" << "\t\t\t" << "Throughput" << '\n';

int minMcs = 0;

int maxMcs = 11;

if (mcs >= 0 && mcs <= 11)

{

minMcs = mcs;

maxMcs = mcs;

}

for (int mcs = minMcs; mcs <= maxMcs; mcs++)

{

uint8\_t index = 0;

double previous = 0;

uint8\_t maxChannelWidth = frequency == 2.4 ? 40 : 160;

for (int channelWidth = 20; channelWidth <= maxChannelWidth; ) //MHz

{

for (int gi = 3200; gi >= 800; ) //Nanoseconds

{

uint32\_t payloadSize; //1500 byte IP packet

if (udp)

{

payloadSize = 1472; //bytes

}

else

{

payloadSize = 1448; //bytes

Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (payloadSize));

}

NodeContainer wifiStaNode;

wifiStaNode.Create (1);

NodeContainer wifiApNode;

wifiApNode.Create (1);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();

YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();

phy.SetChannel (channel.Create ());

WifiMacHelper mac;

WifiHelper wifi;

if (frequency == 6)

{

wifi.SetStandard (WIFI\_STANDARD\_80211ax\_6GHZ);

Config::SetDefault ("ns3::LogDistancePropagationLossModel::ReferenceLoss", DoubleValue (48));

}

else if (frequency == 5)

{

wifi.SetStandard (WIFI\_STANDARD\_80211ax\_5GHZ);

}

else if (frequency == 2.4)

{

wifi.SetStandard (WIFI\_STANDARD\_80211ax\_2\_4GHZ);

Config::SetDefault ("ns3::LogDistancePropagationLossModel::ReferenceLoss", DoubleValue (40));

}

else

{

std::cout << "Wrong frequency value!" << std::endl;

return 0;

}

std::ostringstream oss;

oss << "HeMcs" << mcs;

wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager","DataMode", StringValue (oss.str ()),

"ControlMode", StringValue (oss.str ()));

Ssid ssid = Ssid ("ns3-80211ax");

mac.SetType ("ns3::StaWifiMac",

"Ssid", SsidValue (ssid));

NetDeviceContainer staDevice;

staDevice = wifi.Install (phy, mac, wifiStaNode);

mac.SetType ("ns3::ApWifiMac",

"EnableBeaconJitter", BooleanValue (false),

"Ssid", SsidValue (ssid));

NetDeviceContainer apDevice;

apDevice = wifi.Install (phy, mac, wifiApNode);

// Set channel width, guard interval and MPDU buffer size

Config::Set ("/NodeList/\*/DeviceList/\*/$ns3::WifiNetDevice/Phy/ChannelWidth", UintegerValue (channelWidth));

Config::Set ("/NodeList/\*/DeviceList/\*/$ns3::WifiNetDevice/HeConfiguration/GuardInterval", TimeValue (NanoSeconds (gi)));

Config::Set ("/NodeList/\*/DeviceList/\*/$ns3::WifiNetDevice/HeConfiguration/MpduBufferSize", UintegerValue (useExtendedBlockAck ? 256 : 64));

// mobility.

MobilityHelper mobility;

Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();

positionAlloc->Add (Vector (0.0, 0.0, 0.0));

positionAlloc->Add (Vector (distance, 0.0, 0.0));

mobility.SetPositionAllocator (positionAlloc);

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

mobility.Install (wifiApNode);

mobility.Install (wifiStaNode);

/\* Internet stack\*/

InternetStackHelper stack;

stack.Install (wifiApNode);

stack.Install (wifiStaNode);

Ipv4AddressHelper address;

address.SetBase ("192.168.1.0", "255.255.255.0");

Ipv4InterfaceContainer staNodeInterface;

Ipv4InterfaceContainer apNodeInterface;

staNodeInterface = address.Assign (staDevice);

apNodeInterface = address.Assign (apDevice);

/\* Setting applications \*/

ApplicationContainer serverApp;

if (udp)

{

//UDP flow

uint16\_t port = 9;

UdpServerHelper server (port);

serverApp = server.Install (wifiStaNode.Get (0));

serverApp.Start (Seconds (0.0));

serverApp.Stop (Seconds (simulationTime + 1));

UdpClientHelper client (staNodeInterface.GetAddress (0), port);

client.SetAttribute ("MaxPackets", UintegerValue (4294967295u));

client.SetAttribute ("Interval", TimeValue (Time ("0.00001"))); //packets/s

client.SetAttribute ("PacketSize", UintegerValue (payloadSize));

ApplicationContainer clientApp = client.Install (wifiApNode.Get (0));

clientApp.Start (Seconds (1.0));

clientApp.Stop (Seconds (simulationTime + 1));

}

else

{

//TCP flow

uint16\_t port = 50000;

Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", localAddress);

serverApp = packetSinkHelper.Install (wifiStaNode.Get (0));

serverApp.Start (Seconds (0.0));

serverApp.Stop (Seconds (simulationTime + 1));

OnOffHelper onoff ("ns3::TcpSocketFactory", Ipv4Address::GetAny ());

onoff.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

onoff.SetAttribute ("PacketSize", UintegerValue (payloadSize));

onoff.SetAttribute ("DataRate", DataRateValue (1000000000)); //bit/s

AddressValue remoteAddress (InetSocketAddress (staNodeInterface.GetAddress (0), port));

onoff.SetAttribute ("Remote", remoteAddress);

ApplicationContainer clientApp = onoff.Install (wifiApNode.Get (0));

clientApp.Start (Seconds (1.0));

clientApp.Stop (Seconds (simulationTime + 1));

} Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (simulationTime + 1));

Simulator::Run ();

uint64\_t rxBytes = 0;

if (udp)

{

rxBytes = payloadSize \* DynamicCast<UdpServer> (serverApp.Get (0))->GetReceived ();

}

else

{

rxBytes = DynamicCast<PacketSink> (serverApp.Get (0))->GetTotalRx ();

}

double throughput = (rxBytes \* 8) / (simulationTime \* 1000000.0); //Mbit/s

Simulator::Destroy ();

std::cout << mcs << "\t\t\t" << channelWidth << " MHz\t\t\t" << gi << " ns\t\t\t" << throughput << " Mbit/s" << std::endl;

//test first element

if (mcs == 0 && channelWidth == 20 && gi == 3200)

{

if (throughput < minExpectedThroughput)

{

NS\_LOG\_ERROR ("Obtained throughput " << throughput << " is not expected!");

exit (1);

}

}

//test last element

if (mcs == 11 && channelWidth == 160 && gi == 800)

{

if (maxExpectedThroughput > 0 && throughput > maxExpectedThroughput)

{

NS\_LOG\_ERROR ("Obtained throughput " << throughput << " is not expected!");

exit (1);

}

}

//test previous throughput is smaller (for the same mcs)

if (throughput > previous)

{

previous = throughput;

}

else

{

NS\_LOG\_ERROR ("Obtained throughput " << throughput << " is not expected!");

exit (1);

}

//test previous throughput is smaller (for the same channel width and GI)

if (throughput > prevThroughput [index])

{

prevThroughput [index] = throughput;

}

else

{

NS\_LOG\_ERROR ("Obtained throughput " << throughput << " is not expected!");

exit (1);

}

index++;

gi /= 2;

}

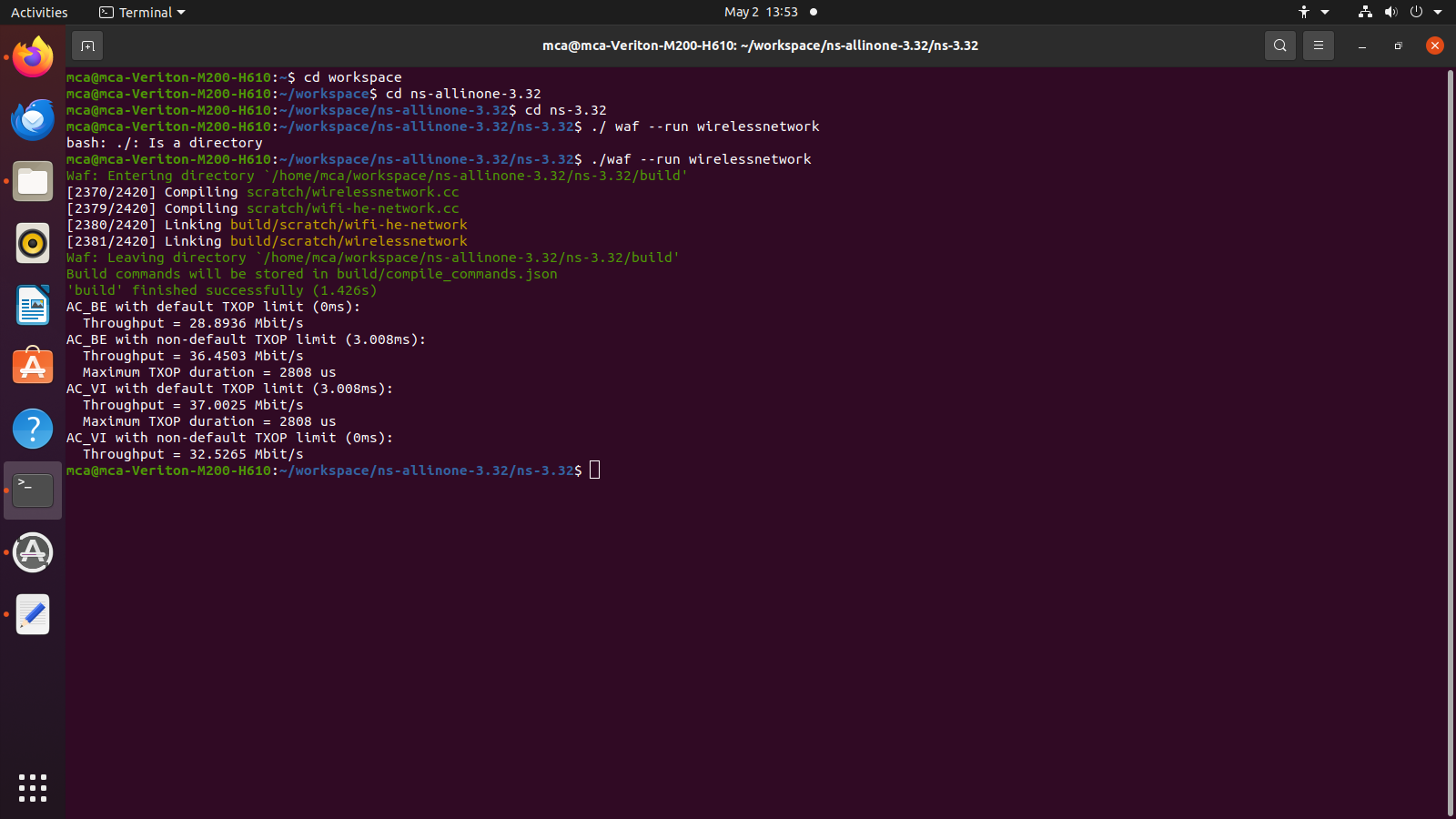
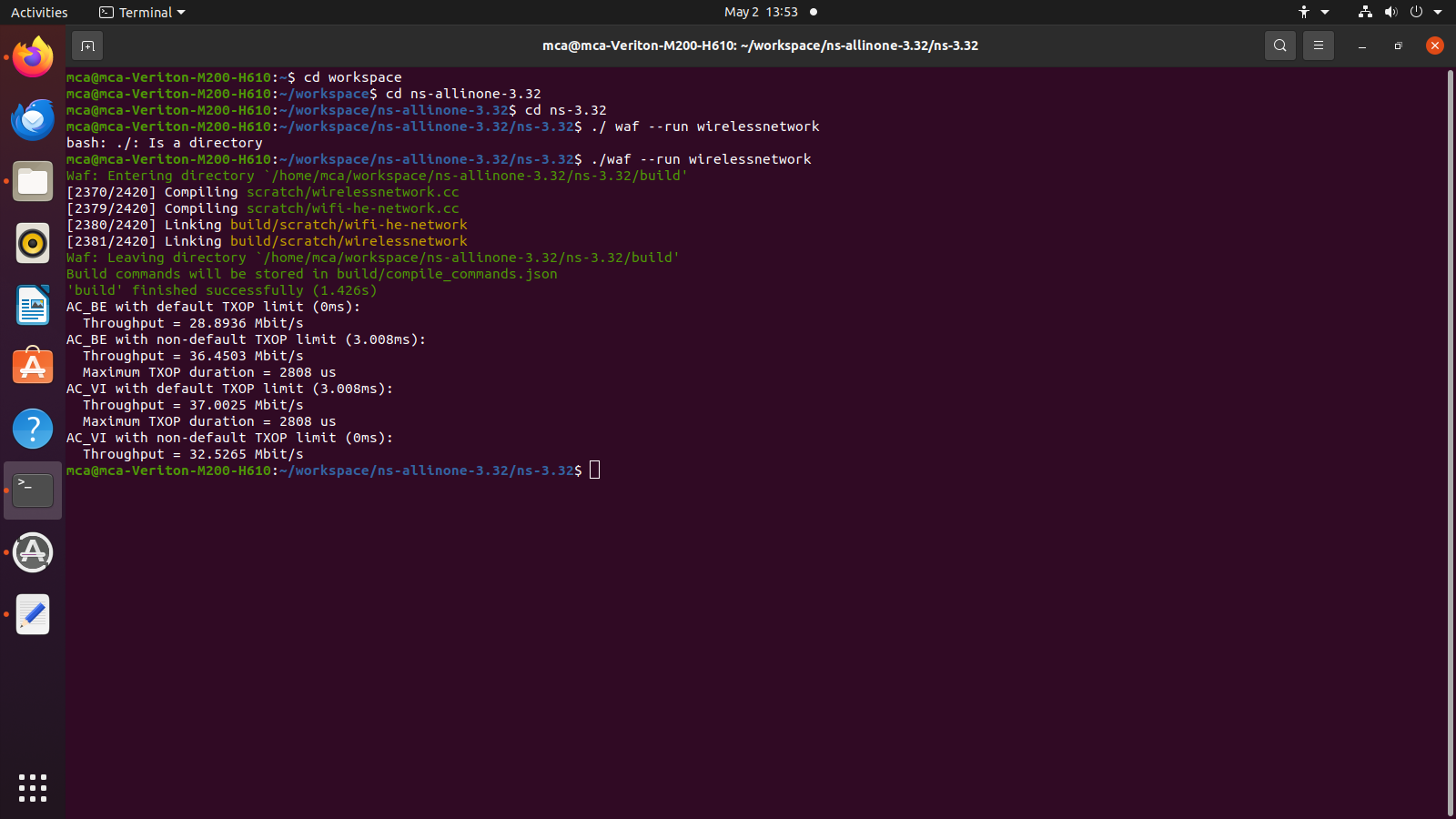
channelWidth \*= 2;

}

}

return 0;

}

**Output: -**

**Conclusion: -**

Successful implementation of wireless network wifi-80211e-txop.