به نام خدا

گزارش کار پروژه پایانی درس شبکه های بی سیم

عنوان پروژه :

شبیه سازی 6 نمودار از نمودار های تز

***Performance study of 802.11n WLAN and MAC enhancements in ns-3***

***استاد مربوطه : دکتر محمد نصیری***

***محمد پیشدار***

***نرگس رضایی***

***دانشگاه بو علی سینا***

***تابستان 95***

فهرست مطالب

|  |  |
| --- | --- |
| صفحه | عنوان |
| 3 | سناریو 1 |
| 8 | پیاده سازی (اصول) |
| 18 | نمودار های Fragmentation Enhancement |
| 28 | نمودار های Impact of Edca Parameter |

ما در این پژوهش موفق شدیم حدود 19 نمودار از نمودار های فصل 4 را که به تعداد 13 مورد از آنها در این گزارش شرح داده شده اند

شبیه سازی مجدد نماییم

برای این کار از نمودار های قسمت بهبود با قطعه بندی مطرح شده در فصل 4 پایان نامه معرفی شده شروع و برای استخراج نمودار های 21 ، 22 و 23 سناریوی زیر را پیاده سازی کرده ایم

توپولوژی شبیه سازی به این شکل می باشد که دو نود در سیستم وجود دارند . که یکی از اینها از نوع Station و دیگری از نوع Access Point می باشد که بر روی نود Station یک Application OnOff نصب شده است که ترافیک از نوع Udp و با دسته بندی های مختلف (Voice,Video,Best Effort,Background)را با نرخ 24mbps و با اندازه بسته 2264 بایت تولید و به نود Access point که بر روی آن نیز یک PacketSink App با سوکت udp و پورت 80 نصب شده است ارسال می کند . این شبیه سازی بر اساس استاندارد 802.11g با استفاده از مدل Log distance propagation loss در باند فرکانسی 2.4 GHZ و با تنظیم Reference loss برابر با 40.047db و در مدت 30 ثانیه انجام شده است .

برای هر دسته مفهوم فرصت ارسال(Txop) به شکل زیر نیز لحاظ شده است که پیاده سازی آن در ادامه شرح داده خواهد شد .

|  |  |
| --- | --- |
| دسته | TXOP (زمان میلی ثانیه ) |
| AC\_VO | 1.5 |
| AC\_VI | 3 |
| AC\_BE | 0 |
| AC\_BK | 0 |

اضافه کردن ماژول های مورد نیاز در شبیه سازی از جمله ماژول های اینترنت ، حرکت ،شبکه و ....

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/mobility-module.h"

#include "ns3/config-store-module.h"

#include "ns3/wifi-module.h"

#include "ns3/internet-module.h"

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

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

#include "ns3/applications-module.h"

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

NS\_LOG\_COMPONENT\_DEFINE ("WifiSimpleAdhocGrid");

using namespace ns3;

تعریف متغییر های سراسری برای انجام محاسبات

uint32\_t count =0;

uint32\_t average =0;

NodeContainer c;

uint32\_t totalRxBytesCounter = 0;

ApplicationContainer sinkApp;

اجرای این تابع به هنگام در یافت هر بسته در Packet Sink Application

void ReceivePacket (Ptr<const Packet> packet, const Address &)

{

NS\_LOG\_UNCOND ("Received one packet!");

QosTag q\_tag;

if (packet->PeekPacketTag(q\_tag))

{

NS\_LOG\_UNCOND ("Packet Tag value " << (int)q\_tag.GetTid());

}

count++;

محاسبه ی گذرداد و تعداد بسته های دریافت شده

totalRxBytesCounter += packet->GetSize ();

totalRxBytesCounter+= 1;

if (count==1000)

std::cout<<"time "<<Simulator::Now ().GetSeconds();

NS\_LOG\_UNCOND ("count basteha daryaft shode " << count);

NS\_LOG\_UNCOND("th " << Simulator::Now ().GetSeconds()<< "\t"<<((totalRxBytesCounter\*8.0)/1000000)/Simulator::Now ().GetSeconds());

NS\_LOG\_UNCOND("average packet transmition time " << Simulator::Now ().GetSeconds()<< "\t"<<((Simulator::Now ().GetSeconds())\*1000)/count);

}

تابع برای اضافه کردن برچسب نوع دسته به بسته های خروجی OnOff Application

void

TagMarker (uint8\_t tid, Ptr<const Packet> packet)

{

QosTag qosTag;

qosTag.SetTid(tid);

packet->AddPacketTag (qosTag);

}

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

{

تعریف تعداد کل نود ها و تعداد نود های ارسال کننده و دریافت کننده

std::string phyMode ("DsssRate1Mbps");

//uint32\_t packetSize = 2264; // bytes

uint32\_t numNodes = 2; // by default, 5x5

uint32\_t sinkNode = 0;

uint32\_t sourceNode = 1;

ایجاد نود ها

c.Create (numNodes);

// The below set of helpers will help us to put together the wifi NICs we want

ایجاد Wifi Helper

WifiHelper wifi=WifiHelper::Default ();

ایجاد لایه ی فیزیکی

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();

(YansWifiPhyHelper::DLT\_IEEE802\_11\_RADIO);

ایجاد و تنظیم کانال و مدل Prppagation Loss برای آن

YansWifiChannelHelper wifiChannel=YansWifiChannelHelper::Default ();

//wifiChannel.SetPropagationDelay (“ns3::ConstantSpeedPropagationDelayModel”);

// wifiChannel.AddPropagationLoss (“ns3::FriisPropagationLossModel”);

//wifiChannel.AddPropagationLoss(“ns3::LogDistancePropagationLossModel”,”Exponent”, DoubleValue (40));

wifiPhy.SetChannel (wifiChannel.Create ());

ایجاد و تنظیم مک با قابلیت پشتیبانی از کیفیت سرویس

// Add a QoS upper mac, and disable rate control

QosWifiMacHelper wifiMac = QosWifiMacHelper::Default ();

تنظیم استاندارد

wifi.SetStandard (WIFI\_PHY\_STANDARD\_80211g);

تنظیم Wifi Manager

wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

// wifi.SetRemoteStationManager ("ns3::IdealWifiManager", "RtsCtsThreshold", UintegerValue (1000));

تنظیم SSid برای Station و Access Point و نصب مک و لایه فیزیکی بر روی نو د های مربوطه

Ssid ssid = Ssid (“ns-3-ssid”);

wifiMac.SetType (“ns3::StaWifiMac”,

“Ssid”, SsidValue (ssid),

“ActiveProbing”, BooleanValue (false));

// install Wifi on Wifistatnodes

NetDeviceContainer staDevices;

staDevices = wifi.Install (wifiPhy, wifiMac, c.Get(1));

wifiMac.SetType (“ns3::ApWifiMac”,

“Ssid”, SsidValue (ssid));

NetDeviceContainer apDevices;

apDevices = wifi.Install (wifiPhy, wifiMac, c.Get(0));

تعریف مدل حرکت نود ها به صورت ثابت و با فاصله ی مشخص نسبت به هم

/\* Setting mobility model \*/

MobilityHelper mobility;

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

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

positionAlloc->Add (Vector (260, 0.0, 0.0));

mobility.SetPositionAllocator (positionAlloc);

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

mobility.Install (c.Get(0));

mobility.Install (c.Get(1));

نصب پشته ی پروتکی اینترنت بر روی نود ها

InternetStackHelper internet;

internet.Install ©;

تخصیص آدرس IP به نود ها

Ipv4AddressHelper ipv4;

NS\_LOG\_INFO (“Assign IP Addresses.”);

ipv4.SetBase (“10.1.1.0”, “255.255.255.0”);

ipv4.Assign(staDevices);

Ipv4InterfaceContainer i = ipv4.Assign (apDevices);

تعریف و نصب نرم افزا Packet Sink بر روی نود Access Point جهت دریافت بسته های ارسالی

InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);

PacketSinkHelper recvSink("ns3::UdpSocketFactory", local);

sinkApp = recvSink.Install(c.Get (sinkNode));

sinkApp.Start (Seconds (0.0));

sinkApp.Stop (Seconds (30.0));

تعریف و نصب نرم افزار ONOFF بر روی نود Station جهت ارسال داده

InetSocketAddress remote = InetSocketAddress (i.GetAddress (sinkNode, 0), 80);

OnOffHelper onh(“ns3::UdpSocketFactory”, remote);

تعریف ویژگی های نرم افزار OnOff از جمله اندازه بسته ، نرخ ارسال و..

onh.SetAttribute (“PacketSize”, UintegerValue (2264));

onh.SetAttribute (“OnTime”, StringValue (“ns3::ConstantRandomVariable[Constant=1]”));

onh.SetAttribute (“OffTime”, StringValue (“ns3::ConstantRandomVariable[Constant=0]”));

onh.SetAttribute (“DataRate”,DataRateValue (DataRate (“24Mbps”)));

ApplicationContainer source\_apps =onh.Install(c.Get (sourceNode));

source\_apps.Start (Seconds (0.5));

source\_apps.Stop (Seconds (30.0));

Call تابع ReceivePacket در کلاس Packet Sink به صورت تابع اشاره شده در ابتدا ی پیاده سازی

Config::ConnectWithoutContext("/NodeList/0/ApplicationList/0/$ns3::PacketSink/Rx",MakeCallback (&ReceivePacket));

فراخوانی تابع اضافه کردن برچسب به بسته ها (اشاره شده در ابتدای پیاده سازی) با استفاده از QosTag

Ptr<OnOffApplication> onoffapp;

onoffapp = DynamicCast<OnOffApplication>(source\_apps.Get(0));

onoffapp->TraceConnectWithoutContext("Tx", MakeBoundCallback (&TagMarker, AC\_VI));

توقف شبیه سازی و اجرا و تخریب شبیه ساز پس از پایان کار

Simulator::Stop (Seconds (30.0));

Simulator::Run ();

Simulator::Destroy ();

return 0;

}

در این سناریو همانطور که مشاهده می گردد برای افزودن برچسب دسته ی ترافیک در کلاس Onoff Application به این شکل در سناریو عمل کرده ایم.

Ptr<OnOffApplication> onoffapp;

onoffapp = DynamicCast<OnOffApplication>(source\_apps.Get(0));

onoffapp->TraceConnectWithoutContext("Tx", MakeBoundCallback (&TagMarker, AC\_VI));

همچنین تابع زیر را نیز به سناریو اضافه کرده ایم

void

TagMarker (uint8\_t tid, Ptr<const Packet> packet)

{

QosTag qosTag;

qosTag.SetTid(tid);

packet->AddPacketTag (qosTag);

}

همچنین کلاس qostag.h را نیز به کلاس onoff اضافه نموده ایم .

با اضافه کردن این برچسب ها به بسته های ارسالی و استفاده از Qos Wifi Mac به طور پیش فرض 4 صف مجزا به ازای هر دسته ایجاد و بسته ها به آن صف ها بر اساس برچسب اضافه شده هدایت می گردند .

نکته : با تعریف لایه فیزیکی به طور پیشفرض مدل log distance propagation تنظیم و در کلاس Propagation loss model و در قسمت Reference مقدار مورد نظر در تعریف سناریو را در قسمت صفت های مدل log distance propagation model اضافه کرده ایم .

برای پیاده سازی این سناریو ابتدا تغییراتی را در سورس های NS3 به شکل زیر داده ایم .

پیاده سازی مفهوم TXOP :

ابتدا Txop های تعریف شده را به عنوان یک صفت از نوع زمان به شکل زیر به کلاس Wifi –Remote –Station-Manager اضافه می نماییم .

.AddAttribute ("TxopVi", "Txop time (in milliSeconds) for packets that have been tagged with AC\_VI"

"This value will not have any effect on some rate control algorithms.",

TimeValue (Seconds (0.003008)),

MakeTimeAccessor (&WifiRemoteStationManager::m\_txopVI),

MakeTimeChecker ())

.AddAttribute ("TxopVO", "Txop time (in milliSeconds) for packets that have been tagged with AC\_VO"

"This value will not have any effect on some rate control algorithms.",

TimeValue (Seconds (0.001507)),

MakeTimeAccessor (&WifiRemoteStationManager::m\_txopVO),

MakeTimeChecker ())

.AddAttribute ("TxopBK", "Txop time (in milliSeconds) for packets that have been tagged with AC\_VI"

"This value will not have any effect on some rate control algorithms.",

TimeValue (Seconds (0)),

MakeTimeAccessor (&WifiRemoteStationManager::m\_txopBK),

MakeTimeChecker ())

.AddAttribute ("TxopBE", "Txop time (in milliSeconds) for packets that have been tagged with AC\_VI"

"This value will not have any effect on some rate control algorithms.",

TimeValue (Seconds (0)),

MakeTimeAccessor (&WifiRemoteStationManager::m\_txopBE),

MakeTimeChecker ())

برای پیاده سازی مفهوم Txop ، همانظور که در فصل 4 پایان نامه مربوطه توضیح داده شده از مفهوم Addaptive Fragmentation Threshold استفاده کرده ایم به این شکل که بسته هایی که برای مدیریت ( قطعه بندی در صورت لزوم) وارد کلاس Wifi-Remote-Station-manager می شوند بر اساس بر چسب توضیح داده شده در مرحله قبل که در Onoff App به بسته ها اضافه می شد . اندازه ی Fragmentation Threshold متفاوتی برای آنها تخمین زده می شود .

تخمین : حاصل ضرب Txop هر دسته در نرخ ارسال لحظه ای استاندارد در لایه ی فیزیکی

در این شبیه سازی چون از استاندارد 802.11g استفاده شده است در این استاندارد 11 نرخ مختلف به شکل زیر داریم

1،2،5.5،6،9،11،18،24،36،48،54 واحد همه ی نرخ ها مگا بیت بر ثانیه می باشد .

بر اساس الگوریتم ها ی کنترل نرخ تنظیم شده . استفاده ی متفاوتی از این نرخ ها در فرستنده با توجه به شرایط کانال (Loss با افزایش فاصله ) می گردد و در هر لحظه ممکن است از نرخ متفاوتی برای ارسال استفاده شود .

در این شبیه سازی از 3 Wifi –manager استفاده شده است :

1-Cara

2-Aarf

3-Ideal

دو Wifi Manager اول با الگوریتم هایی بر اساس ارسال بسته و در صورت دریافت تصدیق آن شرایط کانال را خوب و در صورت عدم دریافت تصدیق شرایط کانال را بد تشخیص می دهند . و در شرایط خوب نرخ افزایش یافته و در شرایط بد نرخ ارسال را کاهش می دهند .

اما Wifi Manager سوم بر اساس میزان Snr سینگال ارسالی و دریافتی کار میکند و در صورتی که این سیگنال از حدی کمتر باشد وضعیت کانال بد و در غیر این صورت وضعیت کانال را خوب تشخیص می دهد .

در تر سیم نمودار ها از هر سه Wifi – Manager مطرح شده استفاده گشته است .

برای پیاده سازی Addaptive Fragmentation threshold مطابق با توضیحات پایان نامه معرفی شده در فصل چهارم کلاس wifi –Remote-Station و Edca txopn رو تغییر داده ایم

به این شکل که تابع Need Fragmetation در کلاس Wifi –remote –station-manager را به صورت زیر تغییر داده ایم :

این تابع نیاز به قطعه قطعه شدن بسته ی ورودی را با توجه به fragmentation threshold تشخیص می دهد .

bool

WifiRemoteStationManager::NeedFragmentation (Mac48Address address, const WifiMacHeader \*header,

Ptr<const Packet> packet)

{

QosTag q\_tag;

if (packet->PeekPacketTag(q\_tag))

{

if (q\_tag.GetTid()==0 && m\_txopBE.IsZero()==false){

DoSetTxopThreshhold(m\_txopBE);ac=0;

}

if (q\_tag.GetTid()==1 && m\_txopBK.IsZero()==false){

DoSetTxopThreshhold(m\_txopBK);ac=1;

}

if (q\_tag.GetTid()==2 && m\_txopVI.IsZero()==false){

DoSetTxopThreshhold(m\_txopVI);ac=2;

}

if (q\_tag.GetTid()==3 && m\_txopVO.IsZero()==false ){

DoSetTxopThreshhold(m\_txopVO);ac=3;

}

}

NS\_LOG\_FUNCTION (this << address << packet << \*header);

if (address.IsGroup ())

{

return false;

}

WifiRemoteStation \*station = Lookup (address, header);

ثبت نرخ ارسال لحظه ای در لایه ی فیزیکی و همچنین محاسبه متوسط نرخ ارسال

std::cout<<"rate "<< GetCurrentRate (wf1.GetUid())<<std::endl;

shomarande++;

average\_rate+=(GetCurrentRate (uid)/100000);

NS\_LOG\_UNCOND ("average rate"<< average\_rate/shomarande);

bool normally = (packet->GetSize () + header->GetSize () + WIFI\_MAC\_FCS\_LENGTH) > GetFragmentationThreshold ();

NS\_LOG\_DEBUG ("WifiRemoteStationManager::NeedFragmentation result: " << std::boolalpha << normally);

return DoNeedFragmentation (station, packet, normally);

}

void

WifiRemoteStationManager::DoSetFragmentationThreshold (uint32\_t threshold)

{

NS\_LOG\_FUNCTION (this << threshold);

بررسی حداقل اندازه ی هر قطعه

if (threshold < 256)

{

/\*

\* ASN.1 encoding of the MAC and PHY MIB (256 ... 8000)

\*/

NS\_LOG\_WARN ("Fragmentation threshold should be larger than 256. Setting to 256.");

m\_nextFragmentationThreshold = 256;

}

else

{

/\*

\* The length of each fragment shall be an even number of octets, except for the last fragment if an MSDU or

\* MMPDU, which may be either an even or an odd number of octets.

\*/

بررسی محدویت فرد بودن مقدار Threshold

if (threshold % 2 != 0)

{

NS\_LOG\_WARN ("Fragmentation threshold should be an even number. Setting to " << threshold - 1);

m\_nextFragmentationThreshold = threshold - 1;

}

else

{

m\_nextFragmentationThreshold = threshold;

}

}

}

در این تابع با توجه به ورود بسته ها بر اساس نوع دسته بندی برچسب آنها تابع Doset txop Threshold که این تابع را به کلاس Wifi-remote-station-manager با تغییر متناظر در کلاس .h آن اضافه کرده ایم . با مقدار ورودی Txop دسته ی مشخص فراخوانی می گردد . و همچنین دسته ی فعلی در متغییر سراسری Ac ذخیره می گردد .

این تابع با توجه به نرخ لحظه ای لایه فیزیکی و مقدار Txop ورودی مقدار Fragmentation threshold را تغییر میدهد .

به دست آوردن مقدار نرخ لحظه ای لایه ی فیزیکی :

برای این کار ابتدا شماره ی uid ، Wifi mode که به صورت لحظه ای حالت نرخ ارسال لایه ی فیزیکی را نشان می دهد ، را به دست آوردیم .

برای این کار ابتدا متغییر سراسری wf1 از نوع wifimode را تعریف می نماییم و سپس به تابع زیر دستور قرمز رنگ شده را اضافه نموده ایم تا حالت فعلی نرخ ارسال در لایه ی فیزیکی در متغییر uid که آن نیز به صورت سراسری تعریف شده است قرار بگیرد .

WifiTxVector

WifiRemoteStationManager::GetDataTxVector (Mac48Address address, const WifiMacHeader \*header,

Ptr<const Packet> packet, uint32\_t fullPacketSize)

{

NS\_LOG\_FUNCTION (this << address << \*header << packet << fullPacketSize);

if (address.IsGroup ())

{

WifiTxVector v;

v.SetMode (GetNonUnicastMode ());

v.SetTxPowerLevel (m\_defaultTxPowerLevel);

v.SetChannelWidth (m\_wifiPhy->GetChannelWidth ());

v.SetShortGuardInterval (m\_wifiPhy->GetGuardInterval ());

v.SetNss (1);

v.SetNess (0);

v.SetStbc (false);

return v;

}

if (!IsLowLatency ())

{

HighLatencyDataTxVectorTag datatag;

bool found;

found = ConstCast<Packet> (packet)->PeekPacketTag (datatag);

NS\_ASSERT (found);

//cast found to void, to suppress 'found' set but not used

//compiler warning

(void) found;

return datatag.GetDataTxVector ();

}

به دست آوردن Wifimode جاری

wf1=DoGetDataTxVector (Lookup (address, header), fullPacketSize).GetMode();

if(uidsetcheck==false){

uid=wf1.GetUid();

}

return DoGetDataTxVector (Lookup (address, header), fullPacketSize);

}

سپس با تعریف تابع زیر

uint32\_t

WifiRemoteStationManager::GetCurrentRate (uint32\_t id){

if (id==0)

return 1000000;

else if (id==1)

return 1000000;

else if (id==2)

return 2000000;

else if (id==3)

return 5500000;

else if (id==4)

return 6000000;

else if (id==5)

return 9000000;

else if (id==6)

return 11000000;

else if (id==7)

return 12000000;

else if (id==8)

return 18000000;

else if (id==9)

return 24000000;

else if (id==10)

return 36000000;

else if (id==11)

return 48000000;

else if (id==12)

return 54000000;

else

return 0;

}

و فرستادن مقدار uid به این تابع ، می توان نرخ ارسال را با توجه به تعریف خود استاندارد در کلاس yans Wifi phy و با توجه به حالت فعلی استخراج نمود .

نکته : به دلیل تفاوت Ideal Wifi manager در الگوریتم نسبت به دو Wifi manager دیگر برای استخراج uid لازم بود که دستور زیر را به تابع زیر در کلاس Ideal Wifi Manager و در هنگام تنظیم نرخ در این کلاس حالت نرخ ارسال در لایه فیزیکی به متغییر uid در کلاس Wifi –remote –Station Manager منتقل گردد .

بنابر این در کلاس Ideal Wifi Manager دستور رنگی شده را اضافه می کنیم .

WifiTxVector

IdealWifiManager::DoGetDataTxVector (WifiRemoteStation \*st, uint32\_t size)

{

IdealWifiRemoteStation \*station = (IdealWifiRemoteStation \*)st;

//We search within the Supported rate set the mode with the

//highest snr threshold possible which is smaller than m\_lastSnr

//to ensure correct packet delivery.

Double maxThreshold = 0.0;

WifiMode maxMode = GetDefaultMode ();

پیدا کردن بالا ترین نرخ سازگار تعریف شده در استاندارد با توجه به محدودیت Snr تعریف شده در الگوریتم

for (uint32\_t I = 0; I < GetNSupported (station); i++)

{

WifiMode mode = GetSupported (station, i);

double threshold = GetSnrThreshold (mode);

محدودیت تعریف شده در الگوریتم

if (threshold > maxThreshold

&& threshold < station->m\_lastSnr)

{

پیدا کردن بالا ترین نرخ سازگار و ارسال شماره ی Wifi-Mode آن به کلاس Wifi-Remote-Station-manager

setUidForIdeal(i);

maxThreshold = threshold;

maxMode = mode;

}

}

uint32\_t channelWidth = GetChannelWidth (station);

if (channelWidth > 20 && channelWidth != 22)

{

//avoid to use legacy rate adaptation algorithms for IEEE 802.11n/ac

channelWidth = 20;

}

بازگشت Wifi-TxVector

return WifiTxVector (maxMode, GetDefaultTxPowerLevel (), GetLongRetryCount (station), false, 1, 0, channelWidth, GetAggregation (station), false);

}

که این تابع اضافه شده در کلاس Wifi –remote –Station –Manager تنظیم مقدار Uid را انجام می دهد .

در نتیجه تابع Dosettxop Threshold به صورت زیر می گردد .

void

WifiRemoteStationManager::DoSetTxopThreshhold(Time txop1){

uint32\_t gam=GetCurrentRate (uid);

gam=gam/8;

SetFragmentationThreshold((txop1.GetSeconds()\*gam));

}

حال برای استخراج نمودار های 21 فصل 4 تابع Getfragment Size در کلاس Edcatxop را به صورت زیر تغییر می دهیم و مقدار fragment های ارسالی را log می زنیم و متوسط می گیریم .

uint32\_t

EdcaTxopN::GetFragmentSize (void)

{

NS\_LOG\_FUNCTION (this);

std::cout<<"khaaaaaar= "<<m\_stationManager->GetFragmentSize (m\_currentHdr.GetAddr1 (), &m\_currentHdr,

m\_currentPacket, m\_fragmentNumber)<<std::endl;

co++;

l=l+ m\_stationManager->GetFragmentSize (m\_currentHdr.GetAddr1 (), &m\_currentHdr,

m\_currentPacket, m\_fragmentNumber);

std::cout<<"kol meghdar= "<<l<<"shomare "<<co<<"taghsim "<<l/co<<std::endl;

return m\_stationManager->GetFragmentSize (m\_currentHdr.GetAddr1 (), &m\_currentHdr,

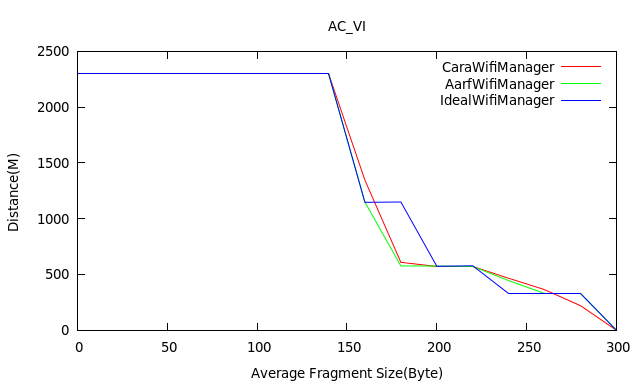
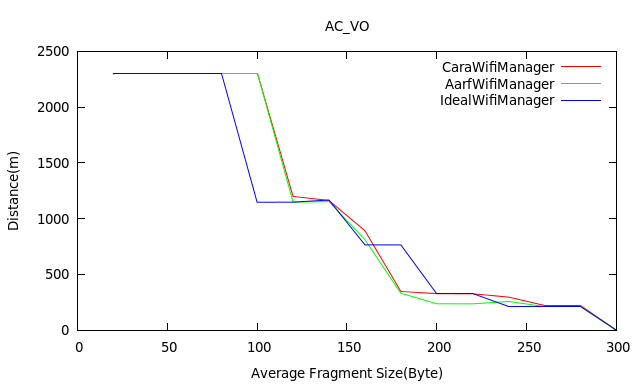
m\_currentPacket, m\_fragmentNumber);

}

نکته این تابع به ازای هر بسته اجرا می گردد .

با خروجی گرفتن . مقدار آخرین log ثبت شده مقدار متوسط اندازه ی قطعه ها را نشان می دهد .

نمودار های به دست آمده مشابه نمودار های 21 فصل 4 پایان نامه معرفی شده

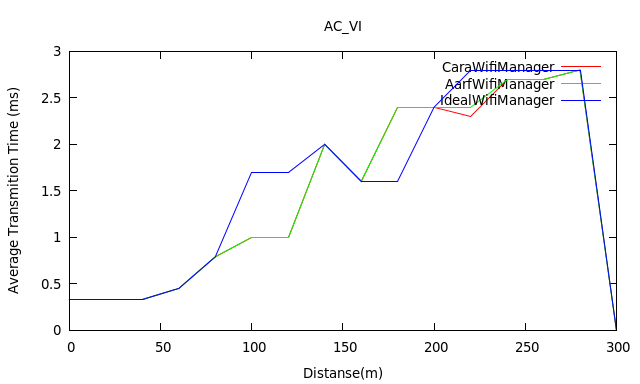


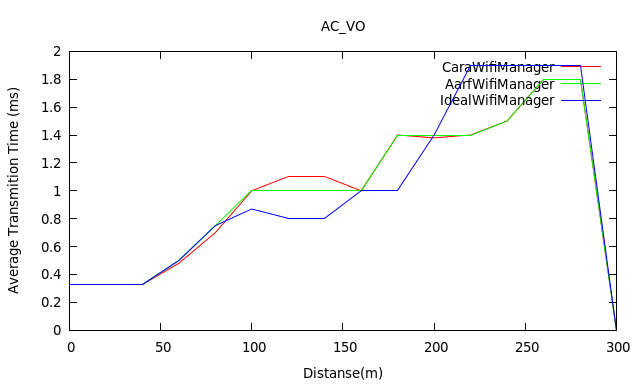
برای به دست آوردن نمودار های 22 پایان نامه ی معرفی شده متوسط زمان انقال به شکل زیر محاسبه شده

متوسط اندازه ی fragment ها تقسیم بر متوسط اندازه ی نرخ ارسال لایه فیزیکی در کل شبیه سازی

برای به دست آوردن متوسط نرخ ارسال در هنگامی که نرخ لحظه ای را به دست می آوردیم خود آن و متوسط لحظه ای آن را ثبت می کنیم . و مقدار آخری که به عنوان متوسط در شبیه سازی بر می گردد متوسط نرخ کل شبیه سازی می باشد .

بنابر این نمودار های شکل 22 را نیز به صورت زیر استخراج نموده ایم .





برای شبیه سازی مجدد نمودار های 23 پایان نامه معرفی شده با نصب Packet Sink Application بر روی Access Point مقدار Throughput کل سیستم را در لایه کاربرد توسط تابع زیر و دستور Connect در سناریو برای هر اجرا در شبیه سازی های بالا محاسبه کرده ایم .

دستور Connect

Config::ConnectWithoutContext("/NodeList/0/ApplicationList/0/$ns3::PacketSink/Rx",MakeCallback (&ReceivePacket));

تابع Receive که به ازای دریافت هر بسته در Access Point یک بار اجرا می گردد .

void ReceivePacket (Ptr<const Packet> packet, const Address &)

{

NS\_LOG\_UNCOND ("Received one packet!");

QosTag q\_tag;

if (packet->PeekPacketTag(q\_tag))

{

NS\_LOG\_UNCOND ("Packet Tag value " << (int)q\_tag.GetTid());

}

count++;

totalRxBytesCounter += packet->GetSize ();

totalRxBytesCounter+= 1;

if (count==1000)

std::cout<<"time "<<Simulator::Now ().GetSeconds();

NS\_LOG\_UNCOND ("count basteha daryaft shode " << count);

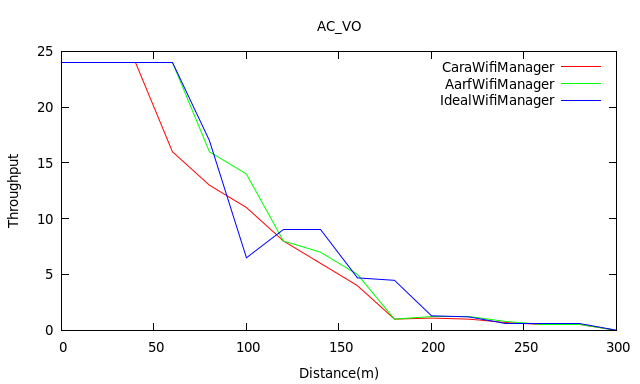
NS\_LOG\_UNCOND("th " << Simulator::Now ().GetSeconds()<< "\t"<<((totalRxBytesCounter\*8.0)/1000000)/Simulator::Now ().GetSeconds());

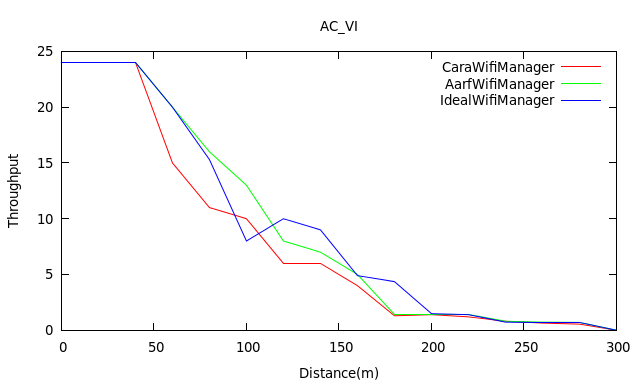
NS\_LOG\_UNCOND("average packet transmition time " << Simulator::Now ().GetSeconds()<< "\t"<<((Simulator::Now ().GetSeconds())\*1000)/count);

}

در این تابع تعداد بسته های دریافتی ، نوع بر چسب آنها در صورت عدم قطعه بندی و برداشته نشدن برچسب آنها و کارایی بر اساس مگا بیت بر ثانیه محاسبه می گردد .

بنابر این نمودار های زیر را استخراج کرده ایم .





برای نمودار های شکل 24 سناریو را به شکل زیر (قسمت قرمز رنگ) تغییر داده ایم و اینبار سه کاربرد که سه ترافیک با دسته های AC\_VI,AC\_VO,AC\_BK بر روی Station نصب کرده ایم . و به روش های مطرح شده در قبل نمودار های زیر را استخراج کرده ایم .

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/mobility-module.h"

#include "ns3/config-store-module.h"

#include "ns3/wifi-module.h"

#include "ns3/internet-module.h"

#include "ns3/olsr-helper.h"

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

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

#include "ns3/applications-module.h"

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

NS\_LOG\_COMPONENT\_DEFINE ("WifiSimpleAdhocGrid");

using namespace ns3;

uint32\_t count =0;

std::ofstream myfile ("project.txt",std::ofstream::binary);

uint32\_t average =0;

NodeContainer c;

uint32\_t totalRxBytesCounter = 0;

ApplicationContainer sinkApp;

void ReceivePacket (Ptr<const Packet> packet, const Address &)

{

NS\_LOG\_UNCOND ("Received one packet!");

QosTag q\_tag;

if (packet->PeekPacketTag(q\_tag))

{

NS\_LOG\_UNCOND ("Packet Tag value " << (int)q\_tag.GetTid());

}

count++;

totalRxBytesCounter += packet->GetSize ();

totalRxBytesCounter+= 1;

if (count==1000)

std::cout<<"time "<<Simulator::Now ().GetSeconds();

NS\_LOG\_UNCOND ("count basteha daryaft shode " << count);

NS\_LOG\_UNCOND("th " << Simulator::Now ().GetSeconds()<< "\t"<<((totalRxBytesCounter\*8.0)/1000000)/Simulator::Now ().GetSeconds());

NS\_LOG\_UNCOND("average packet transmition time " << Simulator::Now ().GetSeconds()<< "\t"<<((Simulator::Now ().GetSeconds())\*1000)/count);

}

void

TagMarker (uint8\_t tid, Ptr<const Packet> packet)

{

QosTag qosTag;

qosTag.SetTid(tid);

packet->AddPacketTag (qosTag);

}

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

{

std::string phyMode ("DsssRate1Mbps");

//uint32\_t packetSize = 2264; // bytes

uint32\_t numNodes = 2; // by default, 5x5

uint32\_t sinkNode = 0;

uint32\_t sourceNode = 1;

// disable fragmentation for frames below 2200 bytes

// Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold", StringValue ("1500"));

// turn off RTS/CTS for frames below 2200 bytes

// Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue ("2300"));

// Fix non-unicast data rate to be the same as that of unicast

// Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode", StringValue (phyMode));

c.Create (numNodes);

// The below set of helpers will help us to put together the wifi NICs we want

WifiHelper wifi=WifiHelper::Default ();

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();

// set it to zero; otherwise, gain will be added

// wifiPhy.Set ("RxGain", DoubleValue (-10) );

// ns-3 supports RadioTap and Prism tracing extensions for 802.11b

//wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT\_IEEE802\_11\_RADIO);

YansWifiChannelHelper wifiChannel=YansWifiChannelHelper::Default ();

//wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");

// wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");

//wifiChannel.AddPropagationLoss("ns3::LogDistancePropagationLossModel","Exponent", DoubleValue (40));

wifiPhy.SetChannel (wifiChannel.Create ());

// Add a QoS upper mac, and disable rate control

QosWifiMacHelper wifiMac = QosWifiMacHelper::Default ();

wifi.SetStandard (WIFI\_PHY\_STANDARD\_80211g);

wifi.SetRemoteStationManager ("ns3::IdealWifiManager");

// wifi.SetRemoteStationManager ("ns3::IdealWifiManager", "RtsCtsThreshold", UintegerValue (1000));

Ssid ssid = Ssid ("ns-3-ssid");

wifiMac.SetType ("ns3::StaWifiMac",

"Ssid", SsidValue (ssid),

"ActiveProbing", BooleanValue (false));

//wifiMac.SetMsduAggregatorForAc (AC\_BK, "ns3::MsduStandardAggregator", "MaxAmsduSize", UintegerValue (3839));

// wifiMac.SetMsduAggregatorForAc (AC\_VO, "ns3::MsduStandardAggregator", "MaxAmsduSize", UintegerValue (7935));

//wifiMac.SetMsduAggregatorForAc (AC\_VI, "ns3::MsduStandardAggregator", "MaxAmsduSize", UintegerValue (7935));

// install Wifi on Wifistatnodes

NetDeviceContainer staDevices;

staDevices = wifi.Install (wifiPhy, wifiMac, c.Get(1));

wifiMac.SetType ("ns3::ApWifiMac",

"Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;

apDevices = wifi.Install (wifiPhy, wifiMac, c.Get(0));

/\* Setting mobility model \*/

MobilityHelper mobility;

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

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

positionAlloc->Add (Vector (120, 0.0, 0.0));

mobility.SetPositionAllocator (positionAlloc);

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

mobility.Install (c.Get(0));

mobility.Install (c.Get(1));

InternetStackHelper internet;

internet.Install (c);

Ipv4AddressHelper ipv4;

NS\_LOG\_INFO ("Assign IP Addresses.");

ipv4.SetBase ("10.1.1.0", "255.255.255.0");

ipv4.Assign(staDevices);

Ipv4InterfaceContainer i = ipv4.Assign (apDevices);

InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);

PacketSinkHelper recvSink("ns3::UdpSocketFactory", local);

sinkApp = recvSink.Install(c.Get (sinkNode));

sinkApp.Start (Seconds (0.0));

sinkApp.Stop (Seconds (30.0));

InetSocketAddress remote = InetSocketAddress (i.GetAddress (sinkNode, 0), 80);

OnOffHelper onh("ns3::UdpSocketFactory", remote);

onh.SetAttribute ("PacketSize", UintegerValue (2264));

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

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

onh.SetAttribute ("DataRate",DataRateValue (DataRate ("24Mbps")));

ApplicationContainer source\_apps =onh.Install(c.Get (sourceNode));

source\_apps.Start (Seconds (0.5));

source\_apps.Stop (Seconds (30.0));

Ptr<OnOffApplication> onoffapp;

onoffapp = DynamicCast<OnOffApplication>(source\_apps.Get(0));

onoffapp->TraceConnectWithoutContext("Tx", MakeBoundCallback (&TagMarker, AC\_VO));

OnOffHelper onh2("ns3::UdpSocketFactory", remote);

onh.SetAttribute ("PacketSize", UintegerValue (2264));

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

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

onh.SetAttribute ("DataRate",DataRateValue (DataRate ("24Mbps")));

ApplicationContainer source\_apps2 =onh2.Install(c.Get (sourceNode));

source\_apps2.Start (Seconds (0.5));

source\_apps2.Stop (Seconds (30.0));

Ptr<OnOffApplication> onoffapp2;

onoffapp2 = DynamicCast<OnOffApplication>(source\_apps2.Get(0));

onoffapp2->TraceConnectWithoutContext("Tx", MakeBoundCallback (&TagMarker, AC\_VI));

OnOffHelper onh3("ns3::UdpSocketFactory", remote);

onh.SetAttribute ("PacketSize", UintegerValue (2264));

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

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

onh.SetAttribute ("DataRate",DataRateValue (DataRate ("24Mbps")));

ApplicationContainer source\_apps3 =onh3.Install(c.Get (sourceNode));

source\_apps3.Start (Seconds (0.5));

source\_apps3.Stop (Seconds (30.0));

Ptr<OnOffApplication> onoffapp3;

onoffapp3 = DynamicCast<OnOffApplication>(source\_apps3.Get(0));

onoffapp3->TraceConnectWithoutContext("Tx", MakeBoundCallback (&TagMarker, AC\_BK));

Config::ConnectWithoutContext("/NodeList/0/ApplicationList/0/$ns3::PacketSink/Rx",MakeCallback (&ReceivePacket));

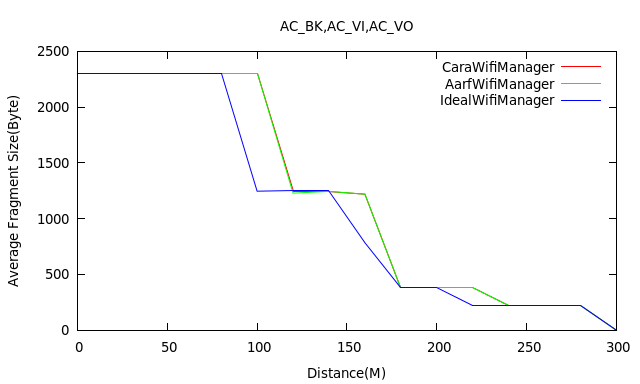
Simulator::Stop (Seconds (30.0));

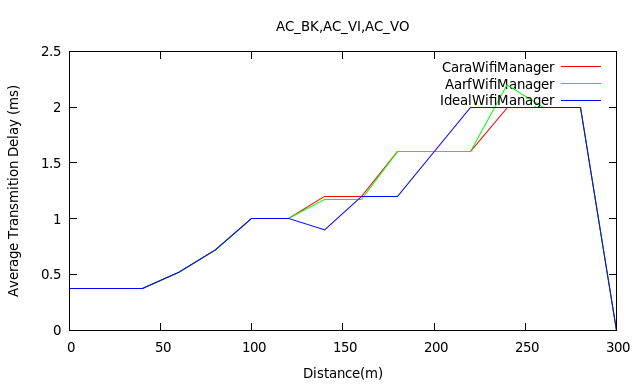
Simulator::Run ();

Simulator::Destroy ();

return 0;

}





حال با پیاده سازی قسمت Impact of Edca Parameter پیش می رویم :

برای پیاده سازی نمودار 32 سناریو ی زیر را با دو نود (یکی Station با نصب OnoffAppو یکی Access Point با نصب Packet Sink) و تولید ترافیک در شرایط بافر پر (نرخ بالا ) پیاده کرده ایم .

برای تغییر مقدار Cwmin و Cwmax میتوان از سناریو با کد زیر این مقادیر را تنظیم کرد . و همچنین روش دوم می توان این مقادیر را از طریق کلاس Regular –Wifi-Mac با تغییر مقادیر استاندارد نیز این کار را انجام داد .

Ptr<NetDevice> dev = node->GetDevice(0);

Ptr<WifiNetDevice> wifi\_dev = DynamicCast<WifiNetDevice>(dev);

Ptr<WifiMac> mac = wifi\_dev->GetMac();

PointerValue ptr;

mac->GetAttribute(“DcaTxop”, ptr);

Ptr<DcaTxop> dca = ptr.Get<DcaTxop>();

dca->SetMinCw(minCw);

dca->SetMaxCw(maxCw);

برای به دست آوردن مقدار متوسط تاخیر بسته ها از تقسیم تعداد بسته ها ی رسیده شده در طول زمان شبیه سازی بر زمان شبیه سازی در واحد میلی ثانیه استفاده کرده ایم . از سناریو ی زیر که در قسمت های قبل قسمت های آن توضیح داده شد استفاده کرده ایم .

#include "ns3/core-module.h"

#include "ns3/point-to-point-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/dca-txop.h"

using namespace ns3;

// declare For Calculate Cw

uint32\_t cwold=15;

uint32\_t countcollision=0;

ApplicationContainer sinkApps;

std::ofstream myfile ("p.txt",std::ofstream::binary);

uint32\_t totalRxBytesCounter = 0;

// Add All Cw

uint32\_t cw\_all=0;

uint32\_t count=0;

ApplicationContainer sinkApp;

NodeContainer wifiStaNodes;

// Calculate Colition and Avrage Cw

void ReceivePacket (Ptr<const Packet> packet, const Address &)

{

NS\_LOG\_UNCOND ("Received one packet!");

count++;

totalRxBytesCounter += packet->GetSize ();

totalRxBytesCounter+= 1;

NS\_LOG\_UNCOND ("count basteha daryaft shode " << count);

NS\_LOG\_UNCOND ("Average Delay "<<count/(Simulator::Now ().GetSeconds()\*1000));

NS\_LOG\_UNCOND ("count basteha daryaft shode " <<Simulator::Now ().GetSeconds());

//NS\_LOG\_UNCOND("th " << Simulator::Now ().GetSeconds()<< "\t"<<((totalRxBytesCounter\*8.0)/1000000)/Simulator::Now ().GetSeconds());

//NS\_LOG\_UNCOND("average packet transmition time " << Simulator::Now ().GetSeconds()<< "\t"<<((Simulator::Now ().GetSeconds())\*1000)/count);

}

int

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

{

// bool verbose = true;

// Number of Server And Wifi

uint32\_t nserver = 1;

uint32\_t nWifi = 1;

// Definition server node

NodeContainer serverNodes;

serverNodes.Create (nserver);

//CsmaHelper serverh;

//serverh.SetChannelAttribute ("DataRate", StringValue ("10Mbps"));

//serverh.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

// NetDeviceContainer serverDevices;

// serverDevices = serverh.Install (serverNodes);

// Defiition Of Wifi Nodes

wifiStaNodes.Create (nWifi);

NodeContainer wifiApNode =serverNodes.Get(0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();

YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();

phy.SetChannel (channel.Create ());

//The SetRemoteStationManager method tells the helper the type of rate control algorithm to use. Here, it is asking the helper to use the AARF algorithm

WifiHelper wifi = WifiHelper::Default ();

wifi.SetRemoteStationManager ("ns3::CaraWifiManager");

wifi.SetStandard (WIFI\_PHY\_STANDARD\_80211g);

// wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",

// "DataMode", StringValue ("OfdmRate2Mbps"));

// wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();

//creates an 802.11 service set identifier (SSID) object that will be used to set the value of the â€œSsidâ€ Attribute of the MAC layer implementation.

//The particular kind of MAC layer is specified by Attribute as being of the "ns3::NqstaWifiMac" type.

//This means that the MAC will use a â€œnon-QoS stationâ€ (nqsta) state machine. Finally, the â€œActiveProbingâ€ Attribute is set to false.

Ssid ssid = Ssid ("ns-3-ssid");

mac.SetType ("ns3::StaWifiMac",

"Ssid", SsidValue (ssid),

"ActiveProbing", BooleanValue (false));

// install Wifi on Wifistatnodes

NetDeviceContainer staDevices;

staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac",

"Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;

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

/\* Setting mobility model \*/

MobilityHelper mobility;

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

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

positionAlloc->Add (Vector (150, 0.0, 0.0));

mobility.SetPositionAllocator (positionAlloc);

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

mobility.Install (wifiStaNodes.Get(0));

mobility.Install (wifiApNode.Get(0));

// Define and install Internet Stack On All Nodes

InternetStackHelper stack;

//stack.Install (serverNodes);

stack.Install (wifiApNode);

stack.Install (wifiStaNodes);

// Define Ip Address

Ipv4AddressHelper address;

address.SetBase ("10.1.3.0", "255.255.255.0");

address.Assign (staDevices);

Ipv4InterfaceContainer i=address.Assign (apDevices);

InetSocketAddress remote = InetSocketAddress (i.GetAddress (0, 0), 80);

OnOffHelper onh("ns3::UdpSocketFactory", remote);

onh.SetAttribute ("PacketSize", UintegerValue (200));

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

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

onh.SetAttribute ("DataRate",DataRateValue (DataRate ("10Mbps")));

InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);

PacketSinkHelper recvSink("ns3::UdpSocketFactory", local);

sinkApp = recvSink.Install(serverNodes.Get (0));

sinkApp.Start (Seconds (0.0));

sinkApp.Stop (Seconds (100.0));

// install Source app on all wifi nodes

ApplicationContainer clientApps ;

clientApps = onh.Install (wifiStaNodes.Get (0));

clientApps.Start (Seconds (0.0));

clientApps.Stop (Seconds (100.0));

// Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

// scheduling For All Wifi Nodes

Config::ConnectWithoutContext("/NodeList/0/ApplicationList/0/$ns3::PacketSink/Rx",MakeCallback (&ReceivePacket));

Simulator::Stop (Seconds (20.0));

Simulator::Run ();

Simulator::Destroy ();

//std::cout<<cw\_all<<std::endl;

//std::cout<<cw\_count<<std::endl;

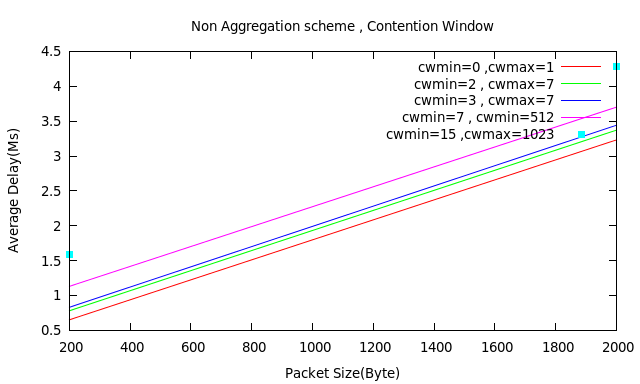
//std::cout<<"-----------"<<std::endl;

//long avg=cw\_all/cw\_count;

//std::cout<<avg<<std::endl;

return 0;

}

در نتیجه نمودار زیر را استخراج کرده ایم 

برای نمودار 33

در این شبیه سازی مقدار Goodput را برابر با ترافیک رسیده به لایه کاربرد در نظر گرفته ایم . برای پیاده سازی این نمودار سناریو استفاده شده در نمودار 21 را بر اساس یک Ac با txop های مختلف از طریق تغییر مقدار صفت Txop تعریف شده در کلاس Wifi-remote-station-manager را مجددا مورد استفاده قرار داده ایم .

مقدار تاخیر را نیز به همان شکل قبل حساب می کنیم .

در نتیجه نمودار های زیر را استخراج نموده ایم .

