به نام خدا

گزارش کار پروژه پایانی درس شبکه های بی سیم

عنوان پروژه :

شبیه سازی ۶ نمودار از نمودار های تز

Performance study of ۸۰۲,۱۱n WLAN and MAC enhancements in ns-۳

استاد مربوطه : دکتر محمد نصیری

محمد پیشدار

نرگس رضایی

دانشگاه بو علی سینا

تابستان ۹۵

فهرست مطالب

ما در این پژوهش موفق شدیم حدود ۱۹ نمودار از نمودار های فصل ۴ را که به تعداد ۱۳ 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 ۴ پایان نامه معرفی شده شروع و برای استخراج نمودار های ۲۱ ، ۲۲ و ۲۳ 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 OnOff Application نصب شده است که ترافیک از نوع Udp و با دسته بندی های مختلف (Voice,Video,Best Effort,Background)را با نرخ mbps۲۴ و با اندازه بسته ۲۲۶۴ بایت تولید و به نود Access point که بر روی آن نیز یک PacketSink App با سوکت udp و پورت ۸۰ نصب شده است ارسال می کند . این شبیه سازی بر اساس استاندارد ۸۰۲,۱۱ g با استفاده از مدل Log distance propagation loss در باند فرکانسی GHZ۲,۴ و با تنظیم Reference loss برابر با ۴۰,۰۴۷ db و در مدت ۳۰ 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	۱,۵
AC_VI	۳
AC_BE	۰
AC_BK	۰

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

```
#include "ns۳/core-module.h"
#include "ns۳/network-module.h"
#include "ns۳/mobility-module.h"
#include "ns۳/config-store-module.h"
#include "ns۳/wifi-module.h"
#include "ns۳/internet-module.h"
#include "ns۳/ipv۴-static-routing-helper.h"
#include "ns۳/ipv۴-list-routing-helper.h"
#include "ns۳/applications-module.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
```

```
NS_LOG_COMPONENT_DEFINE ("WifiSimpleAdhocGrid");

using namespace ns۳;
```

تعریف متغییر های سراسری برای انجام محاسبات

```
uint۳۲_t count =۰;
uint۳۲_t average =۰;
 NodeContainer c;
 uint۳۲_t totalRxBytesCounter = ۰;

 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+= ۱;
if (count==۱۰۰۰)
    std::cout<<"time "<<Simulator::Now ().GetSeconds();


 NS_LOG_UNCOND ("count basteha daryaft shode " << count);
 NS_LOG_UNCOND("th " << Simulator::Now ().GetSeconds()<<
"\t"<<((totalRxBytesCounter*۸,۰)/۱۰۰۰۰۰)/Simulator::Now
().GetSeconds());
 NS_LOG_UNCOND("average packet transmition time " << Simulator::Now
().GetSeconds()<< "\t"<<((Simulator::Now ().GetSeconds())*۱۰۰۰)/count);


}
```

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

```
void
TagMarker (uint۸_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 ("DsssRate۱Mbps");
  //uint۳۲_t packetSize = ۲۲۶۴; // bytes
  uint۳۲_t numNodes = ۲;  // by default, ۵x۵
  uint۳۲_t sinkNode = ۰;
  uint۳۲_t sourceNode = ۱;
```

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_IEEE۸۰۲_۱۱_RADIO);
```

ایجاد و تنظیم کانال و مدل Prppagation Loss برای آن

```
  YansWifiChannelHelper wifiChannel=YansWifiChannelHelper::Default ();
//wifiChannel.SetPropagationDelay
("ns۳::ConstantSpeedPropagationDelayModel");
// wifiChannel.AddPropagationLoss ("ns۳::FriisPropagationLossModel");
//wifiChannel.AddPropagationLoss("ns۳::LogDistancePropagationLossModel"
,"Exponent", DoubleValue (۴۰));
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_۸۰۲۱۱g);
```

تنظیم Wifi Manager

```
wifi.SetRemoteStationManager ("ns۳::AarfWifiManager");
// wifi.SetRemoteStationManager ("ns۳::IdealWifiManager",
"RtsCtsThreshold", UintegerValue (۱۰۰۰));
```

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

```
  Ssid ssid = Ssid ("ns-۳-ssid");
  wifiMac.SetType ("ns۳::StaWifiMac",
               "Ssid", SsidValue (ssid),
               "ActiveProbing", BooleanValue (false));

  // install Wifi on Wifistatnodes
  NetDeviceContainer staDevices;
  staDevices = wifi.Install (wifiPhy, wifiMac,  c.Get(۱));

  wifiMac.SetType ("ns۳::ApWifiMac",
               "Ssid", SsidValue (ssid));

  NetDeviceContainer apDevices;
  apDevices = wifi.Install (wifiPhy, wifiMac, c.Get(۰));
```

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 (۰,۰, ۰,۰, ۰,۰));
  positionAlloc->Add (Vector (۲۶۰, ۰,۰, ۰,۰));
  mobility.SetPositionAllocator (positionAlloc);

  mobility.SetMobilityModel ("ns۳::ConstantPositionMobilityModel");

  mobility.Install (c.Get(۰));
  mobility.Install (c.Get(۱));
```

نصب پشته ی پروتکی اینترنت بر روی نود ها

```
  InternetStackHelper internet;
  internet.Install ©;
```

تخصیص آدرس IP به نود ها

```
  Ipv۴AddressHelper ipv۴;
  NS_LOG_INFO ("Assign IP Addresses.");
  ipv۴.SetBase ("۱۰,۱,۱,۰", "۲۵۵,۲۵۵,۲۵۵,۰");
  ipv۴.Assign(staDevices);
  Ipv۴InterfaceContainer i = ipv۴.Assign (apDevices);
```

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

```
  InetSocketAddress local = InetSocketAddress (Ipv۴Address::GetAny (),
۸۰);
  PacketSinkHelper recvSink("ns۳::UdpSocketFactory", local);
```

```
  sinkApp = recvSink.Install(c.Get (sinkNode));
   sinkApp.Start (Seconds (۰,۰));
   sinkApp.Stop (Seconds (۳۰,۰));
```

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

```
  InetSocketAddress remote = InetSocketAddress (i.GetAddress
(sinkNode, ۰), ۸۰);
  OnOffHelper onh("ns۳::UdpSocketFactory", remote);
```

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

```
  onh.SetAttribute ("PacketSize",  UintegerValue (۲۲۶۴));
  onh.SetAttribute ("OnTime", StringValue
("ns۳::ConstantRandomVariable[Constant=۱]"));
  onh.SetAttribute ("OffTime", StringValue
("ns۳::ConstantRandomVariable[Constant=۰]"));
  onh.SetAttribute ("DataRate",DataRateValue (DataRate ("۲۴Mbps")));
  ApplicationContainer source_apps =onh.Install(c.Get (sourceNode));
  source_apps.Start (Seconds (۰,۵));
  source_apps.Stop (Seconds (۳۰,۰));
```

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

```
Config::ConnectWithoutContext("/NodeList/۰/ApplicationList/۰/$ns۳::Pack
etSink/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(۰));
  onoffapp->TraceConnectWithoutContext("Tx",  MakeBoundCallback
(&TagMarker, AC_VI));
```

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

```
  Simulator::Stop (Seconds (۳۰,۰));
  Simulator::Run ();
  Simulator::Destroy ();

  return ۰;
   }
```

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(۰));
```

```
  onoffapp->TraceConnectWithoutContext("Tx",  MakeBoundCallback
(&TagMarker, AC_VI));
```

همچنین تابع زیر را نیز به سناریو اضافه کرده ایم

```
void
TagMarker (uint۸_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 ۴ 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 NS۳ 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 (۰,۰۰۳۰۰۸)),
          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 (۰,۰۰۱۵۰۷)),
                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 (۰)),
                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 (۰)),
                MakeTimeAccessor
(&WifiRemoteStationManager::m_txopBE),
                MakeTimeChecker ())
```

برای پیاده سازی مفهوم Txop ، همانظور که در فصل ۴ 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 ۸۰۲,۱۱g استفاده شده است در این استاندارد ۱۱ نرخ مختلف به شکل زیر داریم

۱،۲،۵,۵،۶،۹،۱۱،۱۸،۲۴،۳۶،۴۸،۵۴ 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 ۳ Wifi –manager استفاده شده است :

۱- Cara

۲-Aarf

۳-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 (Mac۴۸Address address,
const WifiMacHeader *header,
                                             Ptr<const Packet> packet)
{
    QosTag q_tag;
    if (packet->PeekPacketTag(q_tag))
    {
        if (q_tag.GetTid()==۰ && m_txopBE.IsZero()==false){
DoSetTxopThreshhold(m_txopBE);ac=۰;
        }
        if (q_tag.GetTid()==۱ && m_txopBK.IsZero()==false){
DoSetTxopThreshhold(m_txopBK);ac=۱;
        }
        if (q_tag.GetTid()==۲ && m_txopVI.IsZero()==false){
DoSetTxopThreshhold(m_txopVI);ac=۲;
        }
        if (q_tag.GetTid()==۳ && m_txopVO.IsZero()==false ){
DoSetTxopThreshhold(m_txopVO);ac=۳;
        }
    }

  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 (wf۱.GetUid())<<std::endl;
```

```
  shomarande++;
average_rate+=(GetCurrentRate (uid)/١٠٠٠٠٠);
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 (uint٣٢_t
threshold)
{
  NS_LOG_FUNCTION (this << threshold);
```

بررسی حداقل اندازه ی هر قطعه

```
  if (threshold < ٢٥٦)
    {
      /*
       * ASN.١ encoding of the MAC and PHY MIB (٢٥٦ ... ٨٠٠٠)
       */
      NS_LOG_WARN ("Fragmentation threshold should be larger than ٢٥٦.
Setting to ٢٥٦.");
      m_nextFragmentationThreshold = ٢٥٦;
    }
  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 % ٢ != ٠)
        {
          NS_LOG_WARN ("Fragmentation threshold should be an even
number. Setting to " << threshold - ١);
          m_nextFragmentationThreshold = threshold - ١;
        }
      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 wf۱ 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 (Mac۴۸Address address, const
WifiMacHeader *header,
                                           Ptr<const Packet> packet,
uint۳۲_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 (۱);
      v.SetNess (۰);
      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

```
 wf\=DoGetDataTxVector (Lookup (address, header),
fullPacketSize).GetMode();
 if(uidsetcheck==false){
 uid=wf\.GetUid();
}


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

سپس با تعریف تابع زیر

```
uint۳۲_t
WifiRemoteStationManager::GetCurrentRate (uint۳۲_t id){
    if (id==۰)
return ۱۰۰۰۰۰;
else if (id==۱)
return ۱۰۰۰۰۰;
    else if (id==۲)
return ۲۰۰۰۰۰;
    else if (id==۳)
return ۵۵۰۰۰۰;
    else if (id==۴)
return ۶۰۰۰۰۰;
    else if (id==۵)
return ۹۰۰۰۰۰;
    else if (id==۶)
return ۱۱۰۰۰۰۰;
    else if (id==۷)
return ۱۲۰۰۰۰۰;
    else if (id==۸)
return ۱۸۰۰۰۰۰;
```

```
    else if (id==۹)
return ۲۴۰۰۰۰۰;
    else if (id==۱۰)
return ۳۶۰۰۰۰۰;
    else if (id==۱۱)
return ۴۸۰۰۰۰۰;
    else if (id==۱۲)
return ۵۴۰۰۰۰۰;
    else
        return ۰;
}
```

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, uint۳۲_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 = ۰.۰;
  WifiMode maxMode = GetDefaultMode ();
```

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

```
  for (uint۳۲_t I = ۰; 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;
       }
    }
  uint۳۲_t channelWidth = GetChannelWidth (station);
  if (channelWidth > ۲۰ && channelWidth != ۲۲)
    {
      //avoid to use legacy rate adaptation algorithms for IEEE
۸۰۲,۱۱n/ac
      channelWidth = ۲۰;
                                                              }
```

بازگشت Wifi-TxVector

```
  return WifiTxVector (maxMode, GetDefaultTxPowerLevel (),
GetLongRetryCount (station), false, ۱, ۰, 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 txop۱){
uint۳۲_t gam=GetCurrentRate (uid);
gam=gam/۸;
SetFragmentationThreshold((txop۱.GetSeconds()*gam));


}
```

حال برای استخراج نمودار های ۲۱ فصل ۴ 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 .

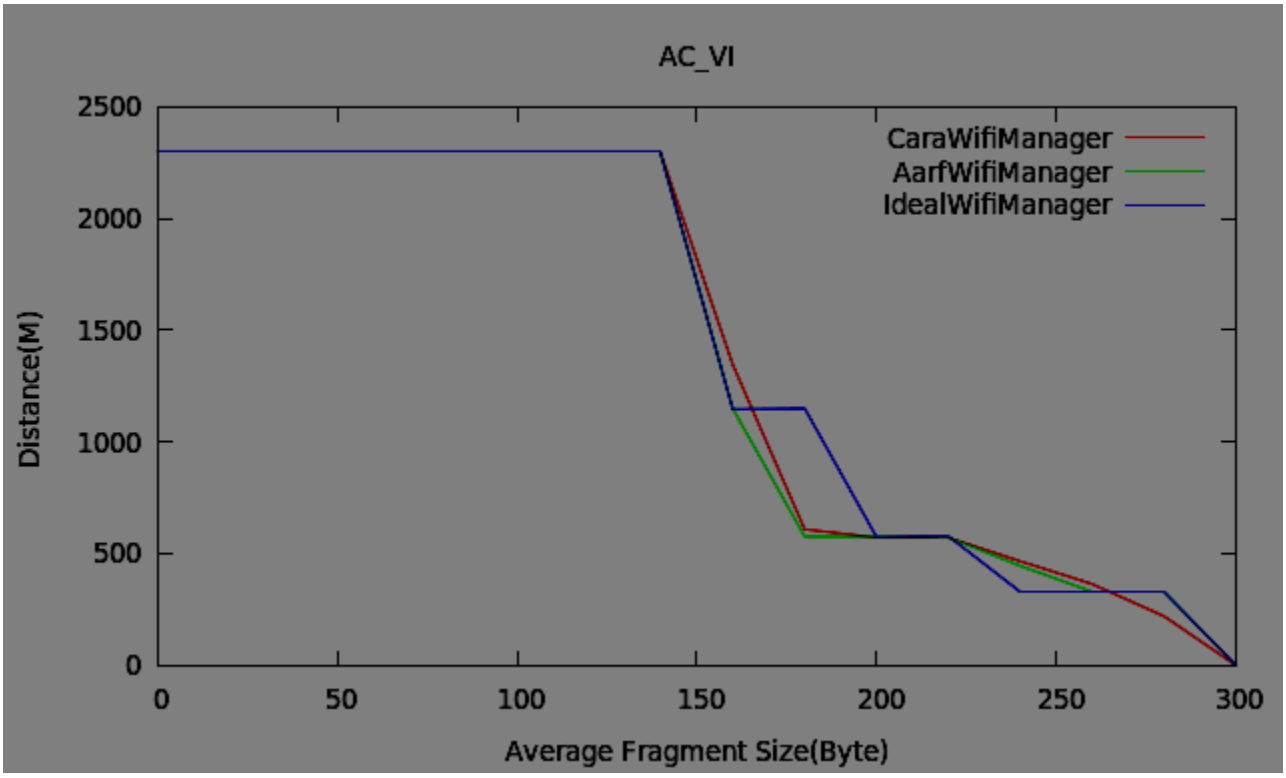
```
uint۳۲_t
EdcaTxopN::GetFragmentSize (void)
{
  NS_LOG_FUNCTION (this);
std::cout<<"khaaaaaar= "<<m_stationManager->GetFragmentSize
(m_currentHdr.GetAddr۱ (), &m_currentHdr,
```

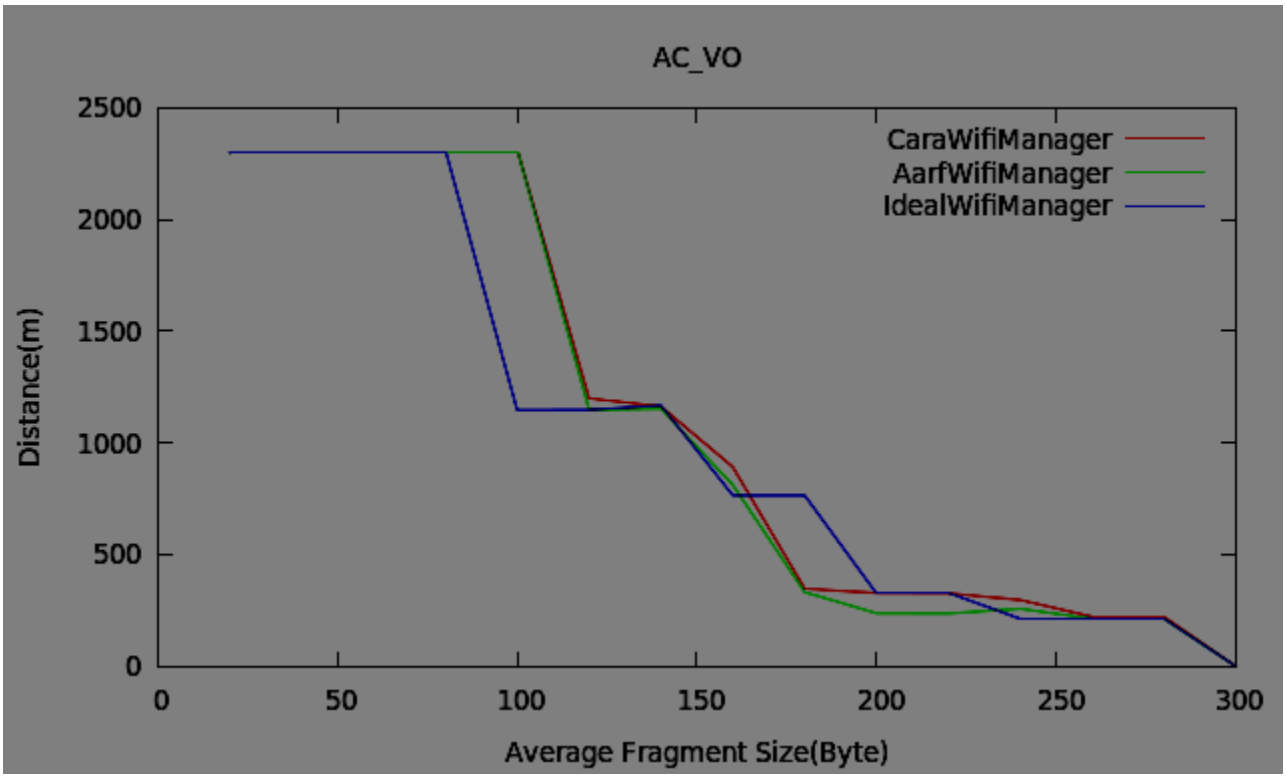
```
                                                    m_currentPacket,
m_fragmentNumber)<<std::endl;

co++;

l=l+ m_stationManager->GetFragmentSize (m_currentHdr.GetAddr۱ (),
&m_currentHdr,
                                        m_currentPacket,
m_fragmentNumber);


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

  return m_stationManager->GetFragmentSize (m_currentHdr.GetAddr۱ (),
&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 ۲۱ فصل ۴ پایان نامه معرفی شده


AC_VI
2500
2000
1500
1000
500
0
Distance(M)
0
50
100
150
200
250
300
Average Fragment Size(Byte)
CaraWifiManager
AarfWifiManager
IdealWifiManager

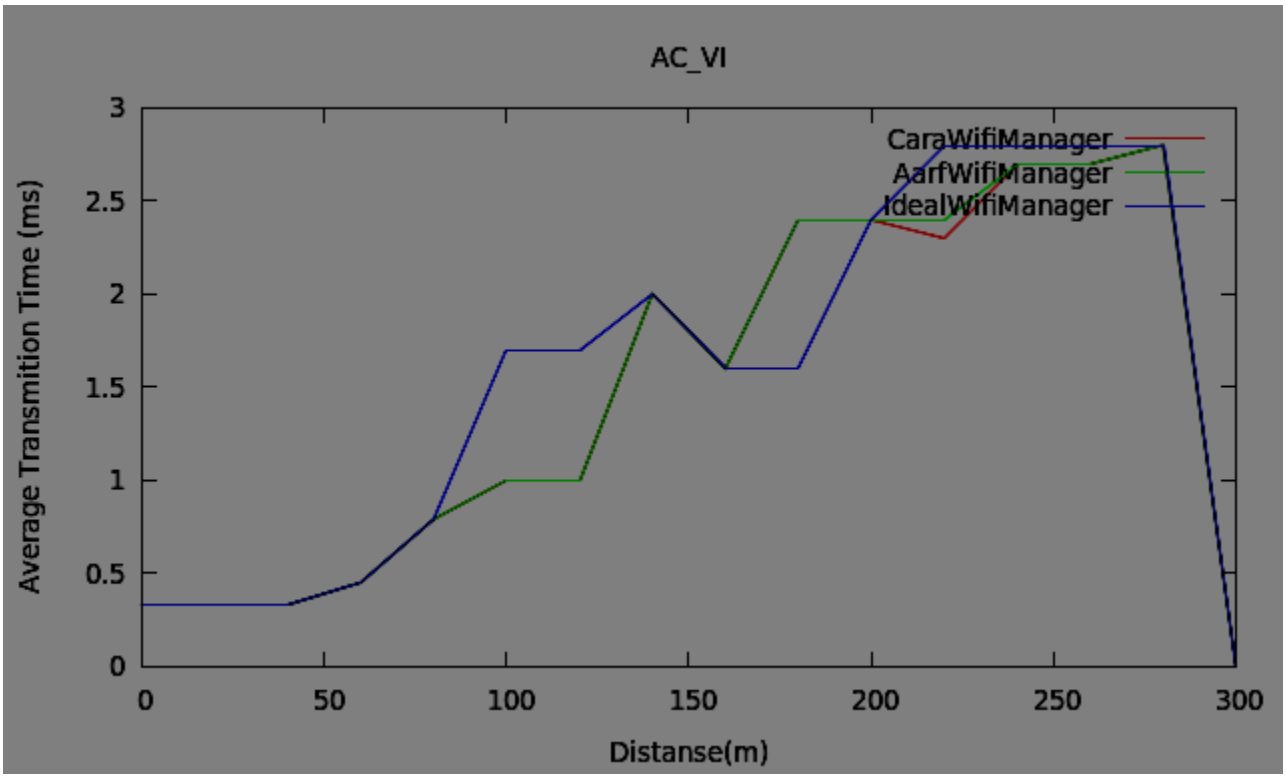

AC_VO
2500
2000
1500
1000
500
0
Distance(m)
0
50
100
150
200
250
300
Average Fragment Size(Byte)
CaraWifiManager
AarfWifiManager
IdealWifiManager

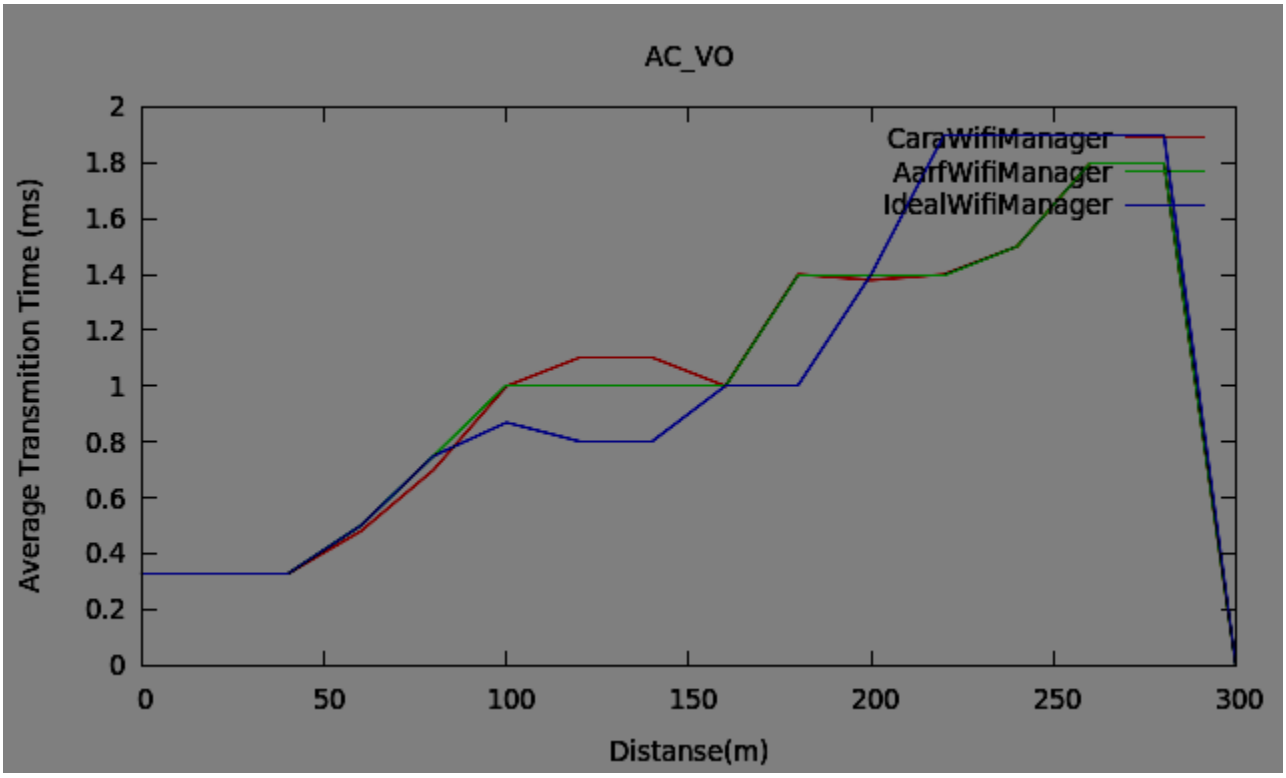
برای به دست آوردن نمودار های ۲۲ پایان نامه ی معرفی شده متوسط زمان انقال به شکل زیر محاسبه شده

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 ۲۲ را نیز به صورت زیر استخراج نموده ایم .





برای شبیه سازی مجدد نمودار های ۲۳ 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/۰/ApplicationList/۰/$ns۳::Pack
etSink/Rx",MakeCallback (&ReceivePacket));
```

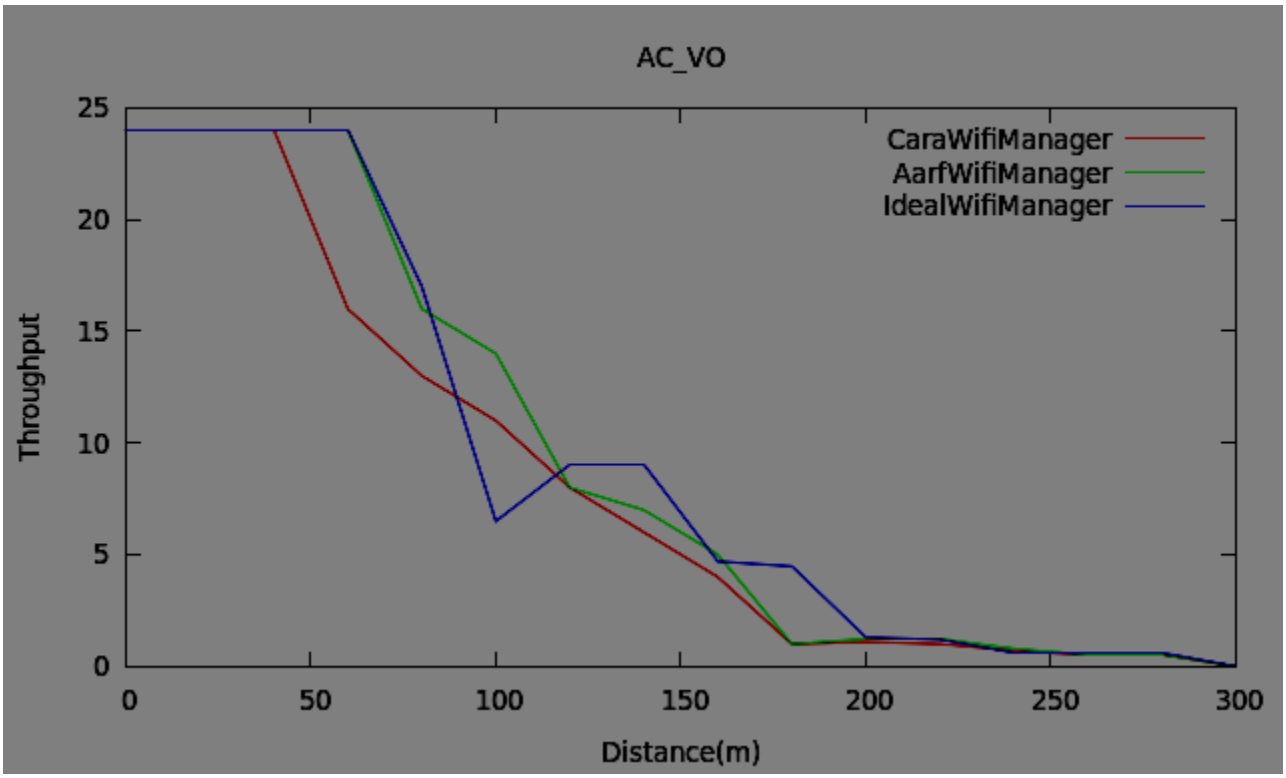
تابع Receive که به ازای دریافت هر بسته در Access Point یک بار اجرا می گردد .

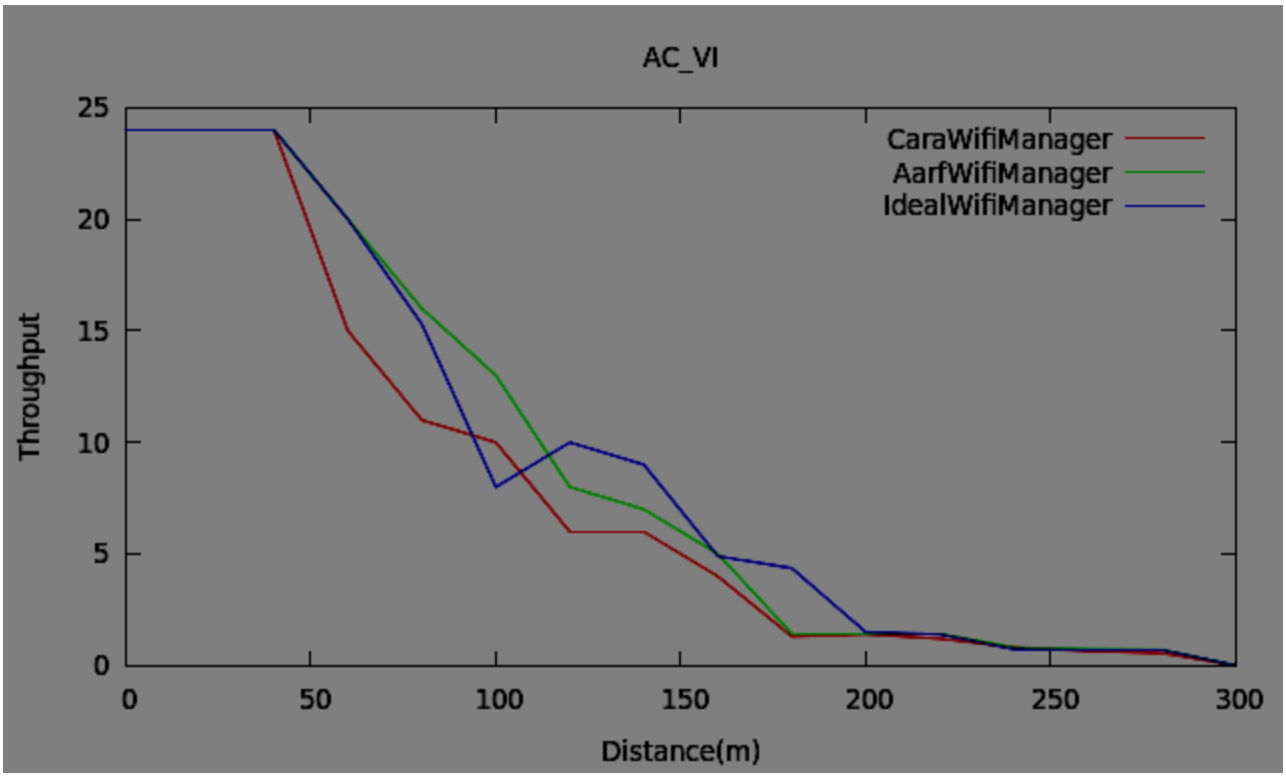
```
void ReceivePacket (Ptr<const Packet> packet, const Address &)
{
  NS_LOG_UNCOND ("Received one packet!");
  QosTag q_tag;
  if (packet->PeekPacketTag(q_tag))
```

```
  {
      NS_LOG_UNCOND ("Packet Tag value " << (int)q_tag.GetTid());
  }
  count++;
    totalRxBytesCounter += packet->GetSize ();
        totalRxBytesCounter+= ١;
if (count==١٠٠٠)
    std::cout<<"time "<<Simulator::Now ().GetSeconds();


 NS_LOG_UNCOND ("count basteha daryaft shode " << count);
 NS_LOG_UNCOND("th " << Simulator::Now ().GetSeconds()<<
"\t"<<((totalRxBytesCounter*٨,٠)/١٠٠٠٠٠)/Simulator::Now
().GetSeconds());
 NS_LOG_UNCOND("average packet transmition time " << Simulator::Now
().GetSeconds()<< "\t"<<((Simulator::Now ().GetSeconds())*١٠٠٠)/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 ۲۴ 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 "ns۳/core-module.h"
#include "ns۳/network-module.h"
#include "ns۳/mobility-module.h"
#include "ns۳/config-store-module.h"
#include "ns۳/wifi-module.h"
#include "ns۳/internet-module.h"
#include "ns۳/olsr-helper.h"
#include "ns۳/ipv۴-static-routing-helper.h"
#include "ns۳/ipv۴-list-routing-helper.h"
#include "ns۳/applications-module.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>

NS_LOG_COMPONENT_DEFINE ("WifiSimpleAdhocGrid");
```

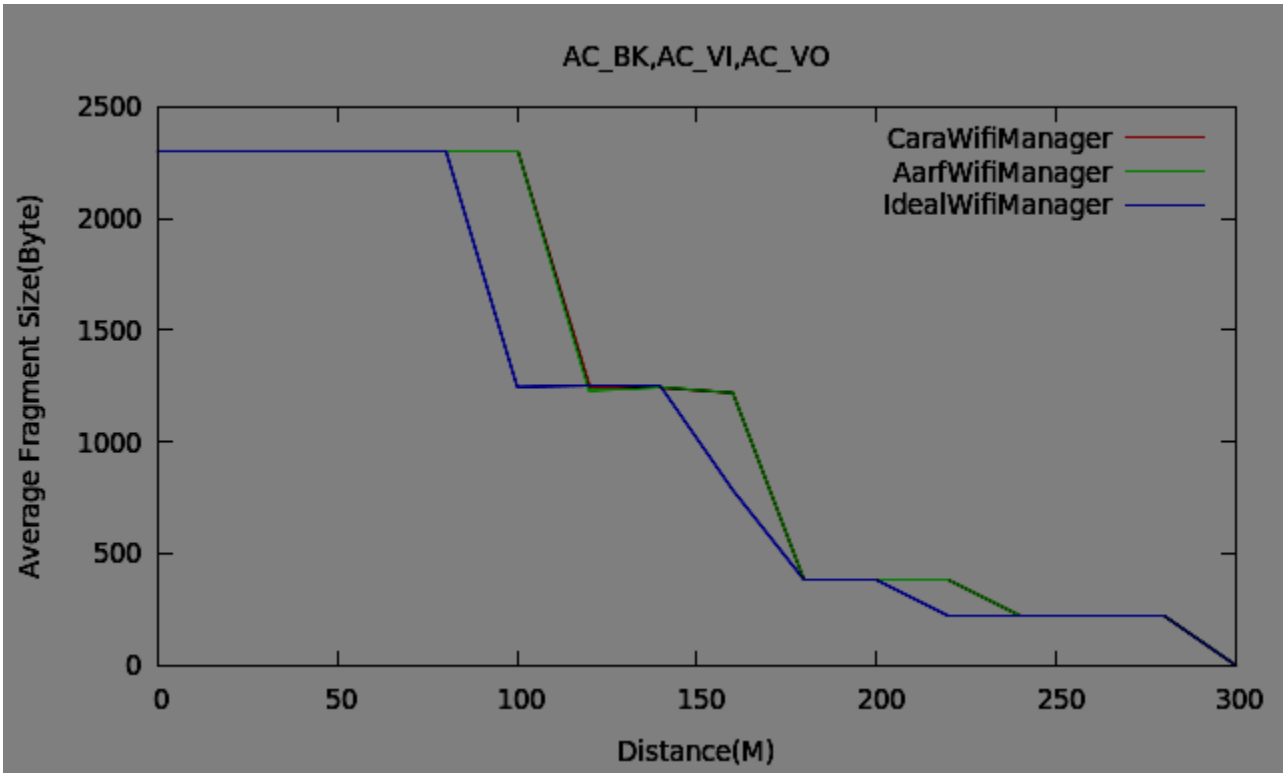
```
using namespace ns٣;
uint٣٢_t count =٠;
std::ofstream myfile ("project.txt",std::ofstream::binary);
uint٣٢_t average =٠;
 NodeContainer c;
 uint٣٢_t totalRxBytesCounter = ٠;

 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+= ١;
if (count==١٠٠٠)
    std::cout<<"time "<<Simulator::Now ().GetSeconds();


 NS_LOG_UNCOND ("count basteha daryaft shode " << count);
 NS_LOG_UNCOND("th " << Simulator::Now ().GetSeconds()<<
"\t"<<((totalRxBytesCounter*٨,٠)/١٠٠٠٠٠)/Simulator::Now
().GetSeconds());
 NS_LOG_UNCOND("average packet transmition time " << Simulator::Now
().GetSeconds()<< "\t"<<((Simulator::Now ().GetSeconds())*١٠٠٠)/count);


}


void
TagMarker (uint٨_t tid, Ptr<const Packet> packet)
{
 QosTag qosTag;
 qosTag.SetTid(tid);
 packet->AddPacketTag (qosTag);
}

int main (int argc, char *argv[])
{
```

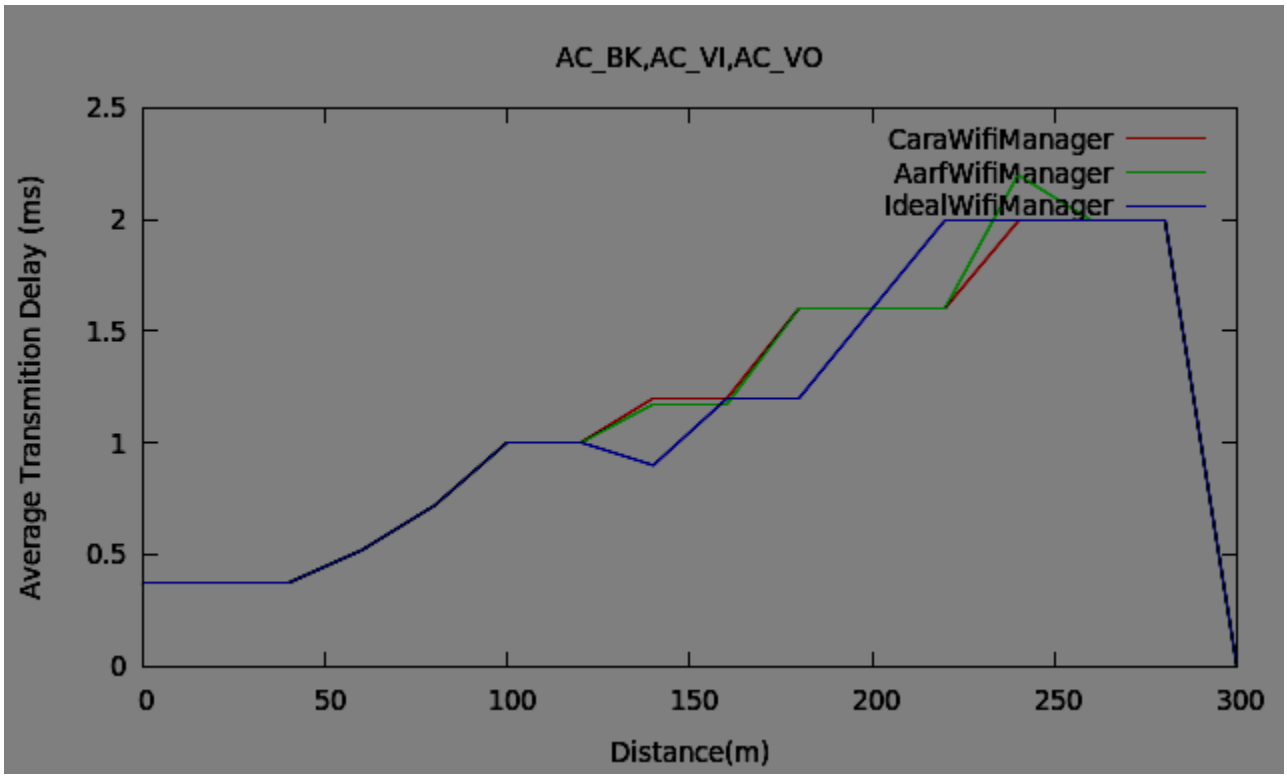
```
  std::string phyMode ("DsssRate١Mbps");
  //uint٣٢_t packetSize = ٢٢٦٤; // bytes
  uint٣٢_t numNodes = ٢;  // by default, ٥x٥
  uint٣٢_t sinkNode = ٠;
  uint٣٢_t sourceNode = ١;


  // disable fragmentation for frames below ٢٢٠٠ bytes
 // Config::SetDefault
("ns٣::WifiRemoteStationManager::FragmentationThreshold", StringValue
("١٥٠٠"));
  // turn off RTS/CTS for frames below ٢٢٠٠ bytes
// Config::SetDefault
("ns٣::WifiRemoteStationManager::RtsCtsThreshold", StringValue
("٢٣٠٠"));
  // Fix non-unicast data rate to be the same as that of unicast
 // Config::SetDefault
("ns٣::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 (-١٠) );
  // ns-٣ supports RadioTap and Prism tracing extensions for ٨٠٢,١١b
  //wifiPhy.SetPcapDataLinkType
(YansWifiPhyHelper::DLT_IEEE٨٠٢_١١_RADIO);

  YansWifiChannelHelper wifiChannel=YansWifiChannelHelper::Default ();
//wifiChannel.SetPropagationDelay
("ns٣::ConstantSpeedPropagationDelayModel");
// wifiChannel.AddPropagationLoss ("ns٣::FriisPropagationLossModel");
//wifiChannel.AddPropagationLoss("ns٣::LogDistancePropagationLossModel"
,"Exponent", DoubleValue (٤٠));
wifiPhy.SetChannel (wifiChannel.Create ());

  // Add a QoS upper mac, and disable rate control
  QosWifiMacHelper wifiMac = QosWifiMacHelper::Default ();
```

```
  wifi.SetStandard (WIFI_PHY_STANDARD_٨٠٢١١g);

wifi.SetRemoteStationManager ("ns٣::IdealWifiManager");
// wifi.SetRemoteStationManager ("ns٣::IdealWifiManager",
"RtsCtsThreshold", UintegerValue (١٠٠٠));
  Ssid ssid = Ssid ("ns-٣-ssid");
  wifiMac.SetType ("ns٣::StaWifiMac",
               "Ssid", SsidValue (ssid),
               "ActiveProbing", BooleanValue (false));
  //wifiMac.SetMsduAggregatorForAc (AC_BK,
"ns٣::MsduStandardAggregator", "MaxAmsduSize", UintegerValue (٣٨٣٩));
 // wifiMac.SetMsduAggregatorForAc (AC_VO,
"ns٣::MsduStandardAggregator", "MaxAmsduSize", UintegerValue (٧٩٣٥));
  //wifiMac.SetMsduAggregatorForAc (AC_VI,
"ns٣::MsduStandardAggregator", "MaxAmsduSize", UintegerValue (٧٩٣٥));

  // install Wifi on Wifistatnodes
  NetDeviceContainer staDevices;
  staDevices = wifi.Install (wifiPhy, wifiMac,  c.Get(١));

  wifiMac.SetType ("ns٣::ApWifiMac",
               "Ssid", SsidValue (ssid));

  NetDeviceContainer apDevices;
  apDevices = wifi.Install (wifiPhy, wifiMac, c.Get(٠));


  /* Setting mobility model */
  MobilityHelper mobility;
  Ptr<ListPositionAllocator> positionAlloc =
CreateObject<ListPositionAllocator> ();

  positionAlloc->Add (Vector (٠,٠, ٠,٠, ٠,٠));
  positionAlloc->Add (Vector (١٢٠, ٠,٠, ٠,٠));
  mobility.SetPositionAllocator (positionAlloc);

  mobility.SetMobilityModel ("ns٣::ConstantPositionMobilityModel");

  mobility.Install (c.Get(٠));
  mobility.Install (c.Get(١));

  InternetStackHelper internet;
  internet.Install (c);

  Ipv٤AddressHelper ipv٤;
  NS_LOG_INFO ("Assign IP Addresses.");
```

```
  ipv۴.SetBase ("۱۰.۱.۱.۰", "۲۵۵.۲۵۵.۲۵۵.۰");
  ipv۴.Assign(staDevices);
  Ipv۴InterfaceContainer i = ipv۴.Assign (apDevices);


  InetSocketAddress local = InetSocketAddress (Ipv۴Address::GetAny (),
۸۰);
  PacketSinkHelper recvSink("ns۳::UdpSocketFactory", local);
  sinkApp = recvSink.Install(c.Get (sinkNode));
   sinkApp.Start (Seconds (۰.۰));
   sinkApp.Stop (Seconds (۳۰.۰));


  InetSocketAddress remote = InetSocketAddress (i.GetAddress
(sinkNode, ۰), ۸۰);
  OnOffHelper onh("ns۳::UdpSocketFactory", remote);
  onh.SetAttribute ("PacketSize",  UintegerValue (۲۲۶۴));
  onh.SetAttribute ("OnTime", StringValue
("ns۳::ConstantRandomVariable[Constant=۱]"));
  onh.SetAttribute ("OffTime", StringValue
("ns۳::ConstantRandomVariable[Constant=۰]"));
  onh.SetAttribute ("DataRate",DataRateValue (DataRate ("۲۴Mbps")));
  ApplicationContainer source_apps =onh.Install(c.Get (sourceNode));
  source_apps.Start (Seconds (۰.۵));
  source_apps.Stop (Seconds (۳۰.۰));

  Ptr<OnOffApplication> onoffapp;
   onoffapp = DynamicCast<OnOffApplication>(source_apps.Get(۰));
   onoffapp->TraceConnectWithoutContext("Tx",  MakeBoundCallback
(&TagMarker, AC_VO));


  OnOffHelper onh۲("ns۳::UdpSocketFactory", remote);
  onh.SetAttribute ("PacketSize",  UintegerValue (۲۲۶۴));
  onh.SetAttribute ("OnTime", StringValue
("ns۳::ConstantRandomVariable[Constant=۱]"));
  onh.SetAttribute ("OffTime", StringValue
("ns۳::ConstantRandomVariable[Constant=۰]"));
  onh.SetAttribute ("DataRate",DataRateValue (DataRate ("۲۴Mbps")));
  ApplicationContainer source_apps۲ =onh۲.Install(c.Get (sourceNode));
  source_apps۲.Start (Seconds (۰.۵));
  source_apps۲.Stop (Seconds (۳۰.۰));
```

```
  Ptr<OnOffApplication> onoffapp٢;
   onoffapp٢ = DynamicCast<OnOffApplication>(source_apps٢.Get(٠));
   onoffapp٢->TraceConnectWithoutContext("Tx",  MakeBoundCallback
(&TagMarker, AC_VI));


   OnOffHelper onh٣("ns٣::UdpSocketFactory", remote);
   onh.SetAttribute ("PacketSize",  UintegerValue (٢٢٦٤));
   onh.SetAttribute ("OnTime", StringValue
("ns٣::ConstantRandomVariable[Constant=١]"));
   onh.SetAttribute ("OffTime", StringValue
("ns٣::ConstantRandomVariable[Constant=٠]"));
   onh.SetAttribute ("DataRate",DataRateValue (DataRate ("٢٤Mbps")));
   ApplicationContainer source_apps٣ =onh٣.Install(c.Get (sourceNode));
   source_apps٣.Start (Seconds (٠,٥));
   source_apps٣.Stop (Seconds (٣٠,٠));


   Ptr<OnOffApplication> onoffapp٣;
    onoffapp٣ = DynamicCast<OnOffApplication>(source_apps٣.Get(٠));
    onoffapp٣->TraceConnectWithoutContext("Tx",  MakeBoundCallback
(&TagMarker, AC_BK));


Config::ConnectWithoutContext("/NodeList/٠/ApplicationList/٠/$ns٣::Pack
etSink/Rx",MakeCallback (&ReceivePacket));




  Simulator::Stop (Seconds (٣٠,٠));
  Simulator::Run ();
  Simulator::Destroy ();

  return ٠;
   }
```


AC_BK,AC_VI,AC_VO
Average Fragment Size(Byte)
Distance(M)
CaraWifiManager
AarfWifiManager
IdealWifiManager
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300


AC_BK,AC_VI,AC_VO
Average Transmition Delay (ms)
Distance(m)
CaraWifiManager
AarfWifiManager
IdealWifiManager
2.5
2
1.5
1
0.5
0
0
50
100
150
200
250
300

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

برای پیاده سازی نمودار ۳۲ سناریو ی زیر را با دو نود (یکی Station با نصب OnoffApp و 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(۰);

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 "ns۳/core-module.h"
#include "ns۳/point-to-point-module.h"
#include "ns۳/network-module.h"
#include "ns۳/applications-module.h"
#include "ns۳/wifi-module.h"
#include "ns۳/mobility-module.h"
#include "ns۳/csma-module.h"
#include "ns۳/internet-module.h"
```

```
#include "ns٣/dca-txop.h"


using namespace ns٣;
// declare For Calculate Cw
uint٣٢_t cwold=١٥;
uint٣٢_t countcollision=٠;
ApplicationContainer sinkApps;
std::ofstream myfile ("p.txt",std::ofstream::binary);
uint٣٢_t totalRxBytesCounter = ٠;
// Add All Cw
uint٣٢_t cw_all=٠;
uint٣٢_t count=٠;

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+= ١;


 NS_LOG_UNCOND ("count basteha daryaft shode " << count);
 NS_LOG_UNCOND ("Average Delay "<<count/(Simulator::Now
().GetSeconds()*١٠٠٠));
 NS_LOG_UNCOND ("count basteha daryaft shode " <<Simulator::Now
().GetSeconds());

 //NS_LOG_UNCOND("th " << Simulator::Now ().GetSeconds()<<
"\t"<<((totalRxBytesCounter*٨,٠)/١٠٠٠٠٠)/Simulator::Now
().GetSeconds());
 //NS_LOG_UNCOND("average packet transmition time " << Simulator::Now
().GetSeconds()<< "\t"<<((Simulator::Now ().GetSeconds())*١٠٠٠)/count);


}
```

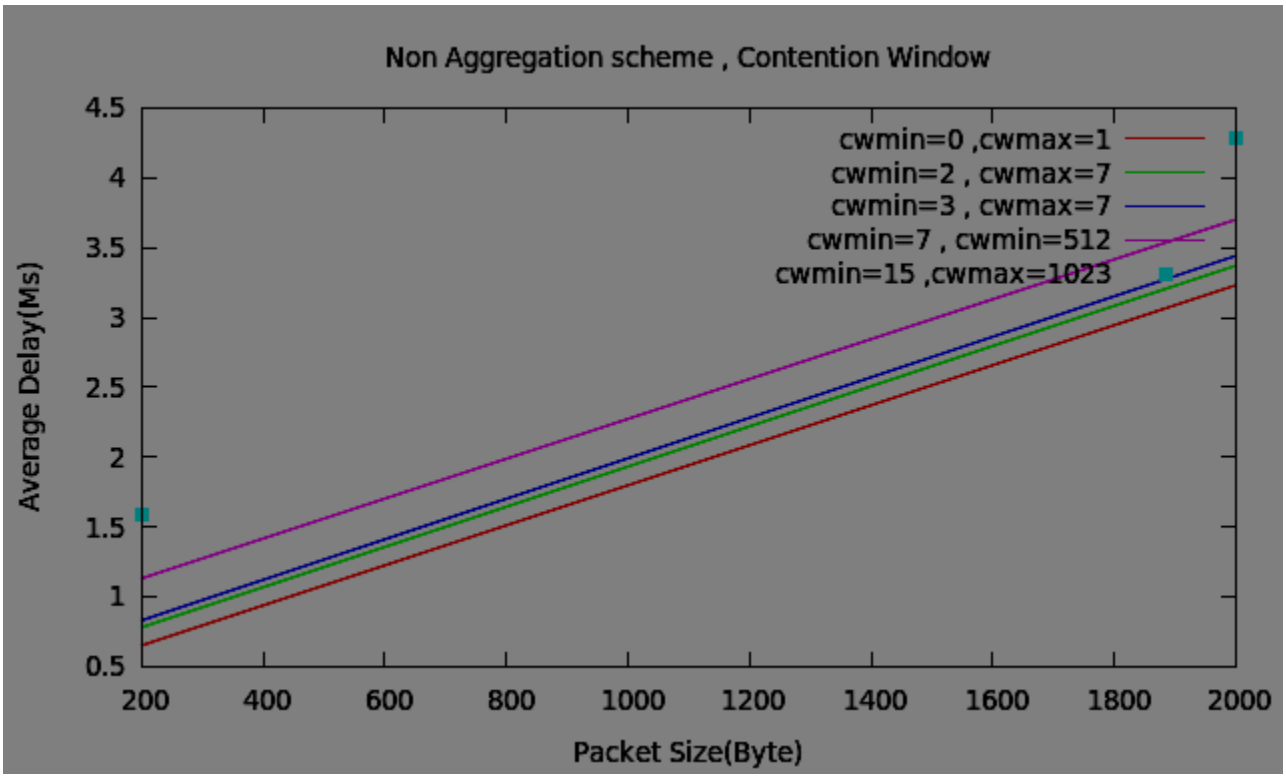
```
int
main (int argc, char *argv[])
{

 // bool verbose = true;
  // Number of Server And Wifi
  uint٣٢_t nserver = ١;
  uint٣٢_t nWifi = ١;



// Definition server node
  NodeContainer serverNodes;
  serverNodes.Create (nserver);
  //CsmaHelper serverh;
 //serverh.SetChannelAttribute ("DataRate", StringValue ("١٠Mbps"));
  //serverh.SetChannelAttribute ("Delay", TimeValue (NanoSeconds
(٦٥٦٠)));

 // NetDeviceContainer serverDevices;
 // serverDevices = serverh.Install (serverNodes);

  // Defiition Of Wifi Nodes

  wifiStaNodes.Create (nWifi);
  NodeContainer wifiApNode =serverNodes.Get(٠);

  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 ("ns٣::CaraWifiManager");
  wifi.SetStandard (WIFI_PHY_STANDARD_٨٠٢١١g);
 // wifi.SetRemoteStationManager ("ns٣::ConstantRateWifiManager",
   //                              "DataMode", StringValue
("OfdmRate٢Mbps"));

// wifi.SetRemoteStationManager ("ns٣::ConstantRateWifiManager");
  NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
```

```
//creates an ٨٠٢,١١ 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 "ns٣::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-٣-ssid");
  mac.SetType ("ns٣::StaWifiMac",
               "Ssid", SsidValue (ssid),
               "ActiveProbing", BooleanValue (false));

  // install Wifi on Wifistatnodes
  NetDeviceContainer staDevices;
  staDevices = wifi.Install (phy, mac, wifiStaNodes);

  mac.SetType ("ns٣::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 (٠,٠, ٠,٠, ٠,٠));
  positionAlloc->Add (Vector (١٥٠, ٠,٠, ٠,٠));
  mobility.SetPositionAllocator (positionAlloc);

  mobility.SetMobilityModel ("ns٣::ConstantPositionMobilityModel");

  mobility.Install (wifiStaNodes.Get(٠));
  mobility.Install (wifiApNode.Get(٠));


  // Define and install Internet Stack On All Nodes
  InternetStackHelper stack;
  //stack.Install (serverNodes);
  stack.Install (wifiApNode);
  stack.Install (wifiStaNodes);

// Define Ip Address

  Ipv٤AddressHelper address;
```

```
  address.SetBase ("١٠,١,٣,٠", "٢٥٥,٢٥٥,٢٥٥,٠");
address.Assign (staDevices);
   Ipv٤InterfaceContainer i=address.Assign (apDevices);


  InetSocketAddress remote = InetSocketAddress (i.GetAddress (٠, ٠),
٨٠);
  OnOffHelper onh("ns٣::UdpSocketFactory", remote);
  onh.SetAttribute ("PacketSize",  UintegerValue (٢٠٠));
 onh.SetAttribute ("OnTime", StringValue
("ns٣::ConstantRandomVariable[Constant=١]"));
  onh.SetAttribute ("OffTime", StringValue
("ns٣::ConstantRandomVariable[Constant=٠]"));
  onh.SetAttribute ("DataRate",DataRateValue (DataRate ("١٠Mbps")));



  InetSocketAddress local = InetSocketAddress (Ipv٤Address::GetAny (),
٨٠);
  PacketSinkHelper recvSink("ns٣::UdpSocketFactory", local);
  sinkApp = recvSink.Install(serverNodes.Get (٠));
   sinkApp.Start (Seconds (٠,٠));
   sinkApp.Stop (Seconds (١٠٠,٠));



// install Source app on all wifi nodes
ApplicationContainer clientApps ;

    clientApps =  onh.Install (wifiStaNodes.Get (٠));
    clientApps.Start (Seconds (٠,٠));
    clientApps.Stop (Seconds (١٠٠,٠));


 // Ipv٤GlobalRoutingHelper::PopulateRoutingTables ();
  // scheduling For All Wifi Nodes
```

```
Config::ConnectWithoutContext("/NodeList/٠/ApplicationList/٠/$ns٣::Pack
etSink/Rx",MakeCallback (&ReceivePacket));


  Simulator::Stop (Seconds (٢٠,٠));

  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 ٠;
}
```

در نتیجه نمودار زیر را استخراج کرده ایم

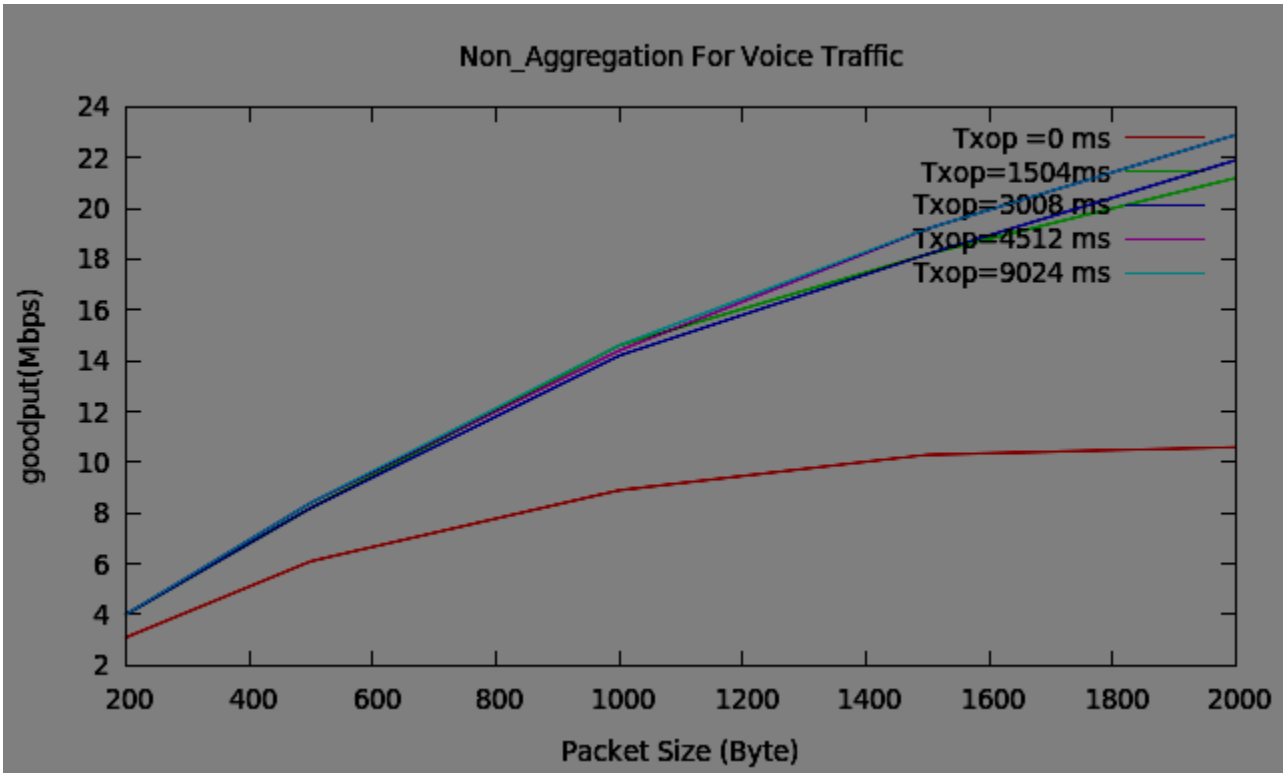


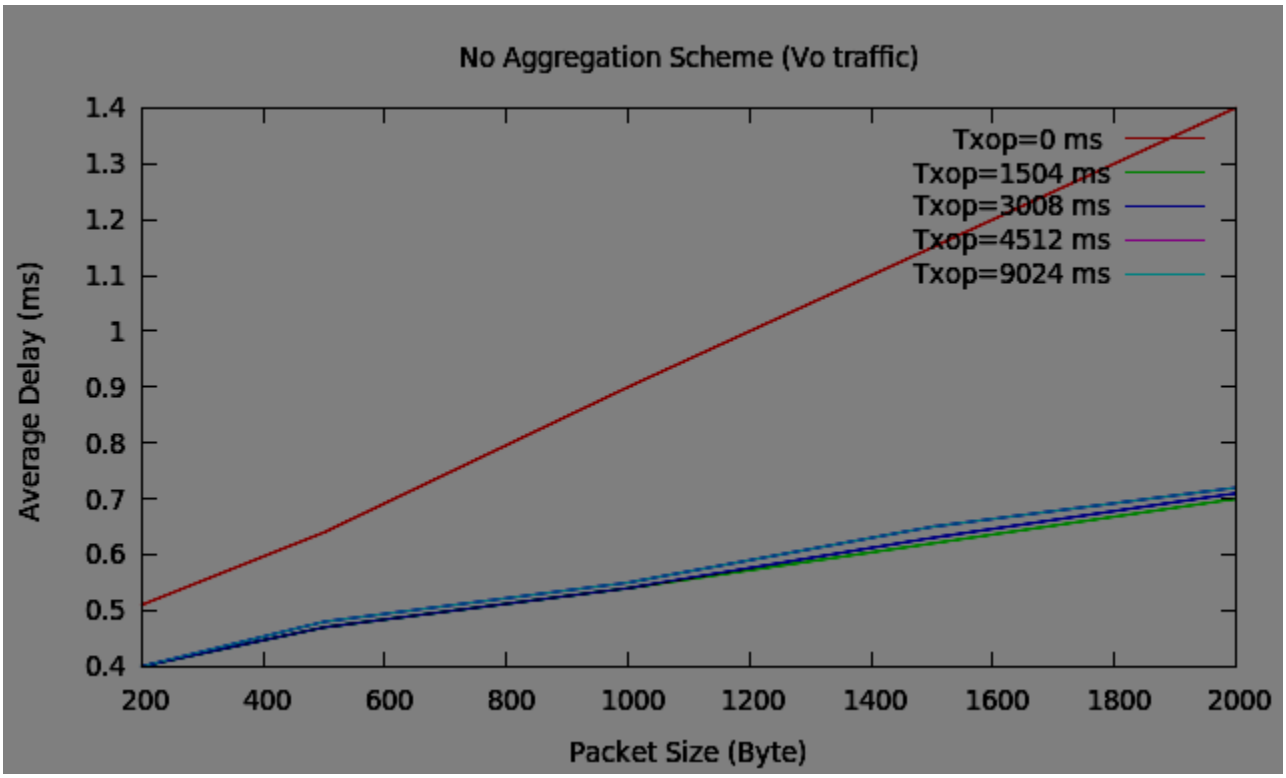
برای نمودار ۳۳

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 ۲۱ 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 .




No Aggregation Scheme (Vo traffic)
Average Delay (ms)
Packet Size (Byte)
Txop=0 ms
Txop=1504 ms
Txop=3008 ms
Txop=4512 ms
Txop=9024 ms
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
200
400
600
800
1000
1200
1400
1600
1800
2000