به نام خدا

گزارش کار پروژه دوم شبکه(CA-TCP-NS2)

اعضای گروه:

1- **آرین حدادی 810196448**
2- **مهدی هادیلو 810196587**

توضیح محتویات فایل آپلود شده:

در فایل آپلود شده برای پروژه یک فایل به نام NS2_TCP.tcl قرار گرفته که کد tcl مرتبط با شبیه سازی است.

یک فایل plot.py هم قرار گرفته که عملیات انجام 10 بار شبیه سازی و میانگین گیری و رسم نمودارها را انجام می دهد . کافیست با دستور python plot.py صرفا ران شده تا شبیه سازی بار دیگر انجام شود.(نیازی به هیچ argument ورودی برای ران کردن این فایل پایتون نیست)

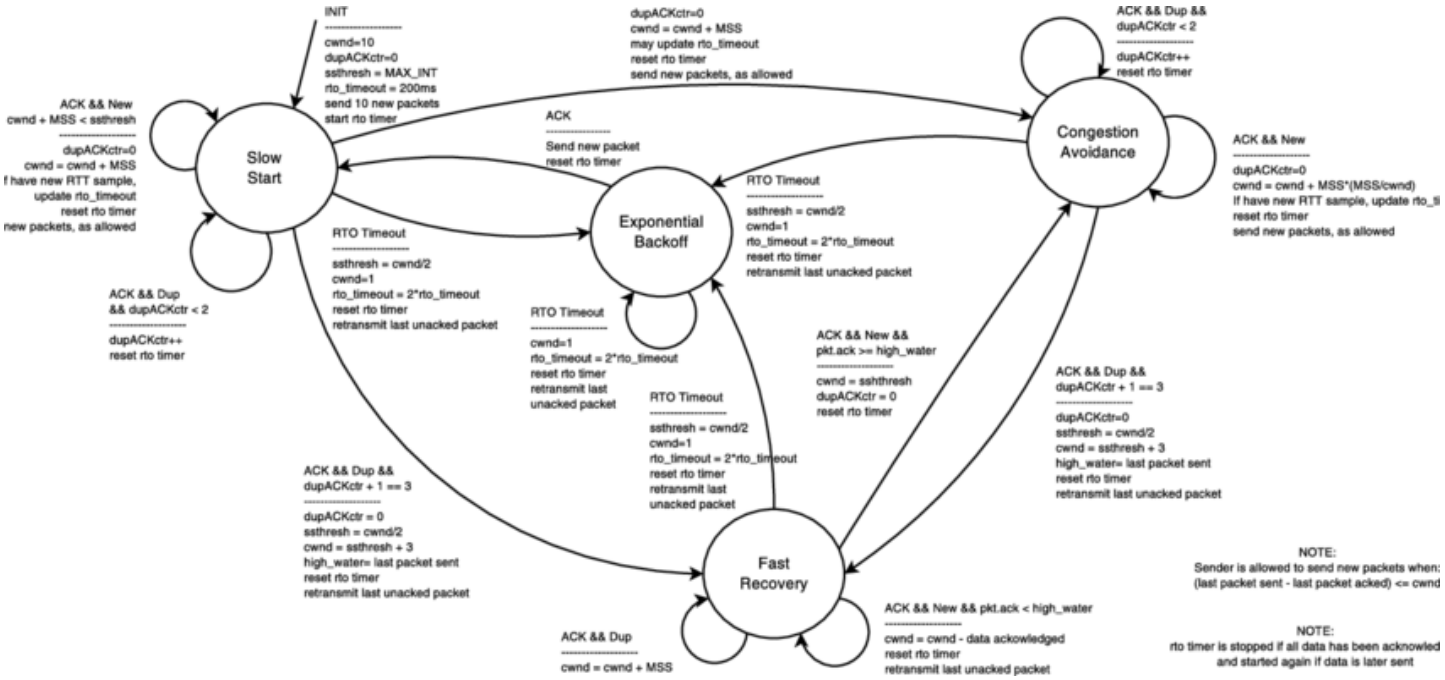
همچنین یک پوشه به نام Plots Data وجود دارد که داده های میانگینی است که برای رسم نمودار به کار می رود.(زیرا در صورت پروژه گفته شده که داده های نمودار در کنار فایل های شبیه سازی تحویل داده شوند.)

با ران کردن کد python با دستور python plot.py شبیه سازی بار دیگر انجام شده و plot ها و داده های آن ها هم در پوشه ای به نام plots قرار داده می شوند.

توضیح مختصر در رابطه با تفاوت های Newreno و Tahoe و Vegas:

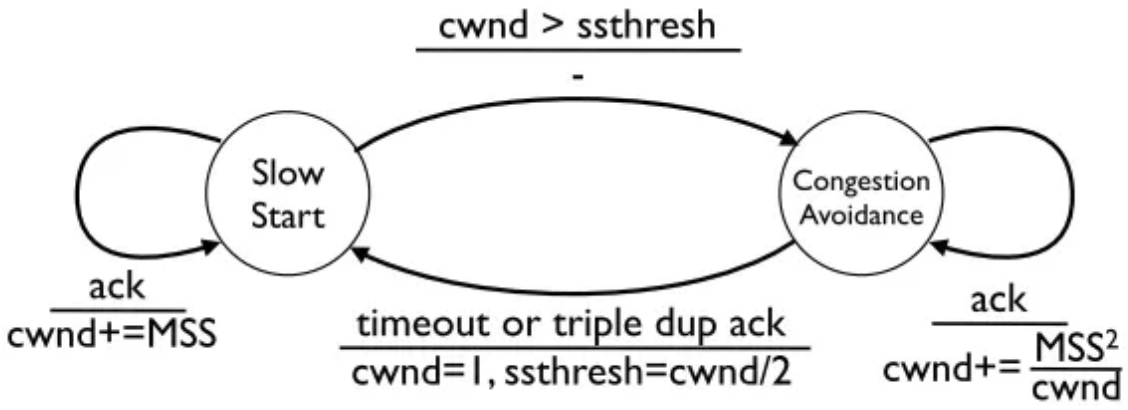
روش Newreno:

این روش نسبت به روش Reno در فاز fast-recovery(که از فاز های مرتبط با congestion control در پروتکل TCP است) عملیات ارسال دوباره پکت ها را اصلاح کرده است.مشکلی که در این روش وجود دارد این است که اگر هیچ packet loss ای وجود نداشته باشد ولی در عوض بسته ها(packet) با بیش از 3 عدد توالی reorder شوند روش Newreno به اشتباه وارد فاز fast recovery می شود.ماشین حالت(state machine) این روش بدین شکل است:



روش Tahoe:

Tahoe به روش کنترل ازدحام(congestion control) پروتکل TCP مربوط است که در paper ای که توسط Van Jacobson نوشته شد پیشنهاد شده است که مبتنی بر اصل "حفاظت از بسته ها(conservation of packets)" است.این الگوریتم بسیار شبیه Reno بوده ولی در بحث نحوه واکنش به ACK های دوگانه(duplicate ACKs) با Reno متفاوت است.ماشین حالت(state machine) این روش بدین شکل است:

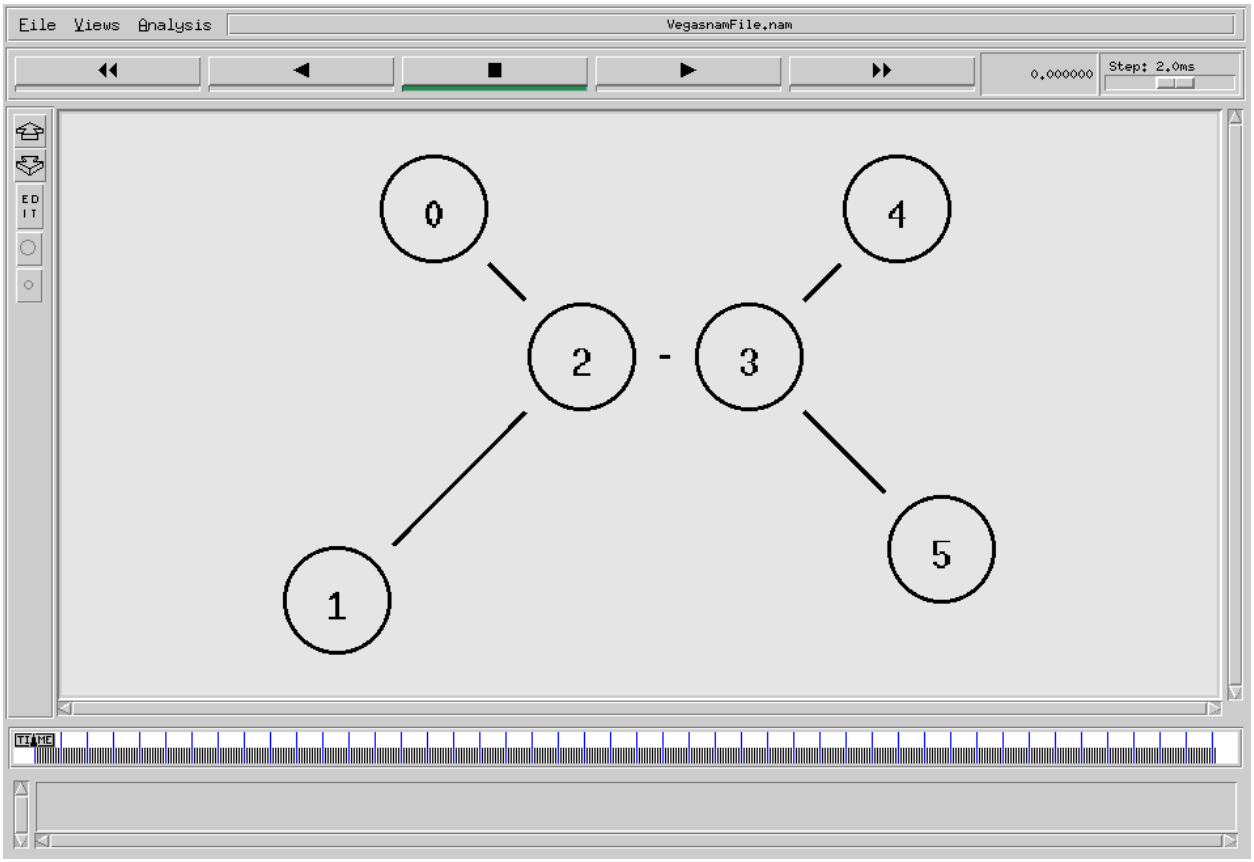


روش Vegas:

این روش اصلاحیه ای بر روش Reno است و توسط محققان دانشگاه Arizona معرفی شده است.این روش به جای packet loss از packet delay برای سیگنالی که مشخص می کند با چه نرخی ارسال بسته ها انجام شود استفاده می کند.این روش برخلاف روش هایی مانند Newreno و Reno به جای آنکه ازدحام را از طریق رخ دادن packet loss بفهمد، از طریق افزایش میزان rtt متوجه می شود.

شرح کار های انجام شده

یک فایل tcl نوشته شده که 2 ورودی به عنوان argv میگیرد که یکی اسم پروتکل بوده و این مورد در نام فایل های خروجی اثر گذار است.ورودی دوم نوع agent ای است از آن instantiate می شود.این دو ورودی برای این گرفته میشوند که هنگام ران کردن بتوان با تعیین این ورودی ها هر 3 نوع TCP را تست کرد.توپولوژی شبکه همانطور که صورت پروژه درخواست کرده بود طراحی شد.و خروجی به این شکل شد:



شماره گذاری گره(node) ها هم همانند شکل داده شده انجام شد لذا شماره گذاری از 1 تا 6 انجام شد.

```
# Creating Six Nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
```

برای عدد رندم به عنوان تاخیر دو لینک بین گره های 2 و 3 و لینک بین گره های 4 و 6 به این طریق انجام شد که proc زیر نوشته شد که خروجی آن عدد رندمی بین 5 و 25 بود و لذا از خروجی آن به عنوان تاخیر دو لینک مذکور استفاده شد.

```
# Defining procedure for generating random numbers between 5 and 25 for links with variable delay values
proc generateRandom {} {
    return [expr round(rand()*20)+5]
}
```

نحوه تاخیر دادن به لینک ها به این شکل بود:

```
# Creating links between the nodes
$ns duplex-link $n1 $n3 100Mb 5ms DropTail
$ns duplex-link $n2 $n3 100Mb [generateRandom]ms DropTail
$ns duplex-link $n3 $n4 100Kb 1ms DropTail
$ns duplex-link $n4 $n5 100Mb 5ms DropTail
$ns duplex-link $n4 $n6 100Mb [generateRandom]ms DropTail
```

که همانطور که مشاهده می شود برای تاخیر لینک بین n2 و n3 و لینک بین n4 و n6 از proc نوشته شده برای تاخیر استفاده شد.

همانطور که در شکل پروژه نشان داده شده است دو روتری که به عنوان نود های 3 و 4 در شبکه ما وجود دارند packet queue size آنها برابر 10 است.از آنجا که در NS2 صف بندی در لینکی که از یک نود خارج می شود انجام می شود لذا در تمام خروجی های دو نود 3 و 4 که روتر های این شبکه هستند میزان صف را برابر 10 قرار می دهیم.کد این بخش بدین شکل است:

```
# Setting queue size for routers
$ns queue-limit $n3 $n1 10
$ns queue-limit $n3 $n2 10
$ns queue-limit $n3 $n4 10

$ns queue-limit $n4 $n3 10
$ns queue-limit $n4 $n5 10
$ns queue-limit $n4 $n6 10
```

سپس دو object از نوع tcp (متناسب با نوع TCP که میخواهیم شبیه سازی کنیم) ایجاد شد که به عنوان فرستنده در نود های 1 و 2 و دو object از نوع TCPSink هم تولید شد تا به عنوان گیرنده در نود های 5 و 6 عمل کنند.

برای مثال کد مربوط به تولید object از نوع فرستنده برای نود 1 بدین شکل است:

```
# Setting Up TCP Agent for Node 1
set tcp1 [new Agent/$agentType]
$tcp1 set class_ 1
$tcp1 set fid_ 1
$tcp1 set ttl_ 64
$ns color 1 Blue
$ns attach-agent $n1 $tcp1
```

که همانطور که مشاهده می شود flow id آن برابر 1 بوده و رنگ مرتبط به بسته های این flow id را آبی قرار داده و ttl برابر 64 (همانطور که در صورت پروژه گفته شده بود) قرار داده شد و سپس این agent به نود 1 attach شد.

توجه شود که متغیر agentType متناسب با نوع TCP که میخواهیم شبیه سازی کنیم متفاوت است و از طریق ورودی به کد tcl هنگام اجرا داده می شود.

```
set protocol [expr [lindex $argv 0]]
set agentType [expr [lindex $argv 1]]
```

همانطور که در کد بالا مشاهده می شود برنامه دو ورودی دارد که اولی نام پروتکل TCP که میخواهیم شبیه سازی کنیم است که در تعیین نام فایل های نتیجه شبیه سازی خروجی به کار می رود و دومی هم نوع agent بوده که هنگامی که میخواهیم دو object از نوع فرستنده instantiate کنیم به کار می رود.

کد مربوط به TCPSink که گیرنده جریان است برای نود 5 هم بدین شکل است:

```
# Setting Up Sink for Node 5
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5
```

که همانطور که مشاهده می شود این TCPSink به نود 5 ام که دریافت کننده جریان با fid برابر یک با فرستنده نود 1 است attach شده است.

پروتکل اپلیکیشن استفاده شده در پروژه FTP است که چون دو جریان از آن داریم دو تا از آن ایجاد شده و هر کدام به یکی از object های TCP وصل(attach) شدند.برای مثال کد مربوط به این جریان برای جریان مرتبط با نود های 1 و 5 بدین شکل است:

```
# Setting Up FTP for TCP Connection Between Node 1 and Node 5
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
```

چون گفته شده بود که جریانات به مدت 1000 ثانیه و 10 بار انجام شوند بدین شکل عمل شد:

```
# Schedule running for FTP
$ns at 0.0 "$ftp1 start"
$ns at 1000.0 "$ftp1 stop"
$ns at 0.0 "$ftp2 start"
$ns at 1000.0 "$ftp2 stop"
```

که یعنی هر دوی جریانات در ثانیه 0 شروع و در ثانیه 1000 متوقف شده و به اتمام برسند.

برای ده بار اجرا کردن آن هم از کد پایتون استفاده شد که ده بار این ها را اجرا کرده و و از نتایج میانگین بگیرند.

اندازه گیری پارامتر های خواسته شده:

برای اندازه گیری پارامتر های خواسته شده در پروژه به این شکل عمل شد:

CWND و RTT: یک proc به نام writeInOutputFile نوشته شد(که در زیر آمده است) که از ابتدای شروع شبیه سازی یعنی زمان 0 هر یک ثانیه صدا زده می شود و هر بار که صدا زده شد مقدار CWND و RTT برای هر کدام از اتصال های TCP(هر اتصال در یک فایل جدا) اندازه گیری شده در فایل نوشته شدند.

```
proc writeInOutputFile {tcpSource traceGoodput outputFile} {
    global ns

    set now [$ns now]
    set cwnd [$tcpSource set cwnd_]
    set rtt [$tcpSource set rtt_]


    set nbytes [$traceGoodput set bytes_]
    $traceGoodput set bytes_ 0
    set goodput [expr ($nbytes * 8.0 / 1000000)]

    # writing cwnd, rtt and goodput for this time unit in output file
    puts $outputFile "$now\t$cwnd\t$rtt\t$goodput"

    # calling writeInOutputFile recursively after 1 second
    $ns at [expr $now + 1] "writeInOutputFile $tcpSource $traceGoodput $outputFile"
}
```

Goodput: برای محاسبه این پارامتر نیاز است که که ببینیم چند بایت به خروجی داده شده است.برای این کار یک class جدید به نام TraceGoodput نوشته شد که از کلاس Application ارث بری می کند.این کلاس دو متد init و recv را override میکند.در متد init مقدار یک فیلد به نام bytes_ برابر صفر قرار داده می شود.در متد recv هم یک ورودی به نام byte داریم که تعداد بایت های جدید دریافت شده است که این مقدار با مقدار قبلی فیلد bytes_ جمع زده می شود.بدین ترتیب در فیلد bytes_ تعداد بایت هایی که instance مربوطه از این کلاس دریافت کرده است ذخیره می شود.

```
# Declaring class "TraceGoodput" as a child class of "Application"
Class TraceGoodput -superclass Application

    # Overriding "init" method for class "TraceGoodput"
    TraceGoodput instproc init {args} {
        $self set bytes_ 0
        eval $self next $args
    }

    # Overriding "recv" method for class "TraceGoodput"
    TraceGoodput instproc recv {byte} {
        $self instvar bytes_
        set bytes_ [expr $bytes_ + $byte]
        return $bytes_
    }
```

دو instance از این کلاس ساخته می شود و هر کدام به یکی از TCPSink ها با دستور attach-agent متصل می شود.(TCPSink ها در این مسئله گره های 5 و 6 هستند که از گره های 1 و 2 به ترتیب به آن ها داده با پروتکل TCP فرستاده می شود)لذا به این شکل می توان تعداد بایت هایی که در مقصد دریافت می شود را در فیلد bytes_ این instance ها ذخیره کرد.

در proc به نام writeInOutputFile که بالاتر بدنه آن قرار داده شده و همانطور که بیان شد هر یک ثانیه از ابتدای شبیه سازی صدا زده می شود مقدار bytes_ هر کدام از این دو instance خوانده شده و سپس مقدار این فیلد برابر 0 قرار داده می شود.به این شکل هر ثانیه اندازه گیری میکنیم که تعداد بایت ها دریافت شده در مقصد(sink) برای هر کدام از این اتصال های TCP چه مقدار بوده و بدین شکل نرخ goodput در هر ثانیه اندازه گیری می شود.

نرخ از دست رفتن بسته(packet loss):برای محاسبه این مقدار بدین شکل عمل شد که با استفاده از دستور traceall تمام packet ها را در طول این انتقال trace شد و در فایلی با پسوند tr. نوشته شد.سپس با استفاده از python این فایل ها خوانده شده و خطوطی که به drop شدن یک packet اشاره میکردند(این خطوط در فایل مذکور بدین شکل نوشته می شوند که اولین حرفی در این خطوط نوشته می شد حرف d است) برای هرکدام از اتصال ها جدا شده و سپس برای هر یک از ثانیه های 1 تا 1000 که زمان شبیه سازی بود تعداد این drop ها شمرده شدند.

نمونه ای از یک خط مرتبط با drop شدن یک packet:

```
d 0.724552 2 3 tcp 1040 ------- 1 0.0 4.0 9 30
```

این خط از چندین بخش تشکیل شده است.

ابتدا d که یعنی این گزارش مربوط به drop شدن یه packet است.

عدد بعدی که 0.724552 است نشان دهنده زمان رخ دادن این drop است.

عدد بعدی که 2 است عدد گره(node) ای را نشان می دهد که فرستنده این packet بوده است.

عدد بعدی که 3 است هم عدد گره(node) ای را نشان می دهد که دریافت کننده این packet بوده است.

مقدار بعدی که tcp است نشان دهنده پروتکل است که در این پروژه پروتکل tcp شبیه سازی شده است.

مقدار بعدی سایز پکت ها را نشان می دهد که 1040 بایت است که 1000 بایت آن سایز خود بسته و 40 بایت آن مربوط به هدر های TCP است.

سپس flag های tcp اورده شده که چون flag ای وجود ندارد ------- نوشته شده است.از flag هایی که میتوانند در این بخش باشند E است که ECN که مخفف Explicit Congestion Notification است را نشان می دهد.

سپس flow id را نشان می دهد که در اینجا 1 است.flow id برابر 1 مربوط به جریان از میزبان 1 به میزبان 5 است.

عدد بعدی که 0.0 است نود مبدا این flow را نشان می دهد.فرم آن node.connectionId است.

عدد بعدی هم که 4.0 است باز به فرم عدد قبلی است که در اینجا 4 شماره گره مقصد است.توجه شود این دو عدد 0-based بوده و متناظر با همان نود های 1 و 5 در توپولوژی داده شده در صورت پروژه هستند.

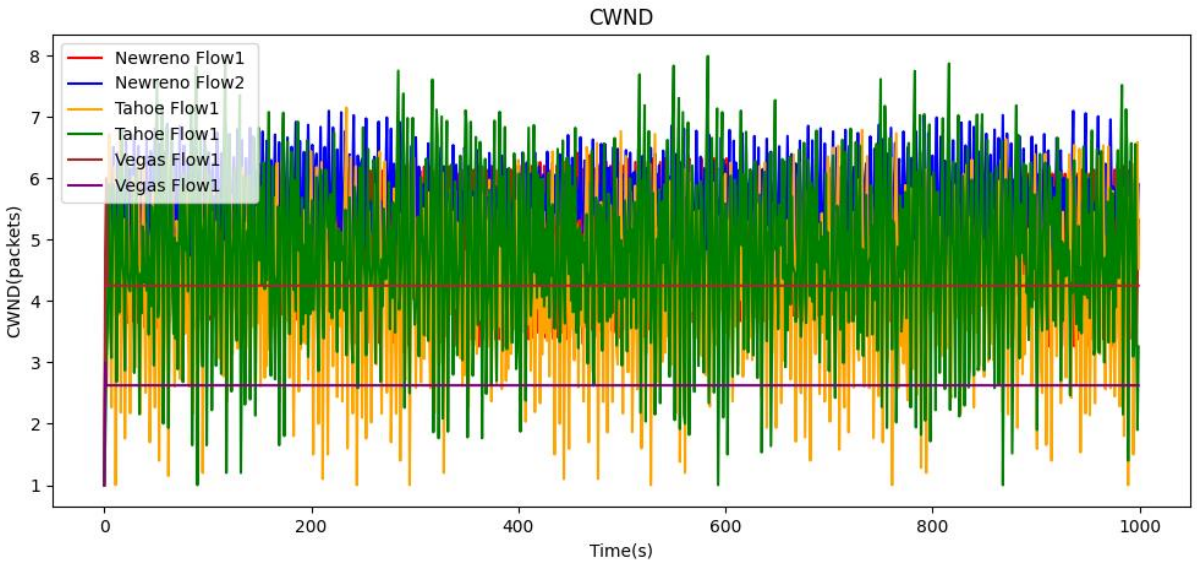
عدد بعدی که 9 است sequence number (عدد توالی) این بسته را نشان می دهد که در پروتکل TCP وجود دارد.

و در نهایت ID این packet که این packet را در کل عملیات شبیه سازی به صورت یکتا(unique) مشخص می کند که در این خط برابر 30 است.

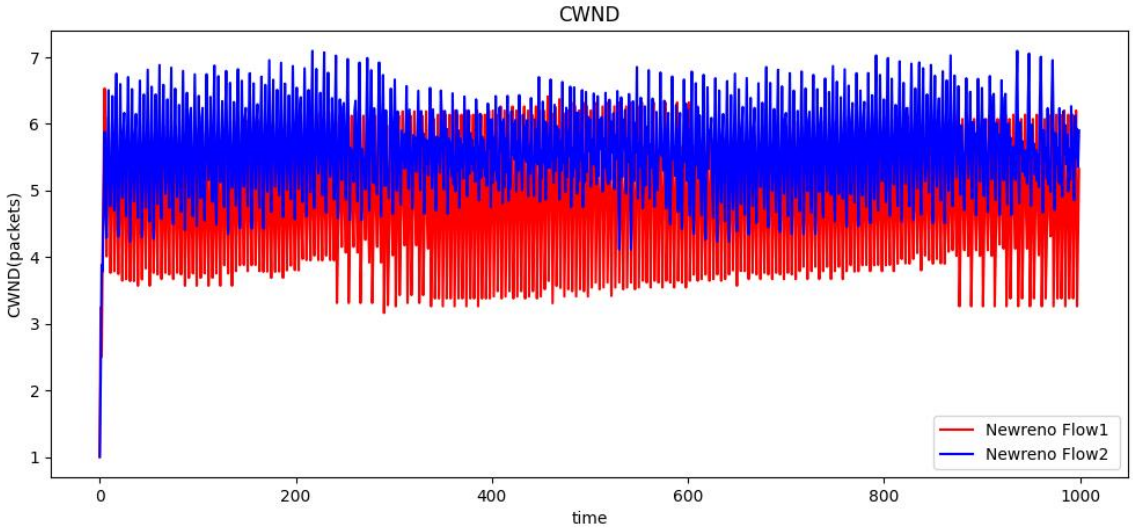
نتایج شبیه سازی

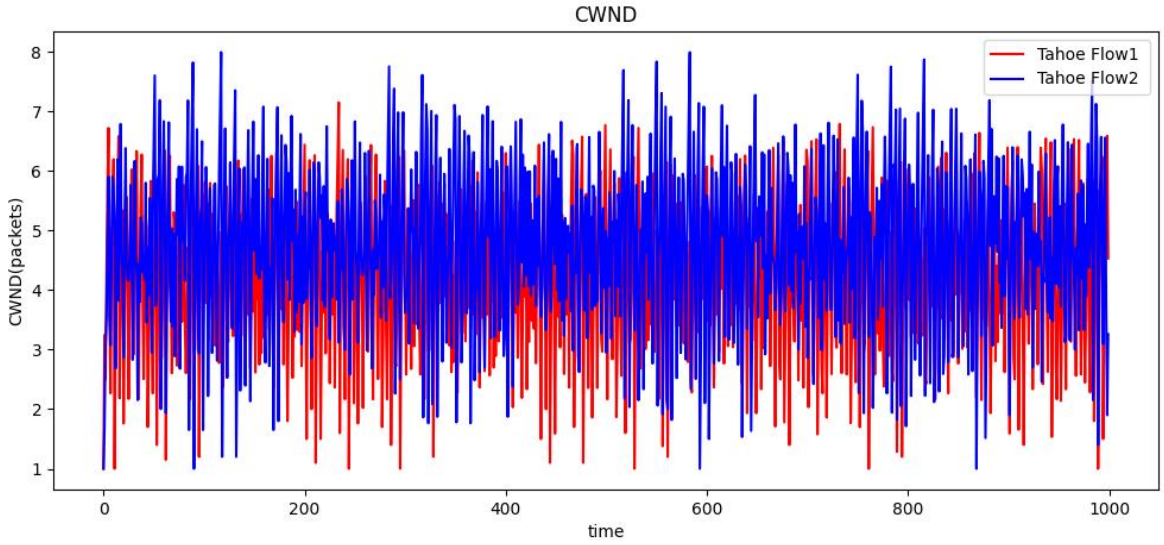
با استفاده از زبان python و کتابخانه pandas فایل های خروجی شبیه سازی که در بالا ذکر شد خوانده شده و با استفاده از کتابخانه matplotlib نمودار های مربوطه رسم شدند.

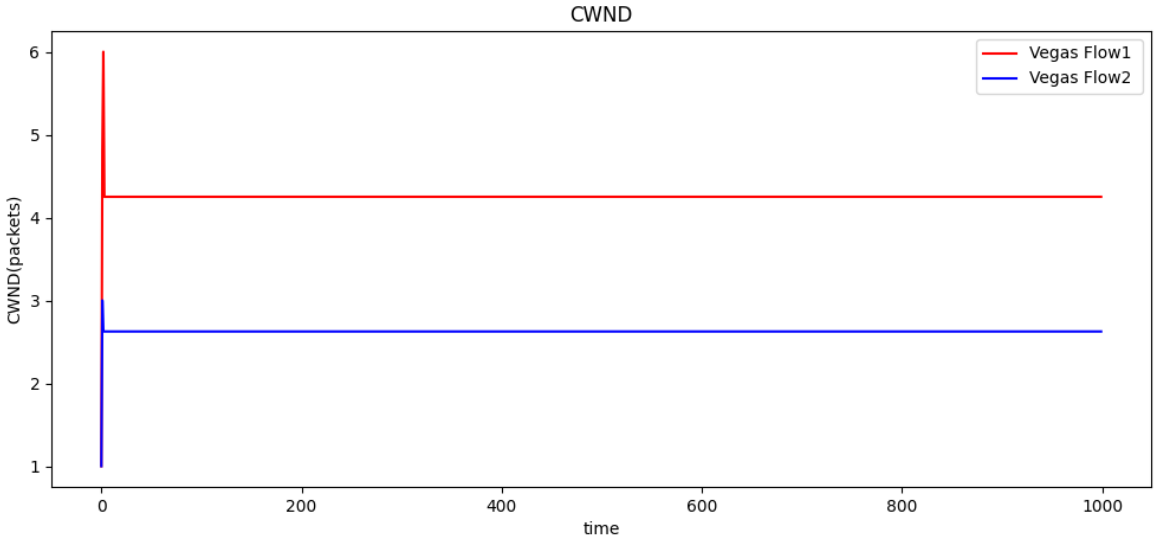
نمودارهای مربوط به CWND:



نمودار های جداگانه مرتبط با هر روش هم بدین شکل شدند:

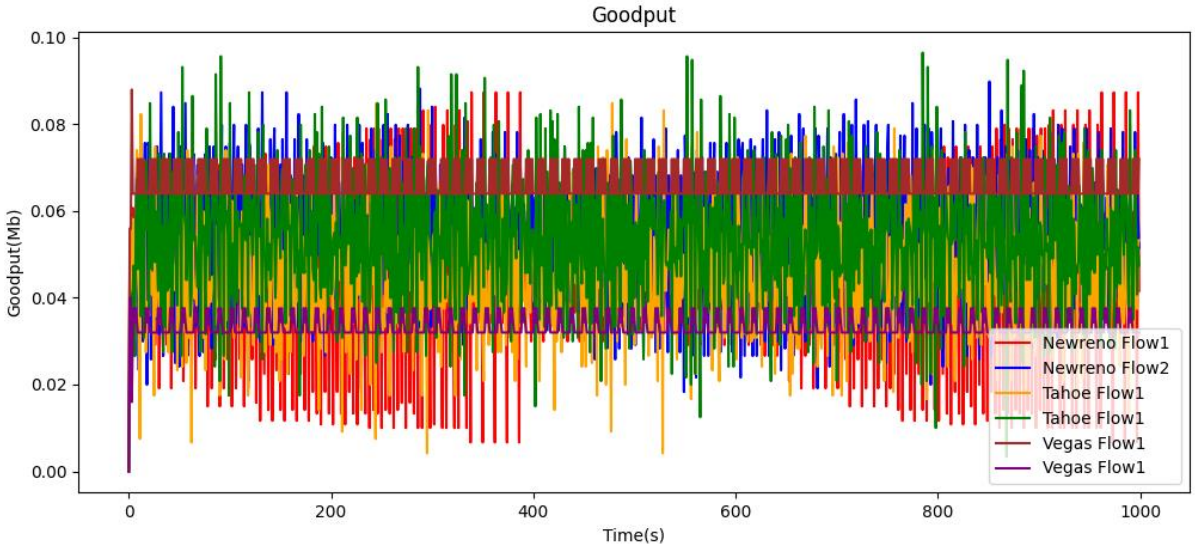




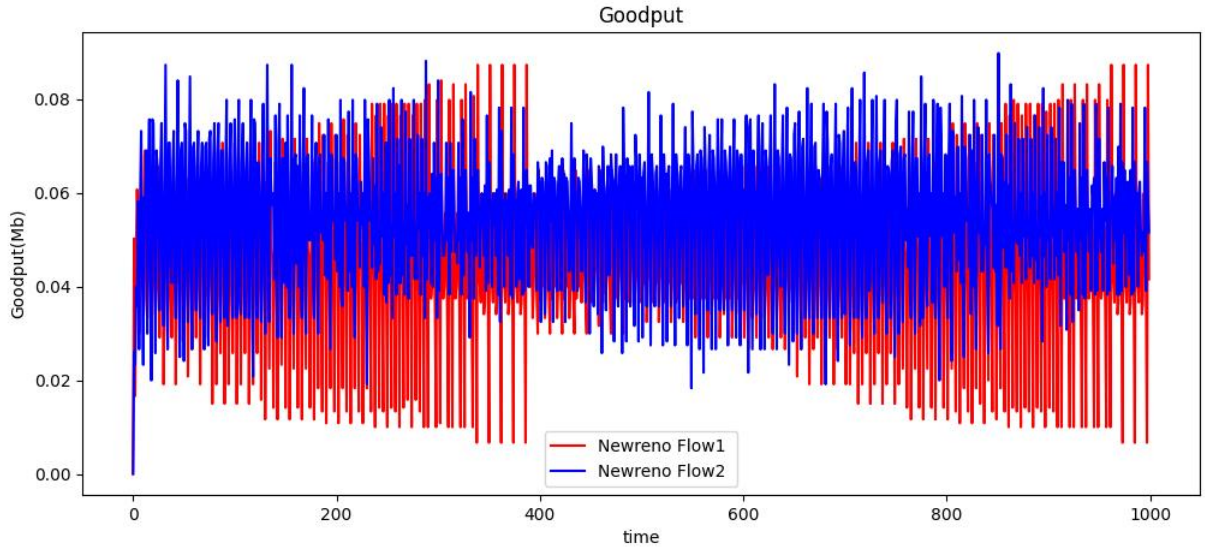


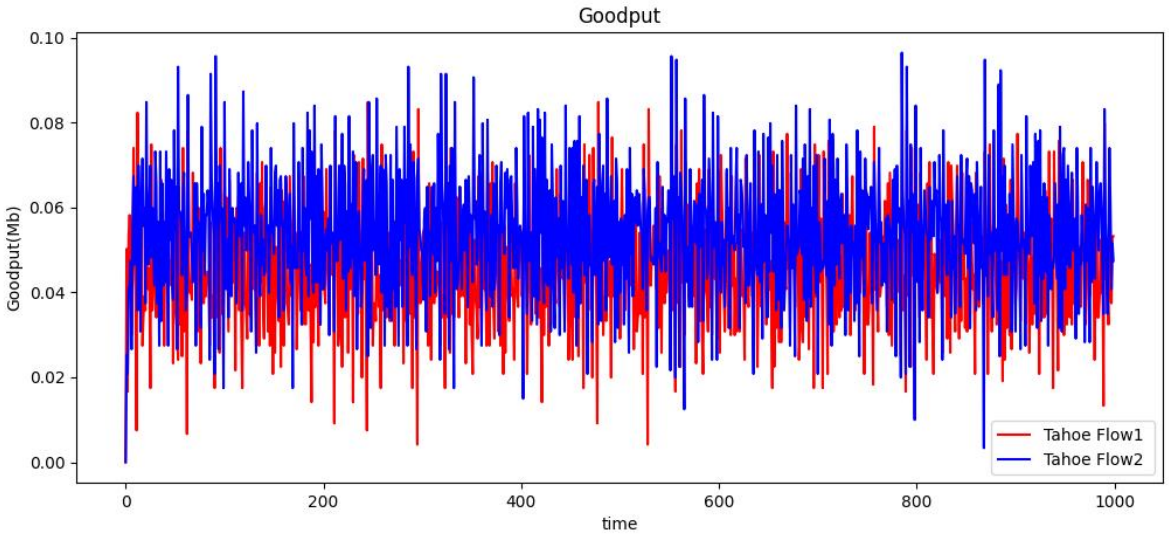
تحلیل:همانطور که در نمودار ها مشاهده می شود در الگوریتم Newreno میزان تغییرات CWND نسبت به Tahoe در طول شبیه سازی کمتر بوده و در Vegas هم بعد از مدت کوتاهی مقدار CWND کوتاه شده است که دلیل آن این است که این روش نرخ ارسال بسته ها را برای congestion avoidance ای که دارد به میزان ثابتی قرار می دهد تا از وقوع packet loss جلوگیری کند.

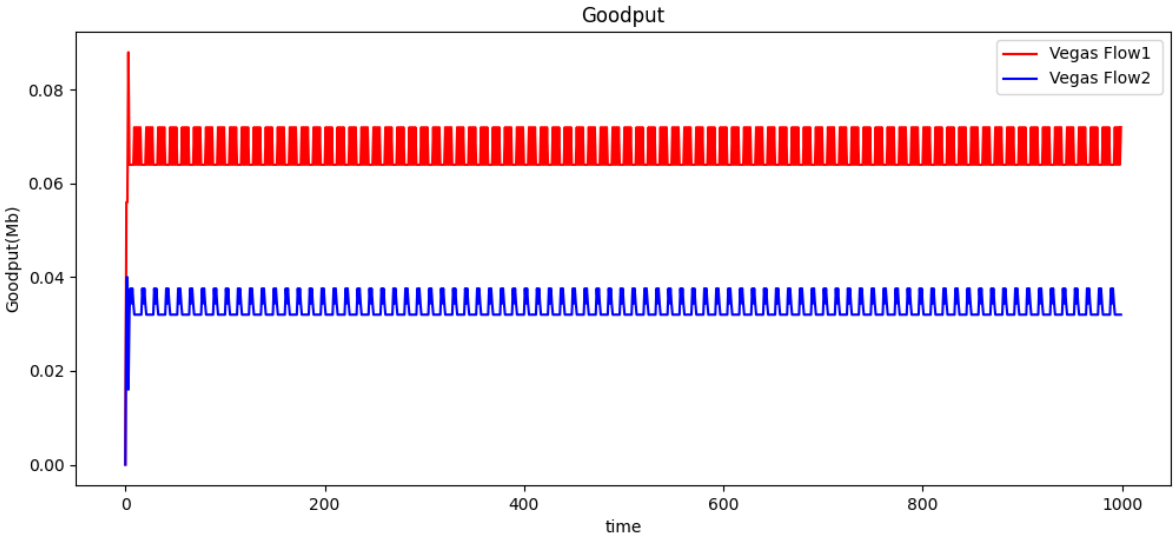
نمودار های مربوط به Goodput:



نمودار های جداگانه هر روش هم بدین شکل شدند:



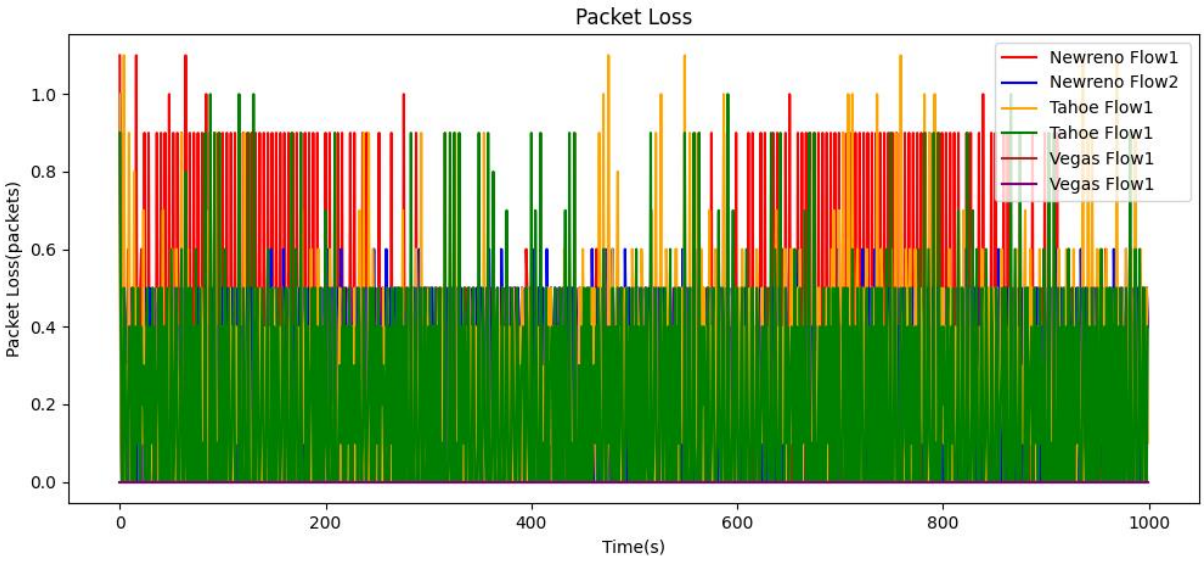




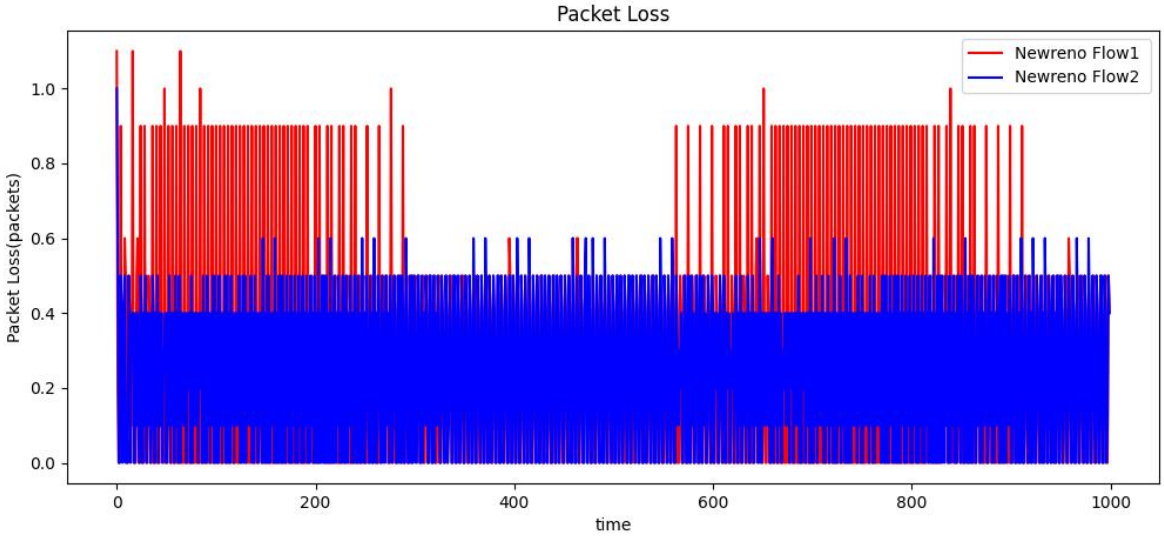
تحلیل:واحد اندازه گیری goodput در این نمودار ها Mb/s است.

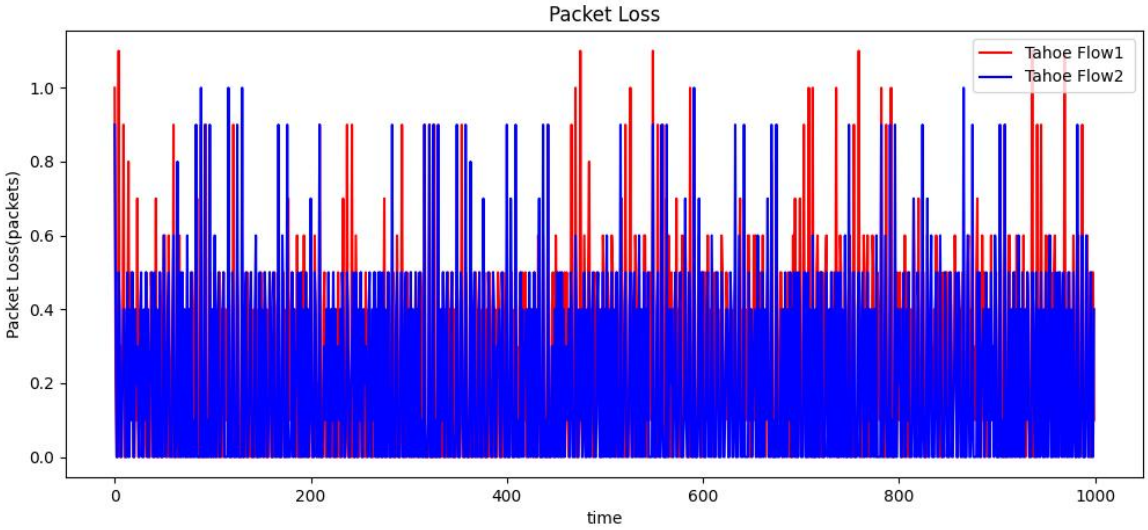
همانطور که در نمودار ها مشاهده می شود در Vegas بعد از مدت کوتاهی به علت آنکه نرخ ارسال بسته ها برابر عدد ثابتی شده و تلاش برای جلوگیری از packet loss انجام می شود نرخ goodput به اندازه دو روش دیگر نوسان نمی کند و نوسان کمتری دارد.

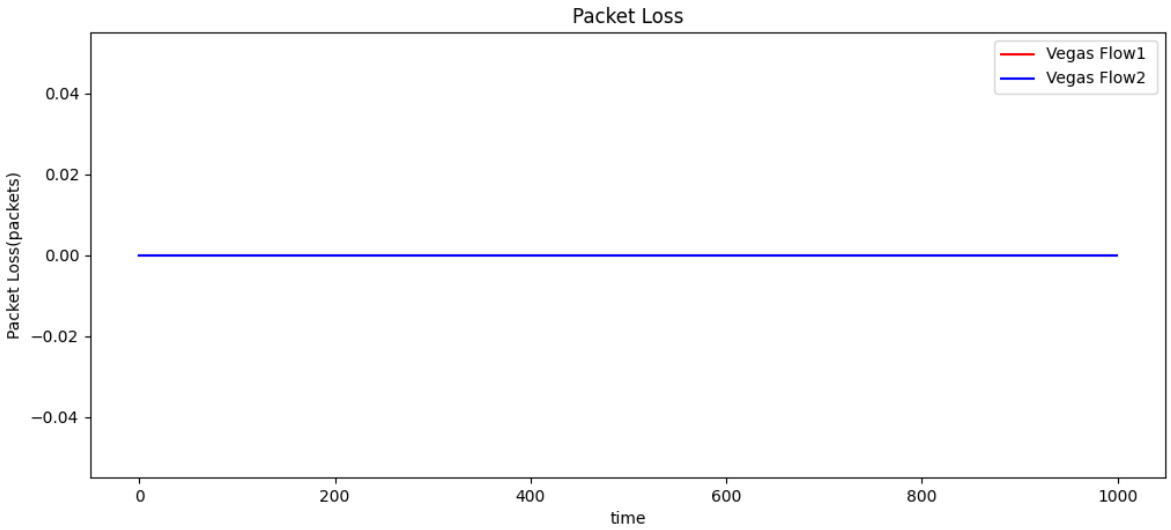
نمودار های مربوط به Packet Loss:



نمودار های جداگانه هر روش هم بدین شکل شدند:

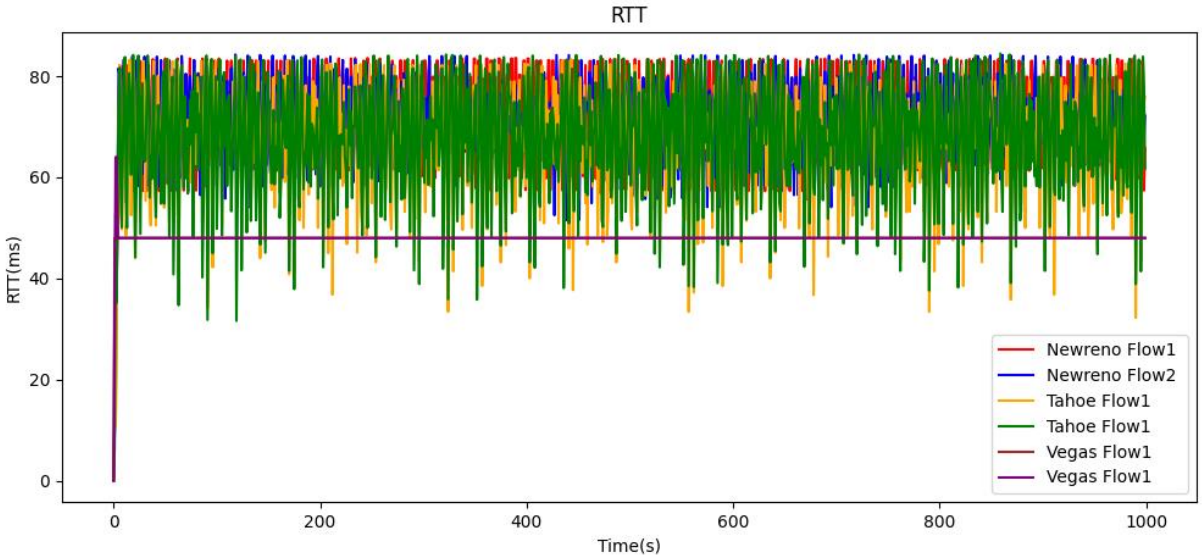




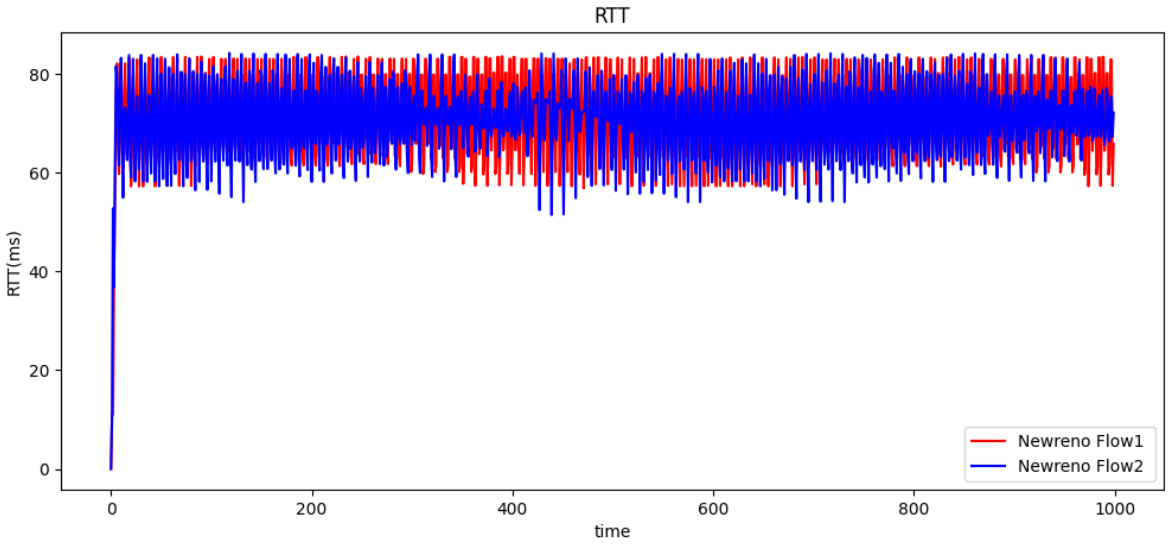


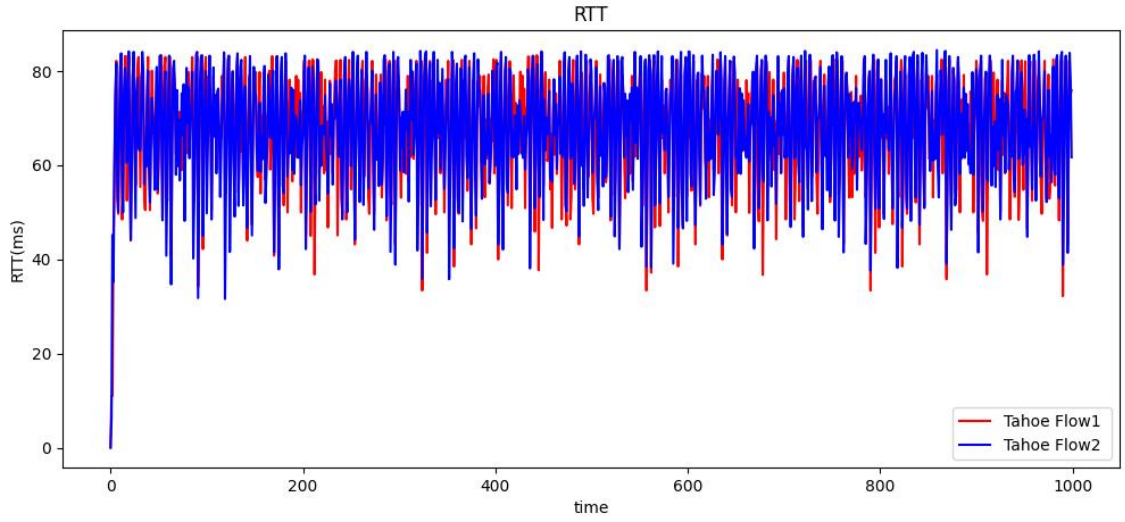
تحلیل:همانطور که در نمودار ها مشاهده می شود در روش Vegas به علت تلاش هایی که برای جلوگیری از packet loss انجام می شود عملا میزان packet loss در کل مدت شبیه سازی برای هر دو flow برابر 0 بوده و هیچ drop ای رخ نداده است.در دو روش دیگر ولی packet loss رخ داده و روش Newreno هم عملکرد بهتری نسبت به Tahoe در کم بودن تعداد drop ها داشته است.

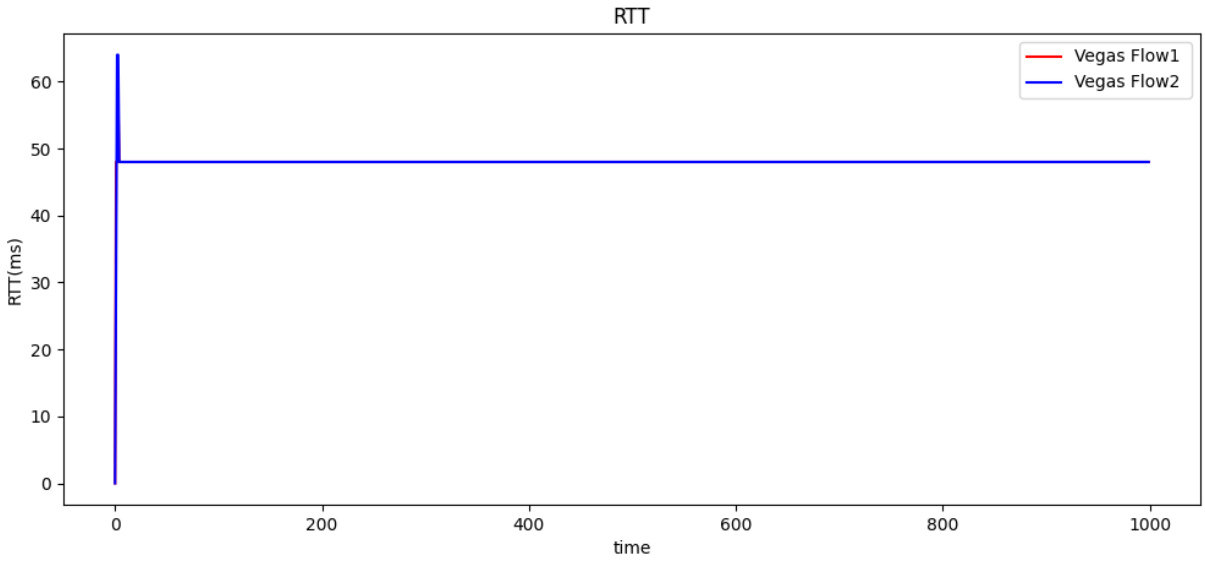
نمودار های مربوط به RTT:



نمودار های جداگانه هر روش هم بدین شکل شدند:







تحلیل:در الگوریتم Tahoe میزان نوسان نسبت به دو الگوریتم دیگر بیشتر است و در Vegas باز هم به علت تلاش های آن برای محاسبه RTT و جلوگیری از افزایش آن مشاهده می شود که بعد مدت کمی این میزان ثابت شده و تا آخر شبیه سازی به همین شکل باقی مانده است زیرا میزان loss برابر صفر بوده و لذا rtt افزایش پیدا نکرده است.

توضیح کد پایتون استفاده شده برای انجام شبیه سازی:

در کد پایتون نوشته شده ده بار این شبیه سازی انجام شده و بعد از هربار انجام شبیه سازی از فایل های خروجی تولید شده نتایج خوانده شده و با مقادیر قبلی جمع زده شدند.

دو تابعی که برای جمع زدن این مقادیر استفاده شدند بدین شکل هستند:

```
def setParams(tcpMethod, cwnd, rtt, goodput, tcpConnectionNum):
    variablesInfo = pd.read_csv(tcpMethod + "VariableTrace" + str(tcpConnectionNum) + ".tr", sep="\t")
    cwndColumnList = list(variablesInfo["cwnd"])
    rttColumnList = list(variablesInfo["rtt"])
    goodputColumnList = list(variablesInfo["goodput"])
    for i in range(len(cwnd)):
        cwnd[i] += cwndColumnList[i]
        rtt[i] += rttColumnList[i]
        goodput[i] += goodputColumnList[i]


def setDrops(tcpMethod, drops1, drops2):
    packetsInfo = pd.read_csv(tcpMethod + "PacketTrace.tr", sep=" ")
    packetsInfo.columns = ["event", "time", "sending", "receiving", "protocol", "size", "flags", "fid",
                           "src", "dst", "sequence Number", "packetId"]
    allDrops = packetsInfo.loc[packetsInfo["event"] == "d"]
    drops1TimeList = list(allDrops.loc[packetsInfo["fid"] == 1]["time"])
    drops2TimeList = list(allDrops.loc[packetsInfo["fid"] == 2]["time"])

    for i in range(len(drops1TimeList)):
        if drops1TimeList[i] < 1000:
            drops1[int(drops1TimeList[i])] += 1

    for i in range(len(drops2TimeList)):
        if drops2TimeList[i] < 1000:
            drops2[int(drops2TimeList[i])] += 1
```

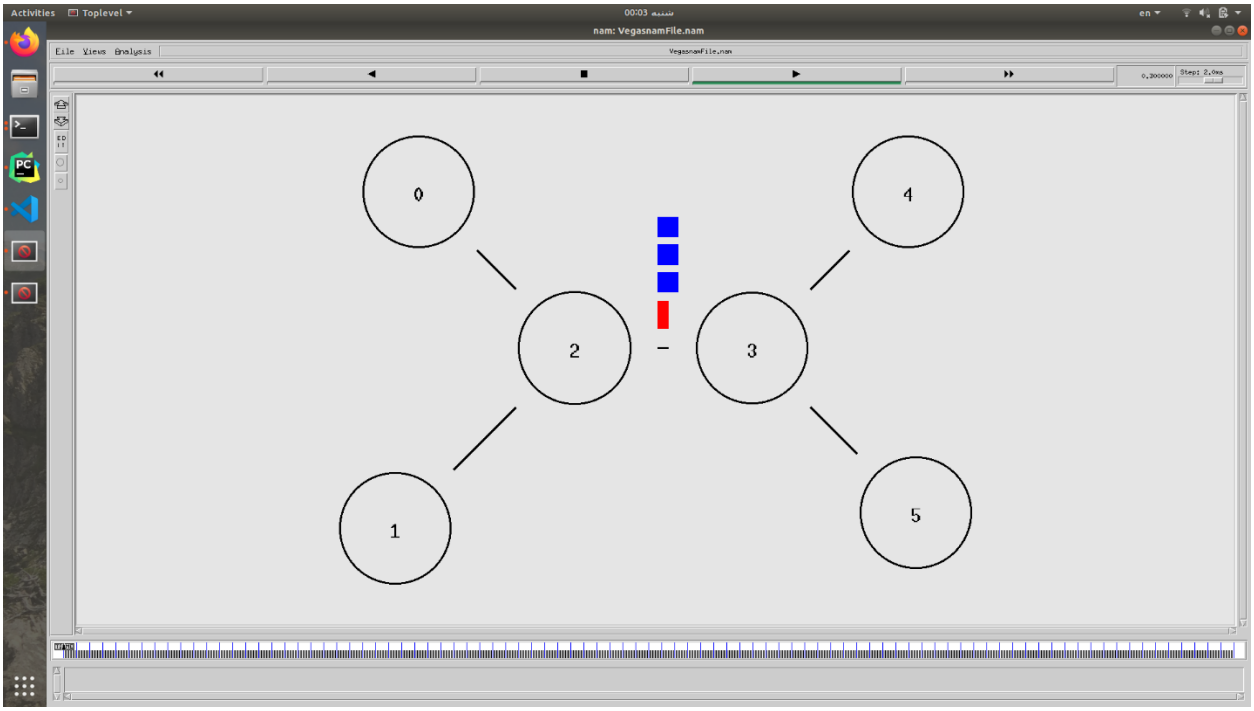
و تابعی که برای صدا زدن دو تابع بالا و انجام شبیه سازی به میزان ده بار و در نهایت میانگین گیری انجام شد تابع زیر بود:

```
def getTcpInfo(tcpMethod):
    cwnd1, rtt1, drops1, goodput1 = [0] * 1000, [0] * 1000, [0] * 1000, [0] * 1000
    cwnd2, rtt2, drops2, goodput2 = [0] * 1000, [0] * 1000, [0] * 1000, [0] * 1000
    createSimulationFolder(tcpMethod)
    for i in range(10):
        os.system("ns NS2_TCP.tcl " + "{" + tcpMethod + "} {" + getAgentType(tcpMethod) + "}")
        setParams(tcpMethod, cwnd1, rtt1, goodput1, 1)
        setParams(tcpMethod, cwnd2, rtt2, goodput2, 2)
        setDrops(tcpMethod, drops1, drops2)
        moveFilesToProperSimulationFolder(tcpMethod)

    for i in range(1000):
        cwnd1[i] /= 10
        rtt1[i] /= 10
        drops1[i] /= 10
        goodput1[i] /= 10
        cwnd2[i] /= 10
        rtt2[i] /= 10
        drops2[i] /= 10
        goodput2[i] /= 10

    return [[cwnd1, rtt1, drops1, goodput1], [cwnd2, rtt2, drops2, goodput2]]
```

که همانطور که مشاهده می شود کد tcl را ران کرده و سپس نتایج را خوانده و در نهایت جمع نتایج بر 10 را محاسبه کرده(عمل میانگین گیری) و خروجی را باز می گرداند.

و در نهایت با استفاده از کتابخانه matplotlib این مقادیر در نمودار های جداگانه ای رسم شدند.

تصویری از nam هنگام شبیه سازی روش Vegas:



پایان