به نام خداوند بخشنده مهربان

موضوع:

گزارش پروژه دوم درس شبکه

شبیه سازی و تحلیل مکانیزم های ازدحام شبکه در tcp

ارائه شده توسط:

دکتر احمد خونساری

در:

دانشکده برق و کامپیوتر دانشگاه تهران

گردآوری شده توسط:

علیرضا زارع نژاد اشکذری

بهار99

توضیح ساختار پروژه و فایل ها :

در پوشه ی اصلی یک فایل main.tcl وجود دارد که حاوی فایل otcl جهت پیاده سازی پروتکل و ساختار اصلی شبکه است.
پوشه ی plot حاوی تمامی نمودار ها و تصاویر شبیه سازی است. در این پوشه به ازای هر agent tcp یعنی به ازای Tahoe ، Newreno ، Vegas و به ازای هر دو flow موجود در شبکه به ازای هر کدام از متغیر های خواسته شده یعنی cwnd و goodput و packet loss و rtt نمودار آورده شده است. پس در این پوشه ۲۸ تصویر png وجود دارد.
همچنین در این پوشه سه دایرکتوری برای سه agent tcp و در داخل هر یک به ازای تمامی متغیر های cwnd و goodput و rtt و droprate یک فولدر وجود دارد که به ازای هر flow دو پوشه , flow0 flow1 وجود دارد که دیتاهای شبیه سازی ناشی از 10 بار اجرا در آن قرار دارد.
همچنین از jupyter notebook برای کشیدن نمودار ها و کد پایتون جهت تحلیل فایل ها و tracefile استفاده شده است.

نحوه ی اجرا:

ایتدا در دایرکتوری اصلی پروژه فایل Main.tcl را اجرا می کنیم. برای این کار کافی است به صورت زیر عمل کنیم. در command line می باست نوع tcp و شماره فایل نولیدی را مشخص کنیم که فایل حاصل در پوشه مربوطه در flow قرار می گیرد. عملا iteration را مشخص می کنیم. به عنوان مثال :
ns main.tcl Tahoe 1
فایل main را اجرا کرده و نوع tcp را Tahoe قرار می دهد و فایل txt.1 مرتبط را در پوشه های زیر قرار می دهد.

Tahoe/cwnd/flow0/1.txt
Tahoe/cwnd/flow1/1/txt
Tahoe/goodput/flow0/1.txt
Tahoe/goodput/flow1/1.txt
Tahoe/Newreno/flow0/1.txt
Tahoe/Newreno/flow1/1.txt

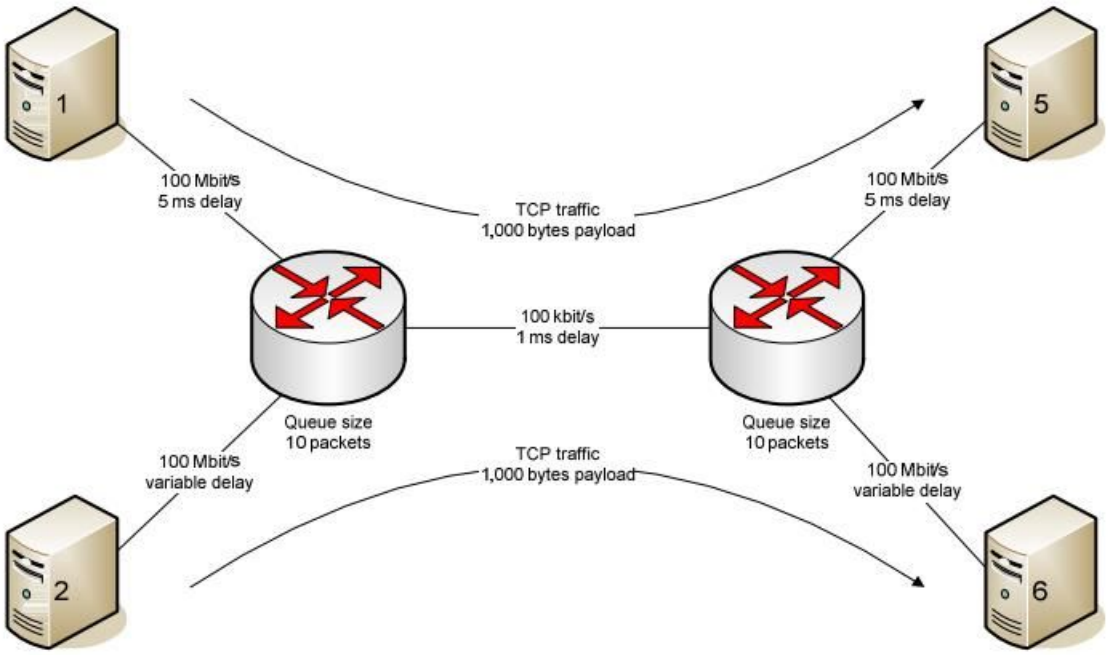
پس از اجرا فایل کافی jupyter notebook را در ترمینال زده و کد پایتون را اجرا کینم تا میانگین دادهی ده بار اجرا به دست آیند و نمودارها کشیده شوند.
به دلیل سادگی به کمک دستور os.system اسکریپت را اجرا می کنیم. کد به صورت زیر خواهد بود. پس برای اجرا دیگر کافی است فایل ژوپیتر را اجرا کنیم.

```
tcp_agents = ["Newreno","Tahoe","Vegas"]
for tcp in tcp_agents:
    for i in range(10):
        os.system("ns main.tcl " + tcp + " " + str(i+1))
```

نمودار های حاصل همان طور که گفته شد در پوشه plot قرار می گیرند.

نحوه پیاده سازی شبکه مطلوب مسئله و نتیجه شبیه سازی:

با توجه به شبکه موجود در صورت پروژه و شکل زیر در فایل basic.tcl به شیوه ای که در ادامه توضیح داده می شود ، سعی میکنیم موارد مطرح شده را به کمک ns2 پیاده کنیم.



ابتدا نود های موجود را می سازیم. با توجه به شکل، نود های 0 و 2 نقش فرستنده و نود های 5 و 6 نقش گیرنده را دارند. دو روتر نیز در شکل وجود دارد.

```
# Create the nodes:
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
```

در ادامه به ایجاد لینک بین نود ها و نسبت دادن transmission rate و propagation delay به آن ها می پردازیم. برای این کار از duplex-link استفاده می کنیم که یک لینک دو طرفه خواهد بود و در این صورت ack ها در مسیر برگشت کاملا قابلیت جابه جایی دارند

با توجه به اینکه تاخیر زمانی برای لینک بین نود 1 و روتر سمت چپ ، و نود 5 و روتر سمت راست مقدار متفاوتی در هر بار اجرا است ، برای این کار یک تابع تعریف می کنیم که عددی رندوم تولید کند.

```
proc randomGenreator {min max} {
    return [expr $min + round(rand() * ($max - $min))]
}
```

در ادامه نحوه ی مقدار دهی لینک ها را در فایل tcl آورده ایم.

```
$ns duplex-link $n0 $n2 100Mb 5ms DropTail
$ns duplex-link $n1 $n2 100Mb [randomGenreator 5 25]ms DropTail
$ns duplex-link $n2 $n3 100Kb 1ms DropTail
$ns duplex-link $n3 $n4 100Mb 5ms DropTail
$ns duplex-link $n3 $n5 100Mb [randomGenreator 5 25]ms DropTail
```

همان طور که در کد بالا آورده شده است لینک بین نود 0 و 2 دارای نرخ انتقال 100 مگابیت بر ثانیه و دیلی 5 میلی ثانیه است. همچنین لینک بین نود 1 و2 دارای نرخ انتقال 100 مگابیت بر ثانیه و دیلی رندوم بین 5 تا 25 میلی ثانیه خواهد بود. لینک بین دو روتر دارای نرخ انتقال 100 کیلو بیت بر ثانیه و دیلی 1 میلی ثانیه است. نرخ انتقال برای لینک بین نود های 3 و 4 برابر 100 مگابیت بر ثانیه و دیلی ۵ میلی ثانیه و در نهایت لینک بین نود 3 و 5 دارای نرخ انتقال 100 مگابیت بر ثانیه و دیلی رندوم است.

با توجه به اینکه گفته شده بود اندازه صف در روتر ها برابر 10 است و توجه به اینکه صف روتر در پورت خروجی آن است ، پس به صورت زیر این کار را انجام دادیم.

```
$ns queue-limit $n2 $n3 10
$ns queue-limit $n3 $n4 10
$ns queue-limit $n3 $n5 10
```

برای شکل بندی و جهت هر کدام از لینک ها تا به صورت شکل بالا در بیاید به صورت زیر عمل می کنیم.

```
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down
```

تا به اینجا نود ها و لینک بین نود ها در شبکه شکل گرفت.حال به سراغ ایجاد دو flow و ایجاد agent های مربوطه برای ارسال و دریافت پکت ها می رویم.

همان طور که در صورت پروژه ذکر گردید ، قرار است شبیه سازی را برای سه agent مختلف tcp انجام دهیم و تحلیل های مربوطه را بنویسیم.

برای این کار می توانیم به شکل زیر عمل کنیم و نوع tcp را متناسب با چیزی که از command line گرفته ایم به وجود آوریم.

```
#set a TCP
if {$variant == "Tahoe"} {
  set tcp0 [new Agent/TCP]
  set tcp1 [new Agent/TCP]
} elseif {$variant == "Newreno"} {
  set tcp0 [new Agent/TCP/Newreno]
  set tcp1 [new Agent/TCP/Newreno]
} elseif {$variant == "Vegas"} {
  set tcp0 [new Agent/TCP/Vegas]
  set tcp1 [new Agent/TCP/Vegas]
}
```

tcpO و tcp1 در حقیقت تا اینجا new شد. حال کافی است که نود های فرستنده را به tcp نسبت دهیم.
در نهایت کاری که می کنیم باید نود شروع و فرستنده پکت ها را برای دو جریان مشخص سازیم. این کار را توسط دستور زیر انجام می دهیم.

```
# Add a TCP sending module to node nO
$ns attach-agent $nO $tcpO
# Add a TCP sending module to node n1
$ns attach-agent $n1 $tcp1
```

همان طور که در صورت پروژه ذکر شده ، باید برای پکت ها time to live را برابر ۶۴ قرار دهیم، این کار را توسط تنظیم کردن ttL برای دو جریان انجام می دهیم. در اینجا برای دو flow یک id در نظر گرفتیم و پکت ها هر flow را با رنگی مشخص کردیم.

```
$tcpO set class_ O
$tcp1 set class_ 1
$ns color O Red
$ns color 1 Blue
$tcpO set ttl_ 64
$tcp1 set ttl_ 64
```

به طور مشابه باید نوع agent گیرنده پکت ها و نود دریافتی هر جریان را به طور جداگانه مشخص کنیم ، که به صورت زیر انجام می شود.

```
#Create a TCP receive agent (a traffic sink) and attach it to N5
set sinkO [new Agent/TCPSink]
$ns attach-agent $n4 $sinkO
```

```
#Create a TCP receive agent (a traffic sink) and attach it to N6
set sink1 [new Agent/TCPSink]
$ns attach-agent $n5 $sink1
```

و در انتها برای شکل گیری جریان O از نود O به نود 5 و جریان 1 از نود 1 به نود 5 به شکل زیر عمل می کنیم.

```
$ns connect $tcp0 $sink0
$ns connect $tcp1 $sink1
```

حال برای اینکه پکت ها شروع به ارسال کنند و جریان داده ای برای هر دو flow شکل بگیرد از ftp به صورت زیر استفاده می کنیم.

```
# Setup a FTP traffic generator on "tcp0"
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set type_ FTP

# Setup a FTP traffic generator on "tcp1"
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
```

در اینجا لازم است توضیحی داده شود. میتوانستم به جای ftp از cbr استفاده کنیم تا ترافیک شبکه را به وجود آورد. از cbr زمانی استفاده می شود که نرخ فرستادن پکت ها بخواهد ثابت باشد و معمولا در udp از آن استفاده می شود ولی tcp به خاطر داشتن مکانیزم کنترل ازدحام تفاوتی ندارد و اغلب لینک ها از ftp استفاده کرده اند.

خب تا در نهایت زمان انجام شبیه سازی را تعیین می کنیم. و با دستور run شبیه سازی را آغاز می کنیم.

```
# Schedule start/stop times
$ns at 0.0   "$ftp0 start"
$ns at 1000.0 "$ftp0 stop"
$ns at 0.0  "$ftp1 start"
$ns at 1000.0 "$ftp1 stop"

# Set simulation end time
$ns at 1000.0 "finish"
```

در اینجا نوشتن فایل tcl به اتمام می رسد.تنها نیاز به trace کردن یک سری variable است برای رسم نمودار و تحلیل به آنها نیاز داریم که در ادامه توضیح داده می شود.

فعلا می خواهیم نتیجه شبیه سازی و فایل های تولیدی را توضیح دهیم.

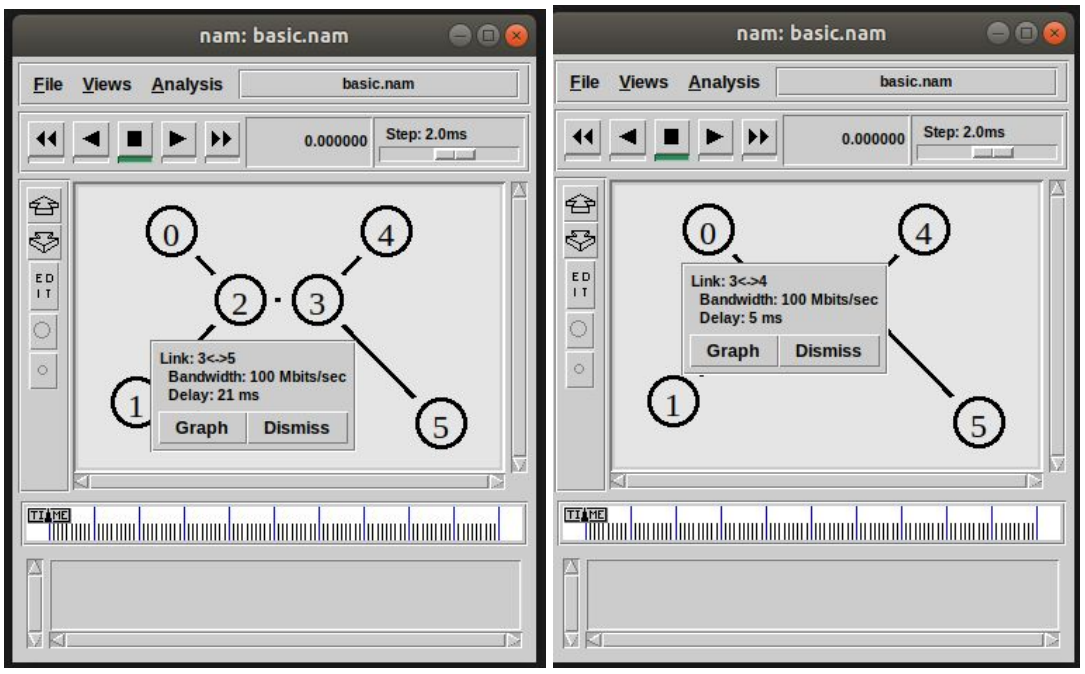
ابتدا ns basic.tcl را در ترمینال وارد می کنیم. خروجی دو فایل خواهد بود. یکی فایل main.nam که برای شبیه سازی از آن استفاده می شود و در دایرکتوری اصلی قرار گرفته و برابر با آخرین اجرا می باشد و دیگری فایل trace که با پسوند .tr مشخص شده و در دایرکتوری مربوطه متناسب با نوع tcp دریافتی از command line در پوشه droprate/tracefiles قرار می گیرد.

```
#Open the nam file basic.nam and the variable-trace file basic.tr
set namfile [open main.nam w]
$ns namtrace-all $namfile
set filetrace "$variant/droprate/tracefiles/$num_run.tr"
puts $filetrace
set tracefile [open $filetrace w]
$ns trace-all $tracefile
```

فایل num_run.tr در حقیقت شامل event ها یعنی اطلاعاتی چون drop ، send ، receive شدن پکت ها در هر نود و یا trace کردن variable هایی است که مشخص می کنیم مثلا cwnd برا تعیین تغییرات سایز پنجره هر جریان tcp.

```
# Let's trace some variables
$tcp0 attach $tracefile
$tcp0 tracevar cwnd_
$tcp1 attach $tracefile
$tcp1 tracevar cwnd_
```

برای اجرای فایل شبیه سازی می توانیم دستور nam basic.nam را در ترمینال بزنیم. نتیجه شبیه سازی به صورت زیر خواهد بود. مثلا به شکل زیر خواهد بود و ملاحظه می کنیم که اطلاعات لینک ها به درستی تعیین شده است.



حال به توضیح نحوه پیاده سازی هر بخش می پردازیم.

CWND یا نرخ تغییر پنجره در tcp

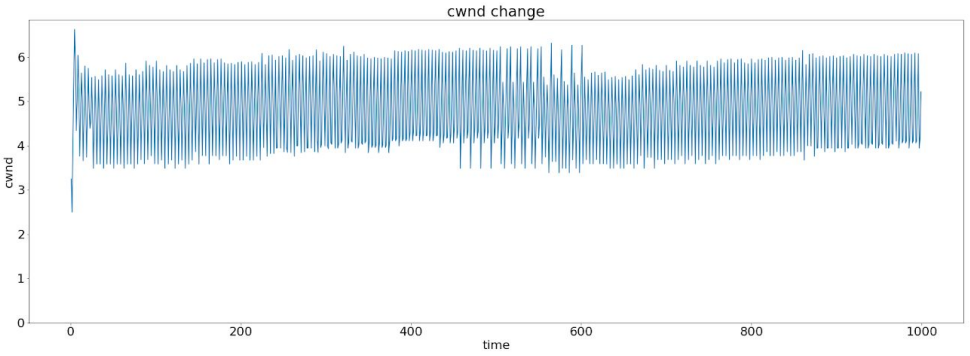
خب با توجه به اینکه در مرحله قبل توپولوژی شبکه را تعیین کردیم. حال با توجه به اطلاعات کی از command line گرفتیم با توجه به کد نوشته شده در زیر به ازای هر جریان عملا یک فایل file_cwnd نسبت داده می شود که در هر ثانیه در فایل مربوطه زمان و سایز پنجره window را نشان می دهد. این سایز را به وسیله ی دستور set cwnd در یافت می کنیم.
فرمت فایل تولیدی در هر خط به شکل زیر است.

time space cwnd_

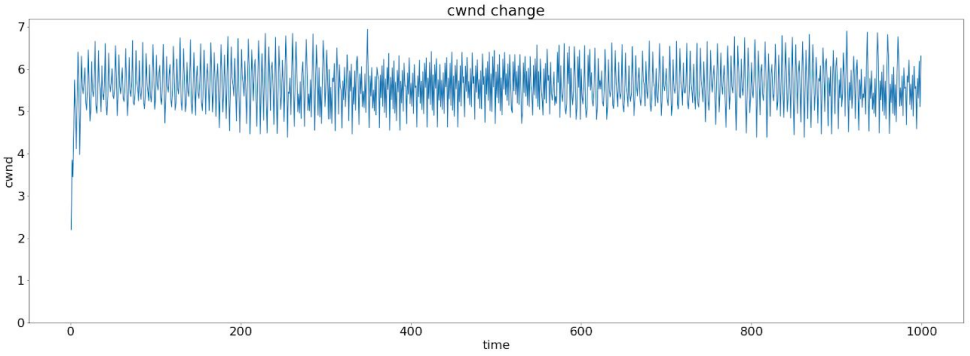
```
proc plotWindow {tcpSource outfile} {
  global ns
  set now [$ns now]
  set cwnd [$tcpSource set cwnd_]

###Print TIME CWND   for  gnuplot to plot progressing on CWND
  puts  $outfile  "$now $cwnd"
  $ns at [expr $now+1] "plotWindow $tcpSource  $outfile"
}
set file_cwnd0 "$variant/cwnd/flow0/$num_run.txt"
set file_cwnd1 "$variant/cwnd/flow1/$num_run.txt"
$ns  at  0.0  "plotWindow $tcp0  $outfile0"
$ns  at  0.0  "plotWindow $tcp1  $outfile1"
```

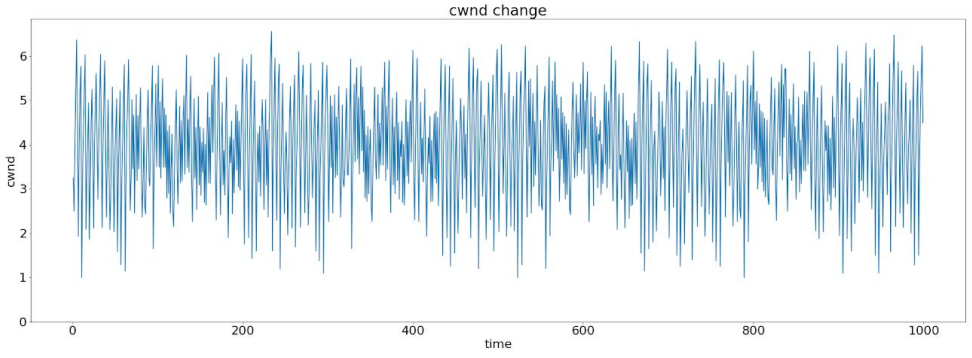
حال پس از اجرا ده بار این اسکریپت به کمک کد پایتون فایل ها را خوانده و میانگین داده ها ی فوق را به دست می آوریم و در فایل mean.txt می ریزیم.
در نهایت به کمک matplotlib و یا gnuplot به سادگی از فایل نهایی نمودار مربوطه را می کشیم.
محور x زمان و محور y شامل اندازه پنجره می باشد.
در زیر نمودار مربوط به هر حالت و نیز نمودار کلی شامل هر سه حالت tcp و هر flow آورده شده است.
کد مربوطه نیز در فایل ژوپیتر آورده شده است.



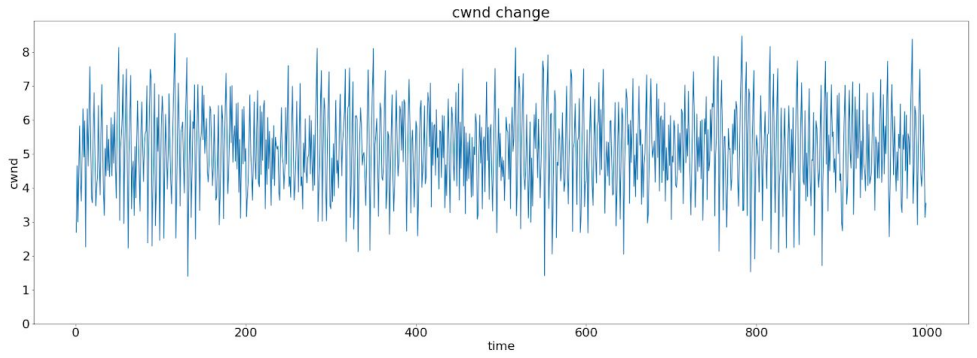
Newreno_cwnd_flow0



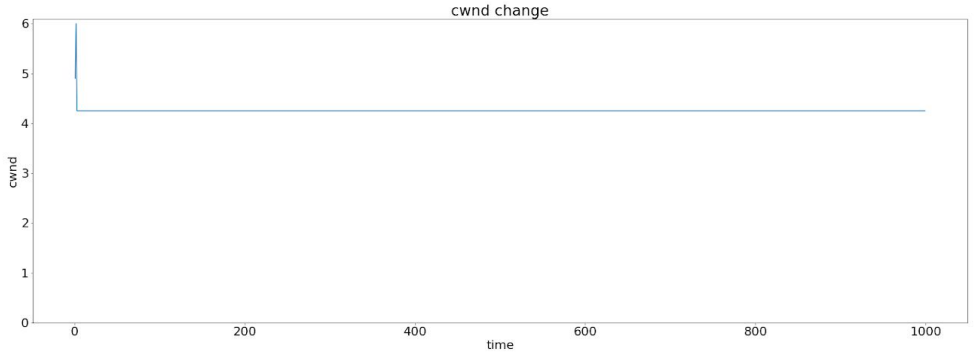
Newreno_cwnd_flow1



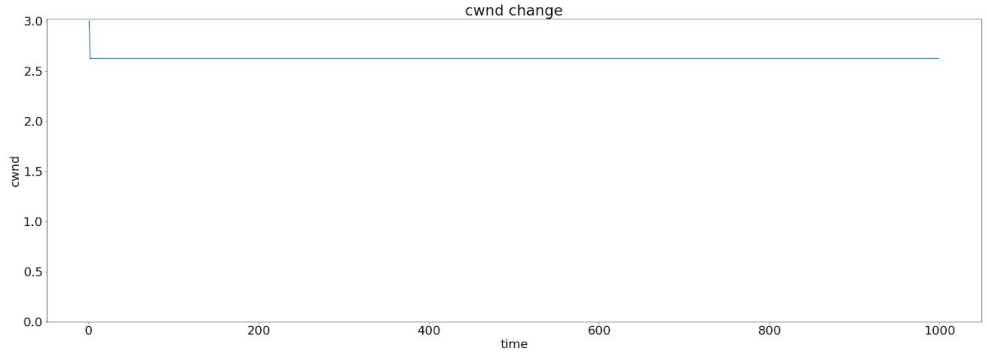
Tahoe_cwnd_flow0



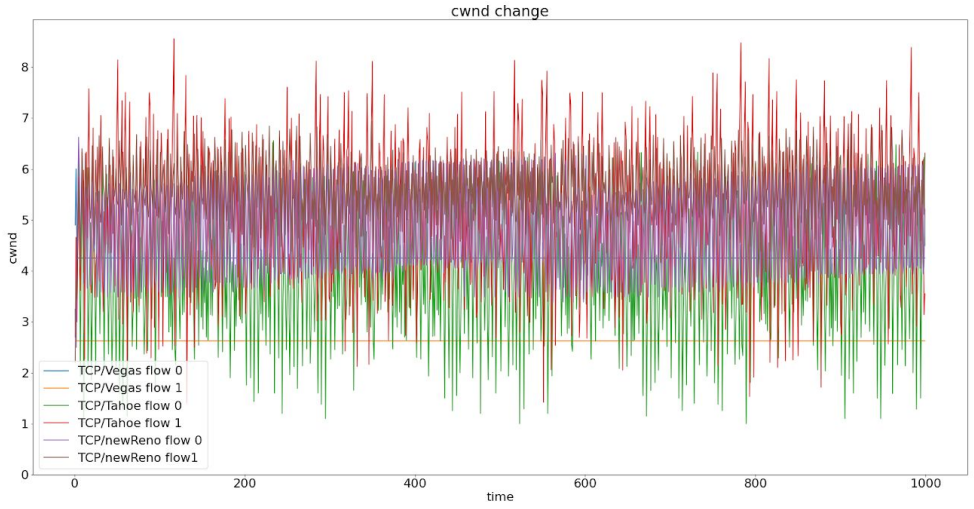
Tahoe/cwnd/flow1



Vegas/cwnd/flow0



Vegas/cwnd/flow1



cwnd

THROUGHPUT , GOODPUT

throughput در حقیقت مقدار دیتایی است که توسط مقصد دریافت می شود. average throughput در حقیقت مقدار throughput در واحد زمان است.
فرض کنیم که reciever در اصل 60Mb دیتا را در 1 دقیقه دریافت می کند.
پس throughput در این بازه ی زمانی برابر 60Mb است و متوسط throughput برابر 1Mb/s است.

دو نوع throughput داریم.در اینجا فقط به تشریح goodput می پردازیم. متوسط دیتایی است که در واحد زمان توسط receiver دریافت می شود به شرطی که retransmission نباشد.

برای اطلاعات مربوط به goodput کافی است که trace application را به tcp sink وصل کنیم.

TraceApp.ns

```
Class TraceApp -superclass Application
TraceApp instproc init {args} {
        $self set bytes_ 0
        eval $self next $args
}
TraceApp instproc recv {byte} {
        $self instvar bytes_
        set bytes_ [expr $bytes_ + $byte]
        return $bytes_
}
```

عبارت superclass- نشان می دهد که کلاس traceapp بچه کلاس application است.کلاس فوق دو متد دارد.

یکی init که توسط اپراتور new فراخوانی می شود وقتی که یک شی از کلاس traceapp ساخته شود. همانند constructor. این تابع متغیر bytes را برابر صفر مقداردهی می کند. این متغیر تعداد بایت های دریافتی را نشان می دهد.
متد دیگر نیز که درحقیقت override شده است ، recv می باشد.این متد مقدار پارامتر ورودی را با متغیر کلاس جمع می کند.

حال کافی است یک object از traceapp بسازیم و آن را به tcp sink وصل کنیم. و عملا tcp sink وظیفه ی فراخوانی متد recv را دارد.

```
set traceapp0 [new TraceApp]      ;# Create a TraceApp object
set traceapp1 [new TraceApp]      ;# Create a TraceApp object

$traceapp0 attach-agent $sink0     ;# Attach traceapp to
$traceapp1 attach-agent $sink1     ;# Attach traceapp to TCPSink
$ns at 0.0 "$traceapp0 start" ;# Start the traceapp object
$ns at 0.0 "$traceapp1 start" ;# Start the traceapp object
```

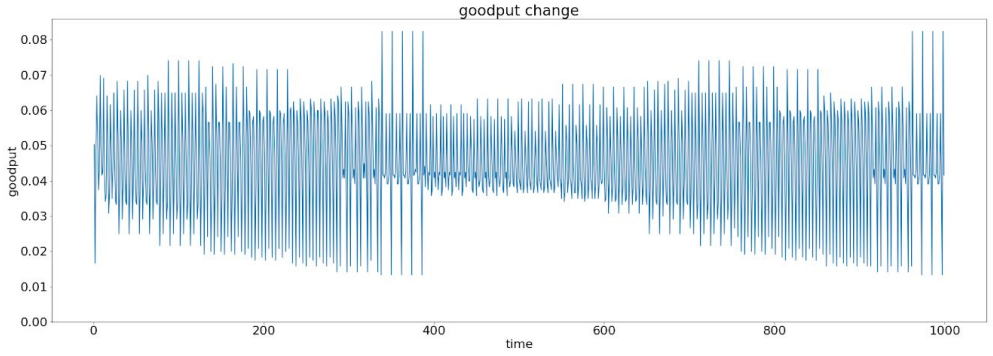
و به طور خلاصه تابع throuput به صورت زیر در خواهد آمد.

```
proc plotThroughput {tcpSink outfile} {
  global ns
  set now [$ns now]
  set nbytes [$tcpSink set bytes_]
  $tcpSink set bytes_ 0
  set time_incr 1.0
  set throughput [expr ($nbytes * 8.0 / 1000000) / $time_incr]
###Print TIME throughput for  gnuplot to plot progressing on throughput
  puts  $outfile  "$now $throughput"
  $ns at [expr $now+$time_incr] "plotThroughput $tcpSink  $outfile"
}
```

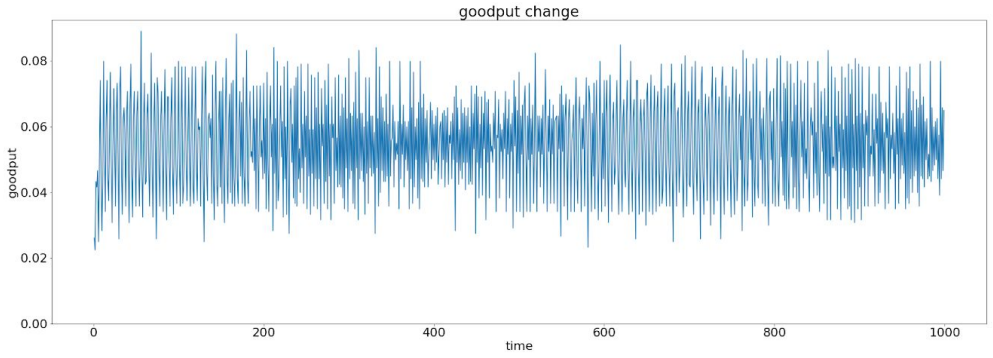
پس به طور خلاصه در goodput بسته ها و پکت هایی که در sequence مرتب دریافت می شوند شمارش می شود و این تاییدی بر تجمعی بودن sequence های دریافتی خواهد بود.
outfile هم مطابق با آنچه از command line گرفته شد ، ساخته می شود.

```
set file_goodput0 "$variant/goodput/flow0/$num_run.txt"
set file_goodput1 "$variant/goodput/flow1/$num_run.txt"
```

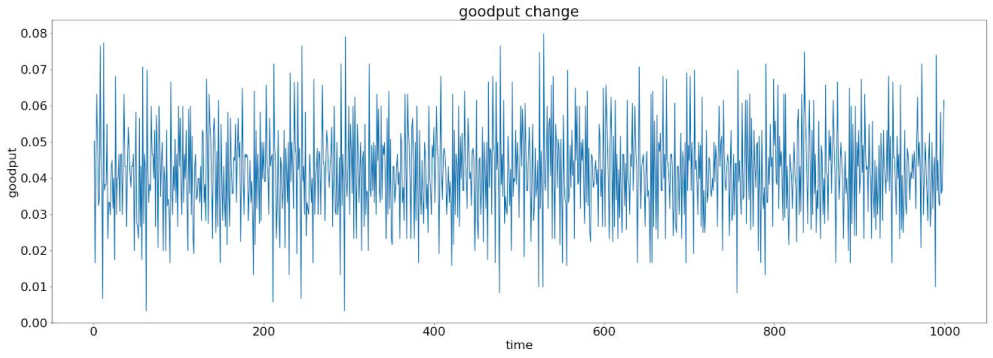
حال نمودارهای خروجی آورده شده است.



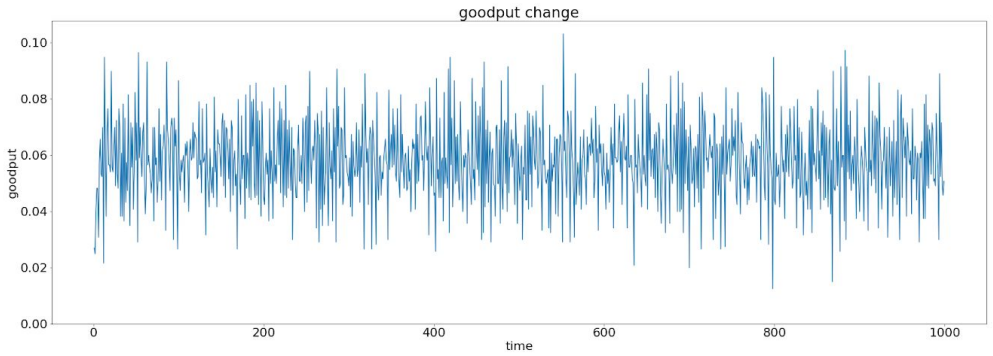
Newreno_goodput_flow0



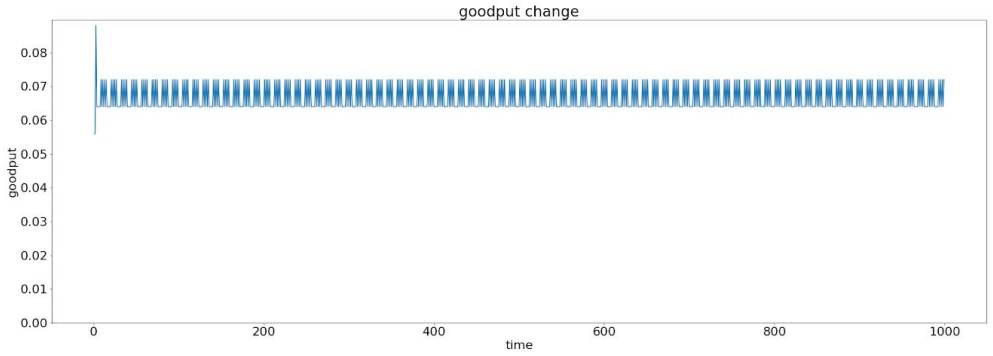
Newreno_goodput_flow1



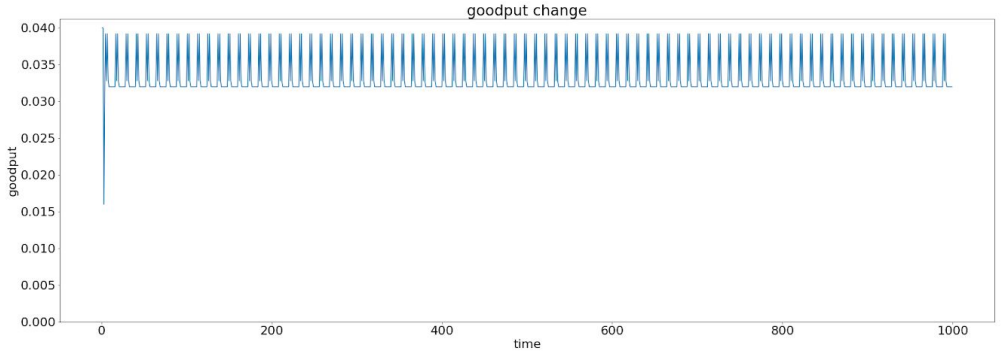
Tahoe_goodput_flow0



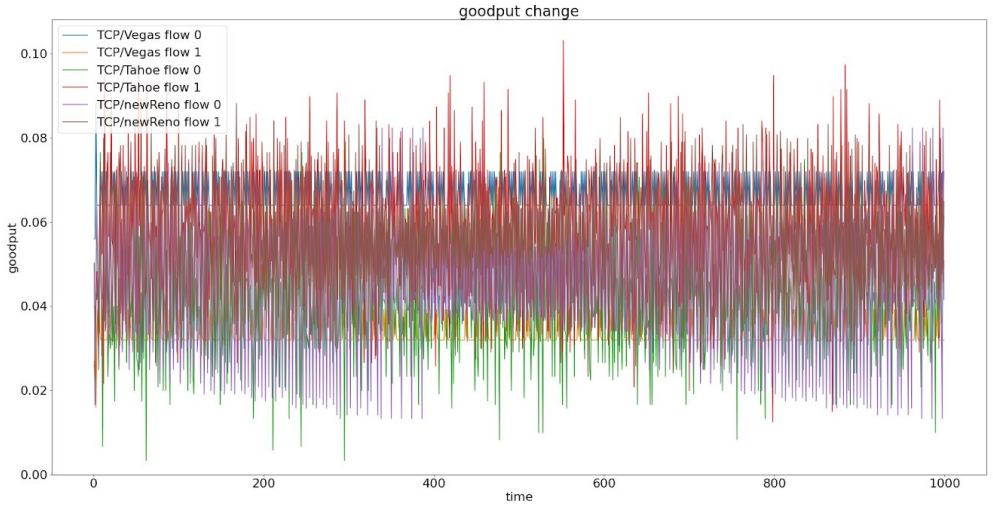
Tahoe_goodput_flow1



Vegas_goodput_flowO



Vegas_goodput_flow1



goodput

RTT

یا round trip delay در حقیقت مدت زمانی است که یک پالس سیگنال و یا یک پکت فرستاده شود و برگردد یا ack آن دریافت شود. و عملا شامل propagation delay خواهد بود.
کاربر اینترنت می تواند با کمک ping مقدار rtt را محاسبه کند.
برای این منظور از agent ping استفاده می کنیم. و همچنین راه دیگری که وجود دارد با استفاده از trace کردن _rtt در فایل خروجی پس از اجرا ns basic.tcl است که اطلاعات rtt را پس از تغییر به ما می دهد.
در اینجا هر دو روش پیاده سازی کد آن آورده شده است.

```
# RTT Calculation Using Ping ------------------------------
set pO [new Agent/Ping]
$ns attach-agent $nO $pO
set p4 [new Agent/Ping]
$ns attach-agent $n4 $p4
#Connect the two agents
$ns connect $pO $p4
set p1 [new Agent/Ping]
$ns attach-agent $n1 $p1
set p5 [new Agent/Ping]
```

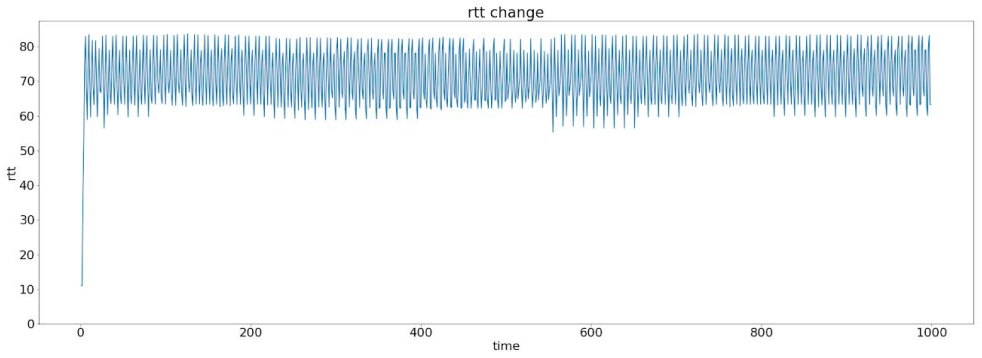
```
$ns attach-agent $n5 $p5
#Connect the two agents
$ns connect $p1 $p5
# Method call from ping.cc file
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received ping answer from \
$from with round-trip-time $rtt ms."
}
# ---------------------------------------------------------
```

خب , برای trace کردن rtt نیز می توانیم به شکل زیر عمل کنیم.

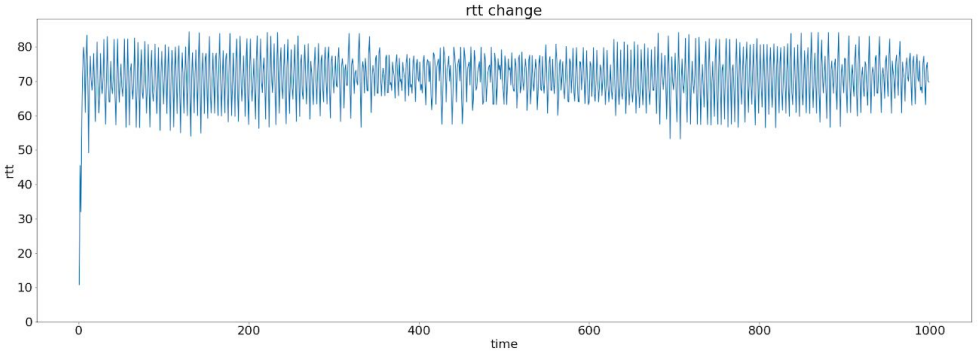
```
set file_rtt0 "$variant/rtt/flow0/$num_run.txt"
set file_rtt1 "$variant/rtt/flow1/$num_run.txt"
#################### CAIC RTT ############################
for {set  i 0} { $i < 1000} {set i [expr {$i + 1}]} {
    $ns at $i "calcRtt $tcp0  $out0 $i"
    $ns at $i "calcRtt $tcp1  $out1  $i"
}
proc calcRtt {tcpSource outfile time} {
    set rtt [$tcpSource set rtt_]
    puts  $outfile  "$time $rtt"
}
```

ابتدا فایل ها متناظر با نوع tcp و شماره فایل ساخته می شود. سپس یک حلقه به 1000 ثانیه داریم که هر بار اطلاعات rtt را از tcp دریافت می کنیم و در فایل میریزیم.

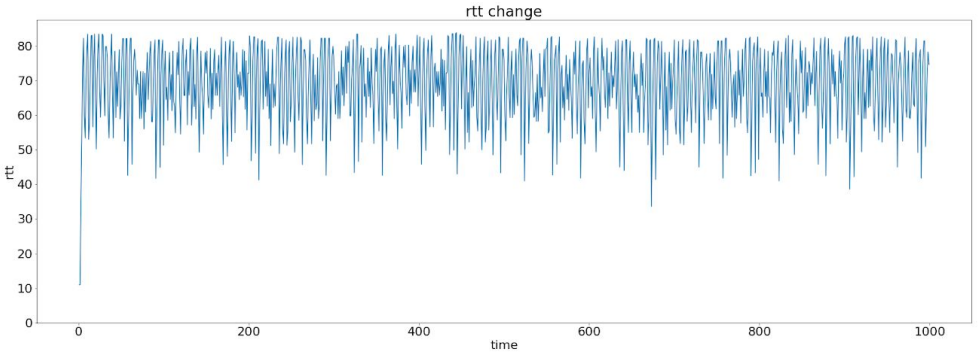
نمودار های در فایل ژوپیتر و در زیر آمده است. در انتها تنها کافی است میانگین گیری کنیم.



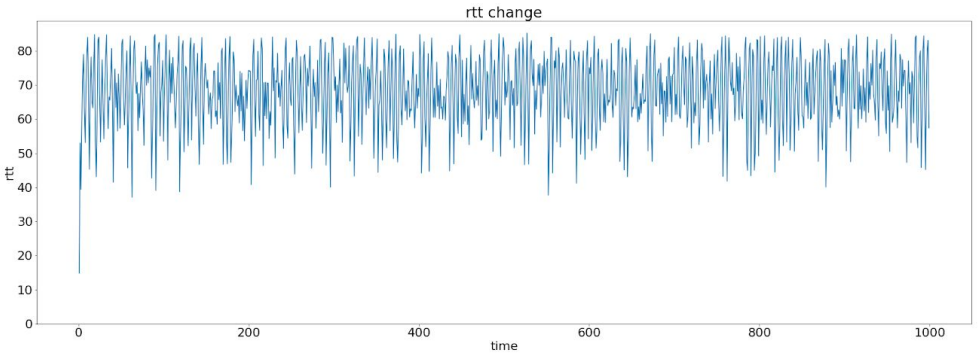
Newreno_rtt_flow0



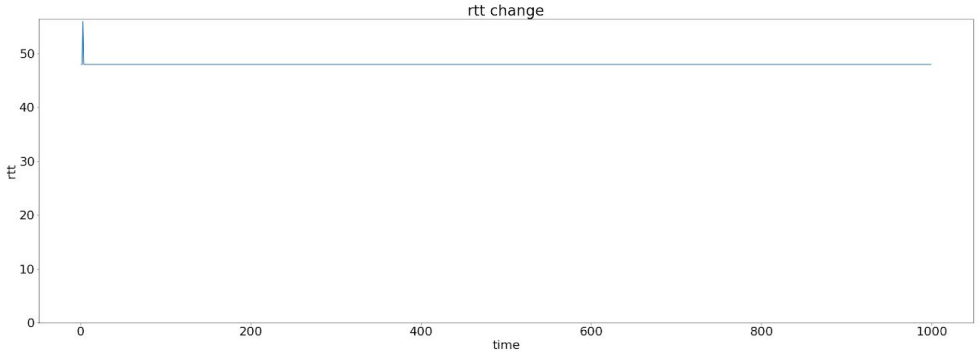
Newreno_rtt_flow1



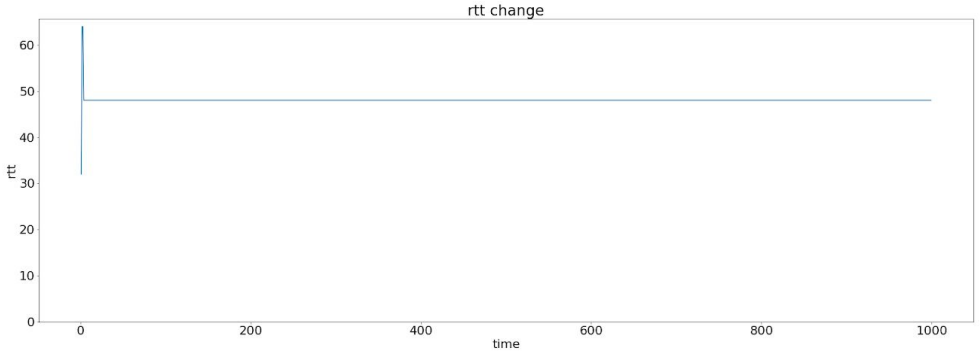
Tahoe_rtt_flow0



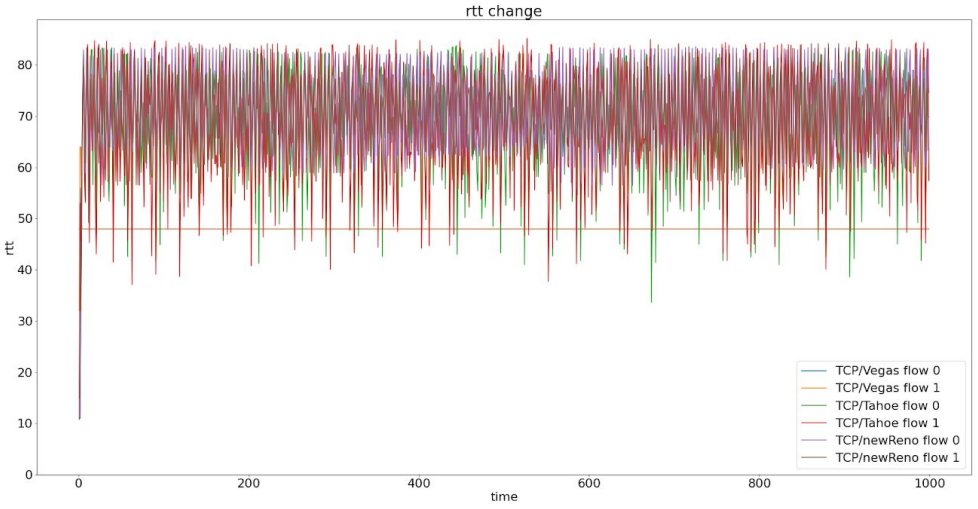
Tahoe_rtt_flow1



Vegas_rtt_flowO



Vegas_rtt_flow1



rtt

DROP RATE

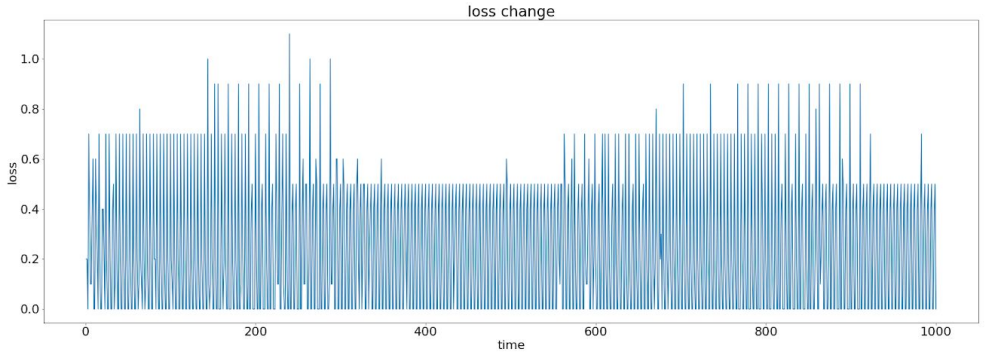
خب در اینجا کمی متفاوت عمل می کنیم و پس از اجرای اسکریپت tcl سعی می کنیم فایل trace درست شده را پارس کنیم و بدین ترتیب عمل کنیم که تعداد بسته های drop شده را در هر ثانیه به دست آوریم. برای مثال در ثانیه دو ، بسته هایی که با -d در فایل وجود دارند را در بازه زمانی 1 تا 2 به دست می آوریم. و به همین ترتیب ادامه می دهیم.
همچنین می توانیم این کار را با شمارش بسته های که فرستاده شده و با تگ - مشخص شده و بسته هایی که ack آن ها دریافت شده انجام دهیم.

پس برای محاسبه نرخ از دست دادن بسته پس از اجرا کد ، از فایل trace شروع به خواندن می کنیم ، بدین ترتیب که در هر ثانیه تفاوت بسته ها فرستاده شده و ack شده را تقسیم به کل بسته های فرستاده شده در آن ثانیه می کنیم.

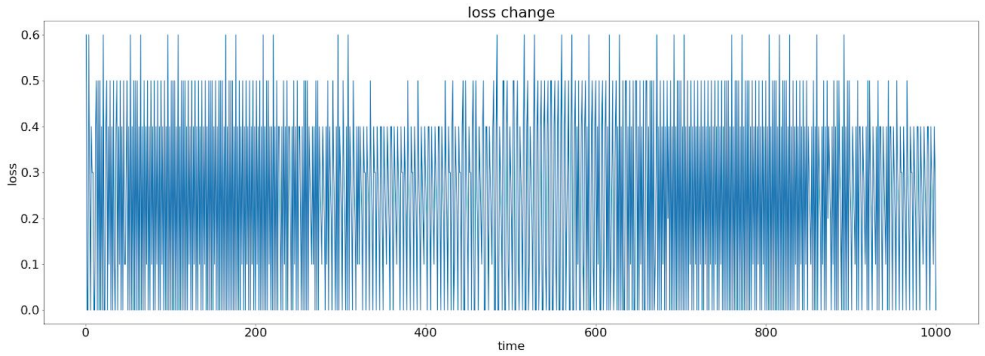
کد آن در زیر آمده است.

```
def getDropRate(tracepath):
    onlyfiles = [f for f in listdir(tracepath) if isfile(join(tracepath, f))]
    num_files = len(onlyfiles)
    for file in onlyfiles:
        drop0 = {}
        drop1 = {}
        count = 0
        for time in range(1001):
            drop0[time] = 0
            drop1[time] = 0
        with open (tracepath + "/" + file) as f:
            for line in f:
                sec = line.split()
                event = sec[0]
                time = float(sec[1])
                from_node = sec[2]
                to_node = sec[3]
                pkttype = sec[4]
                pktsize = int(sec[5])
                flow_id = sec[7]
                src_addr = sec[8]
                dst_addr = sec[9]
                seq_num = sec[10]
                if event == 'd' and flow_id == '0':
                    count+=1
                    drop0[int(float(time))]+=1
                elif event == 'd' and flow_id == '1':
                    count+=1
                    drop1[int(float(time))]+=1
```

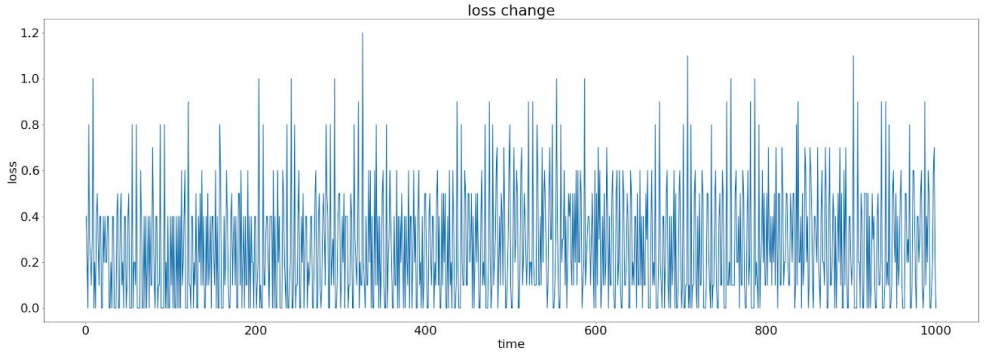
همان طور که در کد دیده می شود ، با خواندن هر فایل تریس در هر ثانیه نگاه می کنیم چند تا loss داشته ایم و این با d مشخص می شود. پس از ذخیره کردن تعداد لاس ها در هر ثانیه در فایل متناظر با فایل تریس ، از آن ها میانگین گرفته و در فایل mean.txt می ریزیم و نهایتا نمودار آن را می کشیم.
در زیر نمودار ها آورده شده اند.



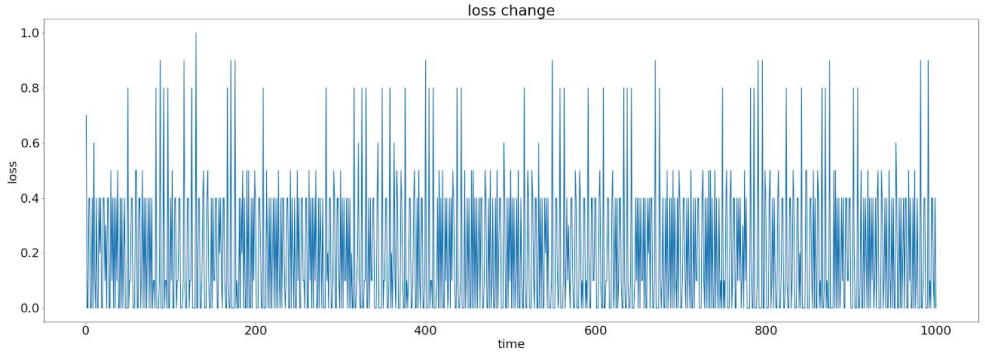
Newreno_droprate_flow0



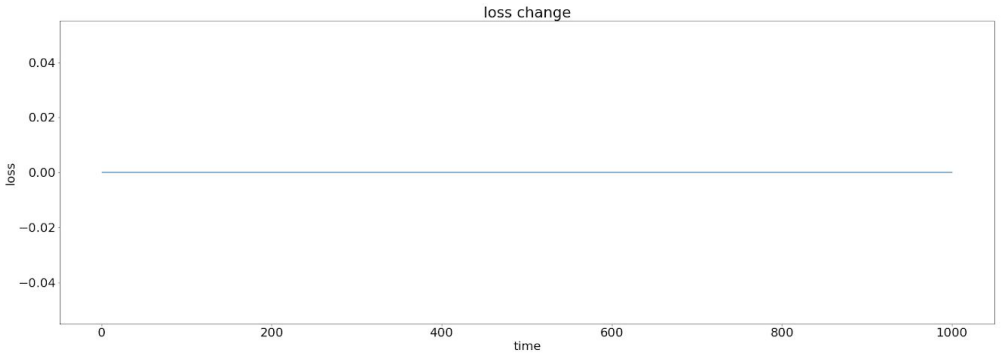
Newreno_droprate_flow1



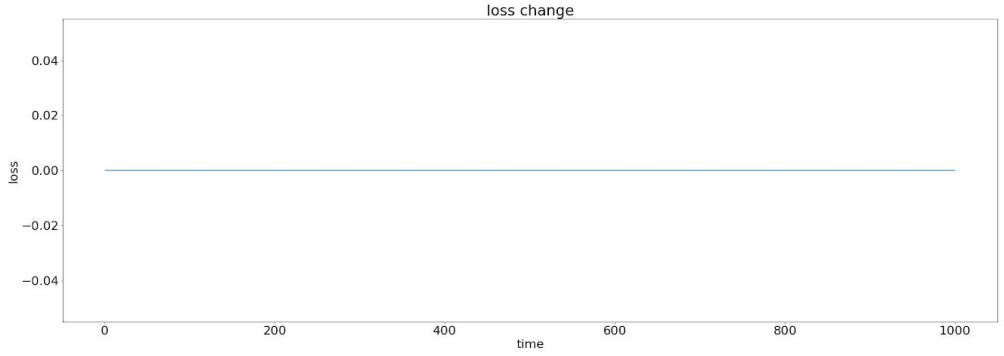
Tahoe_droprate_flow0



Tahoe_droprate_flow1



Vegas_droprate_flow0



Vegas_droprate_flow1

همان طور که از نمودار ها دیده می شود از بین انواع tcp فقط tcp vegas هست که در تمامی حالت مقدار ثابتی را داراست و یا بین دو مقدار نوسان می کند تا مقدار حدودا ثابتی را به خود بگیرد. پس در این نوع tcp ،عملا ازدحام شبکه را در early stage شناسایی می کند و از پکت loss جلوگیری می کند. پس در این نمودار نرخ packet loss برابر صفر است. و cwnd , rtt نیز مقدار ثابتی دارند.

Tahoe به اندازه ی کافی effient نیست به خاطر اینکه هر لحظه پکت لاسی صورت می گیرد به اندازه ی time out صبر می کند و بعد دوباره ارسال می کند و سایز پنجره را به یک کاهش می دهد به خاطر اینکه یک پکت لاس اتفاق افتاده است.

Tahoe is not very much efficient because every time a packet is lost it waits for the timeout and then retransmits the packet. It reduces the size of congestion window to 1 just because of 1 packet loss, this inefficiency cost a lost in high bandwidth delay product links.

Fast Retransmit:

وقتی که فرستنده سه تا duplicate ack دریافت کند آن گاه دوباره پکت ها را باز ارسال می کند بدون آنکه منتظر time out باشد.

Fast Recovery:

در اینجا دیگر سایز پنجره به یک کاهش نمی یابد بلکه به نصف مقدار قبلی می رود. .

TCP New Reno:

TCP New Reno is efficient as compare to Tahoe and Reno. In Reno Fast Recovery exits when three duplicate acknowledgements are received but in New Reno it does not exist until all outstanding data is acknowledged or ends when retransmission timeout occurs. In this way it avoids unnecessary multiple fast retransmit from single window data

TCP Vegas:

TCP Tahoe,Reno and New Reno detects and controls congestion after congestion occurs but still there is a better way to overcome congestion problem i.e TCP Vegas.TCP Vegas detects congestion without causing congestion.

It differs from TCP Reno concerning

- Slow Start
- Packet loss detection
- Detection of available bandwidth

همچنین برای بخش drop rate نیز سعی کردیم بدون توجه به نمودار از طریق کد زیر تعداد بسته های سند شده و ack شده را تقسیم به کل بسته های سند شده کنیم تا عدد drop rate بدست میاد. کد در فایل ژوپیتر اجرا و نتایج نیز قابل رویت است.

In [42]:
```
def getDropRateValue(tracefile, flow):
    totalPacketSent = 0
    totalPacketReceived = 0
    with open (tracefile) as f:
        for line in f:
            sec = line.split()
            event = sec[0]
            time = float(sec[1])
            from_node = sec[2]
            to_node = sec[3]
            pkttype = sec[4]
            pktsize = int(sec[5])
            flow_id = sec[7]
            src_addr = sec[8]
            dst_addr = sec[9]
            seq_num = sec[10]
            if flow_id == flow and flow == '0':
                if event == '-' and from_node == '0':
                    totalPacketSent += 1
                if event == 'r' and to_node == '0' and pkttype == 'ack':
                    totalPacketReceived += 1
            elif flow_id == flow and flow == '1':
                if event == '-' and from_node == '1':
                    totalPacketSent += 1
                if event == 'r' and to_node == '1' and pkttype == 'ack':
                    totalPacketReceived += 1
    if totalPacketSent == 0:
        return 0
    else:
        return float (totalPacketSent - totalPacketReceived) / totalPacketSent * 100
```

In [43]:
```
for tcp in tcp_agents:
    droprate0 = getDropRateValue(tcp + "/droprate/tracefiles/1.tr", '0')
    droprate1 = getDropRateValue(tcp + "/droprate/tracefiles/1.tr", '1')
    print("drop rate => " + tcp + "flow 0 => " , droprate0)
    print("drop rate => " + tcp + "flow 1 => " , droprate1)
```

```
drop rate => Newrenoflow 0 =>  4.076433121019108
drop rate => Newrenoflow 1 =>  4.110027186950264
drop rate => Tahoeflow 0 =>  3.479236812570146
drop rate => Tahoeflow 1 =>  3.4593724859211585
drop rate => Vegasflow 0 =>  0.04797888928871297
drop rate => Vegasflow 1 =>  0.04798464491362764
```