بسم الرحمن الرحیم

گزارش کار پروژه دوم شبکه‌های کامپیوتری

بهار ۹۹

بخش اول:

توضیح کد tcl

ابتدا یک شیئ از کلاس Simulator می‌سازیم که این شیئ ابزار اصلی شبیه‌سازی در این پروژه است:

```
#Create a simulator object
set ns [new Simulator]
```

حال دو رنگ اول و دوم شبیه‌سازی را تعیین می‌کنیم:

```
#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
```

حال اسم فایل شبیه‌سازی با پسوند nam را از ورودی می‌خوانیم و یک فایل با آن اسم می‌سازیم و اشاره‌گر به آن فایل را در متغیر namfile قرار می‌دهیم.

```
#Open the NAM trace file
set namName [lindex $argv 1]
set namfile [open $namName w]
```

حال تعیین میکنیم که trace های مربوط به نمایش انیمیشنی توپولوژی شبکه و حرکت بسته‌ها در فایل namfile نوشته شوند.

```
$ns namtrace-all $namfile
```

حال اسم فایل ذخیره کننده مقدار متغیرهای مطلوب در زمان‌های مشخص را از ورودی می‌خوانیم و یک فایل با این اسم می‌سازیم و آن را در tracefile قرار می‌دهیم. در این فایل trace های تولید شده توسط خودمان را خواهیم نوشت.

```
set traceName [lindex $argv 2]
set tracefile [open $traceName w]
```

حال رشته‌ی auto_ را به traceName اضافه کرده و یک فایل با آن اسم می‌سازیم و اشاره‌گر به آن فایل را در متغیر tracefile_auto قرار می‌دهیم.

```
set traceName_auto [append traceName "_auto"]
set tracefile_auto [open $traceName_auto w]
```

حال با استفاده از دستور زیر تعیین میکنیم که trace های تولید شده توسط خود شبیه‌ساز در فایل tracefile_auto نوشته شوند.

```
$ns trace-all $tracefile_auto
```

یک تابع finish درست میکنیم که در انتهای شبیه‌سازی اجرا شود. در این تابع فایل ها بسته می‌شوند.

```
#Define a 'finish' procedure
proc finish {} {
    global ns namfile tracefile
    $ns flush-trace
    #Close the NAM trace file
    close $namfile
    close $tracefile

    exit 0
}
```

تابع rand که خودمان نوشته‌ایم دو عدد min و max را به عنوان ورودی می‌گیرد و یک عدد رندوم بین آن دو را می‌گرداند:

```
# generate random integer number in the range [0,max]
proc rand {min max} {
   return [expr {$min + int(rand()*(($max-$min)+1))}]
}
```

بیشترین و کمترین تاخیر ممکن لینک‌ها را در دو متغیر زیر ذخیره می‌کنیم:

```
# set min and max random delay
set min_delay 5
set max_delay 25
```

۶ گره موجود در توپولوژی را میسازیم و آن‌ها را در متغیرهای n1 تا n6 نگه می‌داریم:

```
#Create 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]
```

لینک‌ها را به صورت زیر می‌سازیم که ابتدا نوع آن‌ها که به صورت duplex هستند مشخص شده و سپس گره‌های دو سر و سپس ظرفیت و تأخیر انتشار و سپس سیاست برخورد با بسته‌ها به هنگام پر شدن صف روتر مشخص شده که در اینجا DropTail به معنی drop کردن بسته‌ها به هنگام پر شدن صف روتر است.

```
#Create links between the nodes
$ns duplex-link $n1 $n3 100Mb 5ms DropTail
$ns duplex-link $n2 $n3 100Mb [rand $min_delay $max_delay]ms DropTail
$ns duplex-link $n3 $n4 0.1Mb 1ms DropTail
$ns duplex-link $n4 $n5 100Mb 5ms DropTail
$ns duplex-link $n4 $n6 100Mb [rand $min_delay $max_delay]ms DropTail
```

سایر صف روترها را 10 میگذاریم:

```
#Set Queue Size of links
$ns queue-limit $n3 $n4 10
$ns queue-limit $n4 $n5 10
$ns queue-limit $n4 $n6 10
```

برای نمایش گره‌ها به دقیقا به همان شکل داده شده در توپولوژی، محل گره‌ها را به صورت جهت‌های جغرافیایی مشخص می‌کنیم:

```
#Give node position (for NAM)
$ns duplex-link-op $n1 $n3 orient right-down
$ns duplex-link-op $n2 $n3 orient right-up
$ns duplex-link-op $n3 $n4 orient right
$ns duplex-link-op $n4 $n5 orient right-up
$ns duplex-link-op $n4 $n6 orient right-down

#Monitor the queue for link (n2-n3). (for NAM)
```

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

با استفاده از عدد داده شده به برنامه به هنگام اجرا، مشخص میکنیم که نوع کانکشن TCP کدام یک از ۳ نوع است:

```
#Setup a TCP connection:
set tcpNum [lindex $argv 0]
set tcpConType Agent/TCP/Newreno
if {$tcpNum == 1} {
   set tcpConType Agent/TCP/Newreno
} elseif {$tcpNum == 2} {
   set tcpConType Agent/TCP
} elseif {$tcpNum == 3} {
   set tcpConType Agent/TCP/Vegas
}
```

مبدأ کانکشن TCP اول را میسازیم و آن را به گره شماره ۱ وصل میکنیم:

```
#create tcp source1
set tcp_src1 [new $tcpConType]
$tcp_src1 set ttl_ 64
$tcp_src1 set fid_ 1
$ns attach-agent $n1 $tcp_src1
```

مقصد کانکشن TCP اول را میسازیم و آن را به گره شماره ۵ وصل میکنیم:

```
#create tcp sink1
set tcp_sink1 [new Agent/TCPSink]
```

```
$ns attach-agent $n5 $tcp_sink1
$tcp_sink1 set fid_ 1
```

مبدا و مقصد کانکشن TCP اول را به هم متصل میکنیم:

```
#connect source to sink
$ns connect $tcp_src1 $tcp_sink1
```

همین مراحل را برای ساختن کانکشن TCP دوم نیز انجام می‌دهیم:

```
#create tcp source2
set tcp_src2 [new $tcpConType]
$tcp_src2 set ttl_ 64
$tcp_src2 set fid_ 2
$ns attach-agent $n2 $tcp_src2

#create tcp sink2
set tcp_sink2 [new Agent/TCPSink]
$ns attach-agent $n6 $tcp_sink2
$tcp_sink2 set fid_ 2

#connect source to sink
$ns connect $tcp_src2 $tcp_sink2
```

از FTP برای ایجاد ترافیک در شبکه استفاده می‌کنیم. دو ترافیک FTP میسازیم و یکی را به منبع TCP اول و دیگری را به منبع TCP دوم متصل میکنیم:

```
# Setup an FTP over both TCP connections
```

```
set ftp_traffic1 [new Application/FTP]
set ftp_traffic2 [new Application/FTP]
$ftp_traffic1 set type_ FTP
$ftp_traffic2 set type_ FTP
$ftp_traffic1 attach-agent $tcp_src1
$ftp_traffic2 attach-agent $tcp_src2
```

تابع زیر مقدار متغیر های rtt , cwnd , goodput را برای هر دو جریان و در فاصله‌های زمانی 0.1 ثانیه، در فایل ورودی می‌نویسد:

```
proc traceVars {outfile} {
global ns tcp_src1 tcp_src2 tcp_sink1 tcp_sink2
set now [$ns now]                       ;# Read current time
set nbytes1 [$tcp_sink1 set bytes_]     ;# Read number of bytes
set nbytes2 [$tcp_sink2 set bytes_]     ;# Read number of bytes

$tcp_sink1 set bytes_ 0                 ;# Reset for next epoch
$tcp_sink2 set bytes_ 0                 ;# Reset for next epoch



set cwnd1 [$tcp_src1 set cwnd_]
set cwnd2 [$tcp_src2 set cwnd_]

set rtt1 [$tcp_src1 set rtt_]
set rtt2 [$tcp_src2 set rtt_]

set time_incr 1
```

```
### Prints "TIME throughput" in Mb/sec units to output file
set goodput1 [expr ($nbytes1 * 8.0 / 1000000) / $time_incr]
set goodput2 [expr ($nbytes2 * 8.0 / 1000000) / $time_incr]
puts  $outfile  "$now  $goodput1  $goodput2  $cwnd1  $cwnd2  $rtt1  $rtt2"

### Schedule yourself:
$ns at [expr $now+$time_incr] "traceVars  $outfile"
}
```

تابع بالا را با ورودی tracefile صدا میکنیم:

```
traceVars $tracefile
```

هر دو ترافیک FTP را در زمان 0 آغاز می‌کنیم و در زمان 1000 خاتمه می‌دهیم:

```
$ns at 0.0 "$ftp_traffic1 start"
$ns at 0.0 "$ftp_traffic2 start"
$ns at 1000.0 "$ftp_traffic1 stop"
$ns at 1000.0 "$ftp_traffic2 stop"
```

در زمان 1000 تابع finish را صدا میزنیم تا شبیه‌سازی را خاتمه دهد:

```
#Call the finish procedure after 5 seconds of simulation time
$ns at 1000.0 "finish"
```

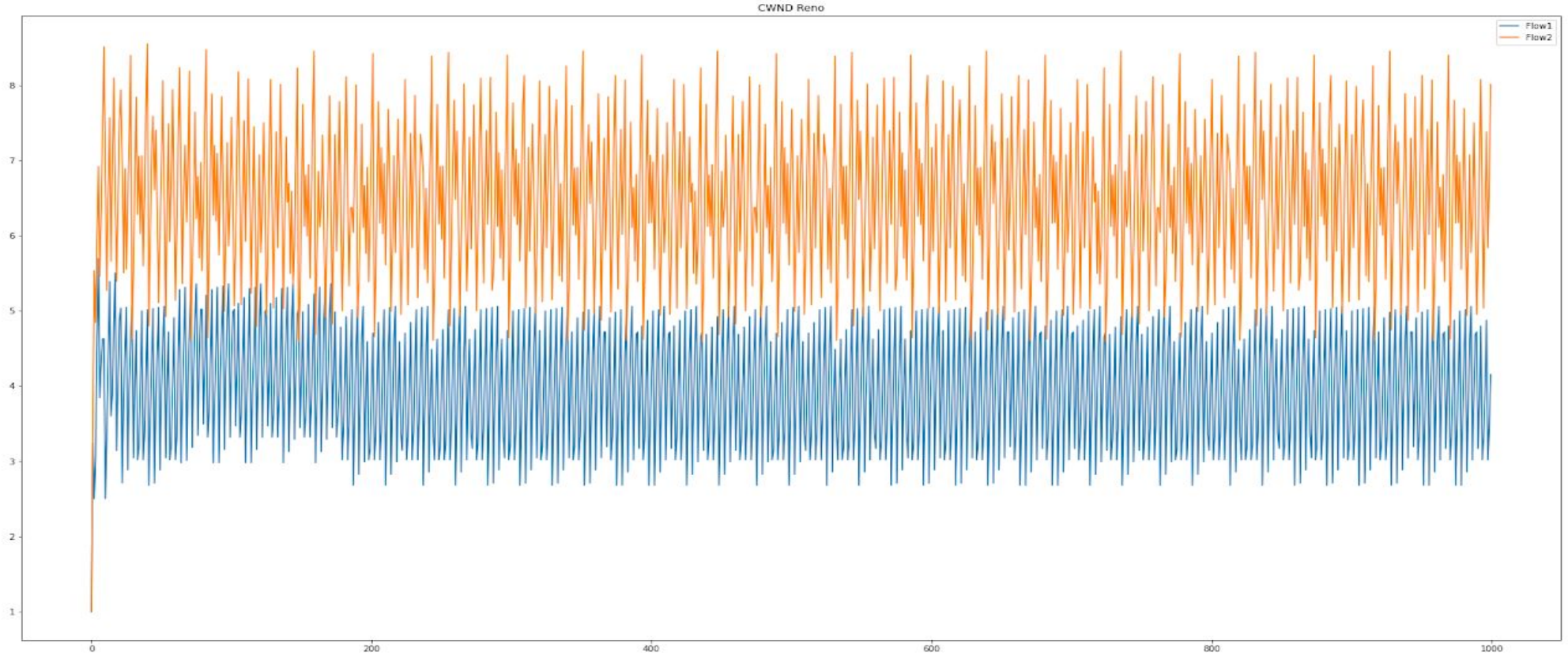
با استفاده از دستور زیر شبیه‌سازی را آغاز می‌کنیم:

```
#Run the simulation
$ns ru
```

بخش دوم:

نمودار های خروجی

نمودار پنجره ازدحام برای TCP-Reno:

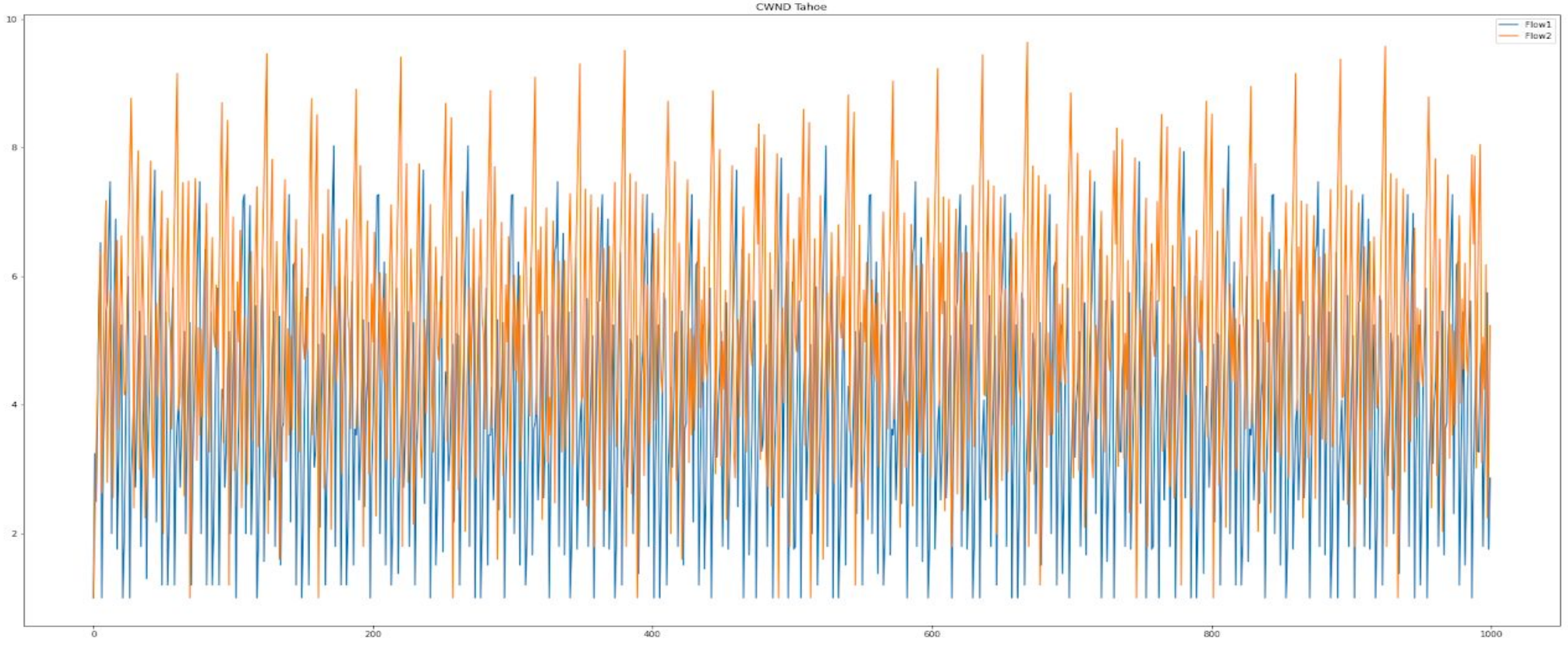


تحلیل:

slow start: در حالت کلی در پروتکل انتقال TCP در ابتدای فرستادن نیاز داریم مقدار گنجایش و توان شبکه برای پیام های ارسالی خود را بدانیم پس برای این که بتوان سریع این مقدار را یافت از باینری سرچ استفاده می‌کنیم و با دریافت هر ACK سایز پنجره فرستنده یک واحد افزایش می‌یابد و این بدین معناست که در هر زمان از RTT عملا سایز پنجره ۲ برابر می‌شود، حال این عمل تکرار می‌شود تا زمانی که اولی loss رخ دهد در آن زمان سایز این پنجره نصف می‌شود و باز همین روند ادامه دارد.

در الگوریتم کنترل ترافیک Reno در انتقال های TCP، در فاز congestion-avoidance با رخداد هر کامل شدن پنجره، یک واحد به سایز آن اضافه می‌شود که عملا بدان معناست که سایز پنجره طبق فرمول cwnd = cwnd + 1/cwnd افزایش می‌یابد و بدین ترتیب فرم افزایش سایز پنجره در الگوریتم Reno در مقایسه با Tahoe سرعت کمتری دارد و تقریبا همیشه به مقدار ثابتی در حالت ماکزیمم می‌رسد و عملا مقادیر اکسترمم ها در روش Reno تقریبا یکسان اند، البته این یکسانی را در نمودار جریان دوم Reno میانگینی از تاخیرهای مختلف را دارد نمی‌توان دید که دلیل اصلی آن همین میانگین گیری و در نتیجه به هم خوردن این اکسترمم ها است. در این در روش همچنین با رخداد هر packet loss سایز این پنجره نصف می‌شود که این می‌شود شکل اره‌ای نمودار در Reno به راحتی مشاهده شود.

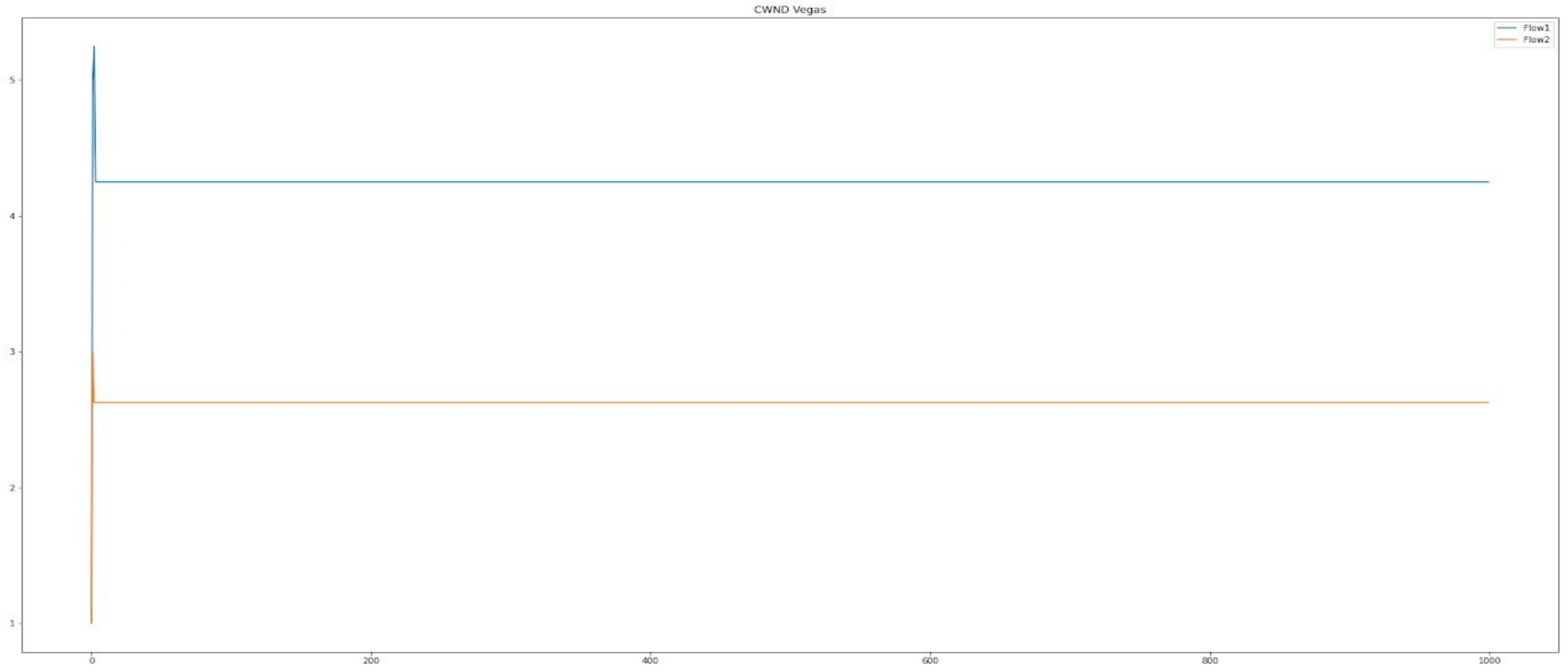
نمودار پنجره ازدحام برای TCP-Tahoe:



تحلیل:

در این روش به ازای هر ack دریافت شده cwnd یک واحد افزایش می‌یابد و به ازای هر packet loss، طول پنجره ۱/۲ می‌شود در نتیجه نمودار تغییرات cwnd همانطور که در نمودار نیز مشاهده می‌شود به شکل دندانه-اره‌ای خواهد بود.

نمودار پنجره ازدحام برای TCP-Vegas:

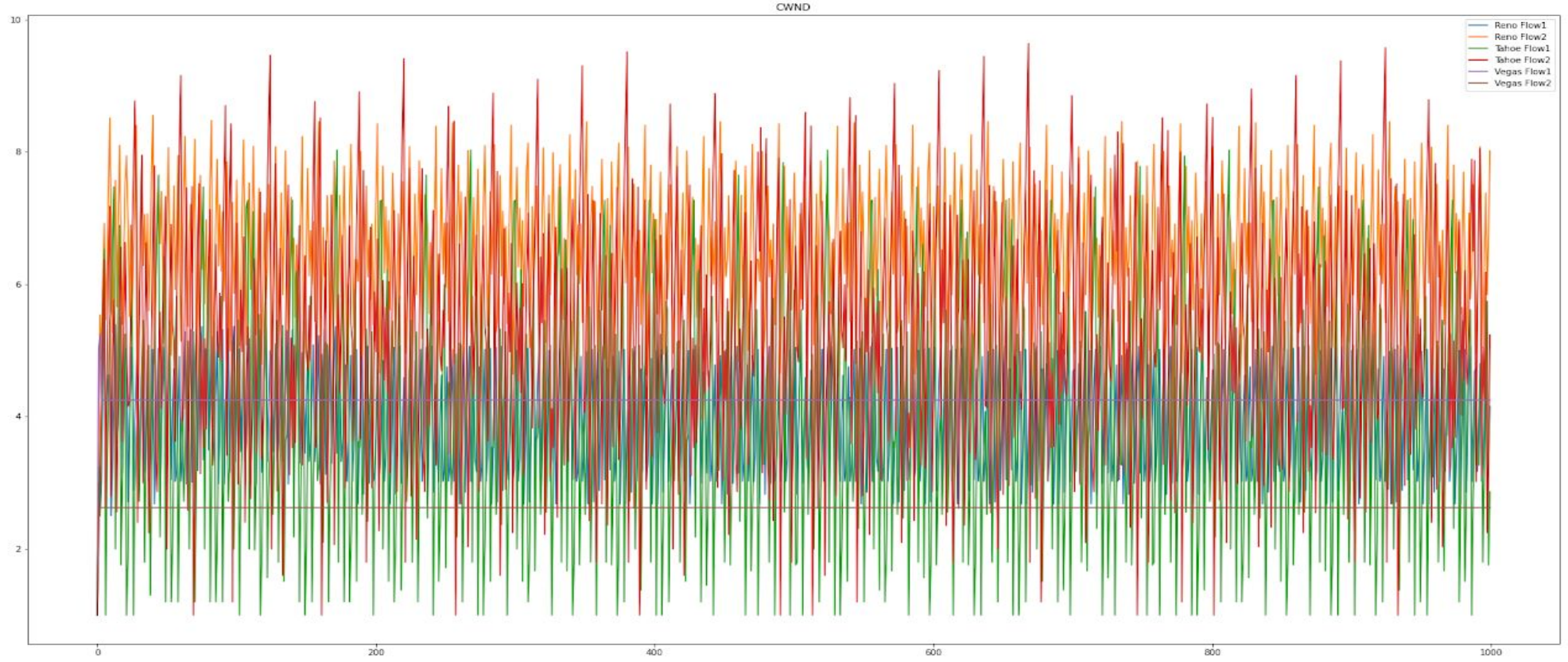


تحلیل:

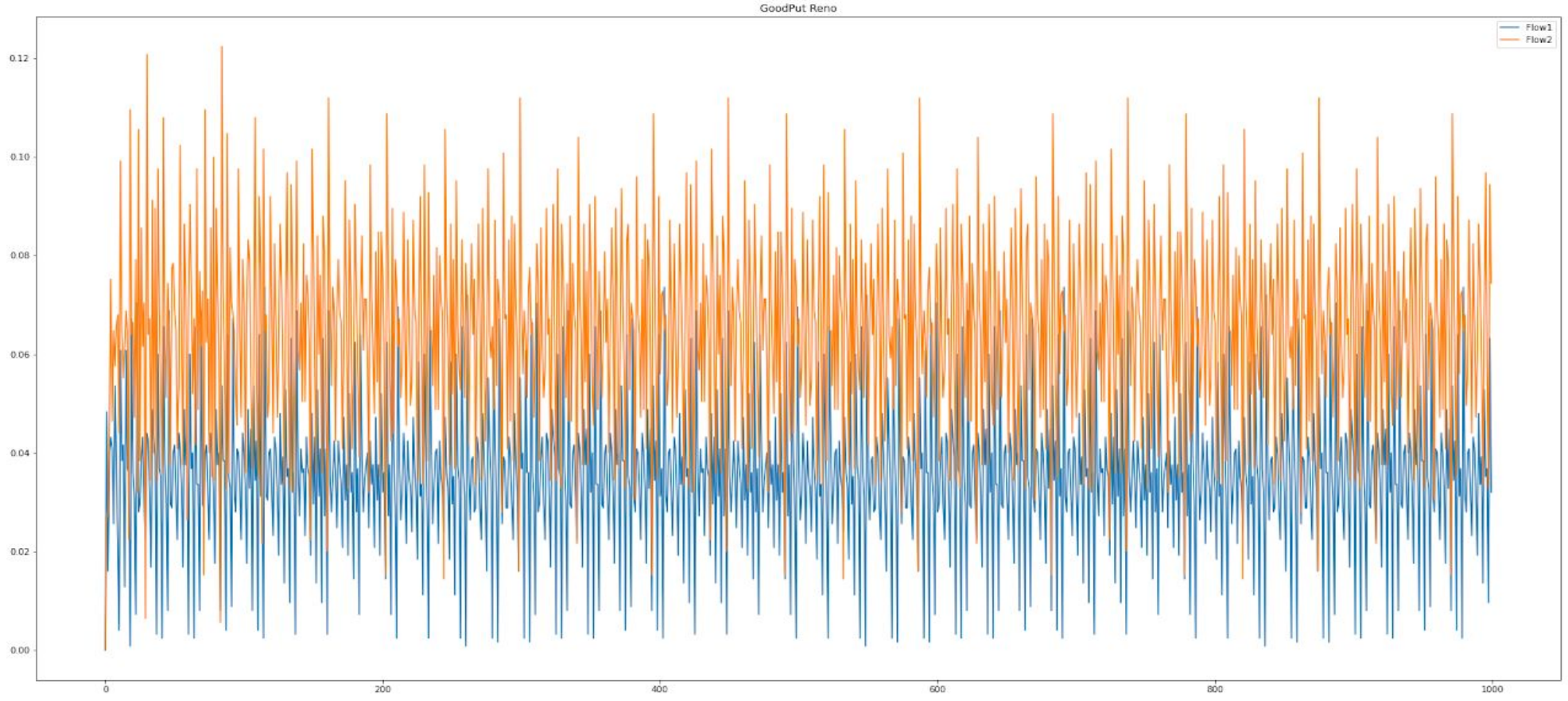
برخلاف TCP Reno و TCP Tahoe که روش‌های کنترلی Loss-based هستند، TCP Vegas یک روش کنترلی Delay-based است. ایده‌ی اصلی این روش این است که با استفاده از تنظیم نرخ ارسال بسته با توجه به تأخیر RTT بسته‌ها، از وقوع packet-loss جلوگیری می‌کند. بنابراین به دلیل عدم وقوع از دست دادن بسته در این روش، طول پنجره(cwnd) در طول زمان ثابت می‌ماند. همانطور که در نمودارصفحه قبل مشاهده می‌شود طول پنجره برای هر دو جریان در طول زمان ثابت است.

در این روش یک RTT مورد انتظار محاسبه می‌شود و در RTT های واقعی با RTT مورد انتظار مقایسه میشود و اختلاف این دو در یک بازه‌ی مشخص نگه‌داری میشود تا از وقوع ازدحام جلوگیری شود.

نمودار کلی پنجره ازدحام:



نمودار good put برای TCP-Reno:

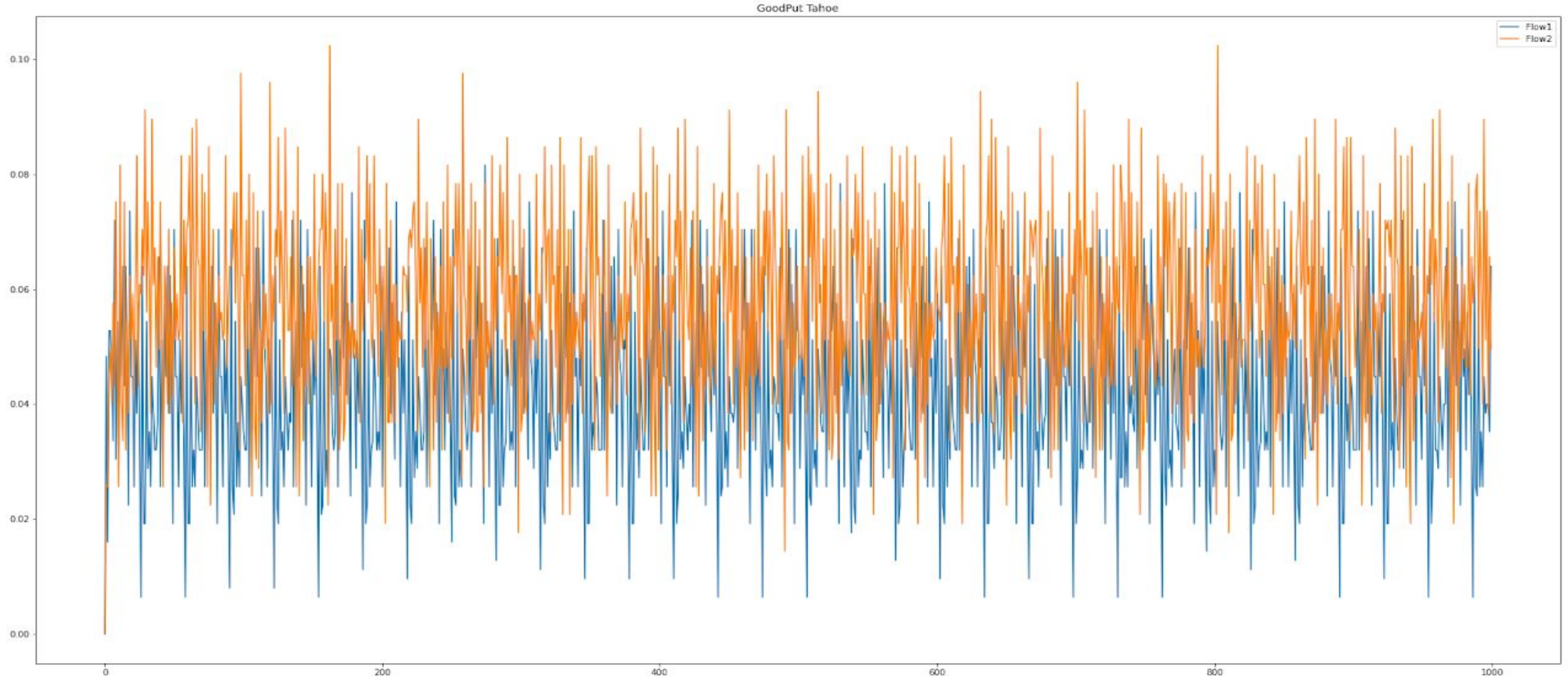


تحلیل:

تعریف GoodPut: مقدار این پارامتر به معنای تعداد بیت‌های مفید که شبکه در مسیری خاص در واحد زمان منتقل کرده است میباشد که بیت‌های سربار پروتکل جزو این بیت‌های مفید به حساب نمی‌آیند همچنین بسته‌های ناشی از ارسال دوباره بسته به دلیل packet loss یا درخواست ارسال دوباره،که به دلیل خطا در بیت‌های ارسالی قبلی رخ داده است نیز جزو این مقادیر مفید نیستند. پس ما در این شبیه‌سازی برای محاسبه این مقدار در اصل تعداد مگابیت‌های دریافتی در واحد زمان توسط هر یک از گره‌های مقصد دو جریان را ذخیره و پلات کرده‌ایم.

با توجه به تغییرات طول cwnd تعداد بسته‌های ارسال شده در بازه‌های زمانی نیز تغییر می‌کنند و در نتیجه تعداد بایت های گرفته شده در مقصد نیز تغییر خواهد کرد در نتیجه همانطور که در نمودار صفحه قبل مشاهده می‌شود شکل نمودار goodput برای Reno همانند نمودار cwnd به صورت sowtooth (اره ای) با ارتفاع peak های تقریبا یکسان خواهد بود.

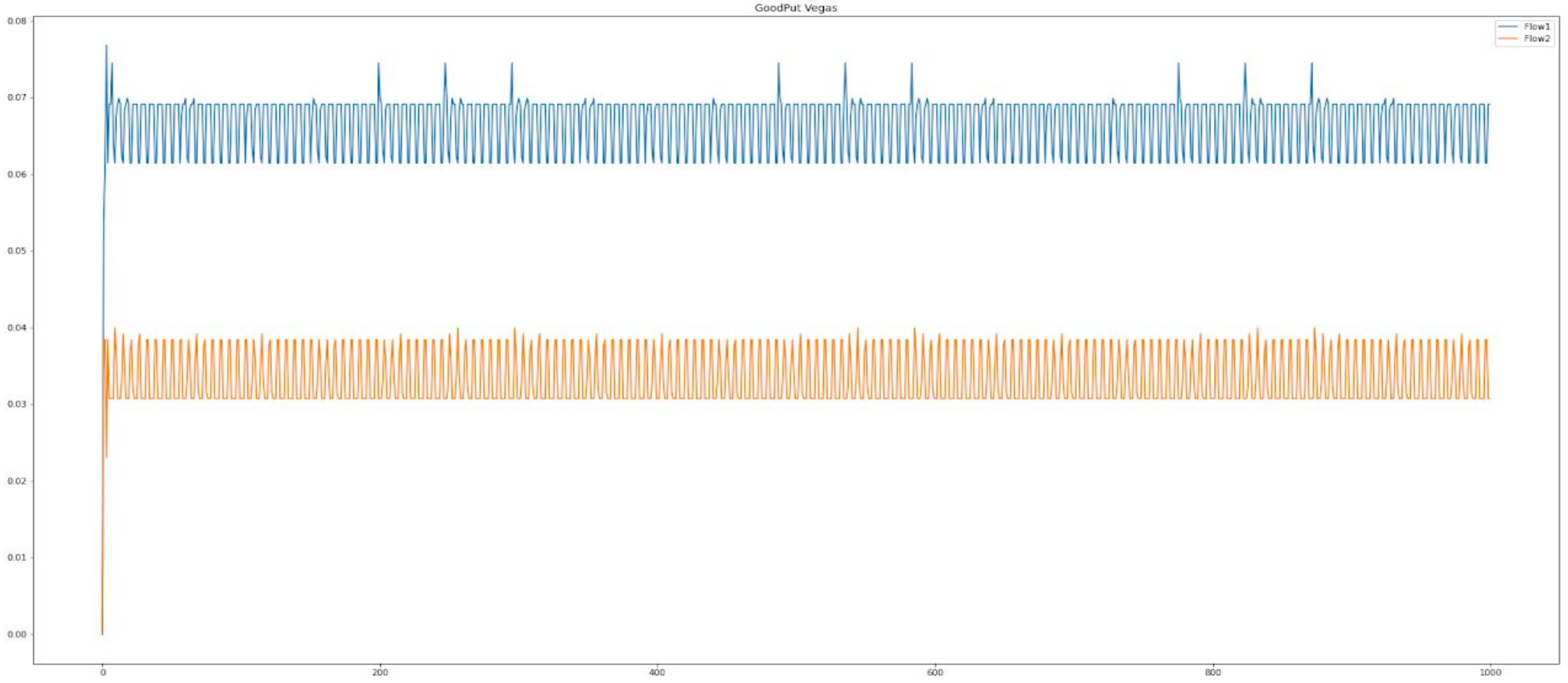
نمودار good put برای TCP-Tahoe:



تحلیل:

با توجه به تغییرات طول cwnd تعداد بسته‌های ارسال شده در بازه‌های زمانی نیز تغییر می‌کنند و در نتیجه تعداد بایت های گرفته شده در مقصد نیز تغییر خواهد کرد در نتیجه همانطور که در نمودارصفحه قبل مشاهده می‌شود شکل نمودار goodput برای TCP Tahoe همانند نمودار cwnd به صورت sowtooth(دندان اره ای) با ارتفاع peak های نامنظم خواهد بود.

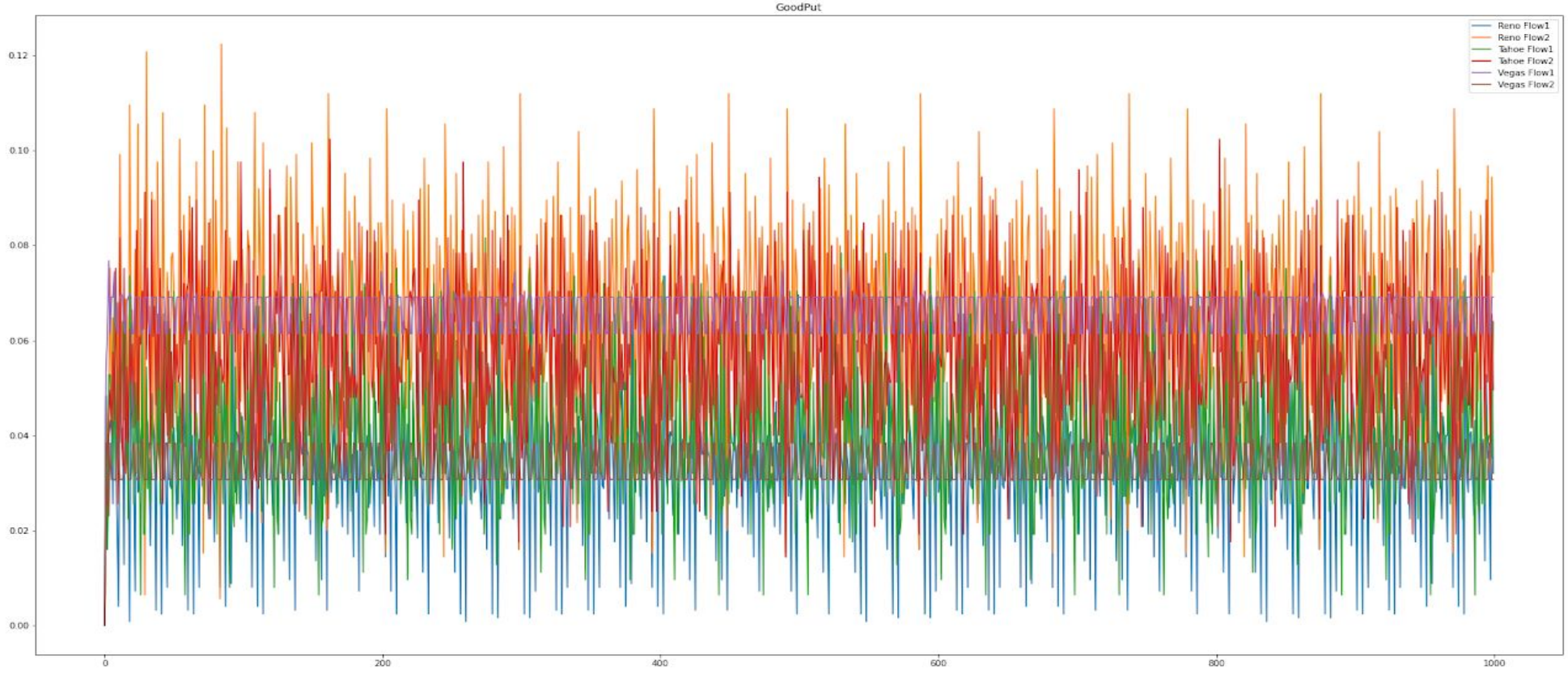
نمودار good put برای TCP-Vegas:



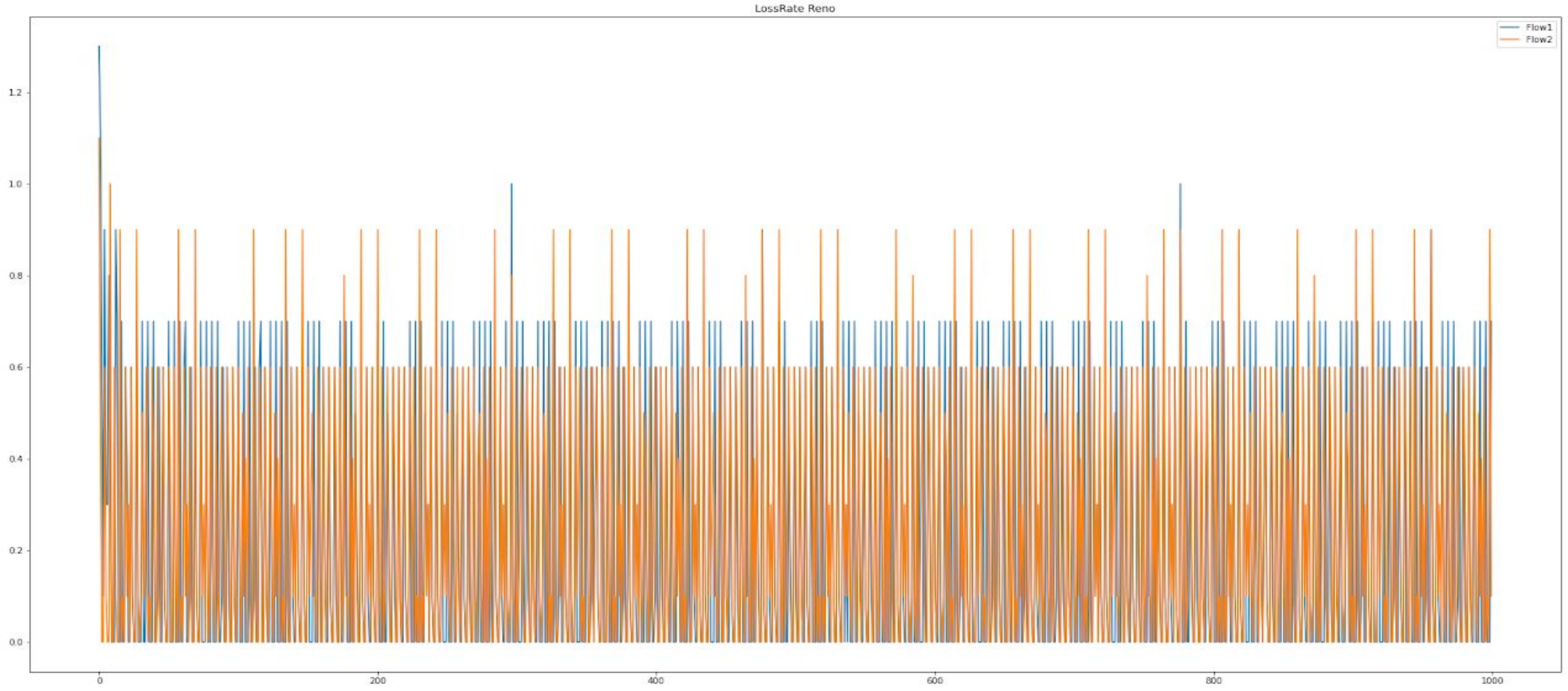
تحلیل و مقایسه:

در مقایسه سه نمودار قبلی می‌بینیم که به طور میانگین در دو پروتکل کنترل ازدحام اول یعنی Tahoe , Reno گذردهی مفید جریان دوم که تاخیر بیشتری در لینک‌های خود دارد بیشتر است در حالی که در پروتکل سوم یعنی Vegas جریان اول که تاخیر کمتری در لینک‌های خود دارد گذردهی مفید بیشتری را تجربه می‌کند همچنین در این پروتکل با نوسانات بسیار کمتری در اندازه این گذردهی مفید برمی‌خوریم که با توجه با ثابت ماندن اندازه RTT , صفر ماندن pack loss ها و همچنین ثابت ماندن اندازه پنجره تعداد بسته‌های ثابتی همواره در واحد زمان منتقل می‌شوند و این مورد نیز به خوبی مشاهده و توجیه می‌شود.

نمودار کلی good put:



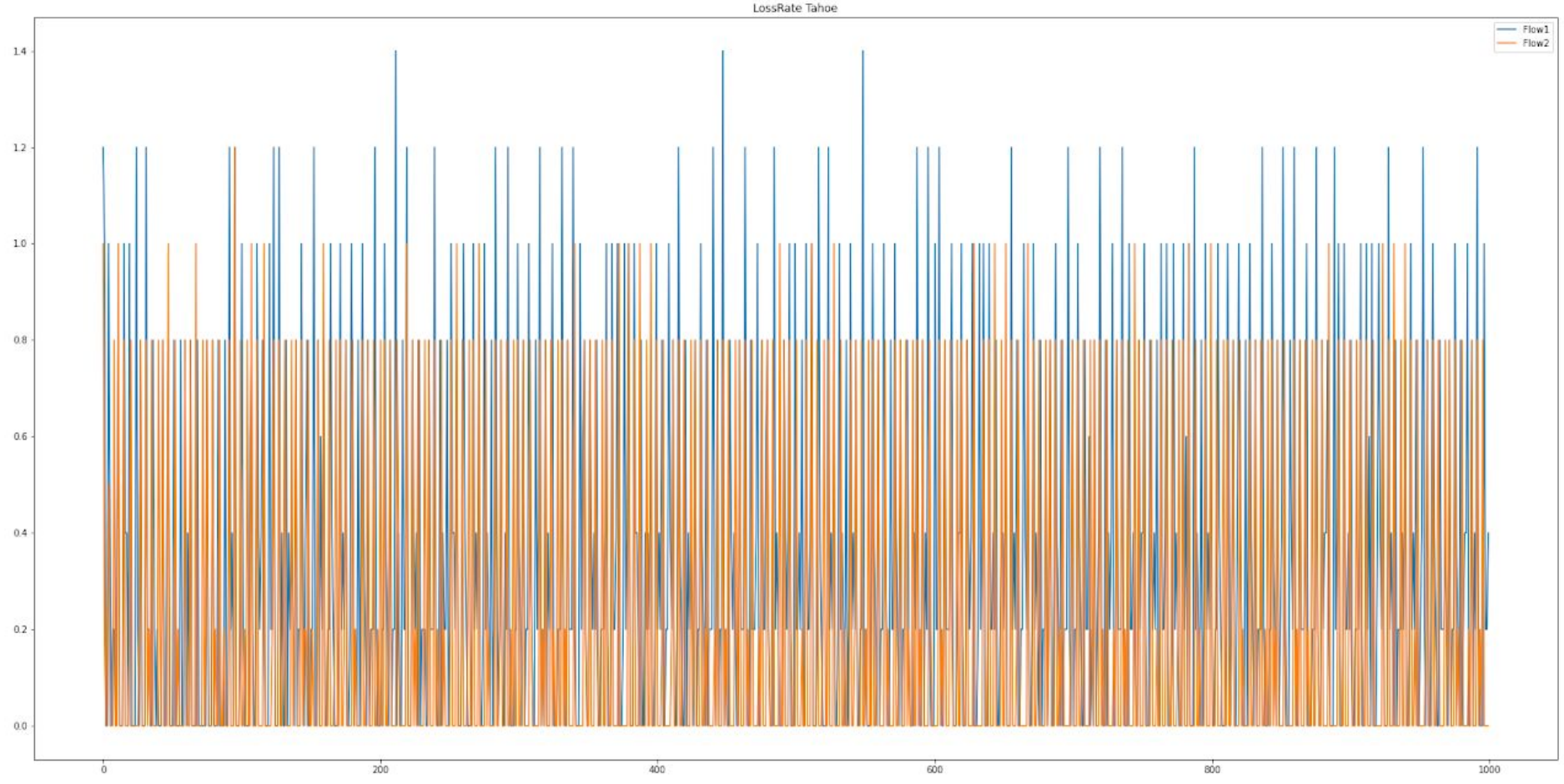
نمودار نرخ از دست‌رفتن بسته برای TCP-Reno:



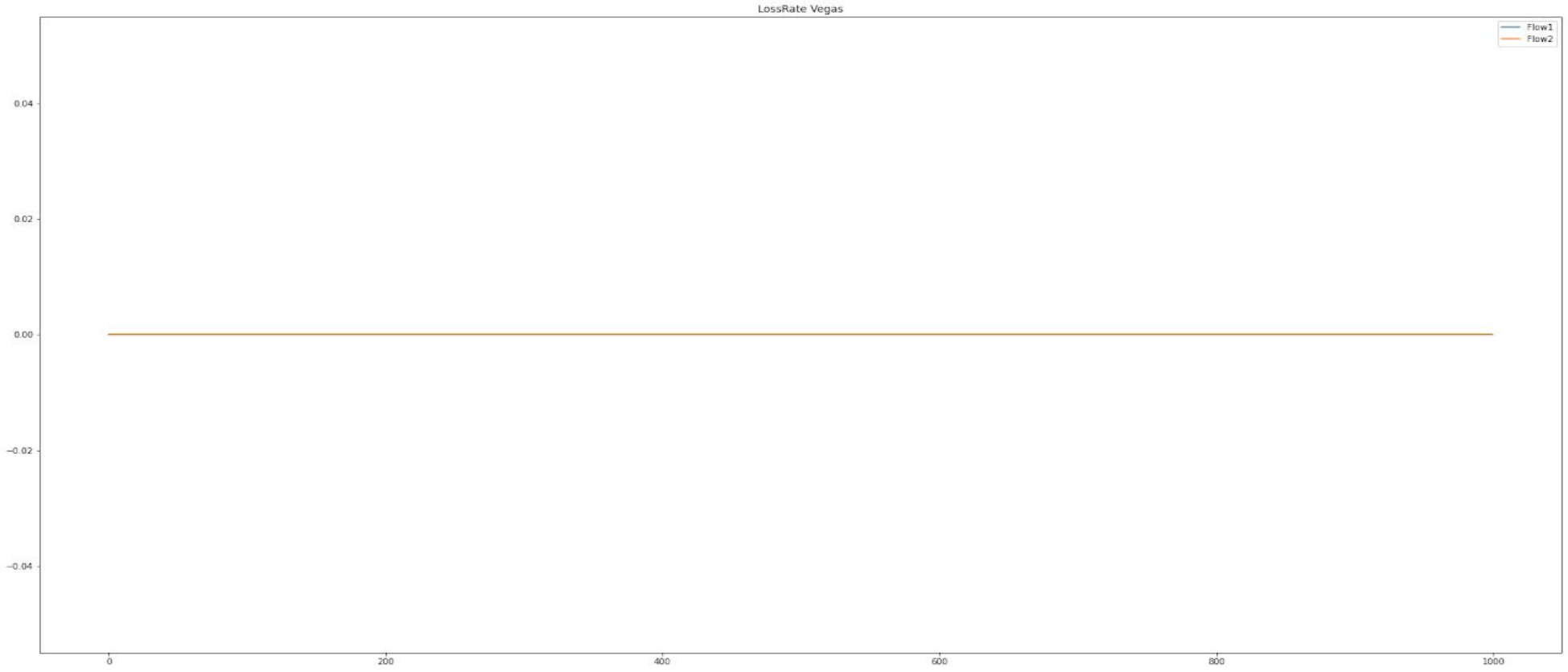
تحلیل:

در تغییراتی که در نمودار های مربوط به مقادیر loss در پروتکل‌های TCP-Reno و TCP-Tahoe است، می‌بینیم که به طور معمول در پروتکل Reno مقادیر کمتری مشاهده می‌شود که این مورد با توجه به دامنه نوسانات کوچکتر سایز پنجره که در Reno مشاهده کردیم و فرمول افزایش اندازه آن توجیه پذیر است و دلیل آن همین است که Reno کمتر باعث ازدحام شبکه و در نتیجه رخداد loss می‌شود.

نمودار نرخ از دست‌رفتن بسته برای TCP-Tahoe:



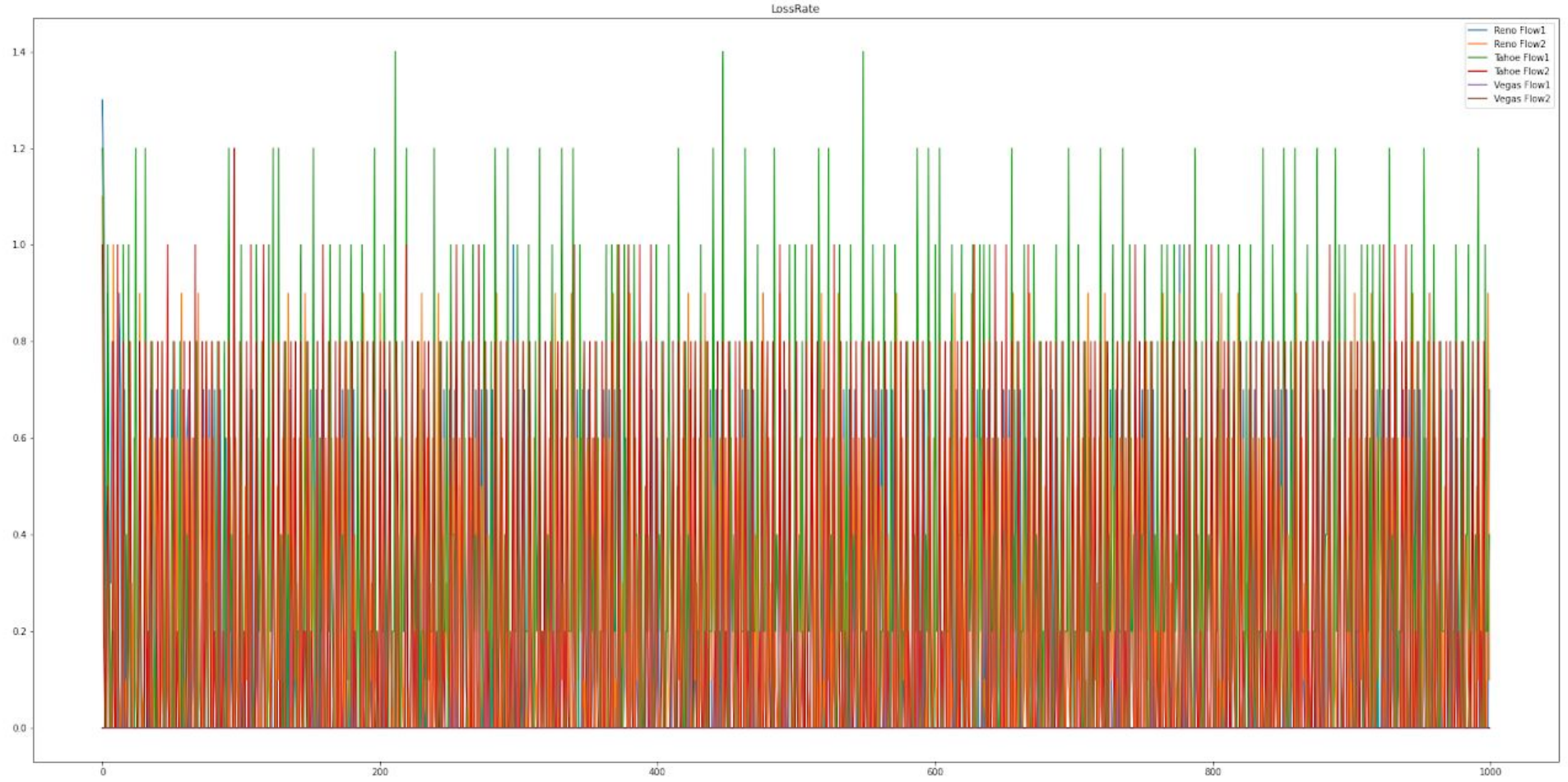
نمودار نرخ از دست‌رفتن بسته برای TCP-Vegas:



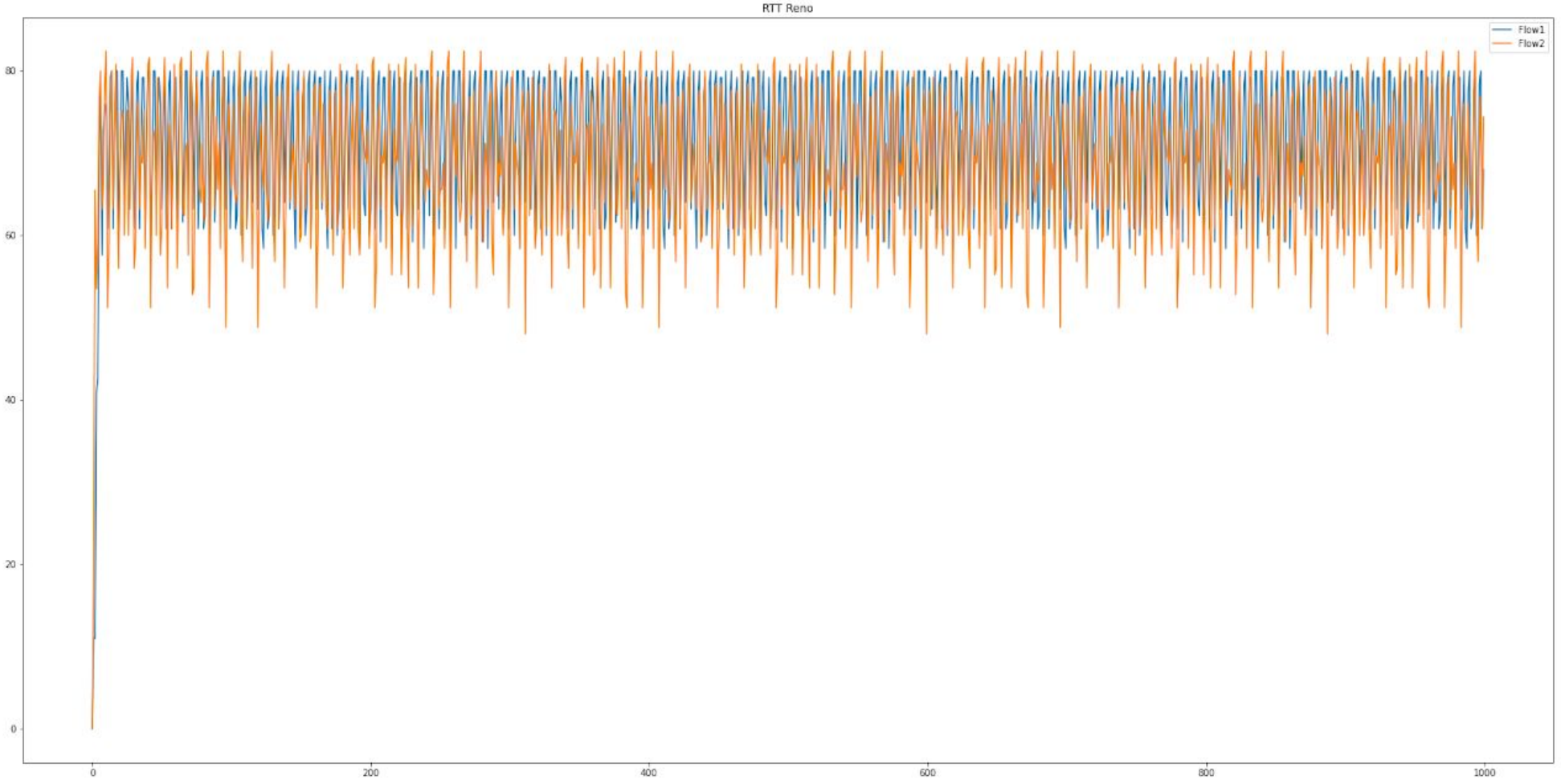
تحلیل:

با توجه با اینکه ایده‌ی اصلی این روش تشخیص ازدحام در روترها قبل از وقوع Loss است، همانطور که از نمودارهای صفحه قبل نیز پیداست packet loss تقریبا اتفاق نمی‌افتد و در نتیجه تعداد packet loss ها در طول زمان همواره صفر خواهد بود.

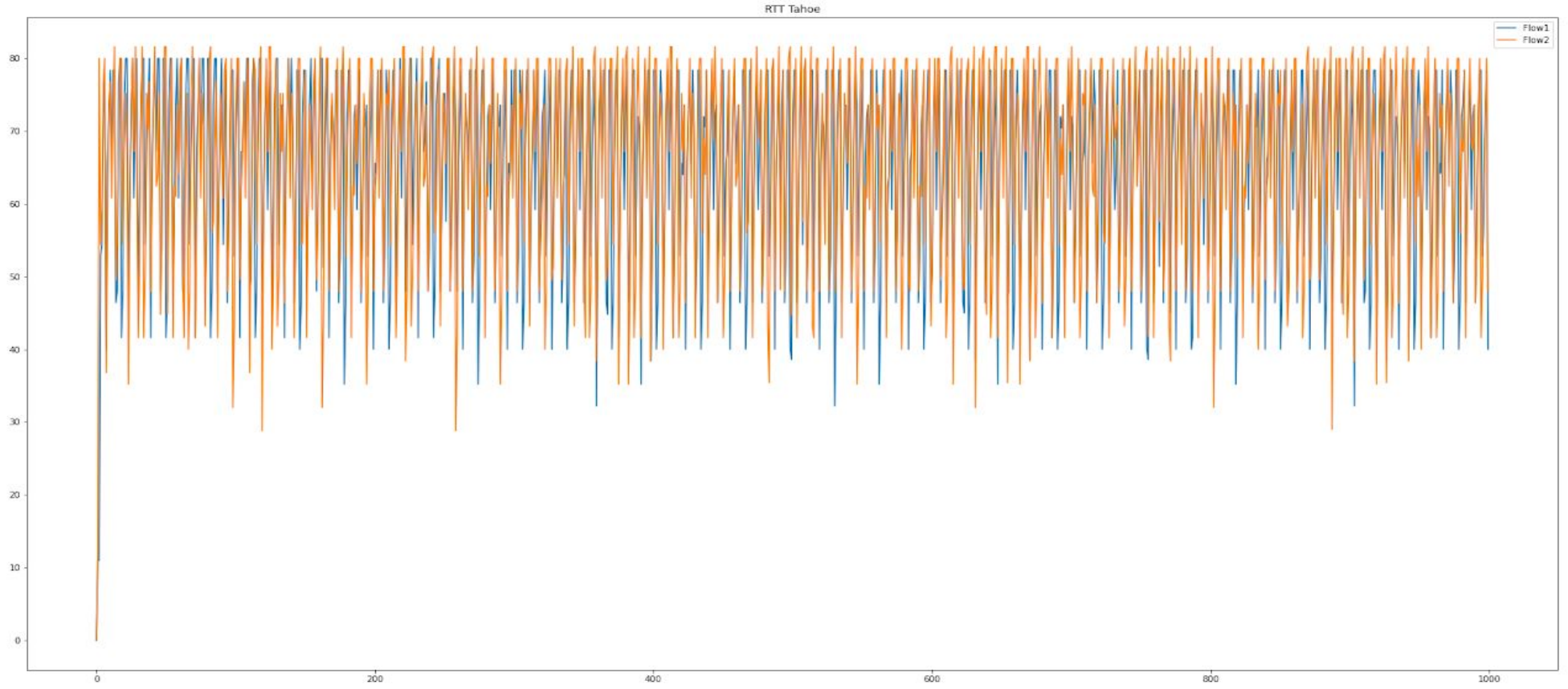
نمودار کلی نرخ از دست‌رفتن بسته:



نمودار نرخ RTT برای TCP-Reno:



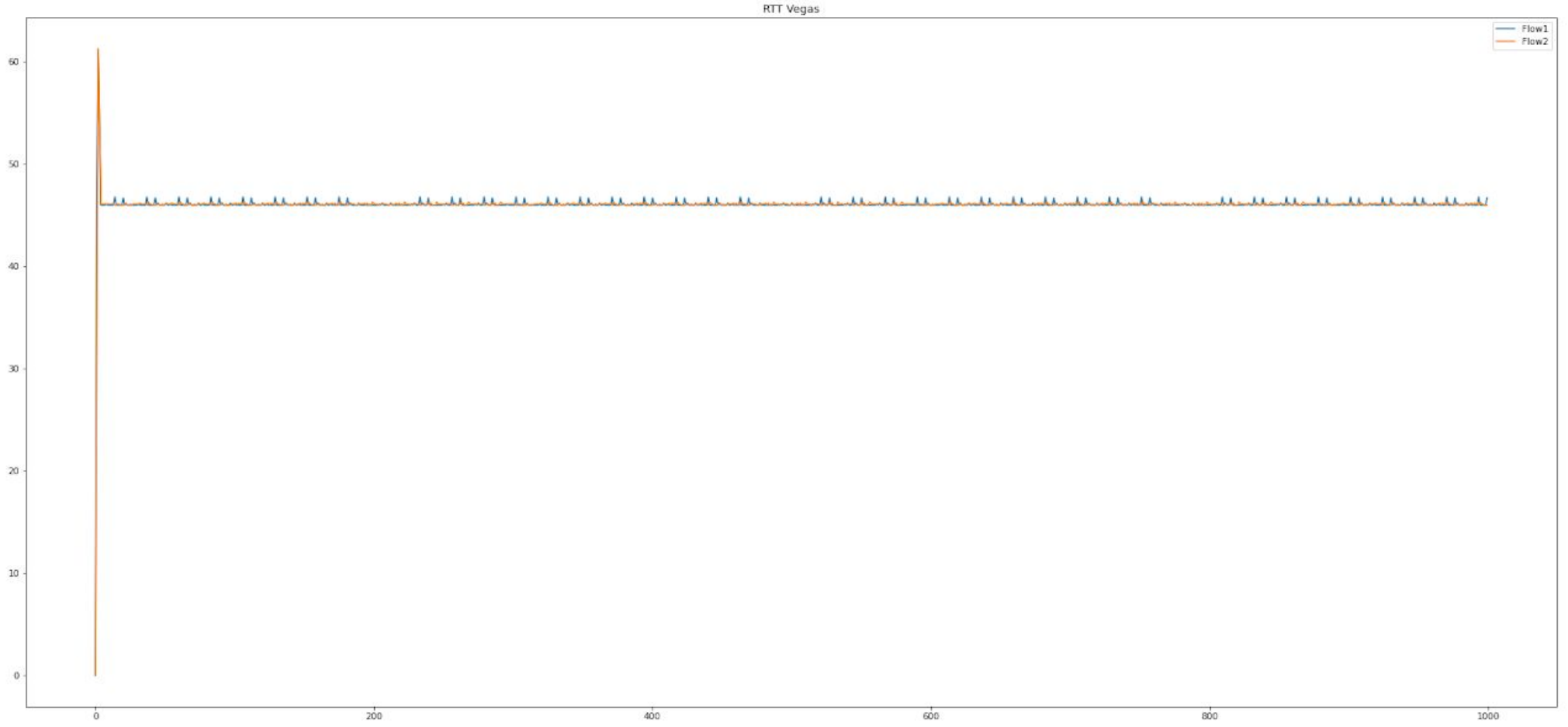
نمودار نرخ RTT برای TCP-Tahoe:



تحلیل:

با مقایسه دو نمودار قبلی که تغییرات RTT دو پروتکل Reno, Tahoe را نشان می‌دهند می‌بینیم که دامنه نوسان این مقادیر برای پروتکل Reno کمتر است که با توجه به دامنه نوسانات کمتر و شیب نوسانات کمتری که تغییرات سایز پنجره Reno داشت این مورد توجیه‌پذیر است و به این شکل مشاهده می‌شود.

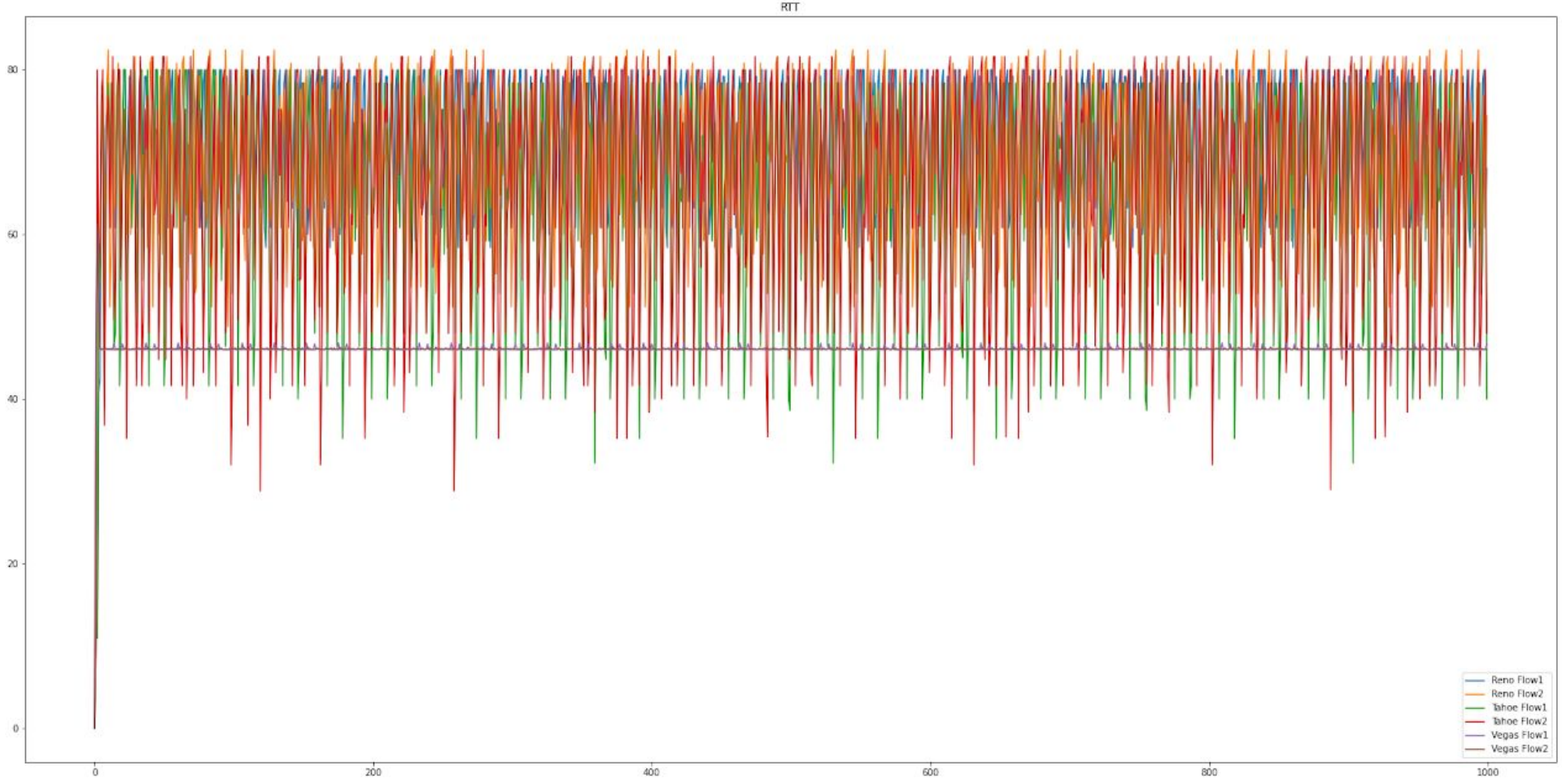
نمودار نرخ RTT برای TCP-Vegas:



تحلیل:

در اینجا می‌بینیم که در مقایسه RTT برای دو جریان در پروتکل Vegas پس از زمان اولیه که پروتکل در حال یافتن کانفیگ مناسب خود است تقریبا مقادیر RTT به طور کامل ثابت می‌شود و مقادیر برابر برای هر دو جریان وجود دارند و جریان اول اندکی نوسانات بیشتری از خود نشان می‌دهد که به نظر می‌آید دلی عدم مشاهده چنین نوسان‌های بزرگی در جریان دوم به طور عمده به دلیل میانگین‌گیری ای باشد که ما از داده‌های ناهمگون جریان دوم گرفته‌ایم و چنین‌نوسان‌هایی در آنجا چون در زمان‌های یکان رخ نداده‌اند بسیار کوچکتر شده‌اند. همچنین این نوسان بیشتر در نمودار مربوط به goodput مشابها بین دو جریان دیده می‌شد که به همین طریق قابل توجیه است.

نمودار کلی نرخ RTT:



تحلیل و مقایسه کلی:

در مقایسه بین ۲ جریان در ۳ پروتکل داریم: در مورد مقادیر CWND و GoodPut جریانی که به طور معمول اندازه پنجره بزرگتری دارد گذردهی مفید بیشتری دارد که این با توجه به این که سایز بزرگتر پنجره به معنای فرستادن تعداد بیشتری بسته روی شبکه قابل توجیه است و همین توقع می‌رود همچنین در ادامه با مقایسه Loss Rate می‌بینیم که جریانی که اندازه پنجره بزرگتری دارد به طور معمول Loss Rate کمتری از جریان دیگر دارد که این نیز با توجه به معیار تغییر و کم شدن سایز پنجره که بر اساس رخداد packet loss است قابل توجیه بوده و همین مورد را نیز متوقع هستیم اما در نهایت اگر به تفاوت میانگین مقادیر RTT بین دو جریان در هر یک از این پروتکل های سه گانه ذکر شده نگاه کنیم می‌بینیم که این مقادیر بین دو جریان تقریبا یکسان است و آن تفاوت‌های معنادار قبلی را بین این پارامتر مشاهده نمی‌کنیم.

توجه شود که در مقایسه Loss Rate چون Vegas این مقدار را همیشه ۰ نگه می‌دارد و مقایسه مقدار Loss Rate بین دو جریان معنایی ندارد و همیشه ۰ است.

بخش سوم:

نحوه‌ی اجرای پروژه

ابتدا با استفاده از دستور زیر برای هر کدام از ۳ نوع TCP، ده بار شبیه‌سازی موجود در فایل script.tcl را اجرا میشود:

```
ns main.tcl
```

با اجرای این دستور فایل های trace و انیمیشن در فولدر out ذخیره می‌شوند.
سپس با باز کردن فایل main.ipynb توسط Jupyter Notebook می‌توانید کد های پایتون تولیدکننده نمودارها را به همراه خروجی‌های متناظر هر قسمت از کد مشاهده نمایید. در صورت عدم فراهم بودن ابزار Jupyter Notebook می‌توانید با اجرای دستور زیر فایل پایتون را اجرا کنید:

```
python3 main.py
```

توجه داشته باشید که همواره بعد از اجرای کامل فایل main.ipynb یا اجرای فایل main.py نمودارهای تولید شده‌ی داخل پوشه plots موجود کنار فایل اجرایی و همچنین فایل csv داده‌هایی که برای رسم نمودار استفاده شده‌اند در پوشه csv_files ذخیره می‌شوند.
همچنین شما می‌توانید به طور کامل تصاویر نهایی مربوط به محتوای فایل main.ipynb را از طریق فایل main.html که داخل پوشه اصل ارسالی ما قرار دارد نیز مشاهده کنید.