به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه‌های کامپیوتری

پروژه‌ی دوم(NS2)

نام و نام خانوادگی	عارف افضلی امین محقق
شماره دانشجویی	810896042 810195579
تاریخ ارسال گزارش	

فهرست گزارش سوالات

نحوه اجرای برنامه

کد TCL مربوط به این قسمت در فایل tcp_new_reno.tcl قرار دارد و کد پایتون به منظور تحلیل خروجی در فایل process_trace_file.py قرار دارد.(به منظور امتحان الگوریتم‌های متفاوت اسم فایل tcl الگوریتم مورد استفاده باید به عنوان command line argument داده شود)

```
python3.5 process_trace_file.py tcp_new_reno.tcl
```

Figure 1 دستور مورد استفاده به منظور شبیه‌سازی **tcp_new_reno**

همچنین برای افزایش سرعت پردازش trace.tr از کتابخانه‌ی multiprocessing پایتون استفاده شد.

TCP NewReno

توضیح الگوریتم

تفاوت عملکرد الگوریتم‌های مختلف tcp در واکنش به time out و یا dupticate ack است.

NewReno در واکنش به time out وارد حالت slow start می‌شود و ssthresh را برابر با نصف congestion window و congestion window جدید را برابر با ۱ قرار می‌دهند.

الگوریتم اولیه‌ی Reno در موجهه با ۳ duplicate ack بر خلاف Tahoe وارد مرحله‌ی slow start نشده و بلکه مستقیما وارد مرحله‌ی congestion control می‌شود. در New Reno به منظور بهبود فرستادن بسته‌ها در مرحله‌ی congestion control به ازای هر duplicate ack که می‌رسد یک بسته‌ی جدید از انتهای congestion window فرستاده می‌شود.

شبیه سازی

به منظور شبیه‌سازی الگوریتم یاد شده از نرم‌افزار ns2 استفاده شد و ساختار زیر پیاده سازی شد.

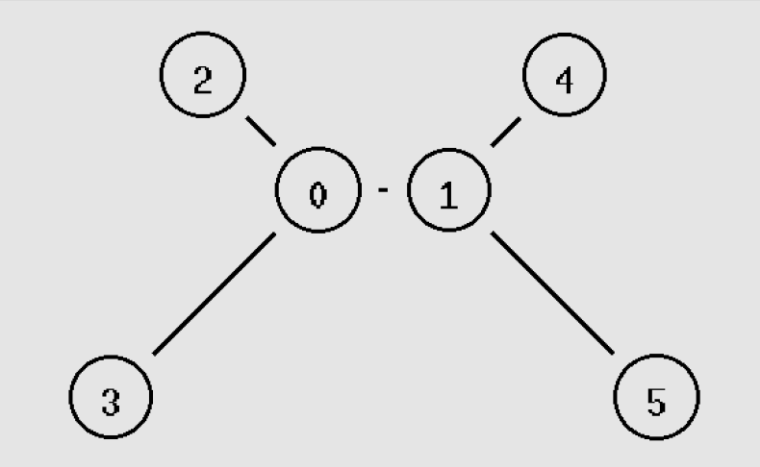


Figure 2 ساختار شبکه‌ی پیاده‌سازی شده در **ns2**

نودهای ۱ و ۲ روتر هستند و نودهای ۲و ۳ مبدا tcp و ۴و ۵ سینک‌های tcp هستند.

به منظور مشخص کردن الگوریتم congestion مورد استفاده TCP در فایل tcl. از دستور زیر استفاده شد.

```
set tcp_1 [new Agent/TCP/Newreno]
```

Figure 3مشخص کردن **TCP_New_reno**

بعد از اجرای فایل tcp_new_reno.tcl(که کد پایتون نوشته شده خودش این فایل را اجرا می‌کند) دو فایل out.nam و trace.tr تولید می‌شود که از خروجی‌های trace.tr به منظور رسم نمودارها استفاده می‌شود.

همچنین با توجه به اینکه تاخیر مربوط به لینک‌های صفر به سه و همچنین یک به پنج باید به صورت تصادفی انتخاب شوند، در کد tcl. از عبارت زیر استفاده شد.

```
set random_delay [expr {5+(rand()*20)}]
```

Figure 4 تولید تاخیر تصادفی بین ۵ تا ۲۵ میلی ثانیه

به منظور کشیدن نمودار CWND از trace کردن متغیر _cwnd در فایل tcl استفاده شد. خروجی مربوط به tcp_1 و tcp_2 به قرار زیر است.(محور x نشان‌دهنده‌ی زمان و محور y مقدار cwnd را نشان می‌دهد)

شکل کلی(مقیاس بزرگ):

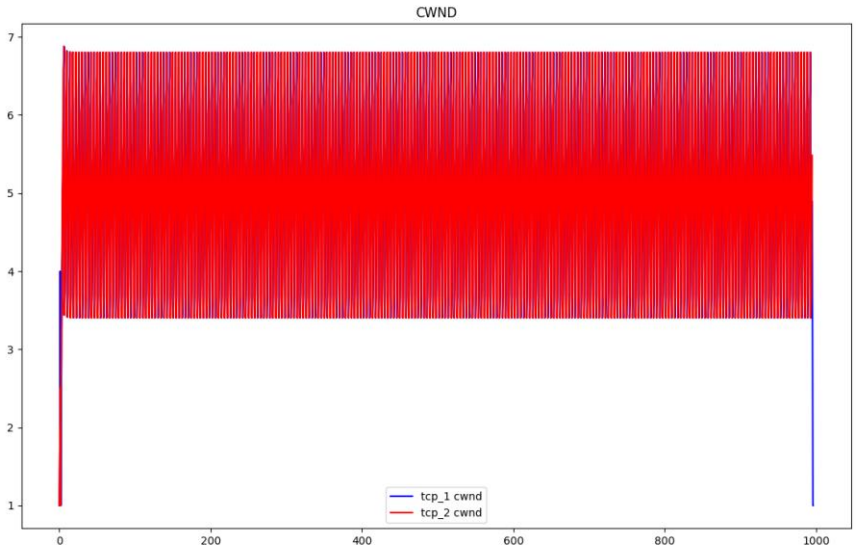


Figure 5نمودار **CWND**(مقیاس بزرگ)

اگر مقیاس را مقداری کوچکتر کنیم شکل زیر حاصل می‌شود.

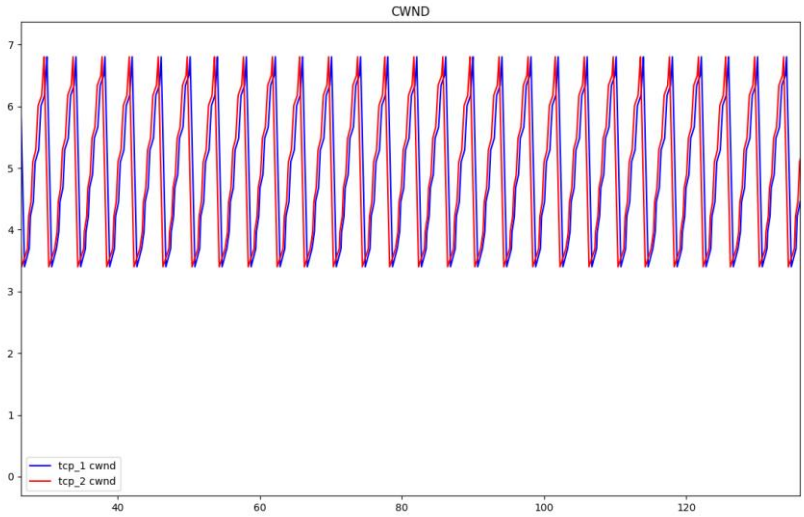


Figure 6 نمودار **CWND**(مقیاس کوچک)

همانطور که از شکل بالا مشخص است و از قبل انتظار داشتیم با گرفتن ۳ duplicate ack مقدار CWND نصف می‌شود و به صورت خطی(additive increase) شروع به افزایش پیدا میکند. در این نمونه هم با توجه به cwnd(مقیاس بزرگ) time out نداشتیم(چرا که اگر داشتیم cwnd باید ۱ میشد)

نمودار بعدی مربوط به RTT است. به منظور کشیدن این نمودار از trace کردن متغیر _rtt در فایل tcl استفاده شد.

محور x نشان دهنده‌ی زمان و محور y میزان rtt را نشان می‌دهد.

نمودار RTT(مقیاس بزرگ):

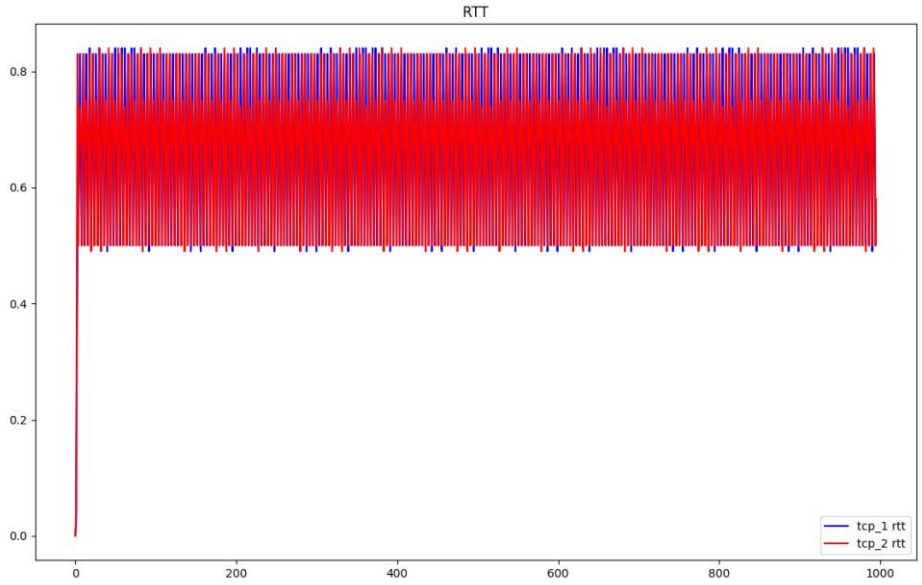


Figure 7 نمودار **RTT** مقیاس بزرگ

نمودار RTT(مقیاس کوچک):

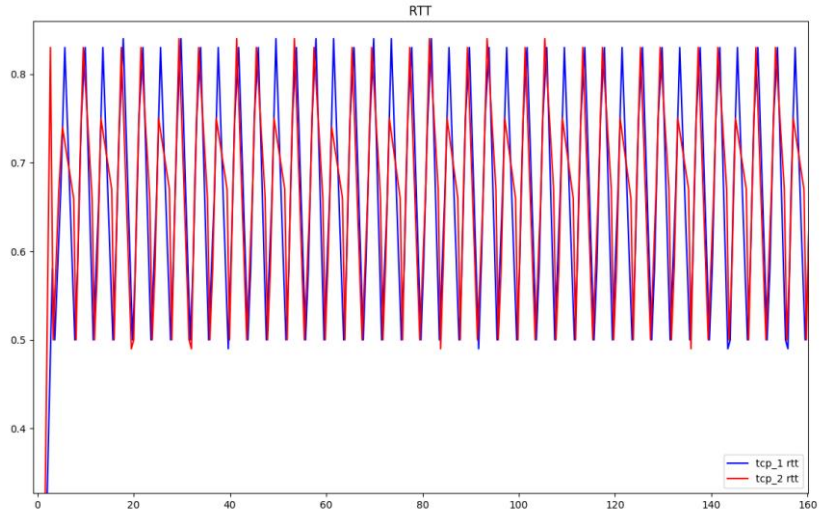


Figure 8 نمودار **RTT** مقیاس کوچک

نمودار بعدی مربوط به good put ration است. برای رسم این نمودار تعداد پکت‌های فرستاده شده را به طور جداگانه محاسبه کرده و هر بار که در trace file میزان _ack گزارش شده بود، عدد ack را تقسیم بر میزان پکت‌ها فرستاده شده تا آن لحظه کردیم.

نمودار good put ratio مقیاس بزرگ:

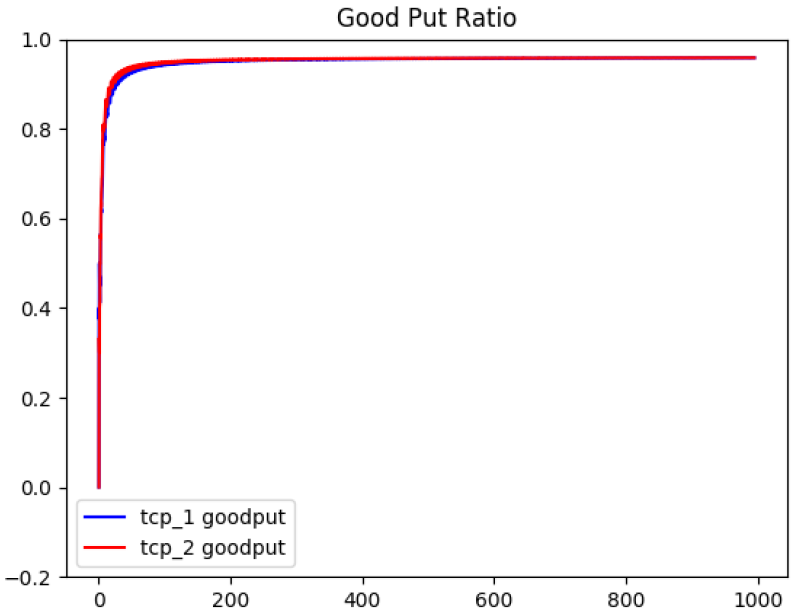
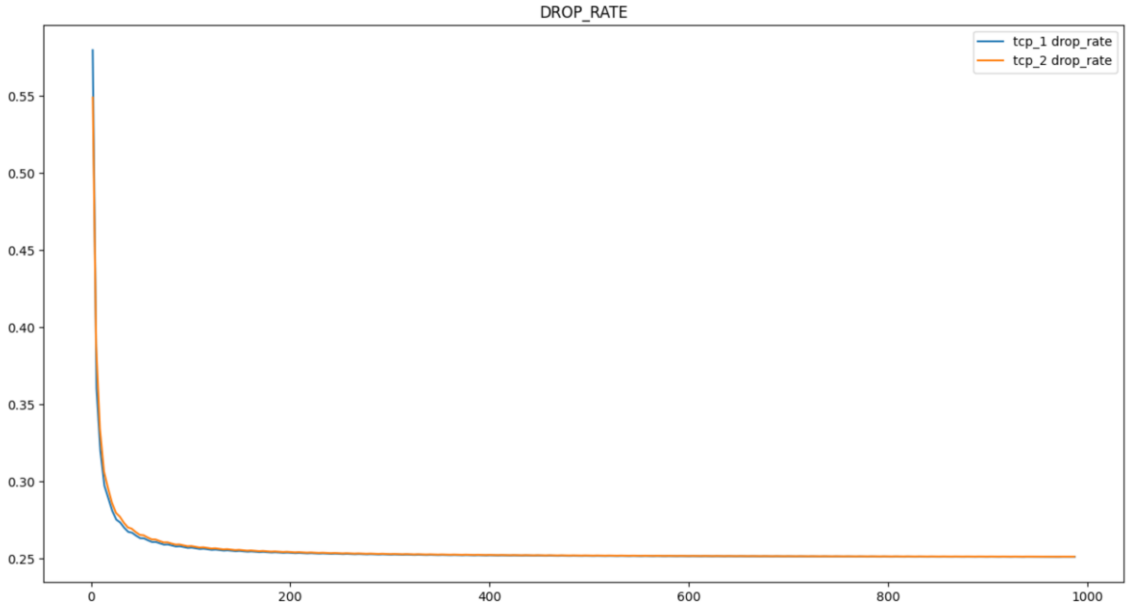


Figure 9 نمودار **goodput ratio** مقیاس بزرگ

نمودار good put ratio مقیاس کوچک:

نمودار نرخ از دست رفتن بسته:



مقدار پکت‌های دراپ شده در مجموع هم در پایان محاسبه شده و چاپ می‌شود.

```
tcp_1 window size: 20, tcp_2 window size: 20
tcp_tick_1 :0.01 last_ack_1 :6181 - last_seq_1 :6181
tcp_tick_2 :0.01 last_ack_2 :6177 - last_seq_2 :6177
CWND plot is complete
Good put ratio plot is complete
RTT plot is complete
Num Drops For TCP_1 = 258
Num Drops For TCP_2 = 258
```

Figure 10 خروجی متنی چاپ شده شبیه سازی

TCP TAHOE

توضیح الگوریتم

هـدف از ایـن مکانیزم تضمین این است که اتصال TCP قادر باشد به موازنه برسد و پس از اینکه به موازنه رسید، اتصال باید از اصل حفظ بسته ها پیروی کند. این اصل می گوید که وقتی اتصالی به موازنه یا تعادل رسید، فقط زمانی می تواند یک بسته را روی شبکه منتقل کند که فیدبک بازگشتی که نشان دهنده خروج یک بسته دیگر از شبکه است را دریافت کرده باشد. اتصال با وارسی شبکه در مورد پهنای باند موجـود و همچنـین تنظـیم یک پنجره ازدحام فرستنده که به تازگی انجام شده باشد، به توازن می رسد. در TCP Tahoe، پنجره ای که فرستنده بکار می برد، حد پایین پنجره گیرنده و همین پنجره ازدحام جدید است. عمده ترین بهبود در کارایی TCP از مکانیزم انتقال مجدد سریع ناشی می شود. پس از دریافت سومین اعلام وصول تکراری انتقال مجدد آغاز می شود، بنابراین TCP Tahoeمجبـور نیـست زمـانی طـولانی بـرای منقضی شدن تایمر انتقال مجدد صبر کند.

شبیه سازی

به منظور شبیه‌سازی کافیست نوع TCPها در فایل tcl. را از Newreno به TCP تغییر دهیم.

```
set tcp_1 [new Agent/TCP]
```

Figure 11 اصلاح انجام شده به منظور **TCP Tahoe**

بعد از اصلاح بالا کد پایتون را اجرا میکنیم.

نمودار پنجره‌ی CWND به صورت زیر است.

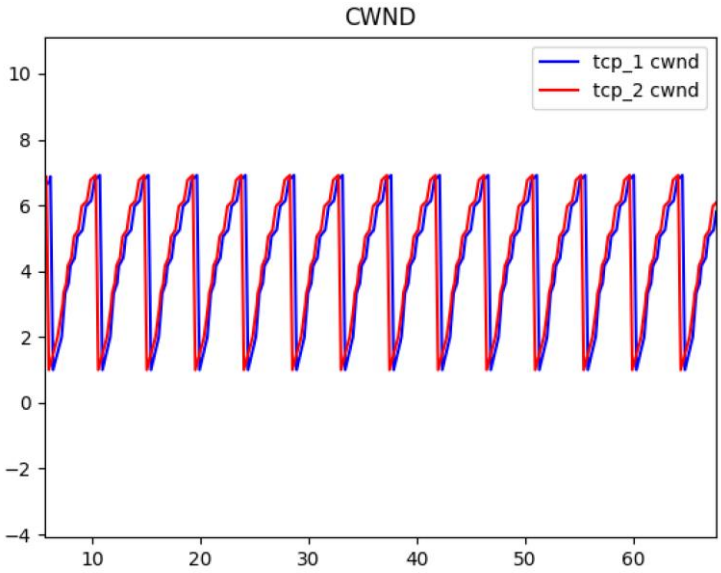


Figure 12 نمودار **CWND**(مقیاس کوچک)

همانطور که از قبل انتظار داشتیم با duplicate ack هم مقدار CWND برابر ۱ شده و سپس با به صورت نمایی تا نصف CWND قبلی بالا رفته و بعد از آن وارد حالت congestion control می‌شود.

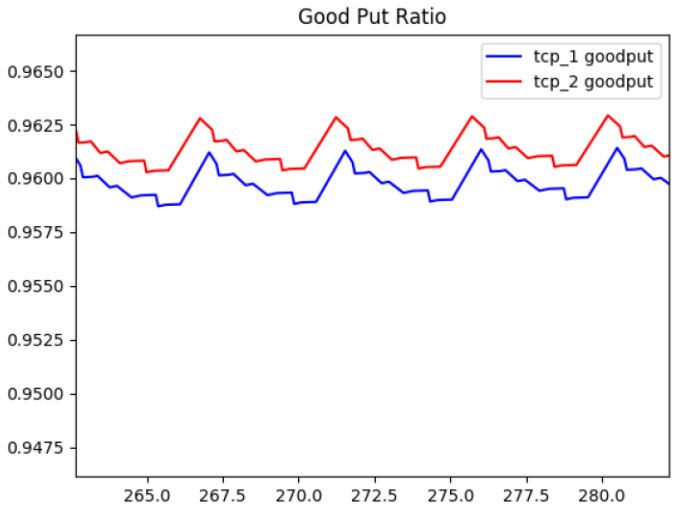


Figure 13 نمودار **good put ration**(مقیاس کوچک)

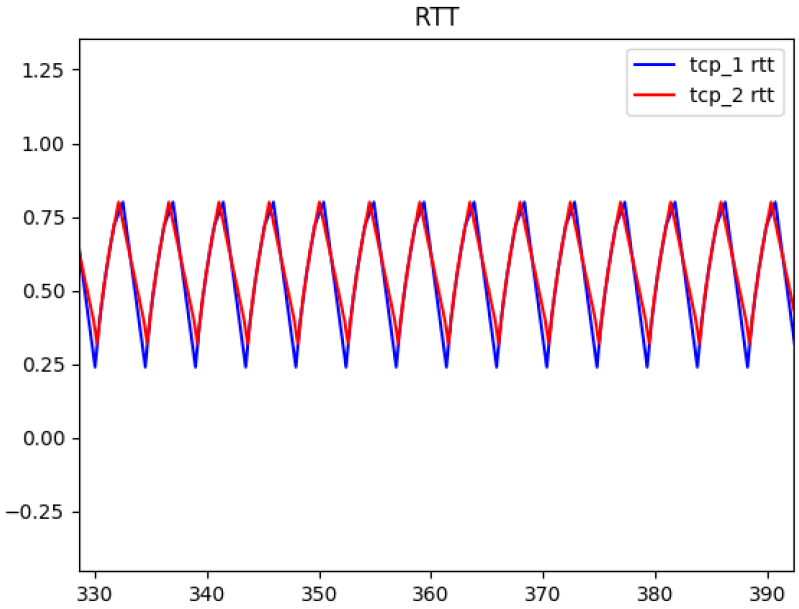
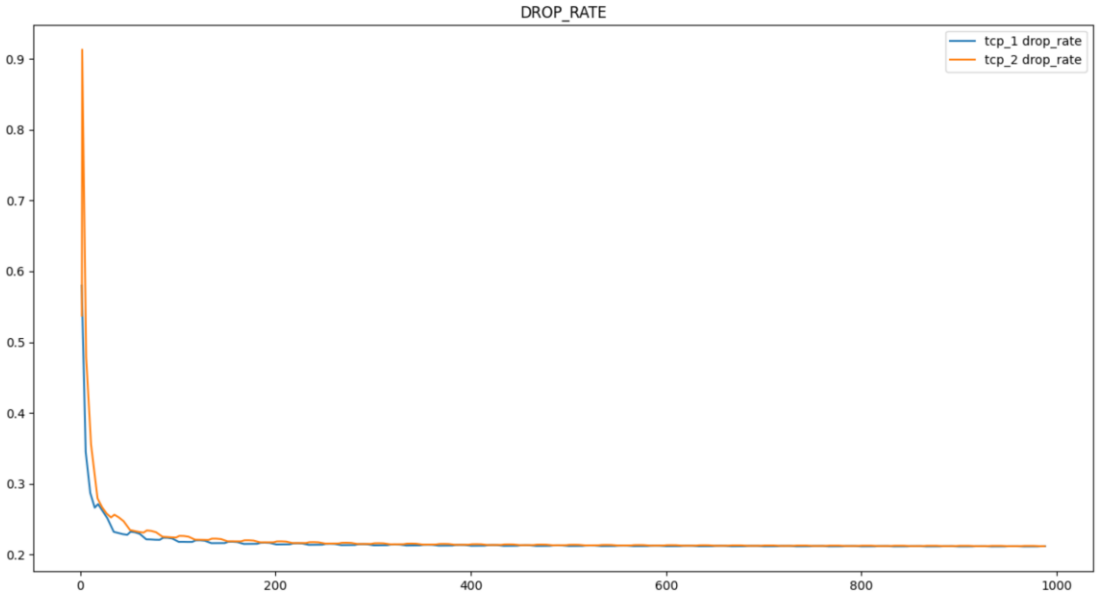


Figure 14 نمودار **RTT**(مقیاس کوچک)

نمودار نرخ از دست رفتن بسته:



در نهایت خروجی چاپ شده هم به شرح زیر است:

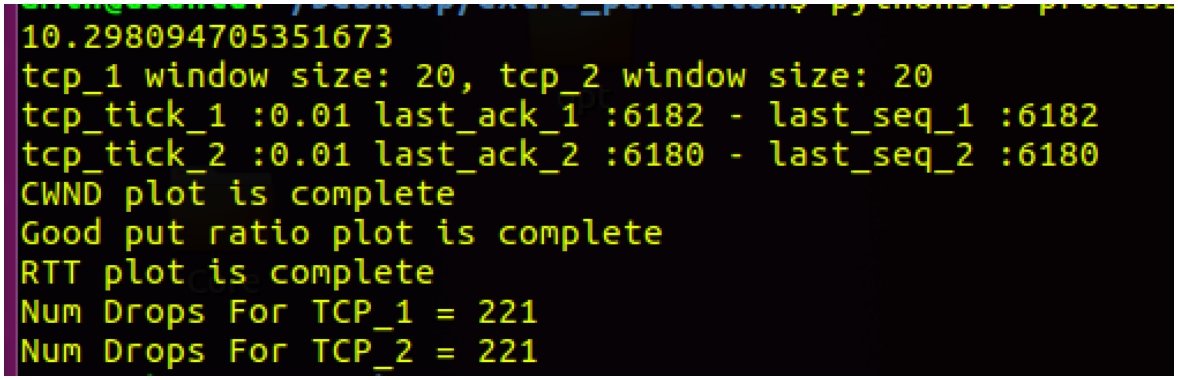


Figure 15 خروجی چاپ شده برنامه(**TCP TAHOE**)

TCP VEGAS

توضیح الگوریتم

TCP Vegas یک دگرگونی اساسی در پیاده سازیهای قدیمی تر TCPداده است و در ویرایش BSD Unix ۳/۴ تعمیم داده شده است. این روش از سه تکنیک به منظور بهبود توان عملیاتی TCPو کاهش بسته های گم شده استفاده می کند. اول TCP Vegas برای تخمین RTTبجای دانه های درشت تایمر TCPاز ساعت سیستم استفاده می کند. تخمین دقیقتر RTT موجب می شود تا ازدحام زودتر تشخیص داده شود، بنابراین TCPمی تواند هر بسته را حتی قبل از دریافت سه تا dupack انتقال مجدد دهد.

دوم ، Vegas با مقایسه توان عملیاتی اندازه گیری شده و توان عملیاتی مورد انتظار که بوسیله اندازه پنجره محاسبه می شود, تغییرات توان عملیاتی را زیر نظر می گیرد. الگوریتم پرهیز از ازدحام از این اطلاعات برای حفظ مقدار بهینه داده در شبکه استفاده می کند. بنابراین، از توان عملیاتی به عنوان شاخصی برای تخمین حالت پایدار استفاده می شود. و در آخراینکه Vegas الگوریتم شروع آهسته را با معرفی یک فاز پنجره ازدحام ثابت در هر رفت و برگشت تغییر داده است. پنجره ازدحام در فاصله بین هر دو رفت و برگشت به طور نمایی افزایش می یابد ودر مدت زمان رفت و برگشت بسته ثابت می ماند. در مدت فاز ثابت، الگوریتم, توان عملیاتی تخمینی و بدست آمده را با هم مقایسه می کند. از اختلاف این دو توان عملیاتی استفاده می شود تا TCPمشخص کندکه باید وارد مُد افزایشی یا مُد کاهشی شود. TCP Vegasزمان بیشتری لازم دارد تا وارد حالت پایدار شود ، ولی برآورد حالت پایدار دقیق تر و درست تر است.

شبیه سازی

به منظور شبیه‌سازی کافیست نوع TCPها در فایل tcl. را از TCP به TCP/Vegas تغییر دهیم.

بعد از اصلاح بالا کد پایتون را اجرا میکنیم.

نمودار پنجره‌ی CWND به صورت زیر است.

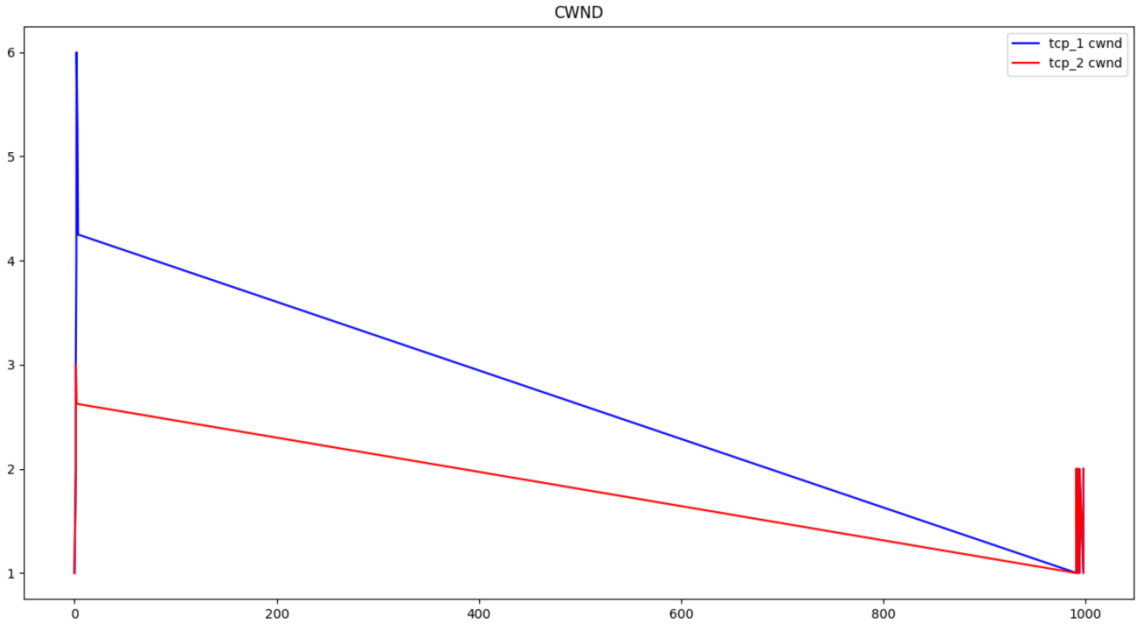


Figure 16 نمودار **CWND**(مقیاس بزرگ)

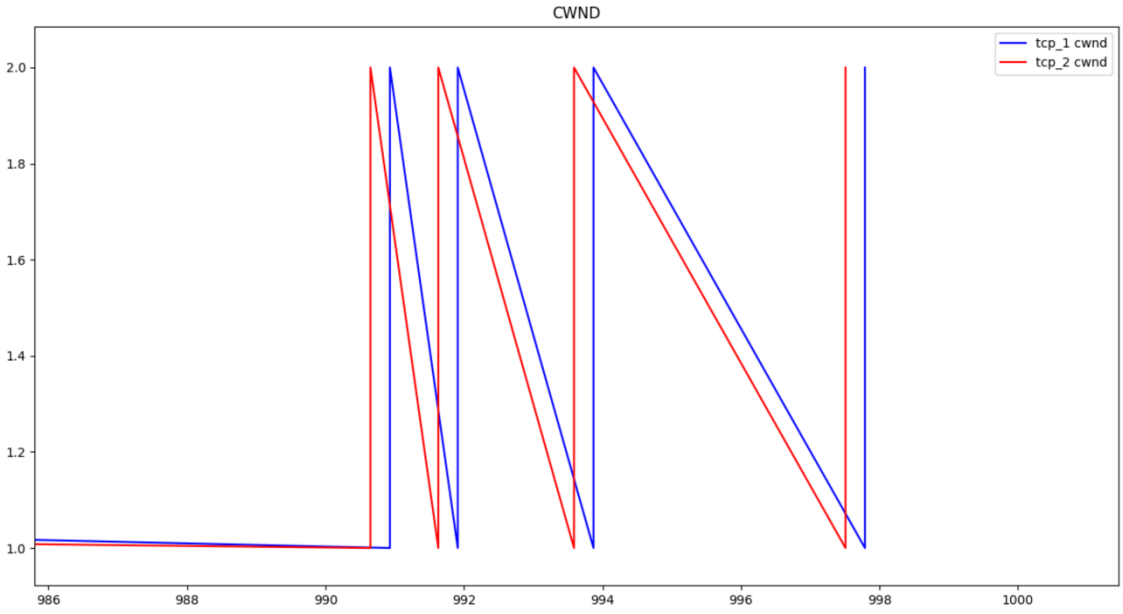


Figure 17 نمودار **CWND**(مقیاس کوچک)

همانطور که از قبل انتظار داشتیم در cwnd، با شروع از صفر و رسیدن به مقدار زیادی که بتوان دقیق تر اندازه‌گیری کرد و نکته دیگر اینکه اون بخشی که به صورت نزولی یک خط صاف است، بدلیل اینکه این

مقدار تغییری نکرده، ثابت است ولی به خاطر اینکه بعد از مدتی که تغییر می‌کند مقدار کمتر از مقدار قبل است و داده ای این بین نبوده، این دو نقطه به صورت خط صاف به هم متصل شده‌اند.

نمودار Good Put را در مقیاس‌های بزرگ و کوچک می‌توان مشاهده کرد و می‌بینیم که بعد از صعود نمایی ثابت می‌ماند.

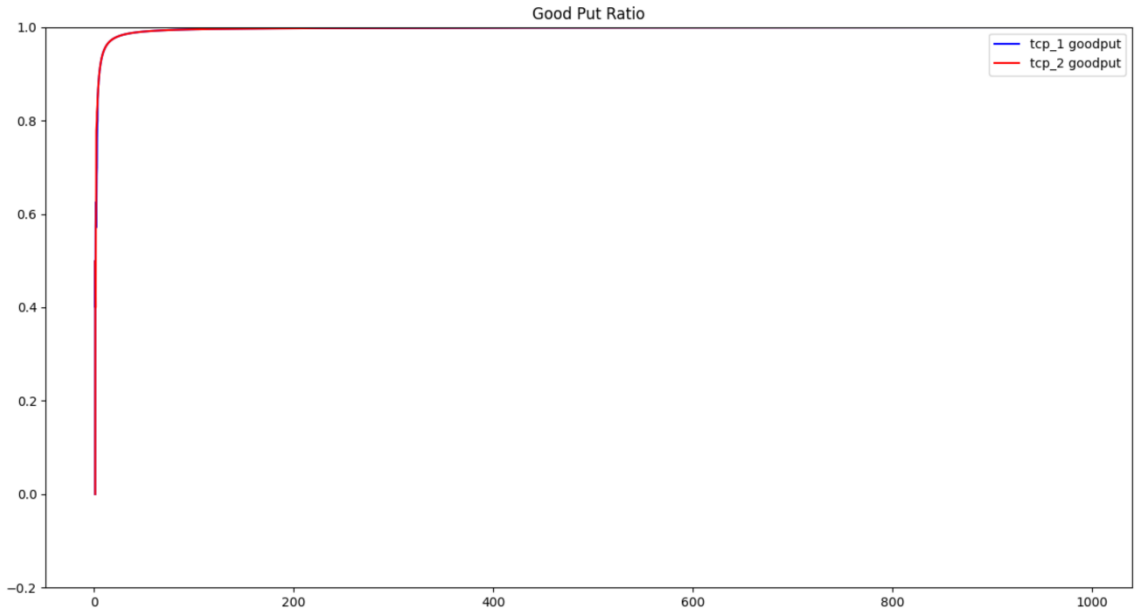


Figure 18 نمودار **good put ration**(مقیاس بزرگ)

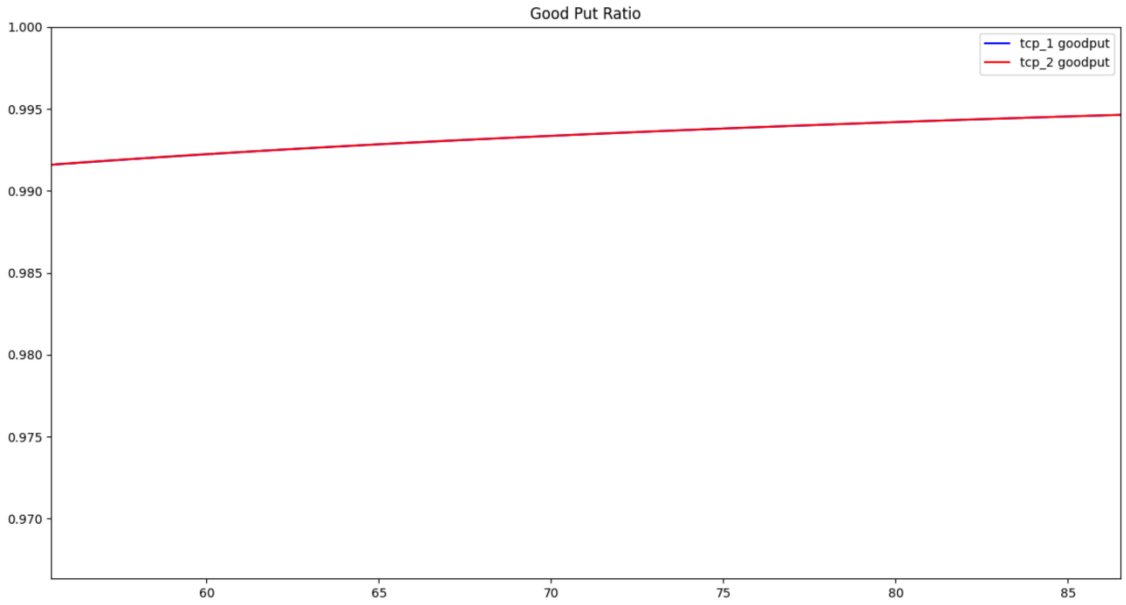


Figure 19 نمودار **good put ration**(مقیاس کوچک)

نمودار RTT را در مقیاس‌های بزرگ و کوچک می‌توان مشاهده کرد:

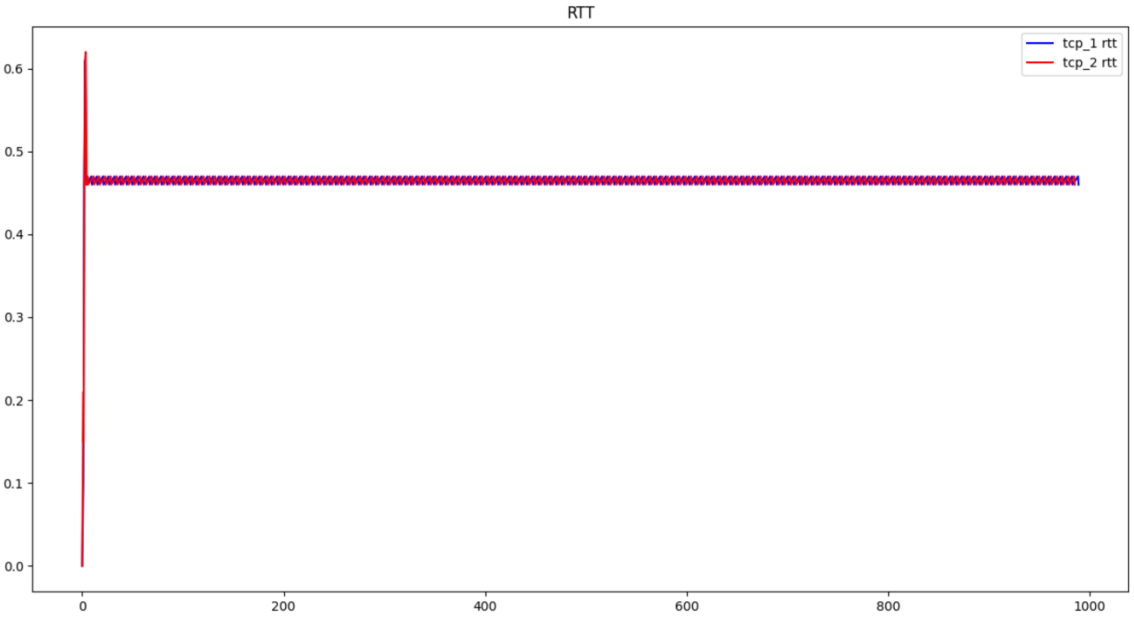


Figure 20 نمودار **RTT**(مقیاس بزرگ)

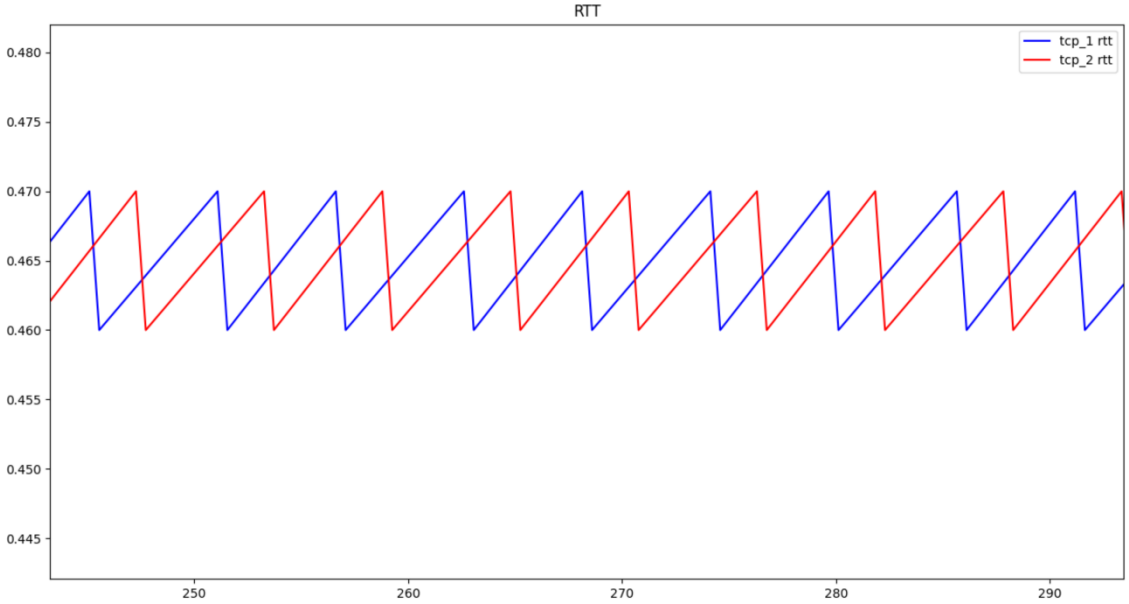
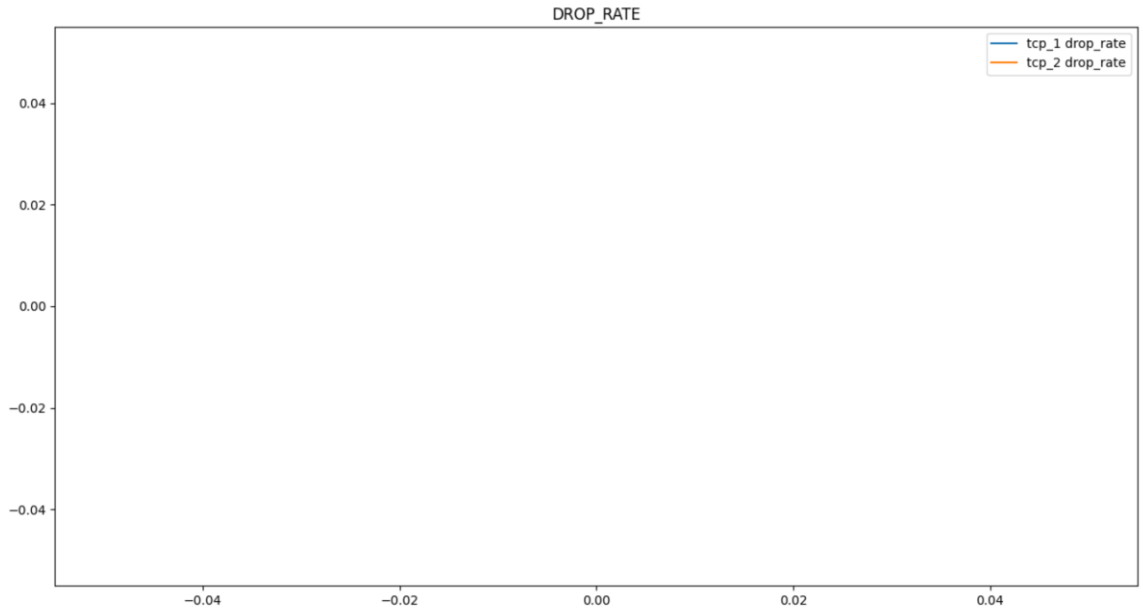


Figure 21 نمودار **RTT**(مقیاس کوچک)

نرخ از دست رفتن بسته در این الگوریتم صفر است:



در نهایت خروجی چاپ شده هم به شرح زیر است که مقدار بسته‌های از دست رفته در مجموع نیز در آن قابل مشاهده است:

```
13.720983792432111
tcp_1 window size: 20, tcp_2 window size: 20
tcp_tick_1 :0.01 last_ack_1 :8588 - last_seq_1 :8588
tcp_tick_2 :0.01 last_ack_2 :4293 - last_seq_2 :4293
CWND plot is complete
Good put ratio plot is complete
RTT plot is complete
Num Drops For TCP_1 = 0
Num Drops For TCP_2 = 0
```

Figure 22 خروجی چاپ شده برنامه(**TCP VEGAS**)

مقایسه هر ۳ روش

در این بخش با نمایش تمام نمودارهای بالا در یک نمودار، می‌توان به مقایسه بهتری از عملکرد این روش ها رسید.

نمودار پنجره‌ی CWND به صورت زیر است.

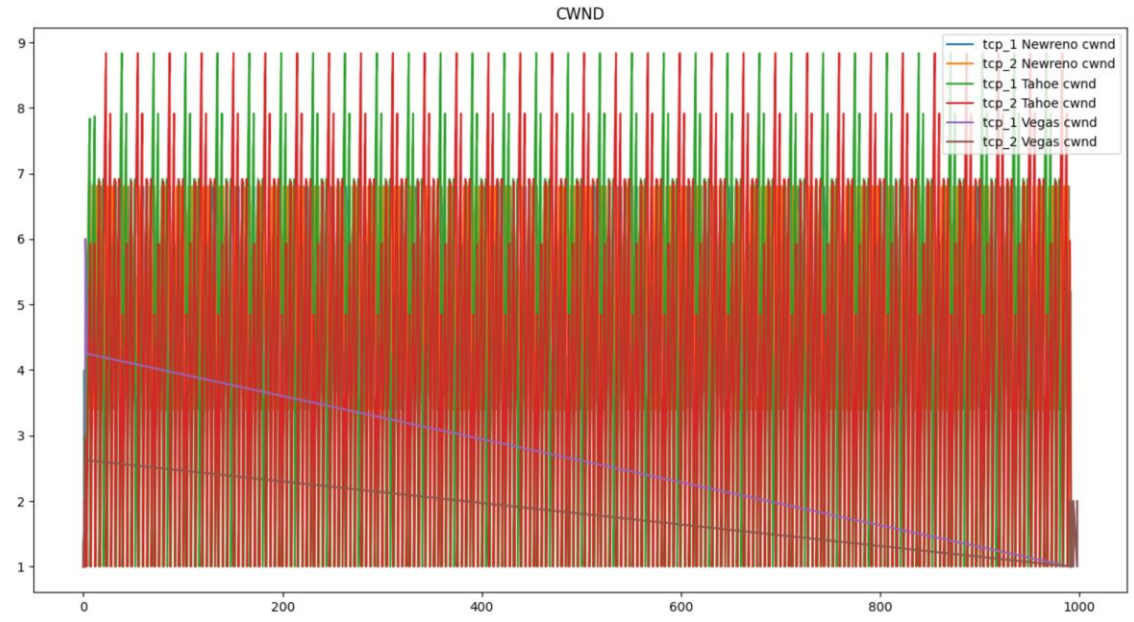


Figure 23 نمودار **CWND**(مقیاس بزرگ)

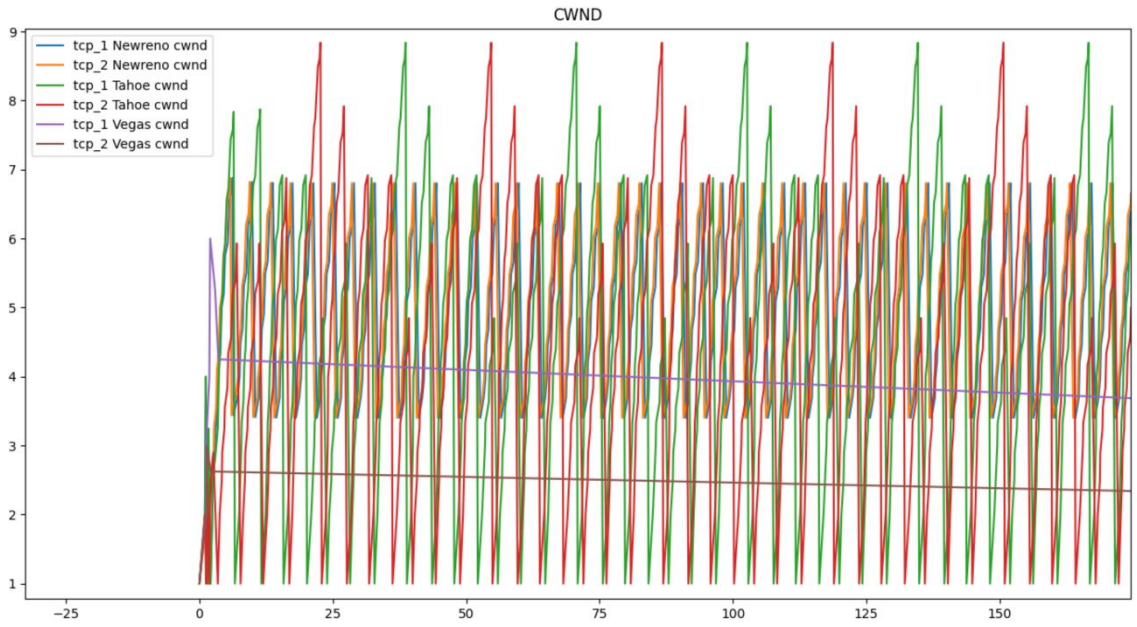


Figure 24 نمودار **CWND**(مقیاس کوچک)

نمودار Good Put را در مقیاس‌های بزرگ و کوچک می‌توان مشاهده کرد:

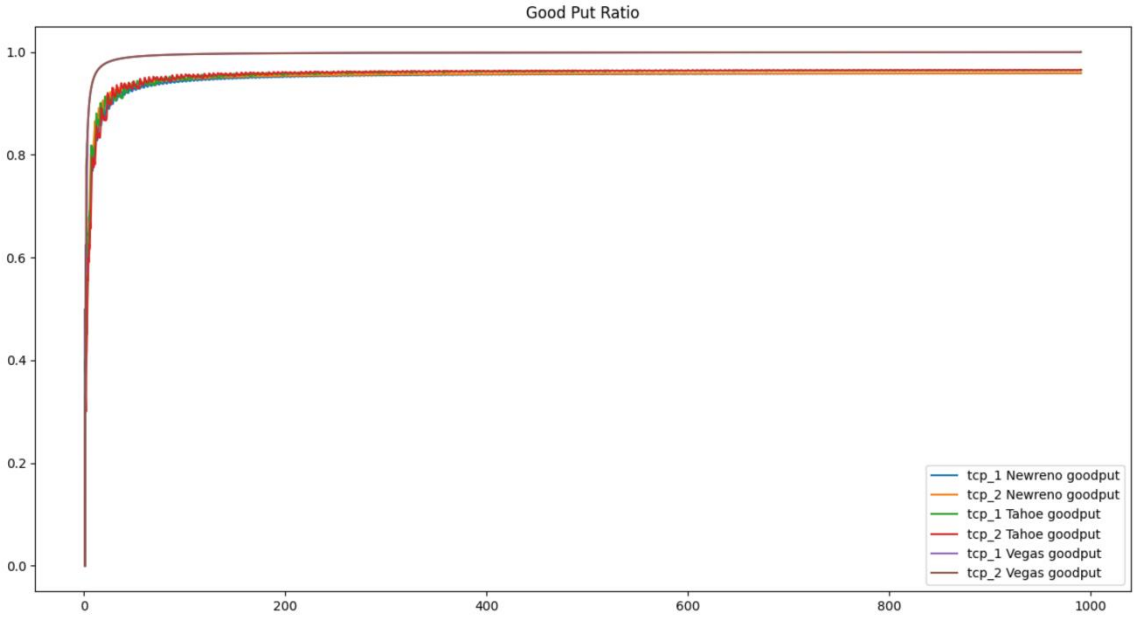


Figure 25 نمودار **good put ration**(مقیاس بزرگ)

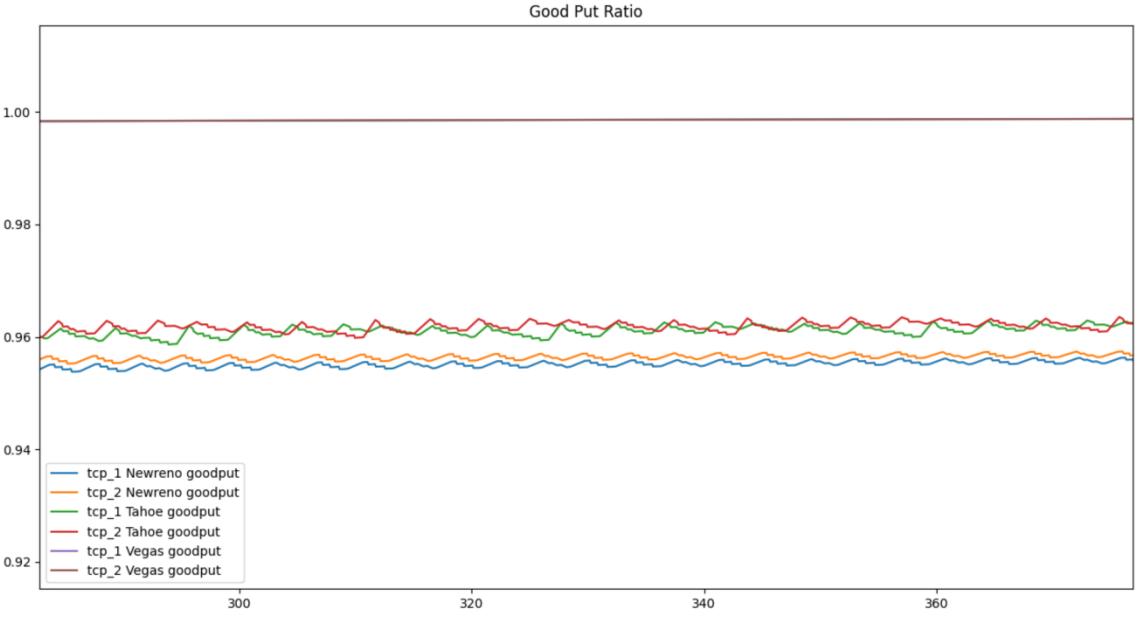


Figure 26 نمودار **good put ration**(مقیاس کوچک)

نمودار RTT را در مقیاس‌های بزرگ و کوچک می‌توان مشاهده کرد:

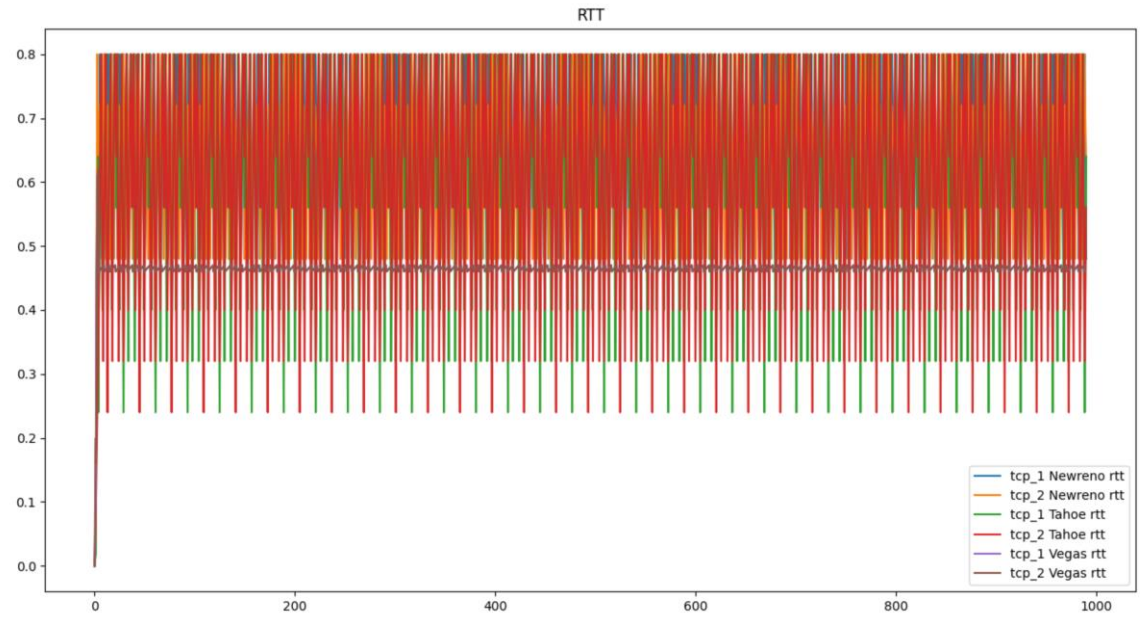


Figure 27 نمودار **RTT**(مقیاس بزرگ)

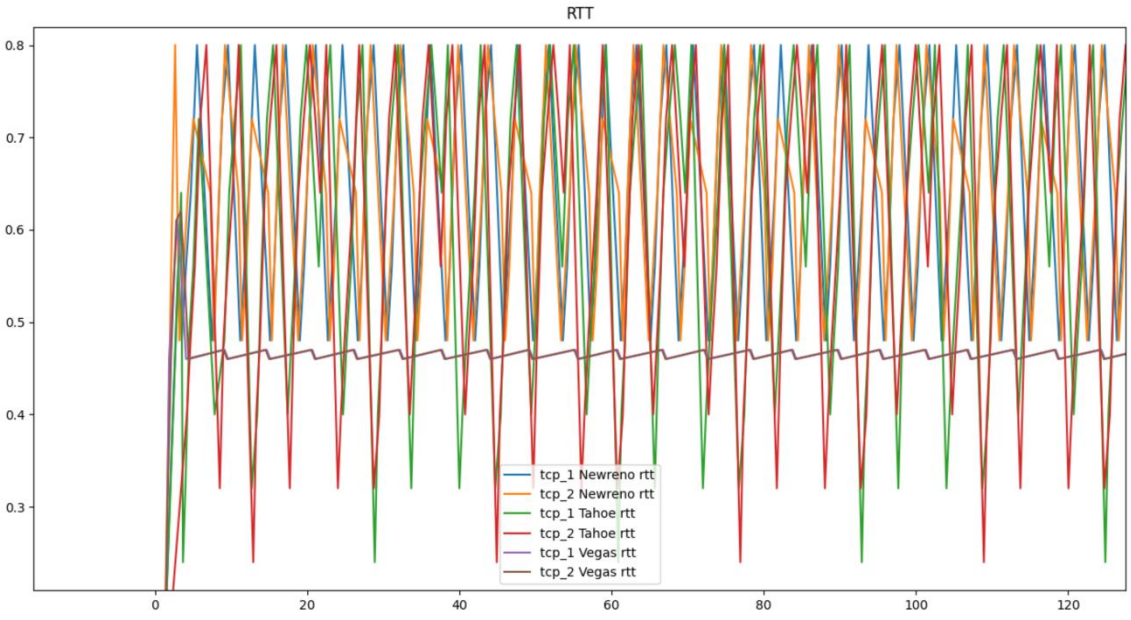
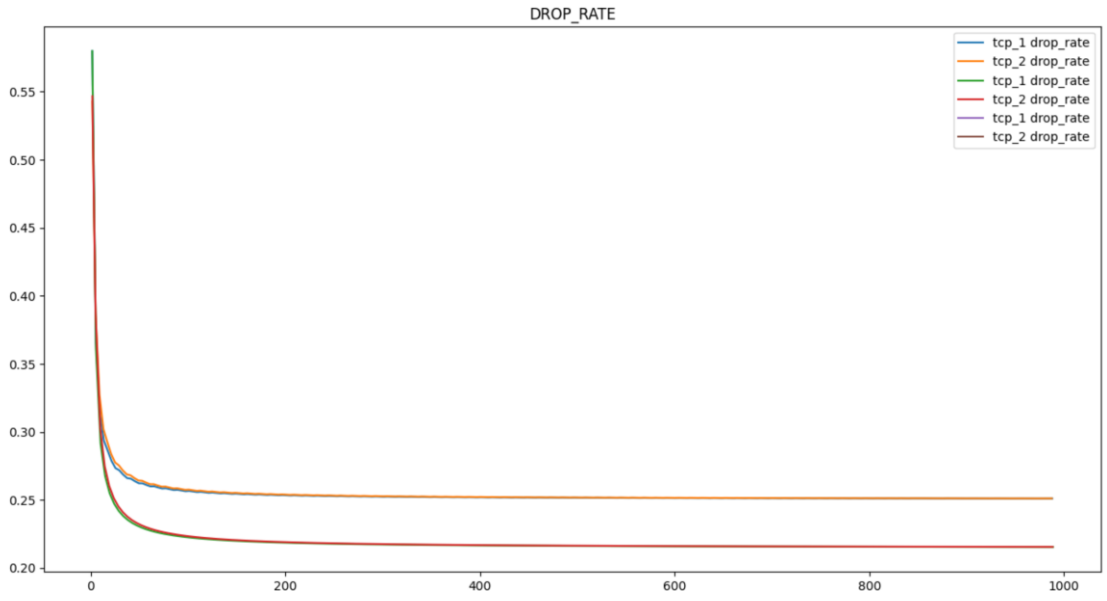
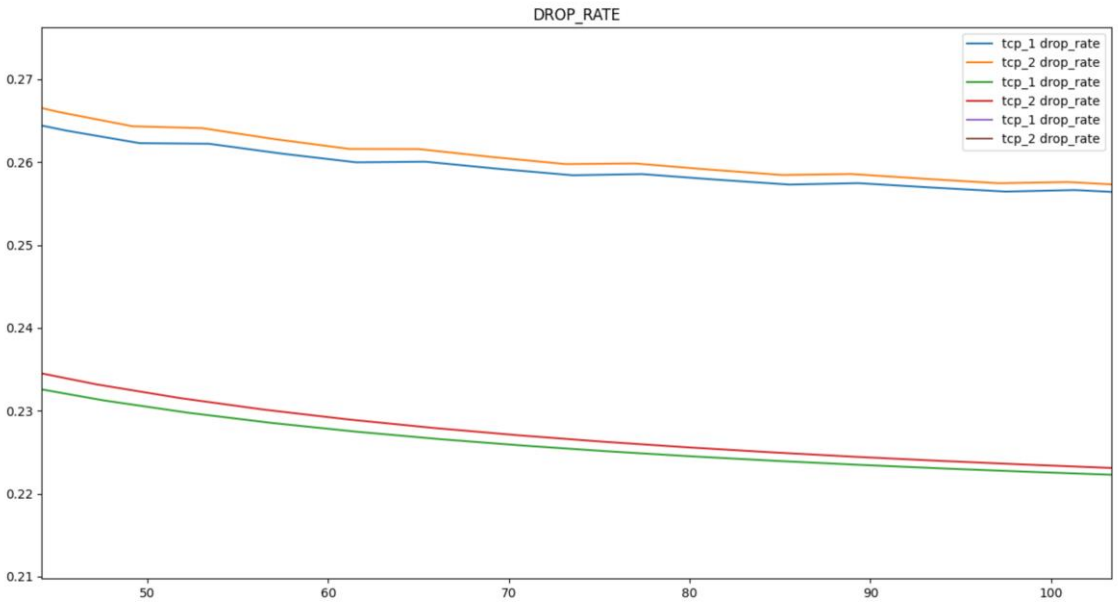


Figure 28 نمودار **RTT**(مقیاس کوچک)

نمودار نرخ از دست رفتن بسته در مقیاس بزرگ:



نمودار نرخ از دست رفتن بسته در مقیاس کوچک:



در نهایت خروجی چاپ شده هم به شرح زیر است که مقدار بسته‌های از دست رفته در مجموع نیز در آن قابل مشاهده است:

```
TCP New-Reno:
16.684221211673794
tcp_1 window size: 20, tcp_2 window size: 20
tcp_tick_1 :0.01 last_ack_1 :6201 - last_seq_1 :6201
tcp_tick_2 :0.01 last_ack_2 :6161 - last_seq_2 :6161
Num Drops For TCP_1 = 258
Num Drops For TCP_2 = 259

TCP Tahoe:
7.2631275384980842
tcp_1 window size: 20, tcp_2 window size: 20
tcp_tick_1 :0.01 last_ack_1 :6182 - last_seq_1 :6182
tcp_tick_2 :0.01 last_ack_2 :6180 - last_seq_2 :6180
Num Drops For TCP_1 = 221
Num Drops For TCP_2 = 221

TCP Vegas:
18.509897474902637
tcp_1 window size: 20, tcp_2 window size: 20
tcp_tick_1 :0.01 last_ack_1 :8588 - last_seq_1 :8588
tcp_tick_2 :0.01 last_ack_2 :4293 - last_seq_2 :4293
Num Drops For TCP_1 = 0
Num Drops For TCP_2 = 0

CWND plot is complete
Good put ratio plot is complete
RTT plot is complete
```

Figure 29 خروجی چاپ شده برنامه