گزارش پروژه‌ی NS2 شبکه‌های کامپیوتری

حسنا سادات آزرم‌سا

810196408

شکیبا بلبلیان خواه

810196426

پیاده‌سازی شبیه‌سازی

در ابتدا برای شبیه‌سازی نیاز است تا در فرمت tcl شبکه‌ی مورد نظر پیاده‌سازی شود. برای پیاده‌سازی مراحل زیر طی خواهد شد:

۱. دو فایل .nam و .trace به منظور ذخیره‌ی اطلاعات مربوط به بسته‌ها و متغیرهای شبکه ایجاد کرده‌ایم. هم‌چنین در ادامه متغیرهای TCP agent مورد نیاز مانند cwnd، rtt ، ack و... را به فایل trace نسبت می‌دهیم تا اطلاعات آن‌ها نیز در فایل ذخیره شود. قابل ذکر است که اطلاعات این متغیرها به محض تغییر یافتن در فایل trace ذخیره خواهد شد.

۲. برای ایجاد روترها و لبه‌های شبکه‌ی مورد نظر، nodeها و linkهای مورد نیاز را تعریف می‌کنیم. در حین تعریف لینک‌ها، ظرفیت، تاخیر و روش مدیریت بسته‌های از دست‌رفته(برای مثال در این پیاده‌سازی در صورت تکمیل بودن ظرفیت صف، اخرین بسته‌ی وارد شده، دور انداخته می‌شود.) نیز تعیین خواهند شد.

- برای ایجاد تاخیر متغیر نیاز دو متغیر random در ابتدا تعریف شده که همواره در بازه‌ی ۵ تا ۲۵ مقدار می‌گیرند و این دو مقدار به دو لینک با تاخیر متغیر، اختصاص داده خواهند شد.

۳. برای لینک‌های بین دو روتر صف با ظرفیت ۱۰ تعریف خواهد شد. از آنجایی که در ns2 صف بر روی لینک تعریف می‌شود، دو صف با ظرفیت ۱۰ بر روی ابتدا و انتهای لینک بین دو روتر تعریف کرده‌ایم تا توپولوژی مسئله درست عمل کند.

۴. در این مرحله نیاز است تا agentهای TCP به دو node مبدا و مقصد اختصاص داده شود. در هر یک از فایل‌های newRenoCode.tcl، vegasCode.tcl و tahoeCode.tcl، نوع TCP تعریف شده به ترتیب newReno، vegas و tahoe (نوع دیفالت خود TCP) خواهد بود.

۵. برای ایجاد جریان، لازم است یک traffic generator به TCP agentهای خود اختصاص دهیم که ftp برای این کار انتخاب شد و به nodeهای مبدا متصل گردید.

۶. برای اجرای شبیه‌سازی نیاز است برای شبکه یک برنامه‌ریزی صورت بگیرد. با توجه به خواسته‌ی صورت پروژه شبکه به گونه‌ای برنامه‌ریزی شد که در ثانیه‌ی ۰ کار خود را آغاز و پس از ۱۰۰۰ ثانیه آن را متوقف کند.

- لازم به ذکر است پس از پایان شبیه‌سازی procedure finish به منظور بستن فایل‌های trace و nam و اجرای فایل انیمیشنی شبیه‌سازی(در صورت نیاز) و همینطور خروج، فراخوانی خواهد شد.

۷. در انتها نیز برای اجرای شبیه‌سازی نیاز است که کد مورد نظر run شود که دستور $ ns run این کار را انجام خواهد داد.

رسم نمودارهای مورد نیاز

برای رسم نمودار متغیرهای خواسته شده، نیاز است تا ابتدا هریک از سه فایل مربوط به TCP agentهای مربوطه، ده مرتبه اجرا شده و میانگین داده‌های آن‌ها در نظر گرفته‌شود. این داده‌های از parse کردن فایل trace به دست خواهد آمد. لازم به ذکر است از آن‌جایی که زمان‌ها در هر یک از اجراها با هم منطبق نیستند، هر ثانیه را به عنوان داده در نظر گرفته و آخرین مقدار موجود در یک ثانیه را ذخیره کرده و میانگین را روی آن اعمال کردیم. فایل analyzer.py اجرای هریک از شبیه‌سازی‌ها برای ده مرتبه، محاسبه‌ی میانگین‌های مورد نظر و هم‌چنین رسم نمودارها را به طور کامل انجام خواهد داد.

- با توجه به توضیحات فوق برای اجرای پروژه کافیست فایل analyzer.py اجرا شود، تا نمودارهای مربوطه نمایش داده‌شود.

  هم‌چنین برای مشاهده‌ی شبیه سازی، اجرای دستور nam newreno.nam/vegas.nam/tahoe.nam در ترمینال کافیست.

تجزیه فایل trace جهت دریافت اطلاعات و رسم نمودار

به صورت کلی در فایل trace دو نوع سطر وجود دارد:

1. Variable-trace information

   که اطلاعات موجود در آن‌ها به ترتیب زیر می‌باشد.

   1. زمان
   2. Node مبدا جریان
   3. Port مبدا
   4. Node مقصد جریان
   5. Port مقصد
   6. نام متغیر trace
   7. مقدار متغیر trace

2. Event record

   که اطلاعات موجود در آن‌ها به ترتیب زیر می‌باشد.

   1. r نشان‌دهنده دریافت بسته، d نشان‌دهنده از دست رفتن بسته، + نشان‌دهنده وارد صف‌شدن بسته و - نشان‌دهنده خارج از صف شدن بسته است.
   2. زمان با واحد ثانیه

3. شماره نود منبع. برای مثال اگر فیلد اول d باشد به معنی آن است که بسته در این نود از دست رفته‌است.
4. شماره نود مقصد. برای مثال اگر فیلد اول r باشد به معنی آن است که بسته توسط این نود دریافت شده‌است.
5. نام پروتکل(مثلا tcp)
6. سایز بسته برحسب بایت
7. تعدادی پرچم مربوط به TCP. در اینجا چون این پرچم‌ها مقداردهی نشده‌اند، با "-" نشان داده می‌شوند.
8. شناسه جریان که برای هر جریان شبکه می‌توانیم تعریف کنیم. برای مثال شناسه جریان از نود ۱ به ۵ را ۱ و شناسه جریان از نود ۲ به ۶ را ۲ قرار دادیم.
9. نود مبدا جریان که بافرمت <پورت نود، شماره نود> مشخص شده‌است.
10. نود مقصد جریان که با همان فرمت قسمت گفته‌شده مشخص می‌شود.
11. شماره توالی بسته.
12. شناسه بسته. زمانی که بسته وارد یک لینک جدید می‌شود به آن یک شناسه جدید اختصاص داده می‌شود اما شماره توالی تغییر نمی‌کند.(برای شبیه‌سازی کاربرد دارد)

- **سایز پنجره**:

متغیر trace موجود در فایل trace با نام cwnd_ اطلاعات مربوط به تغییر سایز پنجره را در اختیارمان می‌گذارد. این اطلاعات به صورت سطرهایی مشابه تصویر زیر در فایل قرار دارند:

```
0.22019  0  0  4  0  cwnd_ 4.000
```

همانطور که ترتیب داده‌ها در ابتدای این بخش بیان شد، برای سایز پنجره تنها به ستون اول(یعنی زمان) و ستون آخر(یعنی سایز جدید پنجره‌ی مورد نظر) نیاز داریم. میانگین نهایی این اطلاعات برای هر یک از جریان‌های Nodeهای 0 به 4 (در شبیه‌سازی شماره‌ی nodeها از صفر آغاز می‌شود) و 1 به 5 در یک دیکشنری ذخیره خواهد شد که شامل داده‌های مربوط به هر سه نوع TCP agent می‌باشد.

- **نرخ Goodput :**

Goodput در شبکه به تعداد بسته‌های دریافت شده توسط node مقصد اطلاق می‌شود. به این منظور نیاز است که تعداد acknowledgment های ارسالی توسط nodeهای مقصد در نظر گرفته شود. Trace variable با نام ack این اطلاعات را در اختیارمان قرار می‌دهد. این متغیر به صورت تجمعی نشان‌دهنده‌ی

تعداد بسته‌های دریافتی از لحظه‌ی صفر تا آن لحظه‌ی مورد نظر است. برای محاسبه‌ی نرخ Goodput نیاز است که در هر لحظه تعداد ackهای دریافتی بر زمان آن تقسیم شود.

```
0.22019  0  0  4  0  ack_ 2
```

تصویر فوق سطر مربوط به متغیر ack در فایل trace است که بیان می‌کند در زمان s0.22019 مقدار این متغیر به 2 تغییر یافته است.

- **نرخ Packet loss:**

طبق تعریف، تعداد بسته‌های از دست رفته از تفاوت تعداد بسته‌های ارسالی و تعداد بسته‌های دریافتی به دست می‌آید. از آنجایی که در فایل‌های trace با فرمت استاندارد تگ‌های s(برای نشان دادن بسته‌های send شده) وجود ندارد[1]، نمی‌توان تعداد بسته‌های ارسالی را به صورت دقیق محاسبه کرد.

اما از طرفی طبق توضیحات ابتدای بخش بیان شد که سطرهای با تگ d در event record ها نشان‌دهنده‌ی بسته‌های drop شده می‌باشند. Drop شدن بسته‌ها علاوه بر overflow شدن صف روترها ممکن است به دلیل خطا، دوباره فرستادن بسته پیش از دریافت ack آن و ... نیز رخ دهد. اما در این پروژه فرض شده drop شدن بسته‌ها به دلیلی غیر از overflow شدن صف، حداقل میزان است و packet loss بر مبنای سطر event با تگ d محاسبه می‌شود.

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

تصویر فوق نشان‌دهنده‌ی یکی از این قبیل سطرهاست. همانطور که در ابتدای بخش تعریف شد هر ستون متعلق به چه متغیری است، برای محاسبه‌ی packet loss نیاز است که زمان(ستون دوم)، node مبدا(ستون نهم) و node مقصد(ستون دهم) را داشته‌باشیم.

- **نرخ RTT:**

RTT به فاصله‌ی میان زمان ارسال بسته تا دریافت acknowledgment آن اطلاق می‌شود. به این منظور هر یک از TCP agentها خود یک trace variable با نام rtt دارند که با لحاظ کردن آن زمان تغییر یافتن مقدار rtt در TCP در فایل trace لحاظ خواهد شد.

```
0.02841  0  0  4  0  rtt_ 0.020
```

همانطور که در ابتدای بخش توضح داده شد، سطر مربوط به rtt مطابق تصویر فوق است. بر این اساس مقدار ستون آخر و زمان در ستون اول، مقادیر مورد نیاز ما برای رسم نمودار مربوط به rtt خواهد بود.

[1] http://nsnam.sourceforge.net/wiki/index.php/NS-2_Trace_Formats

انواع TCPها

TCP Tahoe

TCP tahoe سه خاصیت slow-start،congestion avoidance و الگوریتم fast retransmit را به پروتکل TCP اضافه کرده است. در صورتیکه بازه‌ی زمانی مدنظر tahoe برای دریافت ack از سمت node گیرنده پایان یابد و یا آنکه 3 بار duplicate ack پشت سر هم دریافت کند، این اتفاق را به عنوان packet loss تشخیص می‌دهد. بلافاصله پس از تشخیص packet loss، این tcp فاز fast retransmit خود را آغاز می‌کند که در آن فرستنده ابتدا بسته‌ی loss شده را مجددا ارسال می‌کند، سپس مقدار ssthresh -همان slow-start threshold- خود را به نصف اندازه‌ی فعلی پنجره مقداردهی کرده و در نهایت فاز slow-start را با اندازه‌ی پنجره‌ی 1 شروع می‌کند. در این فاز فرستنده اندازه‌ی پنجره‌ی خود را به صورت خطی به ازای هر ack جدیدی که دریافت می‌کند، زیاد می‌کند(با سرعت نمایی).

به صورت کلی افزایش اندازه‌ی پنجره به صورت خطی و کاهش آن به صورت نمایی در مواجهه با packet loss از ویژگی‌های tahoe به شمار می‌رود. به طور کلی tahoe برای تشخیص congestion وابسته به مشاهده packet loss است.

TCP New Reno

TCP newReno فاز slow-start،congestion avoidance و الگوریتم fast retransmit-recovery مشابه با TCP Reno دارد. با آغاز فاز slow-start در این tcp، فرستنده با هر ack، اندازه‌ی پنجره‌ی خود را یک واحد زیاد می‌کند تا زمانی که به میزان ssthresh برسد. با رسیدن اندازه‌ی پنجره به این مقدار، فاز congestion avoidance تمام خواهد شد. این tcp مشابه tahoe با دریافت 3 تا duplicate ack متوالی، تشخیص می‌دهد که packet loss رخ داده است و وارد فاز fast retransmit می‌شود. در این فاز اندازه‌ی پنجره و ssthresh هر دو به میزان نصف اندازه‌ی پنجره‌ی فعلی مقداردهی می‌شوند. New Reno منتظر می‌ماند تا ack تمامی بسته‌هایی که در مسیر هستند دریافت شود. با این کار از duplicate retransmit یعنی نصف کردن مجدد اندازه‌ی پنجره در صورت مشاهده‌ی یک packet loss دیگر در فاز fast retransmit جلوگیری می‌کند. New Reno مشابه tahoe وابسته به دیدن packet loss برای تشخیص congestion می‌باشد.

TCP Vegas

Vegas بهتر از پیاده‌سازی‌های دیگر tcp عمل می‌کند و دلیل این موضوع آن است که سعی می‌کند ترافیک را قبل از آنکه بسته‌ای گم شود تشخیص دهد. این پیاده‌سازی بیشتر از آنکه ترافیک را با گم شدن بسته‌ها تشخیص دهد، ترافیک را با مقدار rtt تشخیص می‌دهد. به طور کل سه ویژگی متفاوت نسبت به پیاده‌سازی‌های قبلی دارد:

۱) vegas (زمان فرستاده شدن بسته‌ها را ثبت می‌کند. همچنین یک تخمین از rtt هم نگه می‌دارد. هنگامی که یک duplicate ack می‌آید چک می‌کند اگر از زمان فرستاده شدن تا زمان آمدن duplicate ack بیشتر از rtt تخمینی باشد بسته را دوباره می‌فرستد و به این شکل منتظر سه بار duplicate ack نمی‌ماند.

۲) برای تعیین اندازه پنجره برخلاف پیاده‌سازی‌های دیگر براساس مقدار rtt کار می‌کند. که فرمول آن را در زیر مشاهده می‌کنید.

$$Cwnd\ (t+t_A) = \begin{cases} cwnd(t)+1, & \text{if } diff < \alpha/base\_rtt \\ cwnd(t), & \text{if } \alpha/base\_rtt < diff < \beta/base\_rtt \\ cwnd(t)-1, & \text{if } \beta/base\_rtt < diff \end{cases}$$

$$diff = cwnd(t)/base\_rtt - cwnd/rtt$$

که base_rtt کمترین مقدار rtt تا به حال و rtt، مقدار واقعی rtt است.

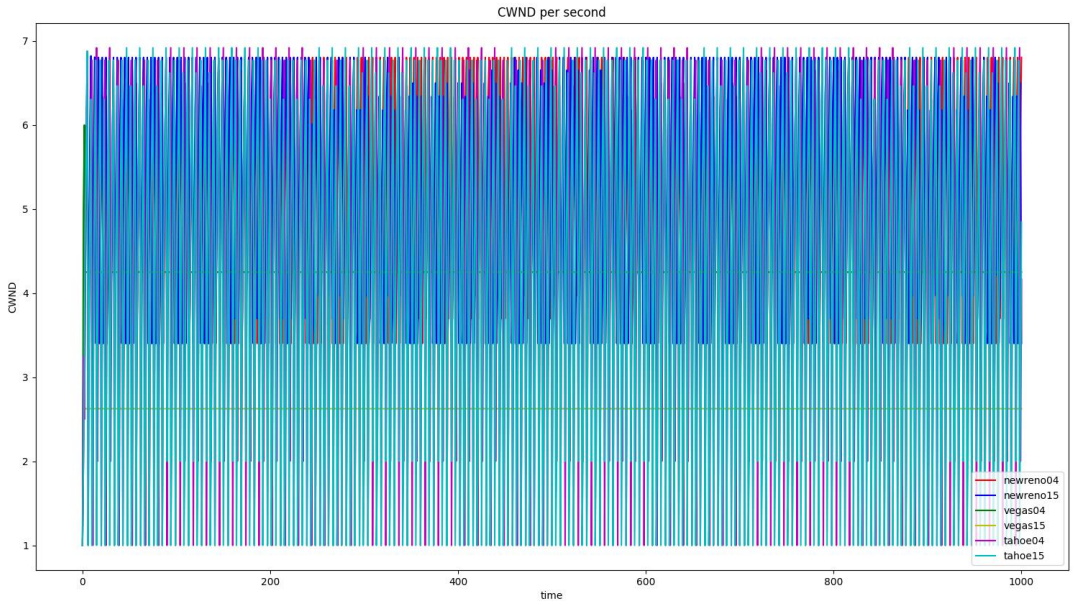
۳) در فاز slow start در هر rtt مقدار گذردهی لینک سنجیده می‌شود و به صورت نمایی زیاد می‌شود تا جایی که تفاوت این مقدار با مقداری که انتظار می‌رود از آستانه‌ای می‌گذرد که در این حالت وارد فاز congestion avoidance می‌شود.

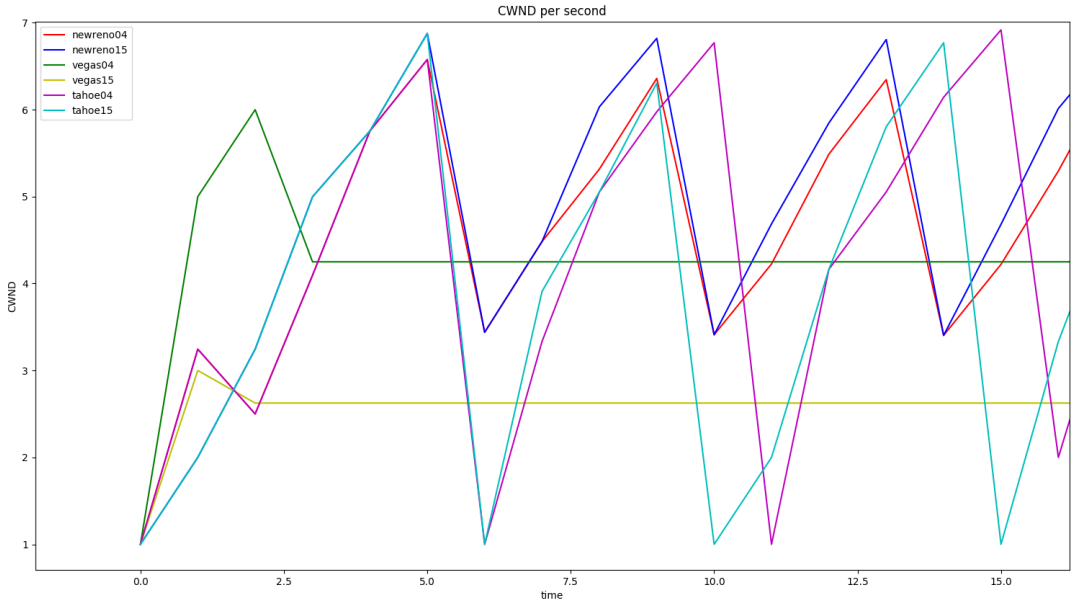
• نمودارهای خروجی

نکته: فرض کرده‌ایم که ده بار اجرا شدن شبیه‌سازی به ازای هر کدام از tcp ها و مدت 1000ثانیه با تاخیرهای متغیر یکسان صورت می‌گیرد(با مقداردهی seed در تولید عدد رندوم به مقداری غیر از 0، عدد رندوم یکسان با هر بار اجرا تولید خواهد شد.) با توجه به این توضیح، تاخیر متغیر لینک بین node های 2و 3 (روتر اولی) در این نمودارها برابر s5.01 و تاخیر لینک بین nodeهای 4 (روتر دومی) و 6، برابر s7.24 می‌باشد.

نکته: برای بهتر نشان داده‌شدن رفتار TCP ها، از محاسبه‌ی نرخ به صورت تجمعی در Goodput و Packet loss استفاده کرده‌ایم. داده‌های مربوط به این دو متغیر در هر زمان به صورت تجمعی ذخیره شده است و برای محاسبه‌ی نرخ، مقدار متغیر در هر ثانیه بر زمان آن لحظه تقسیم شده است. این نوع نشان دادن کمک می‌کند تا رفتار TCP و نوع کنترل ازدحام شبکه توسط آن در طول زمان بهتر نشان داده شود. با این حال نمودار نرخ Goodput و Packet loss در هر لحظه(یعنی مقدار هر متغیر در همان لحظه) در انتهای بخش مربوط به خودشان نشان داده شده است. طبیعتا اندازه‌ی پنجره و RTT از آنجایی که مقدار در لحظه محسوب می‌شوند، به صورت لحظه‌ای نشان داده شده است.

- **اندازه‌ی پنجره**

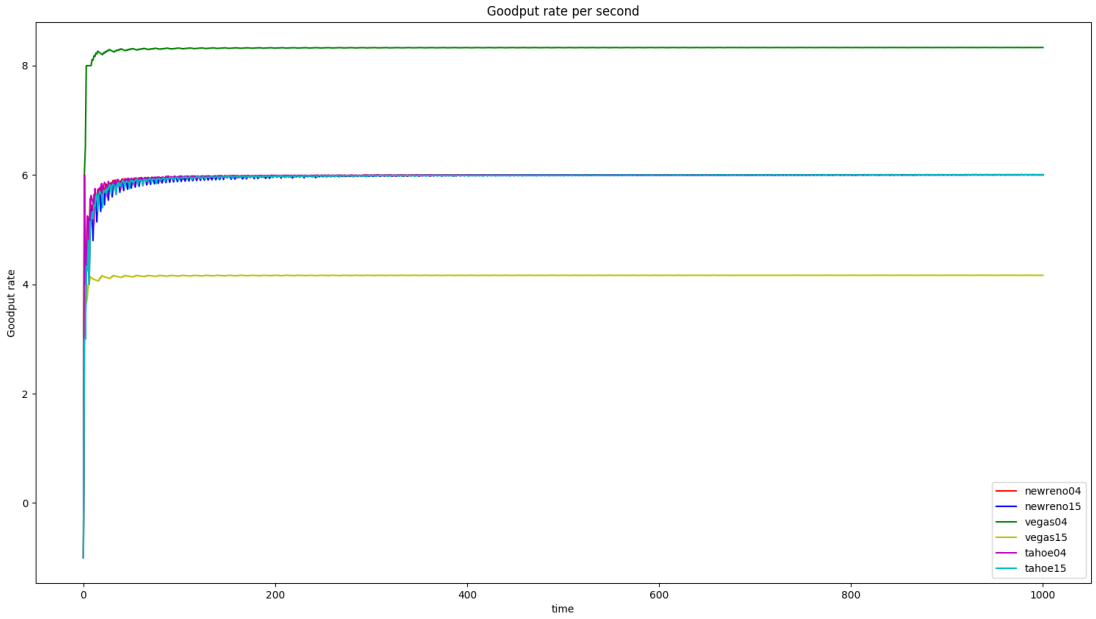


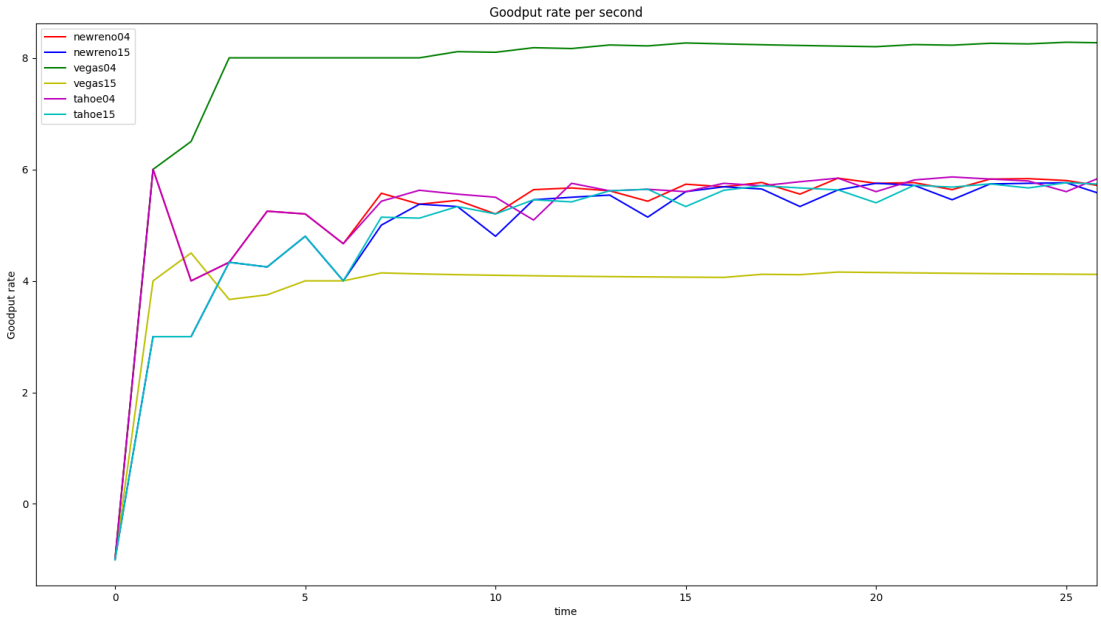


دو تصویر فوق نمودارهای مربوط به تغییرات اندازه‌ی پنجره می‌باشد. (تصویر دوم زوم شده‌ی تصویر اول است.) همانطور که درباره‌ی tcpهای مختلف توضیح داده شد، واضح است که tahoe TCP با مشاهده‌ی packet loss اندازه‌ی پنجره‌ی آن بلافاصله برابر یک خواهد شد و سپس به صورت نمایی زیاد خواهد شد تا به مقدار ssthresh برسد؛ با رسیدن به این مقدار، شیب آن تغییر خواهد کرد و به صورت خطی اندازه‌ی پنجره‌ی آن افزایش خواهد داشت.(نمودارهای صورتی و فیروزه‌ای) این درحالیست که New Reno TCP در صورت مواجهه با network congestion اندازه‌ی پنجره‌ی خود را به نصف کاهش می‌دهد.(نمودارهای قرمز و آبی)

در نهایت TCP vegas به دلیل توانایی قابل توجه در تشخیص زودهنگام network congestion بدون توجه به packet loss و افزایش اندازه‌ی پنجره مبنی بر میزان RTT (دو tcp دیگر بر اساس ackهای دریافتی اندازه‌ی پنجره‌ی خود را افزایش می‌دادند.) حد بالای کمتری نسبت به دو tcp دیگر دارد و هم‌چنین از یک زمانی به بعد دیگر مقدار آن تغییر نخواهد کرد که به نظر می‌رسد با ورود به فاز congestion avoidance به یک مقدار قابل اطمینان و تخمین خوب برای شبکه رسیده‌است که طی آن با مشاهده و بررسی مقادیر جدید RTT نیازی به تغییر اندازه‌ی پنجره وجود نداشته است.

- نرخ Goodput

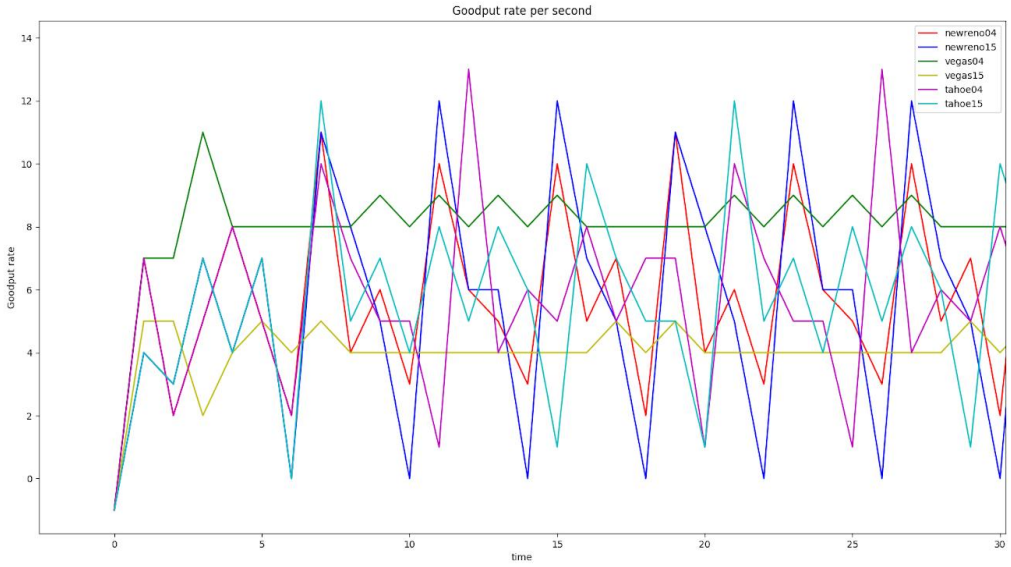




دو تصویر فوق نمودارهای مربوط به نرخ Goodput می‌باشد. (تصویر دوم زوم شده‌ی تصویر اول است.) همانطور که درباره‌ی TCP vegas توضیح داده شد،تغییرات منطقی این TCP در فاز slow start و هم‌چنین congestion avoidance آن موجب می‌شود که packet loss کمتر و در نتیجه تعداد ack بیشتری برای بسته‌های ارسالی دریافت کند و در نهایت در هر لحظه Goodput بیشتری داشته باشد که منجر به بیشتر بودن نرخ تغییرات Goodput نیز خواهد شد.(نمودار سبز).

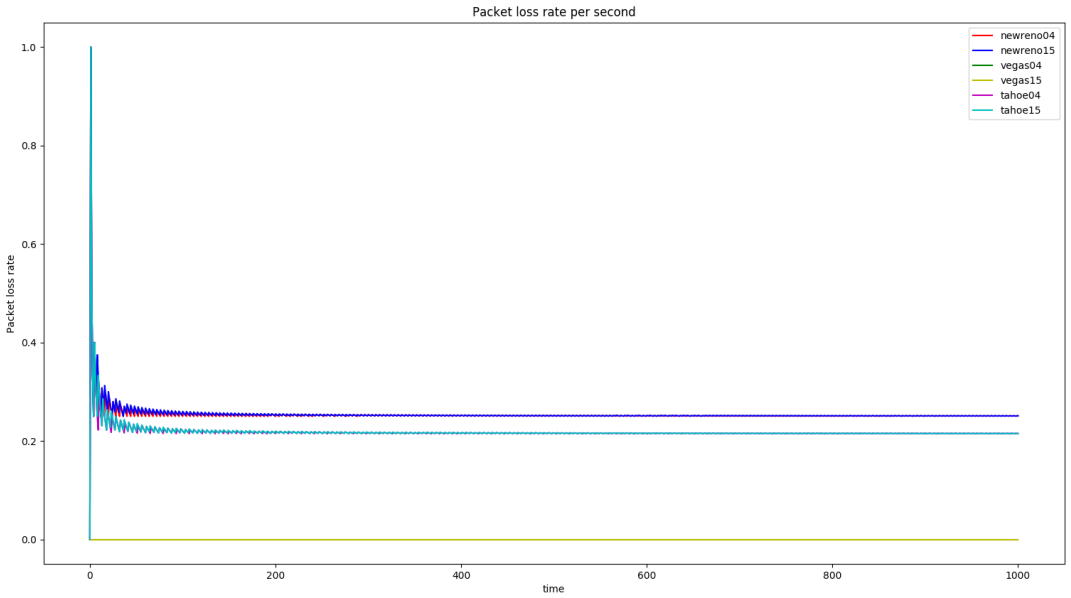
دلیل تفاوت میزان نرخ Goodput در دو نمودار مربوط به مسیرهای nodeهای 0 به 4 و 1 به 5 در TCP Vegas آن است به دلیل وجود تاخیر رندوم و مقادیر تاخیر در نظر گرفته شده برای لینک‌ها،مسیر 1به5 در مجموع تاخیر بیشتر و به واسطه‌ی آن میزان RTT بیشتری برای دریافت بسته و فرستادن ack خواهد داشت. از آن‌جایی که TCP Vegas بر مبنای RTT کار می‌کند با مشاهده بیشتر بودن اندازه‌ی RTT، اندازه‌ی پنجره‌ی آن را کاهش می‌دهد و بدین ترتیب تعداد بسته‌های کمتری در هر مرحله از node 1 به 5 در مقایسه با جریان دیگر ارسال خواهد شد. مطابق توضیحات ابتدای این بخش از آنجایی که packet loss رخ نمی‌دهد افزایش Goodput به صورت خطی خواهد بود اما شیب آن از جریان 0 به 4 کمتر خواهد بود.(از نمودار اندازه‌ی پنجره تفاوت CWND این دو جریان نیز مشهود است.) (نمودار زرد)

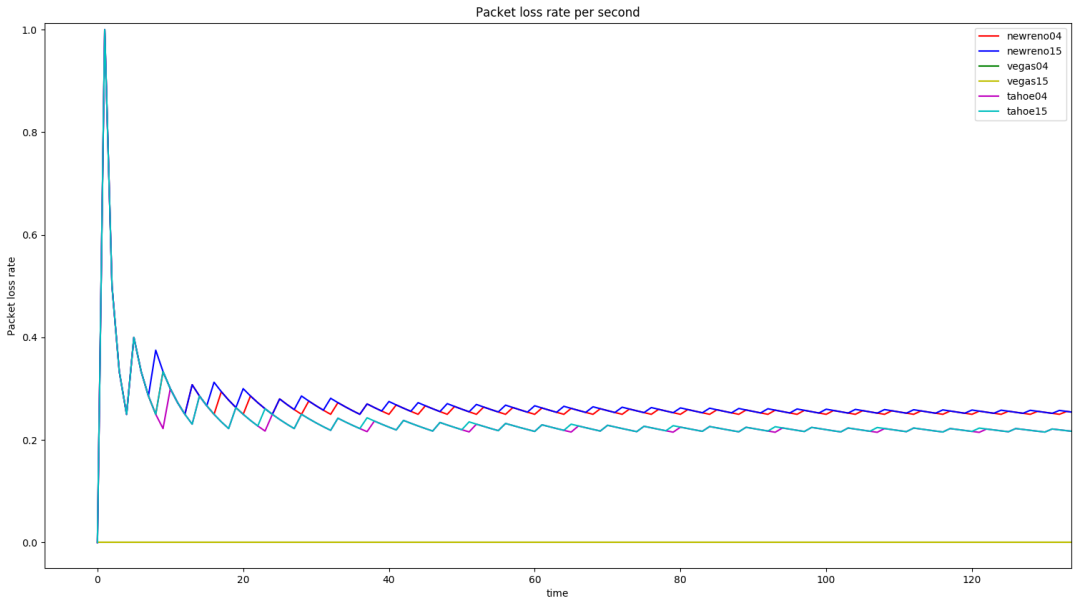
در دو پیاده‌سازی دیگر که اندازه پنجره در نوسان است، حد بالای پنجره هر دو تقریبا یکسان است. همچنین از آنجایی که در tahoe از اندازه پنجره 1 تا ssthresh به صورت نمایی(سریع) زیاد می‌شود در حالی که در همین زمان newreno به صورت خطی(کندتر) از ssthresh خود رشد می‌کند. و این باعث می‌شود به طور میانگین اندازه پنجره‌شان یکسان بوده و در نتیجه تعداد بسته‌های فرستاده نیز یکسان باشد. همچنین حدود نرخ گم شدن بسته در دو پیاده‌سازی نیز حدودا یکسان است(به علت حد بالای پنجره یکسان و سیاست مشابه). در نتیجه به طور میانگین نرخ goodput این دو پیاده‌سازی یکسان شده‌است.



نمودار نرخ لحظه‌ای Goodput

- نرخ Packet Loss

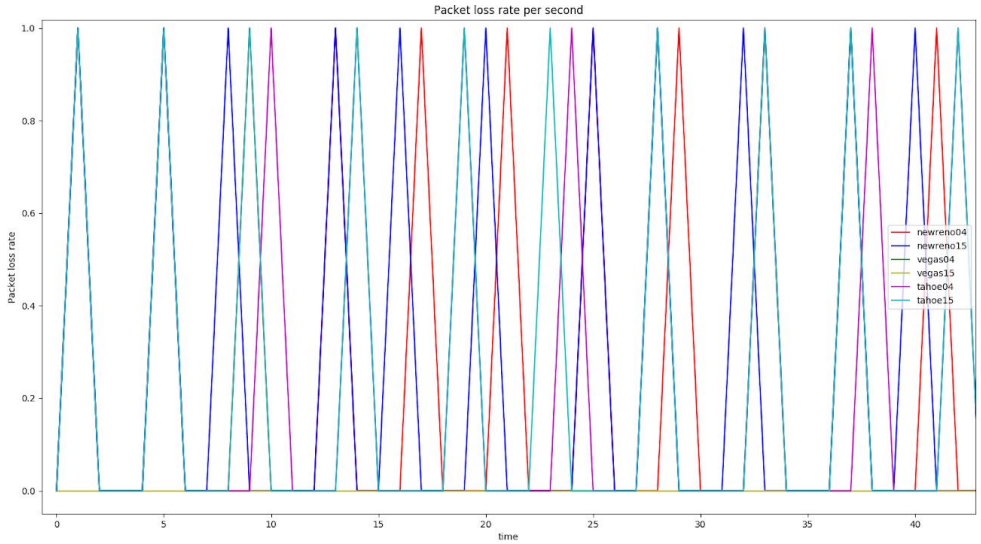




دو تصویر فوق نمودارهای مربوط به نرخ Packet loss می‌باشد. (تصویر دوم زوم شده‌ی تصویر اول است.) در TCP vegas همانطور که تاکید شد سیاست congestion avoidance مستقل از packet loss باعث می‌شود که هیچ بسته‌ای از دست نرود و همواره اندازه‌ی پنجره بر اساس مقدار RTT به گونه‌ای باشد که مانع ایجاد ازدحام و از دست رفتن بسته در شبکه شود. به همین دلیل مشهود است که نرخ packet loss در هر دو جریان متعلق به TCP Vegas در بازه‌ی زمانی صفر تا 1000، صفر است.(دو نمودار سبز و زرد روی هم قرار گرفته‌اند)

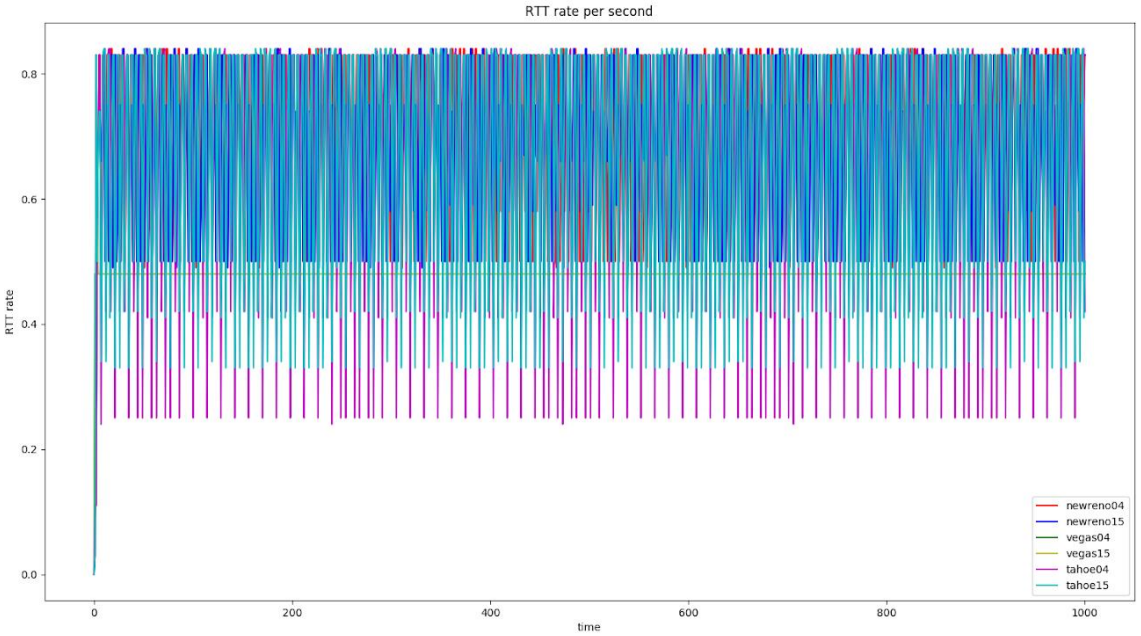
از آنجایی در فاز slow start دو پیاده‌سازی tahoe و newreno بر مبنای packet loss اندازه پنجره‌ی خود را تعیین می‌کنند و در ابتدا میزان بسته‌های ارسالی را آنقدر زیاد می‌کنند تا packet loss رخ دهد(و بتوانند اندازه پنجره

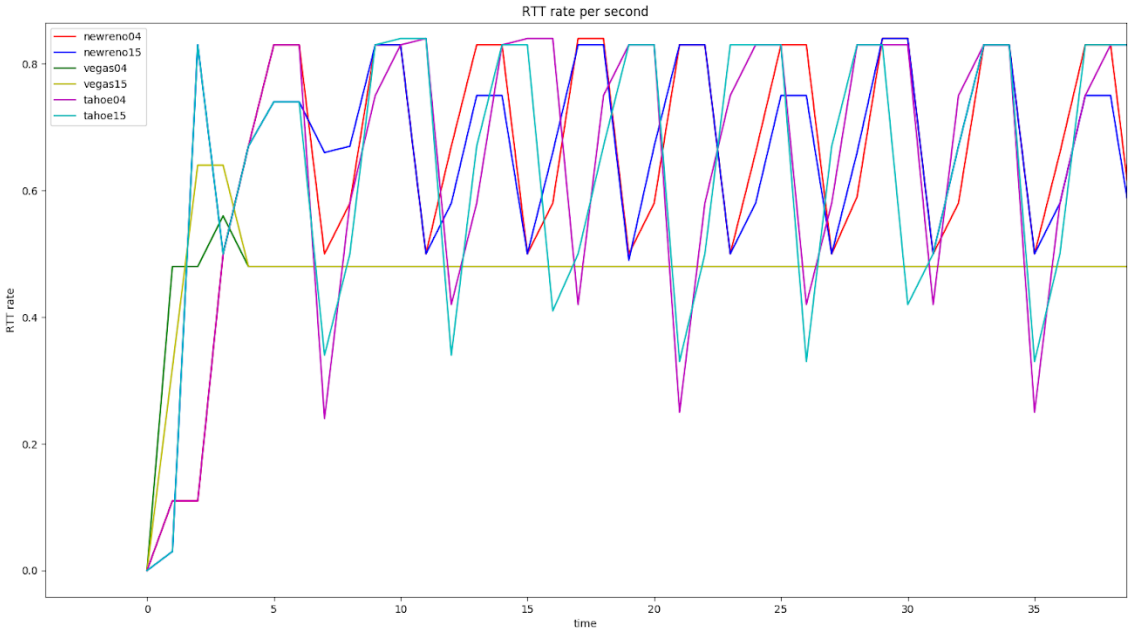
را تنظیم کنند)، در شبیه‌سازی در ابتدا نرخ گم شدن بسته‌ها زیاد شده است که دلیل آن شلوغی شبکه است. اما بعد از تنظیم اندازه پنجره این نرخ کاهش یافته و در نهایت مانند اندازه پنجره در بازه مشخصی نوسان می‌کند. همچنین همان‌طور که ذکر شد چون به طور میانگین اندازه پنجره دو پیاده‌سازی یکسان بوده و تقریبا سیاست‌های مشابهی اتخاذ می‌کنند، نرخ packet loss این دو پیاده‌سازی نیز تقریبا مشابه شده‌است.



نمودار نرخ لحظه‌ای Packet Loss

- نرخ RTT





دو تصویر فوق نمودارهای مربوط به نرخ RTT rate می‌باشد. (تصویر دوم زوم شده‌ی تصویر اول است.) به طور کلی هر سه TCP با شروع شبیه‌سازی، اندازه‌ی پنجره‌ی خود را زیاد می‌کردند تا به مقدار بهینه دست یابند، بدیهی است با افزایش اندازه‌ی پنجره، تعداد بسته‌های موجود در شبکه زیاد شده باعث ایجاد ازدحام و افزایش میزان RTT می‌شود. پس از آن همانطور که سیاست هریک ازTCP ها شرح داده شد، TCP Tahoe با دیدن packet loss اندازه‌ی پنجره‌ی خود را به یک تغییر میداد، بدین ترتیب میزان RTT به صورت چشم‌گیری کم خواهد شد که وجود دره‌هایی در نمودار مربوط به TCP Tahoe ناشی از همین دلیل خواهد بود. پس از این اتفاق، اندازه‌ی پنجره مجددا زیاد خواهد شد که به تبع آن باعث افزایش شلوغی شبکه و زیاد شدن اندازه‌ی RTT نیز می‌شود.(نمودارهای صورتی و آبی فیروزه‌ای)

در TCP NewReno نیز حالتی مشابه TCP Tahoe اتفاق می‌افتد با این تفاوت که NewReno در مواجهه با packet lossاندازه‌ی پنجره‌ی خود را به جای یک، به نصف اندازه‌ی فعلی تقلیل می‌دهد. به همین دلیل دره‌هایی مشابه Tahoeدر نمودار خود دارد اما minimum مقدار آن به اندازه‌ی Tahoe نیست. پس از کاهش اندازه‌ی پنجره، مجددا اندازه‌ی آن به مرور زیاد خواهد شد. که خود منجر به شلوغ‌تر شدن شبکه و افزایش میزان RTT خواهد شد.(نمودار قرمز و آبی)

این در حالیست که در TCP Vegas از آن‌جایی که مدیریت و پیشگیری از ازدحام بسیار قوی و خوب انجام می‌شود از نقطه‌ای به بعد مقدار RTT ثابت مانده و بدون از دست رفته بسته در این بازه‌ی زمانی، کار خود را انجام می‌دهد.(نمودار زرد و سبز)

نتیجه‌گیری

بر اساس شبیه‌سازی انجام شده و نمودارهای به دست آمده این نتیجه حاصل می‌شود که در مدیریت ازدحام شبکه همواره هر چهار عامل اندازه‌ی پنجره، RTT، میزان از دست رفتن بسته و همچنین میزان Goodput بسیار به هم وابسته بوده و روی هم اثرگذار هستند. البته که سیاست TCP موجود در شبکه نیز بسیار حائز اهمیت است و با توجه به TCPهای موجود در این پروژه، این نتیجه حاصل می‌شود که TCP Vegas به نسبت از همه‌ی TCP ها مفیدتر بوده و سیاست‌های بهتری برای مواجهه با network congestion دارد که منجر به عملکرد بهتر آن شده است.