به نام خدا

گزارش پروژه چهارم درس شبکه های کامپیوتری

شبیه سازی NS2

علی بهاری
810196688

امیرعلی رایگان
810197623

بخش اول - تعریف و توضیح سه نوع TCP

- **نوع اول - Cubic :**

در حال حاضر مدل Cubic در سیستم عامل های لینوکس، مک او اس و ویندوز به عنوان مدل پیش فرض به کار برده می شود. این الگوریتم برای جلوگیری از ازدحام شبکه استفاده می شود. Cubic می تواند در برابر تاخیر ها زیاد و اتصال به پهنای باند های زیاد از طریق شبکه، سرعت و اطمینان بیشتری را تضمین کند. دو تفاوت اساسی بین این مدل و مدل های کلاسیک قدیمی تر برای کنترل ازدحام دیده می شود. اول اینکه در فاز پیشگیری از ازدحام اندازه ی پنجره ی ازدحام(Congestion Window) با توجه به cubic function ای که در اختیار داریم پیوسته بیشتر می شود. (به ترتیب concave و convex) این کار کمک میکند تا مشکلات مربوط به bandwidth حل شود.(به خصوص در بخش مربوط به convex)

تفاوت مهم دوم این است که الگوریتم شروع آهسته ی ترکیبی با استفاده از تاخیر ها موجب بیشتر شدن سرعت بزرگ شدن پنجره ی ازدحام می شود تا قبل از اینکه بزرگ شدن آن موجب overflow در صف شود از این اتفاق پیشگیری کند.

اندازه ی پنجره در cubic توسط فرمول زیر محاسبه می شود که در آن B ضریب کاهش ضربی(0.7)، Wmax اندازه ی پنجره درست قبل از آخرین کاهش ، T زمان گذشته از آخرین کاهش برای پنجره ، C ثابت مقیاس گذاری(0.4) و cwnd اندازه ی پنجره ی ازدحام در زمان فعلی می باشد.

$$cwnd = C(T-K)^3 + w_{max}$$

$$where\ K = \sqrt[3]{\frac{w_{max}(1-\beta)}{C}}$$

- **نوع دوم - Reno :**

این الگوریتم وقفه های انتقال مجدد (RTO) و همچنین ACK های دو تایی را به عنوان از دست رفتن پکیج در نظر میگیرد. حال Reno رفتار های مختلفی با ACK های تکراری می تواند داشته باشد. به عنوان نمونه اگر سه ACK تکراری دریافت بشود Reno د رجواب یک انتقال مجدد سریع (fast retransmit) انجام می دهد و از مرحله ی شروع آهسته (slow start) عبور میکند و پنجره ی ازدحام را نصف می کند، آستانه ی شروع آهسته (slow start threshold) را برابر با پنجره ی ازدحام جدید قرار می دهد و وارد مرحله ی بازیابی سریع (fast recovery) می شود.

در این الگوریتم اگر زمان ACK به پایان برسد (RTO timeout) از شروع آهسته (slow start) استفاده می شود و پنجره ی ازدحام به 1 MSS کاهش می یابد.

- **نوع سوم - YeAH :**

این الگوریتم که مخفف کلمات (Yet Another HighSpeed TCP) می باشد در حقیقت یک طراحی نوین برای بالانس کردن نیاز های متفاوت الگوریتم های جدید کنترل ازدحام می باشد. اهداف YeAH در به صورت زیر تعریف می شوند:

1. **High efficiency**
2. **Internal, RTT and Reno fairness**
3. **Resilience to link loss while keeping network elements load as low as possible**

در حالت کلی این الگوریتم در دو شکل کار میکند.

1. حالت سریع یا Fast mode : در این حالت هنگامی که اندازه ی پنجره ی ازدحام کم می شود و صف کوچک، مقدار ازدحام شبکه هم کم می شود در نتیجه با توجه به قانون تهاجمی HSTCP این الگوریتم پنجره ی ازدحام خود را بزرگ تر میکند تا موقعی که ازدحام خیلی بیشتر شود. (High Speed TCP یک پروتکل کننده ی ازدحام است که مشکل استفاده ی کامل از پهنای باند را حل میکند. این مشکل در بیشتر پروتکل های استاندارد tcp در شبکه های bandwidth delay product وجود دارد.)

2. حالت کند یا slow mode : زمانی که سطح ازدحام شبکه زیاد می شود و به معنای دیگر تعداد پکیج های درون صف بیشتر از threshold می شود الگوریتم YeAH وارد فاز کند می شود و با یک الگوریتم decongestion شبیه Reno کار میکند. نحوه ی محاسبه ی backlog در اینجا با الگوریتم Vegas صورت می گیرد.

$$Q = (RTT - BaseRTT) \cdot \frac{cWnd}{RTT}$$

برای تخمین میزان ازدحام شبکه نیز فرمول زیر کمک گرفته می شود.

$$L = \frac{RTT - BaseRTT}{BaseRTT}$$

روند اجرا و توضیحات

فایل تحویل داده شده دارای چندین فایل و پوشه می باشد که اطلاعات مرتبط به آن ها به شرح زیر می باشد:
پوشه cubic: در این پوشه کل فایل های مربوط به trace کردن cubic قرار دارد که شامل فایل trace مربوط به goodput و cwnd و یک فایل allTraces می باشد که از این فایل برای rtt و loss در ادامه استفاده خواهد شد.
پوشه reno: مانند توضیحات قبل می باشد ولی این بار فایل ها برای reno می باشند.
پوشه yeah: مانند قبل این بار برای yeah
پوشه photos: عکس های مربوط به نمودار های خواسته شده در این پوشه قرار دارند
فایل های reno.tcl cubic.tcl yeah.tcl که هر کدام شامل پیاده سازی شبکه گفته شده با الگوریتم کنترل ازدحام مربوطه می باشند همچنین در این فایل ها هر کدام نمودار های گفته شده برای cwnd و goodput را در xgraph رسم می کنند.
فایل loss_plotter.py برای رسم نمودار گفته شده در زمینه نرخ از دست رفتن بسته برای هر سه الگوریتم کنترل ازدحام استفاده می شود. فایل rtt_plotter نیز به همین شکل است فقط این نمودار نرخ rtt را رسم می کند.
فایل clearTraces.sh نیز برای پاک کردن کل trace ها در هر پوشه گفته شده استفاده می شود کافی است در terminal دستور bash clearTraces.sh را اجرا کنید. هدف از قرار دادن این فایل این بود که در دیباگ کردن بعضی اوقات نیاز به پاک کردن کل trace ها و تولید دوباره آن ها با دستور ns داشیم که این فایل کار ما را راحت تر می کرد.

۳ فایل reno.tcl cubic.tcl yeah.tcl در نحوه پیاده سازی شباهت زیادی دارند فقط در موارد کوچکی مانند انتخاب الگوریتم و ذخیره فایل ها در مسیر مد نظر و... تفاوت دارند که آن ها نیز به راحتی قابل تشخیص هستند. روند کلی پیاده سازی به شرح زیر است:
در ابتدا node های شبکه مد نظرمان را که ۶ تا می باشند مانند خطوط ۷۳ تا ۷۸ کد تولید می کنیم سپس اتصالات این نود ها را مانند خطوط ۸۸ تا ۹۴ کد ایجاد می کنیم که به دلیل این که اتصالات دو طرفه می باشند از duplex استفاده می کنیم سپس پهنای باند و تاخیر را نیز وارد می کنیم سپس نوع اتصال که ما از نوع droptail که مانند صف عادی است استفاده کردیم را وارد می کنیم. فقط queue-limit برای router ها را نیز که مقدار آن ۱۰ در نظر گرفته شده است را نیز وارد می کنیم. حال برای به درستی نمایش دادن همه موارد گفته شده در nam خطوط ۹۷ تا ۱۰۳ کد را قرار می دهیم. این دستورات فقط باعث نمایش درست موارد بالا در nam می شوند. حال برای tcp مبدا و مقصد را بعد از تعیین مشخصات گفته شده اش تعیین می کنیم خطوط ۱۰۷ تا ۱۲۱ کد به این کار اختصاص داده شده اند همچنین در این مرحله الگوریتم کنترل ازدحام مدنظر را با دستوری مانند خط ۱۱۳ مشخص می کنیم حال بعد از تمام این مراحل مبدا و مقصد را در خط ۱۲۳ به هم وصل می کنیم سپس ftp را به tcp ساخته شده وصل می کنیم که یک traffic generator می باشد از این ftp در ادامه برای run کردن شبیه سازی استفاده می کنیم. همین اعمال را برای transfer دیگری که داریم به عینه تکرار می کنیم در نهایت با دستوری مانند خط ۱۸۱ اعلام می کنیم که از زمان تعیین شده انتقال packet ها شروع شود. در نهایت بعد از ۱۰۰۰ ثانیه شبیه سازی را متوقف می کنیم.

در کد علاوه بر شبیه سازی دو نمودار مربوط به cwnd و goodput نیز با xgraph رسم می شوند برای این کار از دو proc با نام های plotWindow و plotGoodput استفاده می شود برای cwnd از متغیر cwnd_ که مقدار مدنظر ما را دارد استفاده می کنیم و برای goodput نیز با تعریف کلاسی که در ابتدای کد آورده شده تعداد بایت های را گرفته و با محاسبات انجام شده در خط ۲۱۳ مقدار goodput را به دست آورده ایم این مقادیر گفته شده در لحظات مختلف با فاصله ۰.۱ ثانیه به دست آمده اند که از کل این مقادیر برای رسم نمودار برحسب زمان استفاده می شود.
برای rtt و lossRate از دستور trace-all و tracevar استفاده شده که با قرار دادن اطلاعات در فایل allTraces.tr هر پوشه rtt در ارسال ها و زمان های مختلف را تولید می کند همچنین اتفاقات مختلف مرتبط با packet ها نیز در این فایل ذخیره می شوند. از این فایل برای رسم نمودار مربوط به نرخ rtt و loss در فایل های پایتون مربوطه استفاده می شود.
در فایل های پایتون با توجه به فرمت اطلاعات داخل allTraces.tr مقادیر مد نظر را رسم می کنیم در ابتدا به rtt می پردازیم فرمت یک خط مربوط به rtt به شکل زیر است:

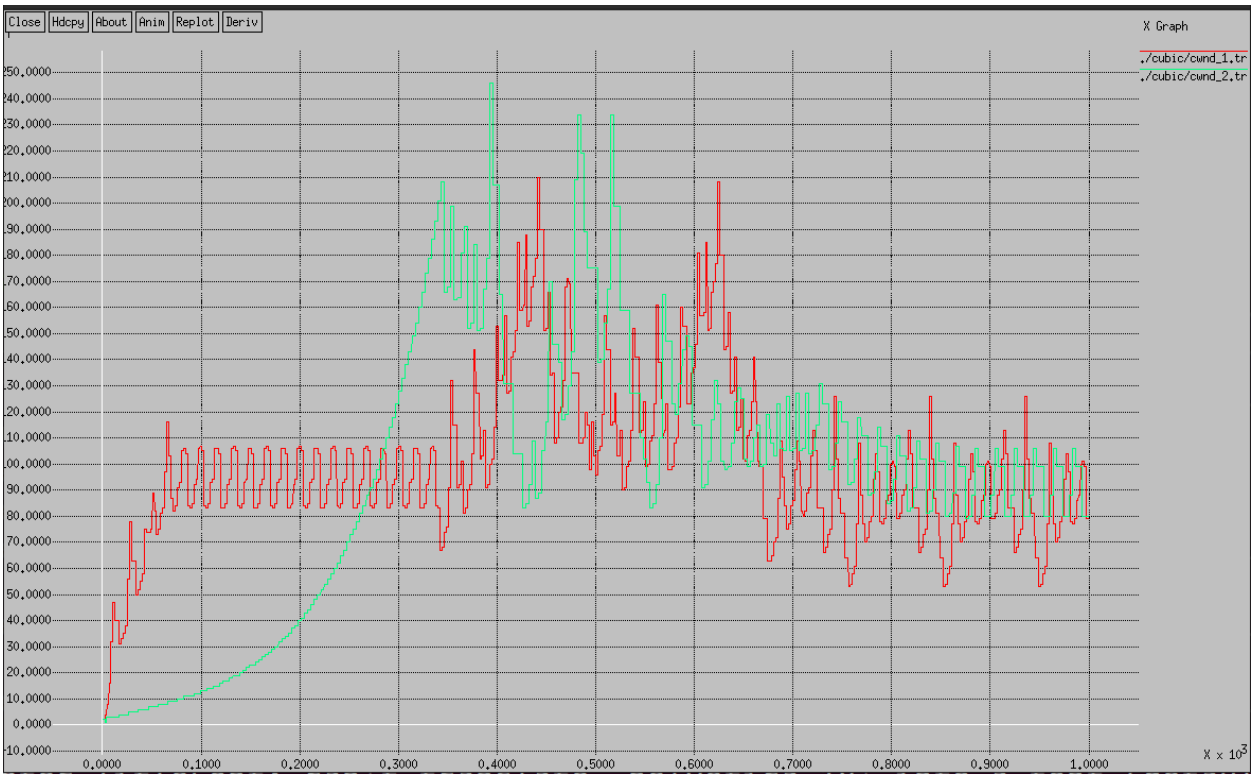
2.20000 0 0 4 0 rtt_ 2.100

از rtt_ متوجه می شویم که مربوط به rtt می باشد از اولین مقدار از سمت چپ زمان فعلی را متوجه می شویم که از این مقدار به عنوان مقادیر محور افقی نمودار استفاده می کنیم. اولین مقدار از سمت راست نیز آخرین مقدار rtt در آن لحظه می باشد که مقدار مد نظر ما می باشد از این مقدار به عنوان مقادیر محور عمودی نمودار استفاده خواهد شد همچنین چون دو ارسال داریم با استفاده از مقدار ستون دوم از چپ می تواند شماره کلاس ارسال که در tcl ثبت کرده بودیم را در نظر بگیریم و با توجه به آن مقادیر را ثبت کنیم. در کد rtt_plotter.py نیز روند به همین شکل است به سادگی این اعمال گفته شده انجام شده اند.
حال فرمت یک خط مربوط به loss به شکل زیر است:

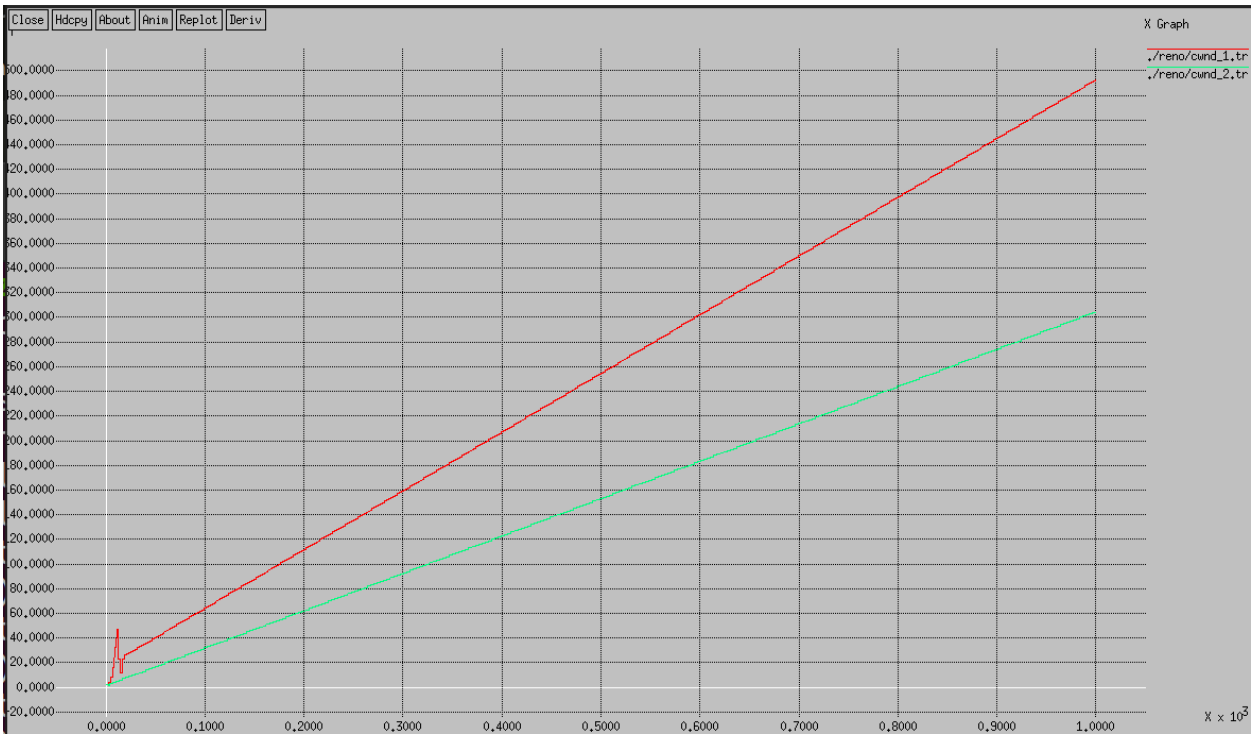
d 9.000119 2 3 tcp 1040 ------- 1 0.0 4.0 49 99

توجه به این نکته در این جا مهم است که سطر های با مقدار اول از چپ d نشان دهنده drop شدن packet می باشند حال drop شدن می تواند دلایل مختلفی داشته باشد که یکی از آن ها packet loss می باشد حال در این جا ما دلایل غیر از packet loss را در نظر نگرفته ایم پس می توان از همین سطر های گفته شده برای به دست آوردن packet loss استفاده کرد به این شکل که هر جا در اول سطر d دیدیم یک packet loss حساب کنیم.
حال چون دو ارسال داریم این بار از مقدار ستون چهارم از راست که نشان دهنده مبدا ارسال است استفاده می کنیم تا ارسال را تشخیص دهیم. این بار نیز زمان که مقدار ستون دوم از چپ است را برای محور افقی در نظر می گیریم حال هر بار d را دیدیم با توجه به مبدا ارسال به تعداد packet های از دست رفته اضافه می کنیم. در کد loss_plotter.py نیز روند به همین شکل است به سادگی این اعمال گفته شده انجام شده اند. در هر دو کد پایتون نمودار های مربوط به هر ۳ الگوریتم کنترل ازدحام رسم می شوند.

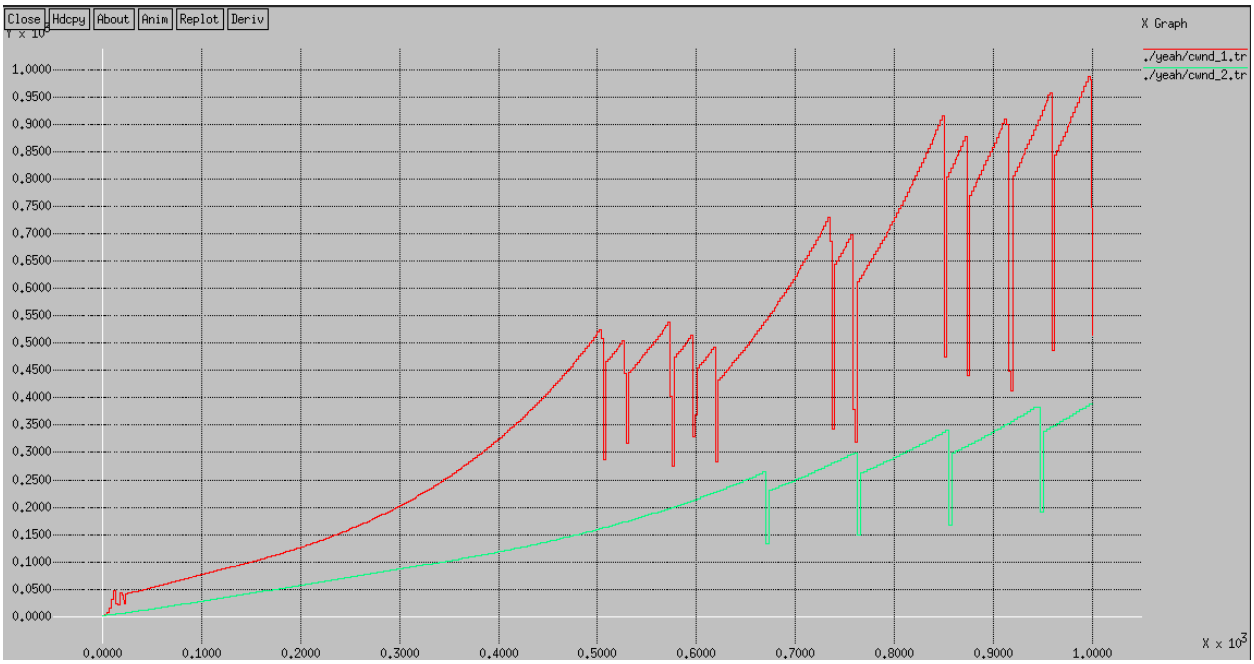
نمودار ها



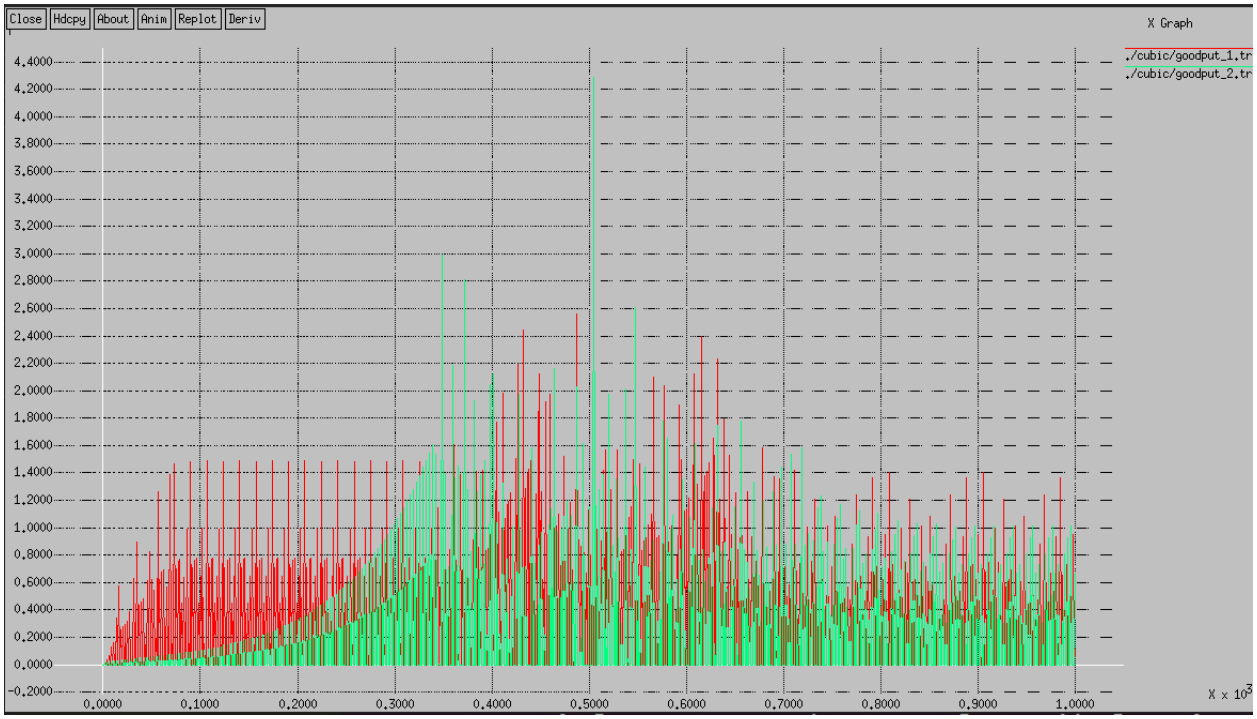
نمودار ۱: نمودار cwnd برای الگوریتم cubic در xgraph



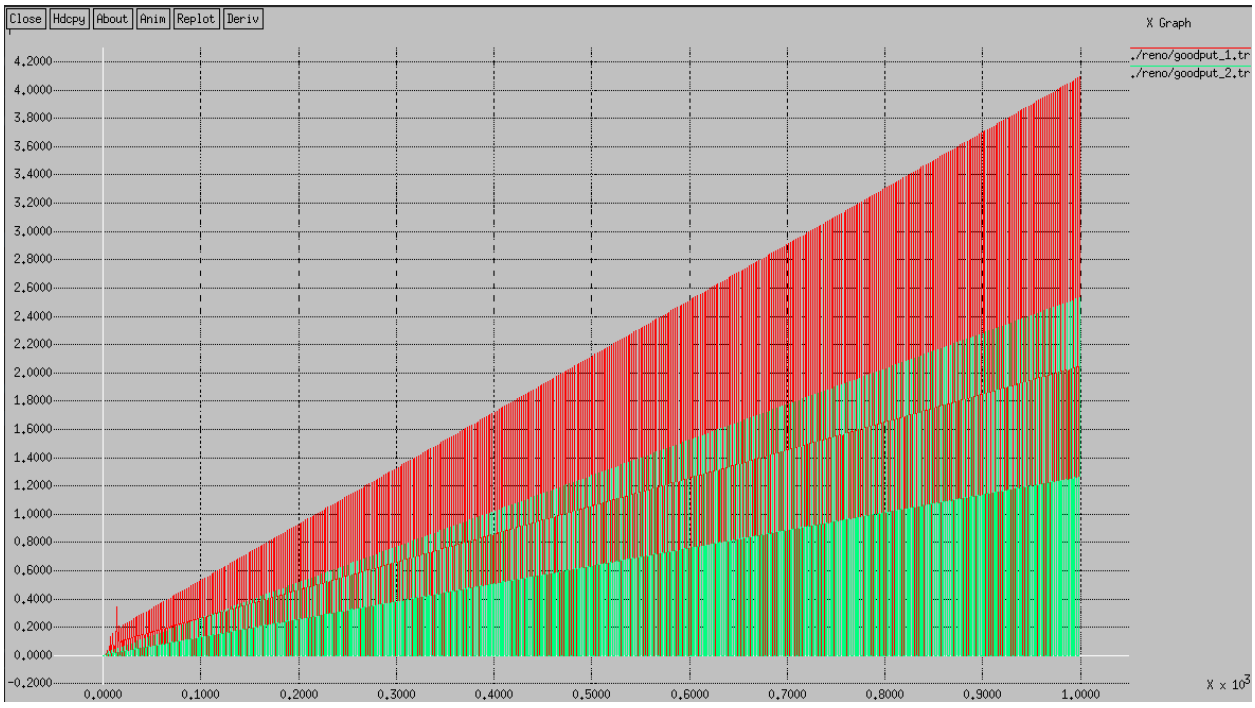
نمودار ۲: نمودار cwnd برای الگوریتم reno در xgraph



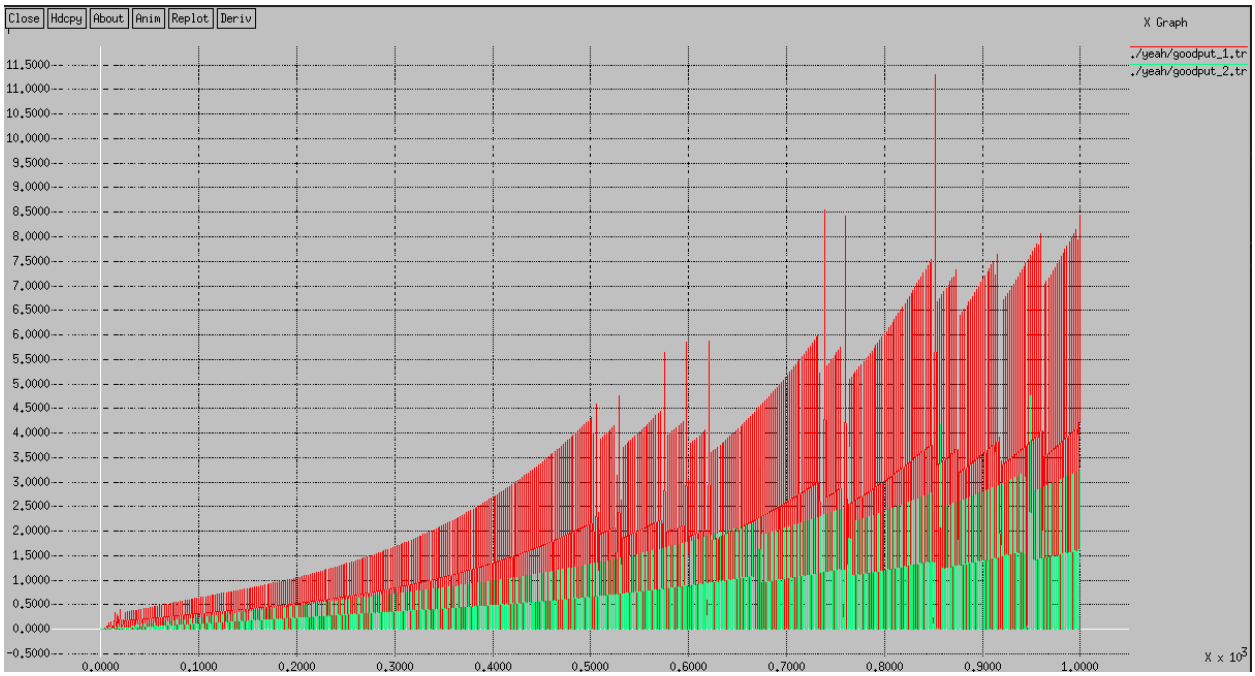
نمودار ۳: نمودار cwnd برای الگوریتم yeah در xgraph



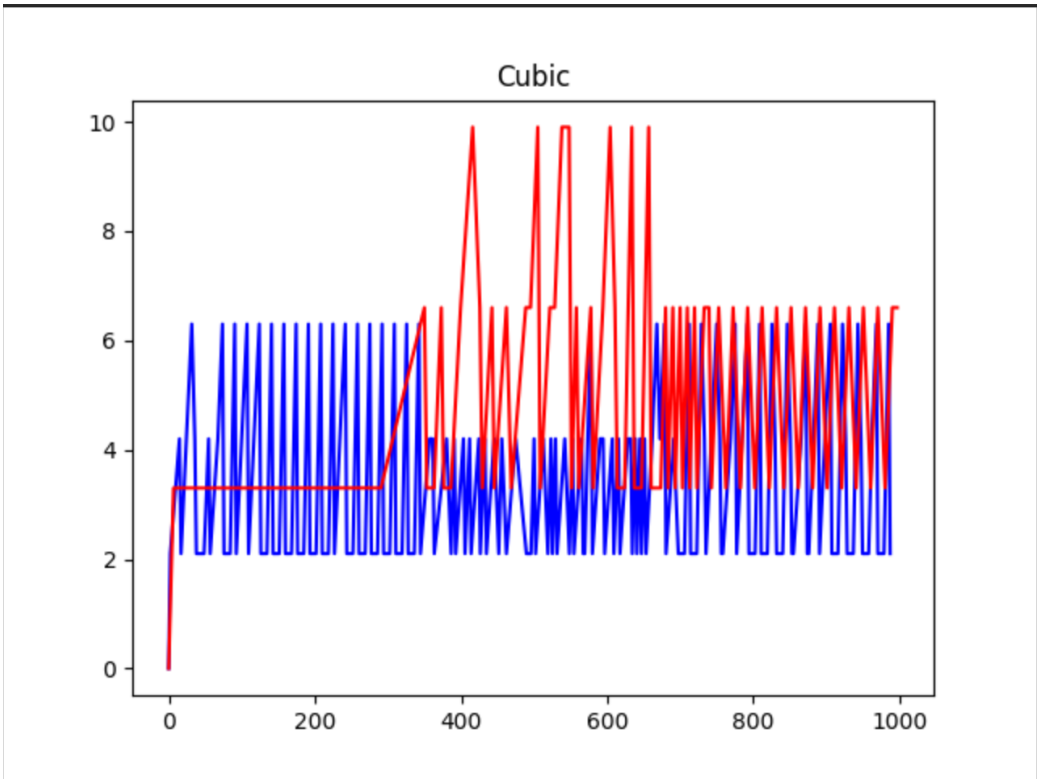
نمودار ۴: نمودار goodput برای الگوریتم cubic در xgraph



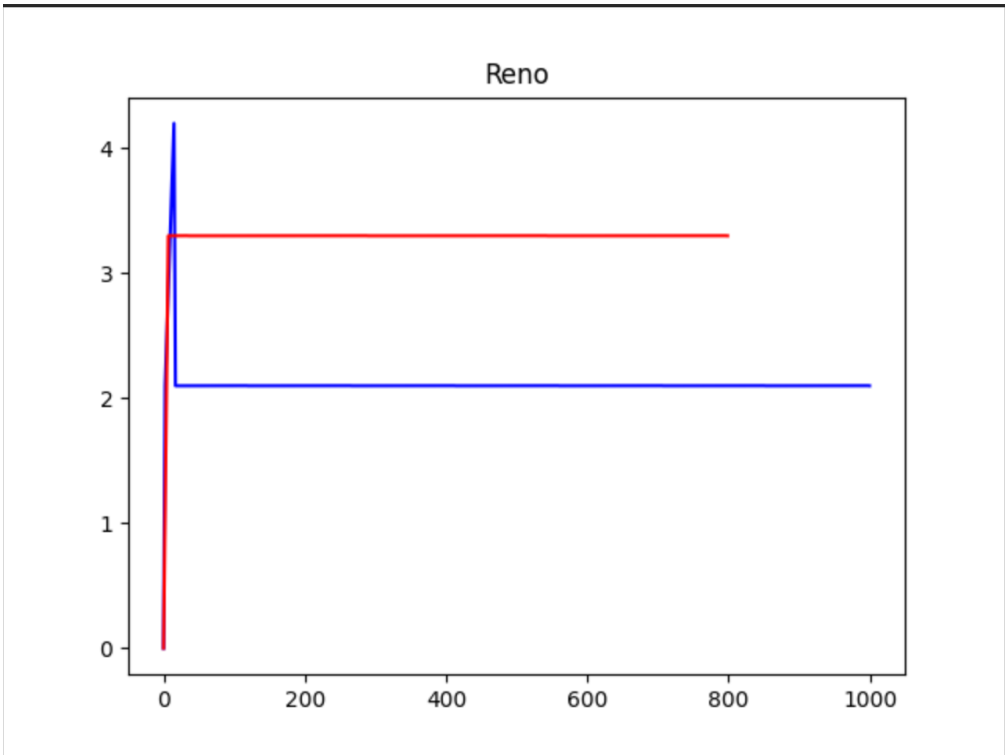
نمودار ۵: نمودار goodput برای الگوریتم reno در xgraph



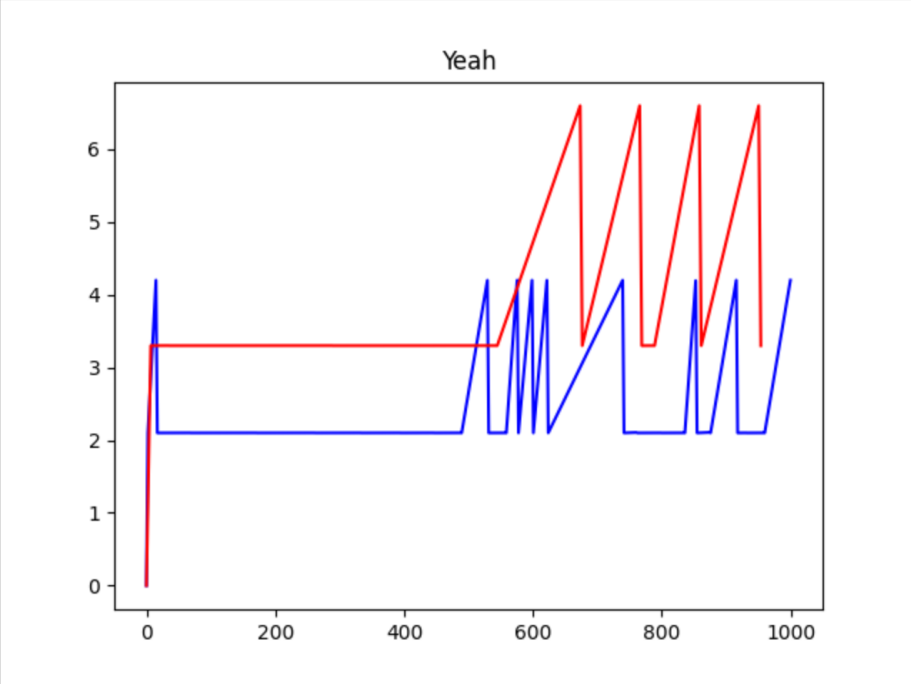
نمودار ۶: نمودار goodput برای الگوریتم yeah در xgraph



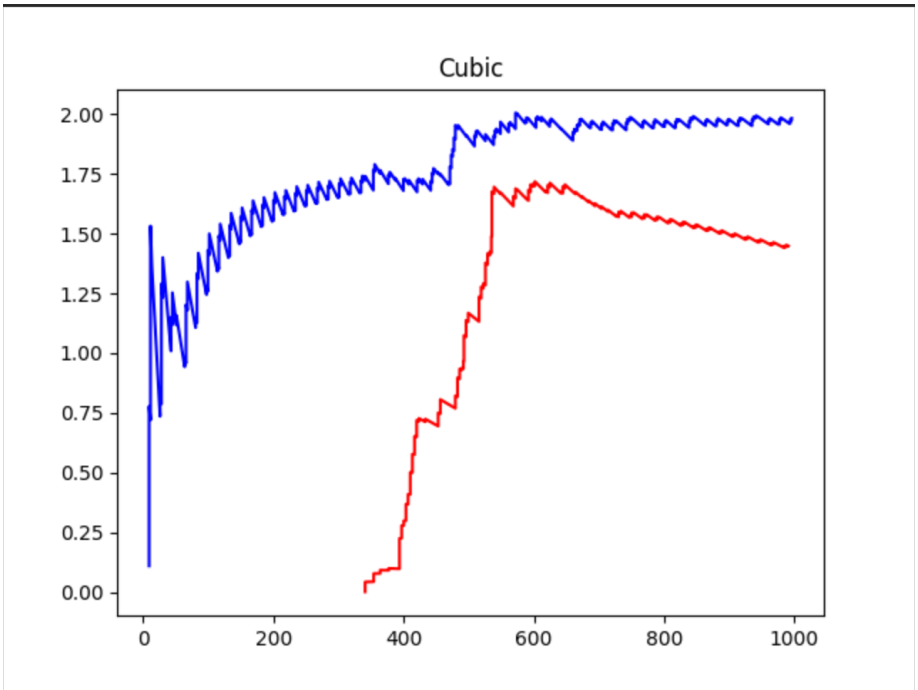
نمودار ۷: نمودار rtt برای الگوریتم cubic در pyplot



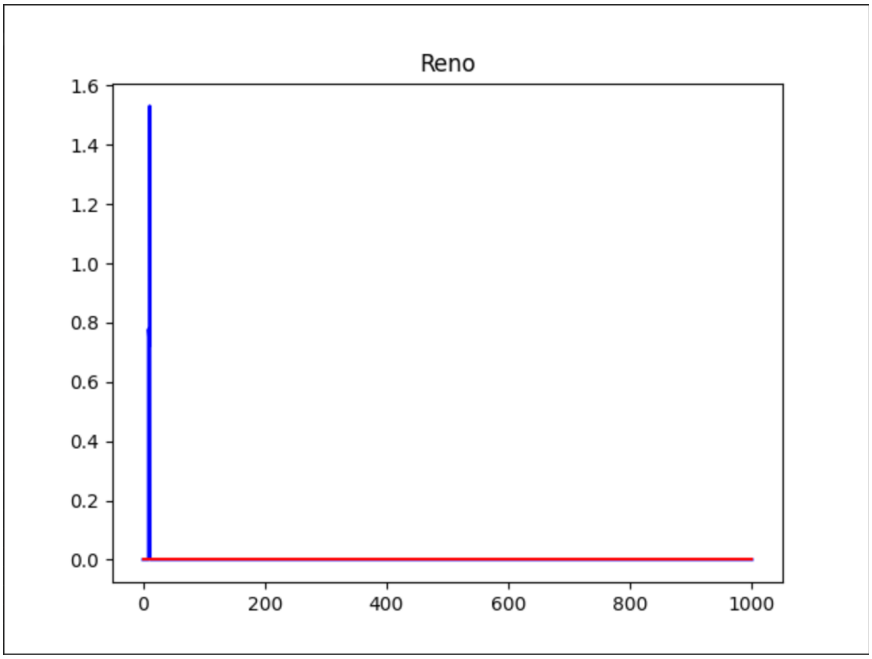
نمودار ۸: نمودار rtt برای الگوریتم reno در pyplot



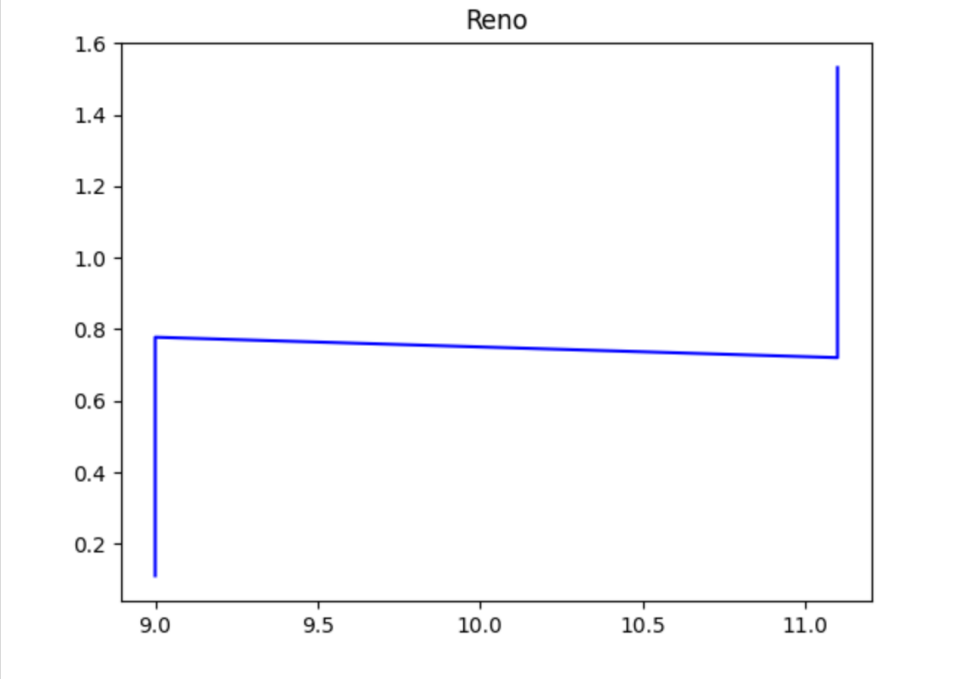
نمودار ۹: نمودار rtt برای الگوریتم yeah در pyplot



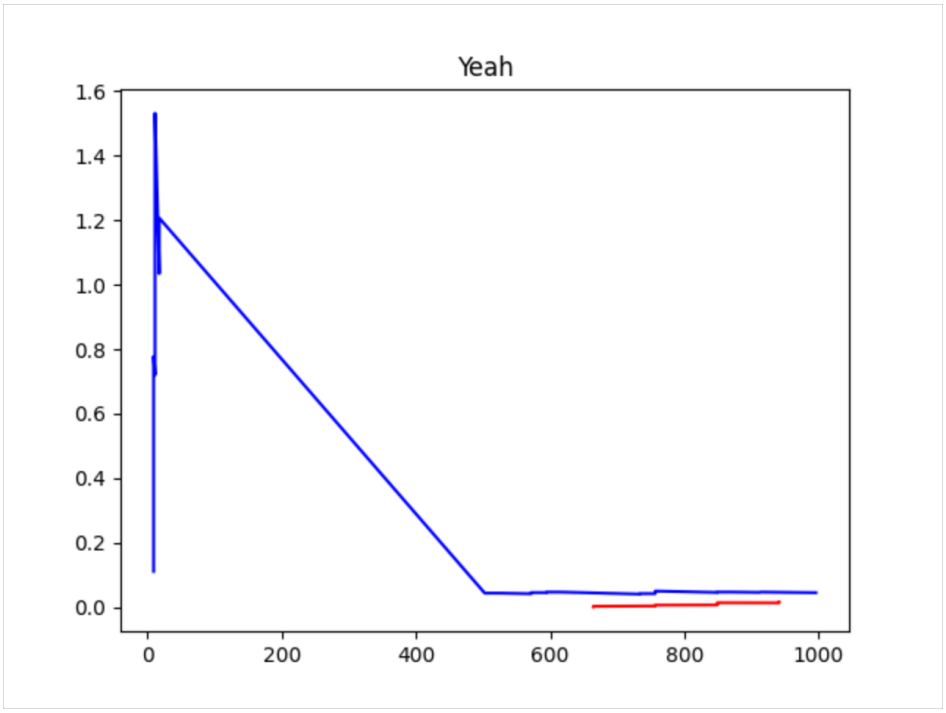
نمودار ۱۰: نمودار packet loss rate برای الگوریتم cubic در pyplot



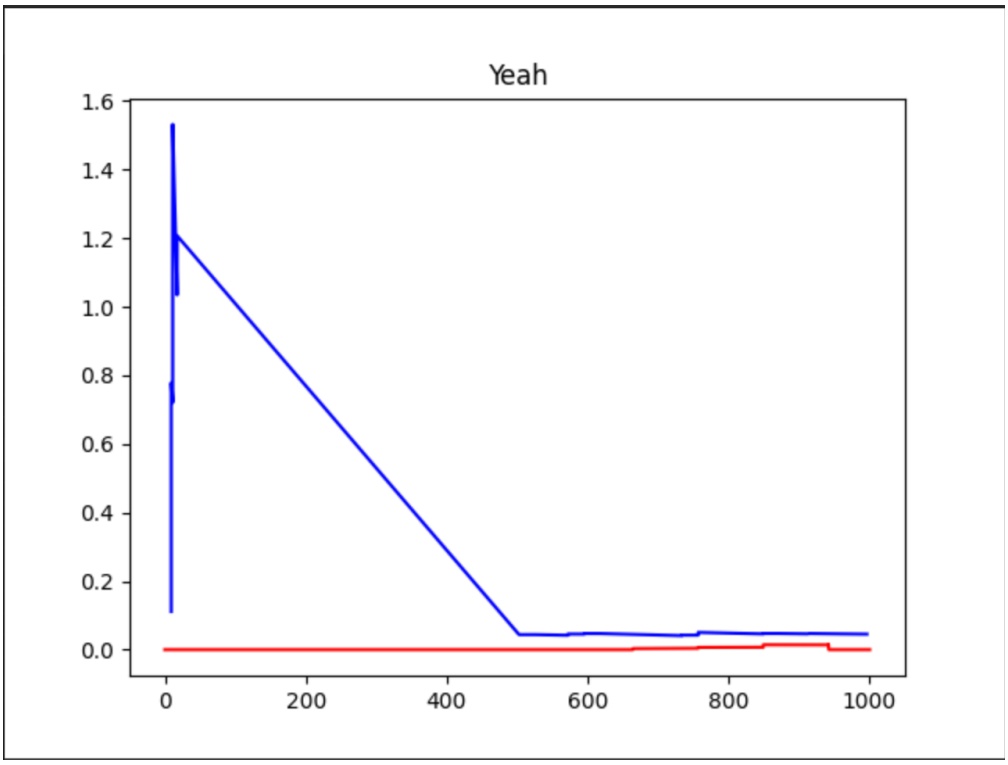
نمودار ۱۱: نمودار packet loss rate برای الگوریتم reno در pyplot



نمودار ۱۲: نمودار packet loss rate برای الگوریتم reno در pyplot زوم شده در بازه ۹ تا ۱۱ ثانیه
(مقدار packet loss rate برای ارسال قرمز رنگ صفر است)



نمودار ۱۳: نمودار packet loss rate برای الگوریتم yeah در pyplot



نمودار ۱۴: نمودار packet loss rate برای الگوریتم yeah در pyplot با نمایی بهتر برای ارسال قرمز رنگ

تحلیل نمودار ها

در نمودار مربوط به cwnd الگوریتم cubic تابع درجه ۳ که گفته شد و نشانش نقطه عطف است قابل مشاهده است. در cubic مقدار cwnd به آخرین congestion بستگی دارد پس هر چه زمان بیشتر بیشتری از congestion آخر بگذرد cwnd با نرخ بیشتری افزایش می یابد. این مورد در نمودار دیده می شود که مقدار cwnd افزایش می یابد.
در نمودار مربوط به cwnd الگوریتم reno این موضوع قابل مشاهده است که اگر acknowledge تمام شود slow start به کار می آید و cwnd یک می شود همچنین نکته دیگر این که در صورتی که ۳ تا acknowledge تکراری دریافت شود اندازه پنجره نصف می شود با توجه به نمودار رسم شده این نصف شدن در ابتدای نمودار قرمز رنگ قابل مشاهده است.
در نمودار مربوط به cwnd الگوریتم yeah در نمودار دومی وارد شدن به slow mode و نصف شدن cwnd معلوم است و مشاهده می شود. این نکته که cwnd سریع زیاد می شود و با روند yeah مطابقت دارد نیز قابل رویت است.

در نمودار مربوط به goodput الگوریتم cubic اولا که درجه سه بودن که قبل هم گفته شد این جا نیز دیده می شود همچنین با توجه به نمودار cwnd همین الگوریتم می بینیم که با افزایش cwnd با افزایش goodput همراه است برعکس این امر نیز دیده می شود.
در نمودار مربوط به goodput الگوریتم reno خطی بودن در نمودار دیده می شود. همچنین با توجه به نمودار cwnd همین الگوریتم نیز می بینیم که مثل cubic با افزایش cwnd مقدار goodput نیز افزایش می یابد برعکسش نیز برقرار است.
برای yeah نیز دقیقا به همان شکلی که گفته شد می باشد و این روند ها در goodput دیده می شود.

با مشاهده rtt برای هر کدام از ۳ الگوریتم می توانیم ببینیم که هر جا cwnd تغییر داشته است نرخ rtt نیز تغییر کرده است مثلا چون در cubic تغییرات cwnd زیاد بوده است می توان در نمودار rtt نیز دید که به شدت تغییراتی در rtt ایجاد شده است. برای reno چون فقط ابتدای روند congestion داشتیم و بعد از آن دیگر congestion ای نداشته ایم و مقدار cwnd با نرخ ثابت افزایش یافته در نمودار rtt یک تغییر محسوس است و بعد از آن دیگر rtt ثابت می شود که منطقی است. برای yeah هم منطق مثل cubic و reno است فقط در yeah تغییرات در cwnd از cubic کم تر و از reno بیشتر است برای همین تعدادی تغییر در rtt دیده می شود که آن نیز منطقی است.

در slow start در شروع به این شکل عمل می شود که آن قدر packet ها ارسال می شود تا یک packet loss اتفاق بیفتد و اقدامات مربوطه را انجام دهد پس با توجه به این نکته در اول کار packet loss rate قاعدتا بالا هست ولی بعد از این که cwnd مناسب set شد این rate قطعا کم تر می شود. موارد گفته شده برای reno و yeah قابل مشاهده است همچنین نوسان کردن این rate نیز در cubic قابل دیدن است.

پس در کل با توجه به مشاهدات بالا هر ۴ معیاری که بررسی شدند به هم وابسته هستند همچنین با توجه به نمونه دیگری از tcp که در کد هم به عنوان قرار داده شد اطلاعات مربوط به tcp هم در شبکه تاثیرگذارند.

آخرین توضیحات

در کد قسمت rtt و packet loss rate در ابتدا با xgraph نمایش داده شده بودند ولی به دلیل این که بعضی اوقات باگ های ریزی داشت تصمیم به استفاده از pyplot گرفته شد و پیاده سازی ها در کد کامنت شد.
دو حالت زوم شده و بهتر رسم package loss rate دو الگوریتم reno و yeah نیز به صورت کامنت شده قرار داده شد.
یک مثال که در شبیه سازی آن بهتر روند ارسال packet ها دیده می شد نیز در کد قرار داده شد.
به دلیل این که out.nam و allTraces.tr ها حجم بالایی داشتند در فایل نهایی قرار داده نشدند با اجرای برنامه آن ها به راحتی تولید خواهند شد.