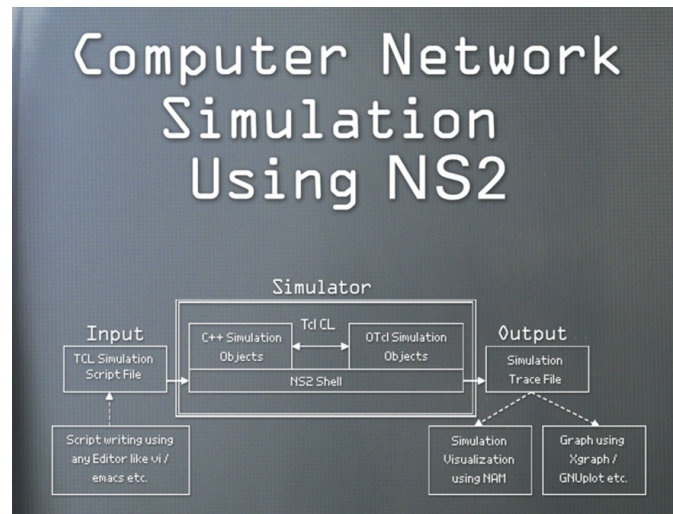


گزارش کار

TCP Simulation



شبکه‌های کامپیوتری

غزل کلهر (۸۱۰۱۹۶۶۷۵)

محمدامین باقرشاهی (۸۱۰۱۹۷۴۶۴)

	فهرست مطالب
2	تعریف و توضیح TCP ها
2	TCP Reno
2	TCP YeAH
3	TCP Cubic
7	توضیح روند اجرا
7	توپولوژی شبکه
9	الگوریتم CUBIC
11	الگوریتم YEAH
14	اجرای نهایی
15	تحلیل و مقایسه‌ی نمودارها
15	تغییرات اندازه پنجره ازدحام
18	نرخ Goodput
21	نرخ RTT
23	نرخ Packet Loss
24	نتیجه گیری
24	مراجع

تعریف و توضیح TCPها

TCP Reno

فرستنده TCP Reno، با استفاده از الگوریتم پنجره لغزان، بسته ها را ارسال می کند. نرخ ارسال بسته به وسیله اندازه پنجره ازدحام کنترل می گردد، که بیانگر حداکثر تعداد بسته هایی است که قبل از دریافت ACK، قابل ارسال می باشند. هنگامی که پنجره ازدحام پر می شود، فرستنده باید قبل از ارسال یک بسته جدید، منتظر رسیدن ACK بماند. ایده کلیدی این الگوریتم این است که برای استفاده از پهنای باند خالی موجود، اندازه پنجره به صورت جمع شونده افزایش یابد، ولی به محض وقوع ازدحام، اندازه پنجره به صورت ضربی کاهش یابد. هر فرستنده با پنجره ای به طول یک بسته، ارتباط را شروع می کند و با دریافت هر ACK، طول پنجره را یک واحد افزایش می دهد. این امر منجر به دو برابر شدن طول پنجره در هر RTT می گردد و به شروع آهسته معروف است. در این مرحله، فرستنده به صورت نمایی نرخ ارسال خود را افزایش می دهد و به سرعت می تواند پهنای باند موجود را پر نماید. هنگامی که اندازه پنجره به حد آستانه شروع آهسته (ssthresh) می رسد، فرستنده وارد مرحله “پرهیز از ازدحام” می گردد. در این مرحله در هر RTT طول پنجره تنها یک واحد افزایش می یابد. یعنی اگر برای همه بسته های ارسال شده در یک پنجره، ACK دریافت شود، به ازای همه آن ها، یک واحد به طول پنجره اضافه می گردد.

هنگامی که فرستنده، از طریق دریافت ACK های تکراری، گم شدن بسته ای را تشخیص می دهد، اندازه پنجره را نصف و اندازه ssthresh را به روز رسانی می کند و با ارسال مجدد بسته گمشده عمل بازایی سریع را انجام می دهد. اگر گم شدن بسته از طریق Time-out شناسایی شود، در آن صورت اندازه پنجره به یک بسته کاهش می یابد و فرستنده دوباره وارد مرحله شروع آهسته می گردد.

TCP YeAH

TCP YeAH یک ایده ابتکاری است که برای ایجاد تعادل بین نیازهای مختلف الگوریتم کنترل ازدحام پیشرفته طراحی شده است. ویژگی های آن به شرح زیر است:

- در حالی که تعداد کمی از رویدادهای ازدحام را ایجاد می کند، به طور کامل از ظرفیت پیوند شبکه های با BDP بالا بهره برداری می کند
- با جریان ها رنو به طور دوستانه رقابت می کند
- به انصاف RTT دست پیدا می کند

● در برابر تلفات تصادفی مقاوم است

● بدون توجه به اندازه بافر، به عملکرد بالایی دست پیدا می‌کند

YeAH بین 2 حالت کار می‌کند: حالت سریع و آهسته. در حالت سریع وقتی اشغال صف کم است و سطح ازدحام شبکه کم است، YeAH بر مبنای قانون تهاجمی HSTCP پنجره ازدحام خود را افزایش می‌دهد. هنگامی که تعداد بسته‌ها در صف فراتر از یک آستانه رشد می‌کند و سطح ازدحام شبکه زیاد است، Yeah وارد حالت آهسته خود می‌شود و با الگوریتم ازدحام به عنوان Reno عمل می‌کند. YeAH از مکانیسم Vegas برای محاسبه عقب ماندگی مانند معادله (۱) استفاده می‌کند. برآورد سطح ازدحام شبکه در معادله (۲) نشان داده شده است.

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

معادله (۱)

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

معادله (۲)

برای اطمینان از دوستانه بودن TCP، در YeAH همچنین الگوریتمی برای تشخیص وجود جریان‌های قدیمی Reno پیاده‌سازی می‌شود. با دریافت 3 ACK تکراری، YeAH در صورت عدم رقابت با جریان‌های Reno، آستانه شروع آهسته آن را مطابق با معادله (۳) کاهش می‌دهد. در غیر این صورت، ssthresh مانند Reno به نصف کاهش می‌یابد:

$$ssthresh = \min\left(\max\left(\frac{cWnd}{8}, Q\right), \frac{cWnd}{2}\right)$$

معادله (۳)

TCP Cubic

الگوریتم CUBIC چیست و علت وجود آن چه بوده است؟

مشکل utilization پایین TCP در شبکه هایی با مسافت طولانی و سریع ثبت شده است. این مشکل ناشی از افزایش آهسته پنجره ازدحام در پی یک رویداد ازدحام در شبکه ای با یک محصول تأخیر پهنای باند بزرگ است. این مشکل اغلب حتی در محدوده اندازه پنجره های ازدحام بیش از صدها بسته مشاهده می شود. این مشکل در تمام استانداردهای TCP به سبک Reno و انواع مختلف آنها، از جمله SCTP، TCP-NewReno، TCP-RENO و TFRC، که از همان عملکرد افزایش خطی برای رشد پنجره استفاده می کنند دیده شده است.

CUBIC در اصل یک اصلاح در الگوریتم کنترل ازدحام استاندارد TCP برای رفع این مشکل است. به طور خاص، CUBIC از یک عملکرد مکعبی استفاده می کند به جای عملکرد افزایش پنجره خطی استاندارد TCP برای بهبود مقیاس پذیری و پایداری در شبکه های سریع و طولانی استفاده می شود. این الگوریتم برای لینوکس در سال 2005 و برای چندین سال توسط جامعه اینترنت به طور گسترده ای مورد استفاده قرار گرفته است. CUBIC از عملکرد افزایش پنجره مشابه BIC-TCP استفاده می کند و طوری طراحی شده است که در استفاده از پهنای باند نسبت به BIC-TCP کمتر تهاجمی و منصفانه تر است و در عین حال نقاط قوت BIC-TCP مانند ثبات، مقیاس پذیری پنجره و انصاف RTT را حفظ می کند. CUBIC جایگزین BIC-TCP به عنوان الگوریتم پیش فرض کنترل ازدحام TCP در لینوکس شده و توسط لینوکس در سطح جهانی مستقر شده است. از طریق آزمایش گسترده در سناریوهای مختلف اینترنت، ما معتقدیم که CUBIC برای آزمایش و استقرار در اینترنت جهانی ایمن است.

در واقع CUBIC یک الگوریتم جلوگیری از ازدحام شبکه برای TCP است که می تواند در برابر تأخیر زیاد نسبت به الگوریتم های دیگر، به اتصالات پهنای باند بالای شبکه ها با سرعت و اطمینان بیشتری دست یابد. به بهینه سازی شبکه های طولانی کمک می کند. CUBIC TCP از سال 2006 به طور پیش فرض در هسته های لینوکس 2.6.19 به بالا پیاده سازی و استفاده می شد. CUBIC مدل کمتر تهاجمی و سیستماتیک تری از BIC TCP است، که در آن اندازه پنجره یک تابع CUBIC از زمان آخرین رویداد ازدحام است. از آنجا که این یک عملکرد CUBIC است، دو جز در رشد پنجره تأثیر دارد. اولین قسمت مقعر (CONCAVE) است که در آن اندازه پنجره به سرعت به اندازه سائز قبل از آخرین رویداد ازدحام بزرگ می شود. بعدی قسمت محدب (CONVEX) است که probe های CUBIC برای پهنای باند بیشتر، ابتدا به آرامی و سپس خیلی سریع انجام می شوند. CUBIC زمان زیادی را بین ناحیه رشد CONCAVE و CONVEX می گذراند که به شبکه امکان می دهد قبل از اینکه CUBIC به دنبال پهنای باند بیشتری باشد، تثبیت شود.

تفاوت عمده دیگر بین CUBIC و بسیاری از الگوریتم های TCP این است که برای افزایش اندازه پنجره به سرعت RTT متکی نیست. اندازه پنجره CUBIC فقط به آخرین رویداد ازدحام بستگی دارد. با الگوریتم هایی مانند TCP New Reno ، جریان هایی با زمان تاخیر رفت و برگشت بسیار کوتاه، (ACK RTT) را سریعتر دریافت می کنند و بنابراین پنجره های ازدحام آنها سریعتر از جریان های دیگر با RTT طولانی تر رشد می کنند. CUBIC امکان عدالت بیشتر بین جریان ها را فراهم می کند ، زیرا رشد پنجره مستقل از RTT است. پس CUBIC پنجره خود را افزایش می دهد تا به زمان واقعی وابسته باشد ، نه به RTT. محاسبه $cwnd$ (پنجره ازدحام) نیز ساده تر از الگوریتم BIC است.

برای تعریف الگوریتم CUBIC ، متغیر های زیر را تعریف می کنیم:

β : ضریب کاهش ضربی

w_{max} : اندازه پنجره درست قبل از آخرین کاهش

T : زمان گذشته از آخرین کاهش پنجره

C : ثابت مقیاس گذاری

$Cwnd$: پنجره ازدحام در زمان فعلی

واحد تمام اندازه های پنجره بر اساس حداکثر اندازه بخش (MSS) است و واحد تمام زمانها ثانیه است. همچنین β باید روی 0.7 تنظیم شود. C باید روی 0.4 تنظیم شود.

اندازه $cwnd$ هم به صورت زیر محاسبه می شود:

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

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

اصول طراحی CUBIC

CUBIC با توجه به اصول طراحی زیر طراحی شده است:

- اصل 1: برای استفاده بهتر و پایداری شبکه ، CUBIC به جای استفاده از فقط یک تابع محدب ، از تابع های مقعر و محدب یک عملکرد مکعبی برای افزایش اندازه پنجره ازدحام استفاده می کند.
- اصل 2: CUBIC به گونه ای طراحی شده است که در شبکه هایی با RTT کوتاه و پهنای باند کوچک که TCP استاندارد عملکرد خوبی دارد ، مانند TCP استاندارد رفتار می کند.

اصل 3: برای انصاف CUBIC ، RTT برای دستیابی به تقسیم پهنای باند خطی بین جریان های RTT مختلف طراحی شده است.

اصل 4: CUBIC به منظور تعادل بین مقیاس پذیری و سرعت همگرایی ، به طور مناسب ضریب کاهش پنجره خود را تنظیم می کند.

کنترل ازدحام CUBIC

cwnd اندازه پنجره ازدحام جریان را نشان دهد و ssthresh آستانه شروع آهسته را نشان دهد.

عملکرد افزایش پنجره

در طی اجتناب از ازدحام پس از یک رویداد ازدحام که در آن یک از اتلاف بسته توسط ACK های تکراری شناسایی می شود یا یک ازدحام شبکه توسط ACK ها با پرچم های ECN-Echo شناسایی می شود ، CUBIC عملکرد افزایش پنجره استاندارد TCP را تغییر می دهد. فرض کنید W_{max} قبل از کاهش پنجره در آخرین رویداد ازدحام ، اندازه پنجره باشد. CUBIC از عملکرد افزایش پنجره زیر استفاده می کند:

$$W_{cubic}(t) = C * (t-K) \wedge 3 + W_{max} \quad (1)$$

که در آن C ثابت برای تعیین میزان تهاجمی افزایش پنجره در شبکه های BDP است ، t زمان سپری شده از ابتدای اجتناب از ازدحام فعلی است ، و K مدت زمانی است که عملکرد فوق طول می کشد تا اندازه پنجره فعلی را به W_{max} در صورت عدم وجود رویدادهای ازدحام بیشتر و محاسبه آن با استفاده از معادله زیر:

$$K = \text{cubic_root}(W_{max} * (1 - \text{beta_cubic}) / C) \quad (2)$$

جایی که beta_cubic عامل کاهش ضرب CUBIC است ، یعنی وقتی یک رویداد ازدحام شناسایی می شود $cwnd$ ، CUBIC خود را به مقدار زیر کاهش می دهد.

توضیح روند اجرا

در این قسمت تمام بخش های کد پروژه را به تفصیل شرح می دهیم.

توپولوژی شبکه

در کد های مربوط به هر کدام از الگوریتم ها، ابتدا توپولوژی شبکه تعریف شده است و این بخش از کد در همه ی آن ها مشترک است. نحوه تعریف این بخش از کد به صورت زیر است:

```
set ns [new Simulator]

# nam sim data
set namfile [open cubic.nam w]
$ns namtrace-all $namfile
set tracefile1 [open cubicTrace.tr w]
$ns trace-all $tracefile1

# on finish
# flush all trace and open nam
proc finish {} {
    global ns namfile
    $ns flush-trace
    #Close the NAM trace file
    close $namfile
    #Execute NAM on the trace file
    # exec nam newreno.nam &
    exit 0
}
```

این بخش از کد مربوط به فایل هایی است که می خواهیم خروجی کد اجرا شده در آن ها قرار گیرد. پس فایل های با تایپ .nam و trace را در اینجا مشخص و نام گذاری می کنیم.


```
# create 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]
```

در این قسمت هر کدام از گره های توپولوژی داده شده در صورت پروژه را تعریف کرده ایم که در مجموع 6 گره شامل میزبان ها و مسیریاب ها بوده است.

```
$ns duplex-link $n1 $n3 4000Mb 500ms DropTail
$ns duplex-link $n2 $n3 4000Mb 800ms DropTail
$ns duplex-link $n3 $n4 1000Mb 50ms DropTail
$ns duplex-link $n4 $n5 4000Mb 500ms DropTail
$ns duplex-link $n4 $n6 4000Mb 800ms DropTail
```


در این بخش از کد، ظرفیت هر لینک و مقدار delay هایی که برای هر یک از آن ها در صورت پروژه تعیین شده بود تعریف شده اند. همچنین هر لینک با توجه به دو گره انتهایی آن تعریف می شود.

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

در اینجا اندازه ی صف در روتر ها که برابر 10 بسته است، تعریف شده است.

```
$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
```

در این بخش نحوه قرار گیری هر کدام از گره های موجود در توپولوژی تعریف می شود. این نحوه قرار گرفتن گره ها دقیقا مانند شکل موجود در صورت پروژه است.



```
# setup simulation colors
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green
```

در اینجا رنگ هر کدام از جریان های خروجی که در پنجره NS2 نشان داده می شود، مشخص می شود.

الگوریتم CUBIC

در این بخش کدهایی که فقط مربوط به الگوریتم CUBIC است را توضیح می دهیم.

```

set source1 [new Agent/TCP/Reno]
$source1 set class_ 2
$source1 set ttl_ 64
$source1 set window_ 200
$source1 set packet_size_ 1000

$ns attach-agent $n1 $source1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n5 $sink1
$ns connect $source1 $sink1
$source1 set fid_ 1

set source2 [new Agent/TCP/Reno]
$source2 set class_ 1
$source2 set ttl_ 64
$source2 set window_ 200
$source2 set packet_size_ 1000

$ns attach-agent $n2 $source2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n6 $sink2
$ns connect $source2 $sink2
$source2 set fid_ 2

$source1 attach $tracefile1
$source1 tracevar cwnd_
$source1 tracevar ssthresh_
$source1 tracevar ack_
$source1 tracevar maxseq_
$source1 tracevar rtt_

$source2 attach $tracefile1
$source2 tracevar cwnd_
$source2 tracevar ssthresh_
$source2 tracevar ack_
$source2 tracevar maxseq_
$source2 tracevar rtt_

set myftp1 [new Application/FTP]
$myftp1 attach-agent $source1

set myftp2 [new Application/FTP]
$myftp2 attach-agent $source2

```

در اینجا دو اتصال tcp و ftp به نام های source0 و source1 و همچنین myftp0 و myftp1 تعریف شده است. یک اتصال بین دو گره 1 و 5 و اتصال دیگر بین گره های 2 و 6 می باشد. همچنین برای تعریف کردن الگوریتم CUBIC ، عبارت select_ca در cubic در کد نوشته است و مقدار اولیه RTT نیز برابر 64 در نظر گرفته شده است. همچنین در این بخش، مقادیری که می خواهیم

در فایل trace وجود داشته باشد، تعریف شده است. این مقادیر شامل cwnd و ssthresh و ack و maxsequence و RTT است. همچنین مقدار cwnd طبق صورت پروژه برابر با 8000 تعریف شده است.

```
$ns at 0 "$ftp1 start"
$ns at 0 "$ftp0 start"

$ns at 1000 "$ftp1 stop"
$ns at 1000 "$ftp0 stop"
```

در اینجا زمان شروع و پایان هر یک از اتصال های tcp و ftp نظیر آن ها، تعیین شده است.

```
# when to stop
$ns at 1000.0 "finish"
```

در این بخش زمان پایان اجرای کل الگوریتم CUBIC تعیین شده است که برابر با 1000 ثانیه می باشد.

```
$ns run
```

این خط کد را اجرا خواهد کرد.

الگوریتم YEAH

تمام بخش های این الگوریتم مانند الگوریتم CUBIC است و فقط بخش زیر در آن تفاوت اندکی دارد.

```

set source1 [new Agent/TCP/Linux]
$ns at 0.0 "$source1 select_ca yeah"
$source1 set class_ 2
$source1 set ttl_ 64
$source1 set window_ 200
$source1 set packet_size_ 1000

$ns attach-agent $n1 $source1
set sink1 [new Agent/TCPSink/Sack1]
$ns attach-agent $n5 $sink1
$ns connect $source1 $sink1
$source1 set fid_ 1

set source2 [new Agent/TCP/Linux]
$ns at 0.0 "$source2 select_ca yeah"
$source2 set class_ 1
$source2 set ttl_ 64
$source2 set window_ 200
$source2 set packet_size_ 1000

$ns attach-agent $n2 $source2
set sink2 [new Agent/TCPSink/Sack1]
$ns attach-agent $n6 $sink2
$ns connect $source2 $sink2
$source2 set fid_ 2

$source1 attach $tracefile1
$source1 tracevar cwnd_
$source1 tracevar ssthresh_
$source1 tracevar ack_
$source1 tracevar maxseq_
$source1 tracevar rtt_

$source2 attach $tracefile1
$source2 tracevar cwnd_
$source2 tracevar ssthresh_
$source2 tracevar ack_
$source2 tracevar maxseq_

$source2 tracevar rtt_

set myftp1 [new Application/FTP]
$myftp1 attach-agent $source1

set myftp2 [new Application/FTP]
$myftp2 attach-agent $source2

```

در این قسمت باید مقدار select_ca را به yeah تغییر دهیم تا الگوریتم yeah اجرا شود و تمام اتصالات tcp و ftp آن مانند الگوریتم CUBIC می باشد. پس دو اتصال tcp خواهد داشت. سائز اولیه cwnd هم برابر با 8000 است.

الگوریتم RENO

تمام بخش های این الگوریتم مانند الگوریتم CUBIC است و فقط بخش زیر در آن تفاوت اندکی دارد.

```
set source1 [new Agent/TCP/Reno]
$source1 set class_ 2
$source1 set ttl_ 64

ns attach-agent $n1 $source1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n5 $sink1
$ns connect $source1 $sink1
$source1 set fid_ 1

set source2 [new Agent/TCP/Reno]
$source2 set class_ 1
$source2 set ttl_ 64
$ns attach-agent $n2 $source2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n6 $sink2
$ns connect $source2 $sink2
$source2 set fid_ 2

$source1 attach $tracefile1
$source1 tracevar cwnd_ 8000
$source1 tracevar ssthresh_
$source1 tracevar ack_
$source1 tracevar maxseq_
$source1 tracevar rtt_

$source2 attach $tracefile1
$source2 tracevar cwnd_ 8000
$source2 tracevar ssthresh_
$source2 tracevar ack_
$source2 tracevar maxseq_
$source2 tracevar rtt_

set myftp1 [new Application/FTP]
$myftp1 attach-agent $source1

set myftp2 [new Application/FTP]
$myftp2 attach-agent $source2
```

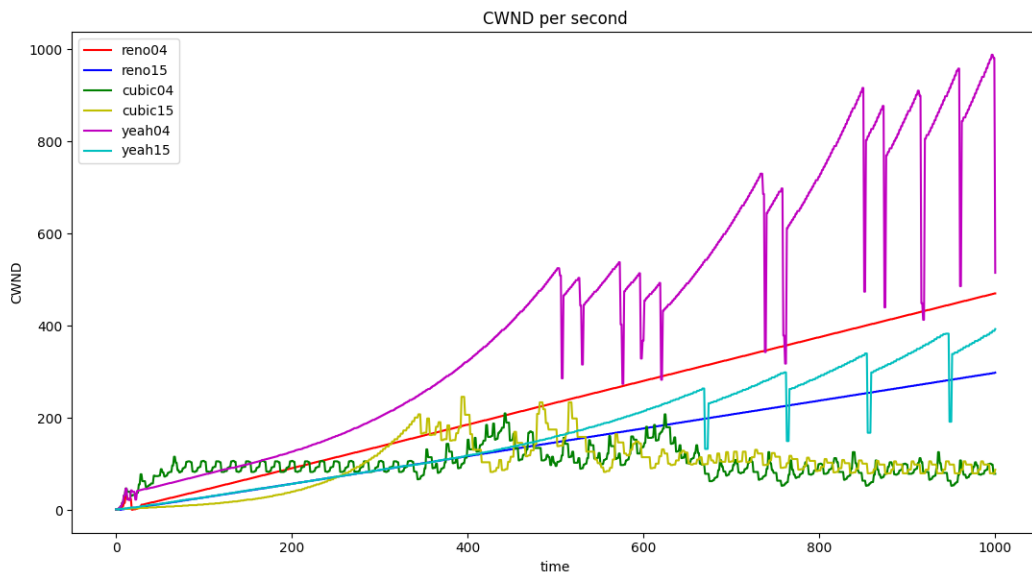
در این بخش دو اتصال tcp و ftp های مربوط به آن تعریف شده است. پس شامل اتصال های tcp با نام source1 و source2 و دو ftp با نام های myftp1 و myftp2 است. همچنین در فایل trace، مقادیر شامل cwnd و ssthresh و ack و maxsequence و RTT تعریف شده است. همچنین مقدار cwnd طبق صورت پروژه برابر با 8000 تعریف شده است.

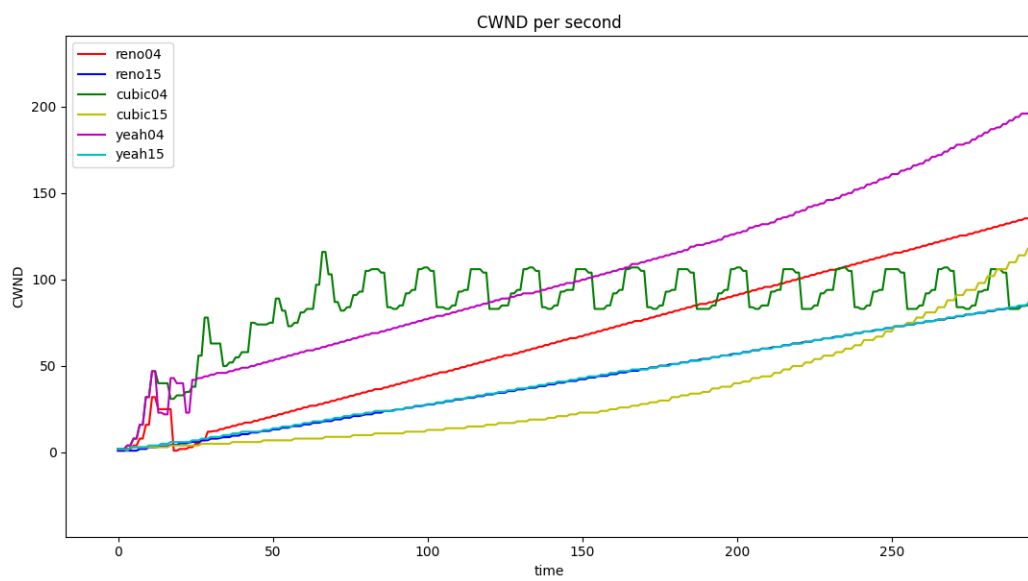
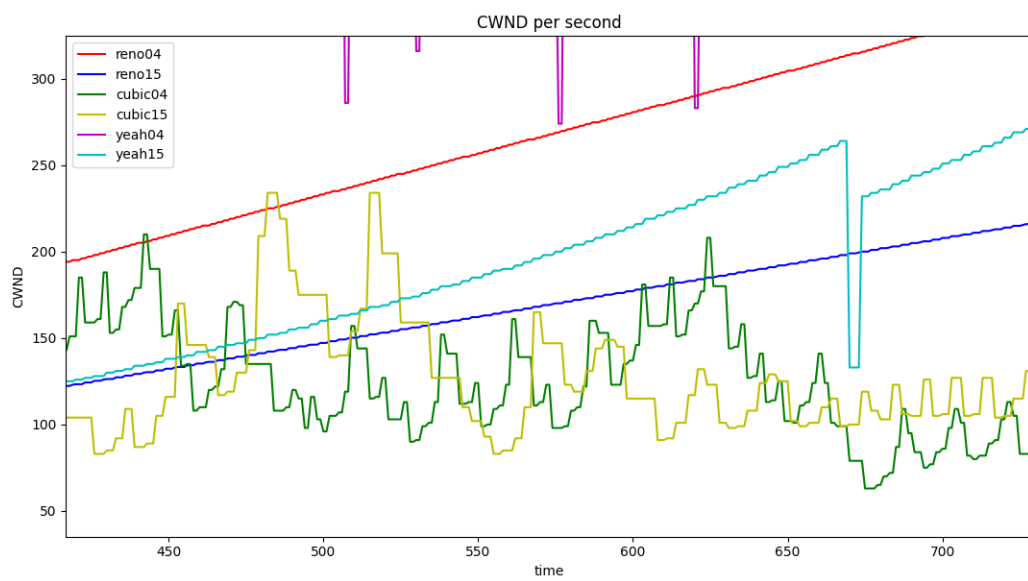
اجرای نهایی

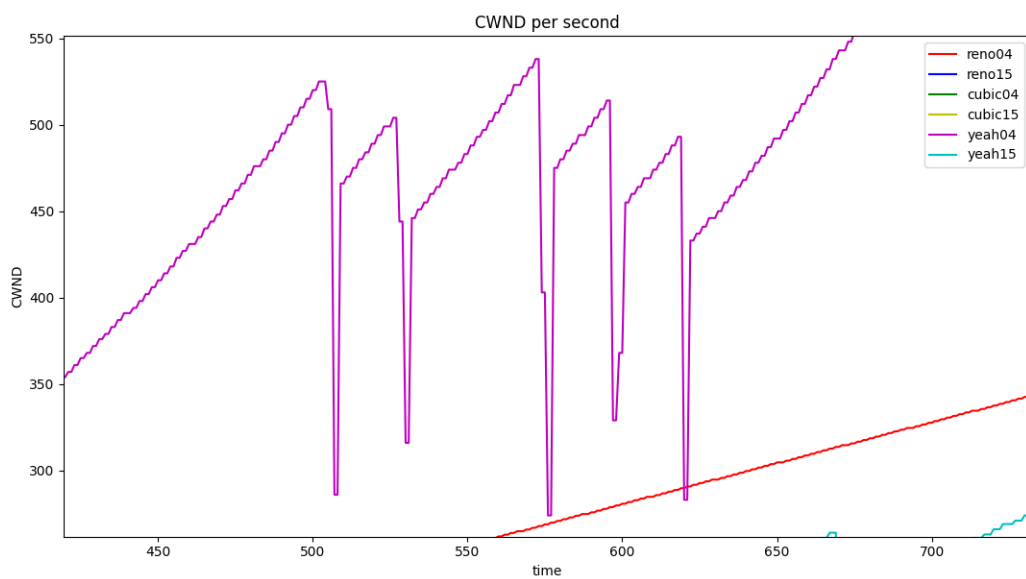
برای اجرا شدن هر یک از این الگوریتم ها به زبان tcl در NS2، در ترمینال کد پایتون نوشته شده را اجرا خواهیم کرد و الگوریتم به درستی ران خواهد شد و نمودار ها پس از چند ثانیه نشان داده می شود. همچنین به عنوان خروجی فایل های nam. و trace. ایجاد خواهد شد.

تحلیل و مقایسه‌ی نمودارها

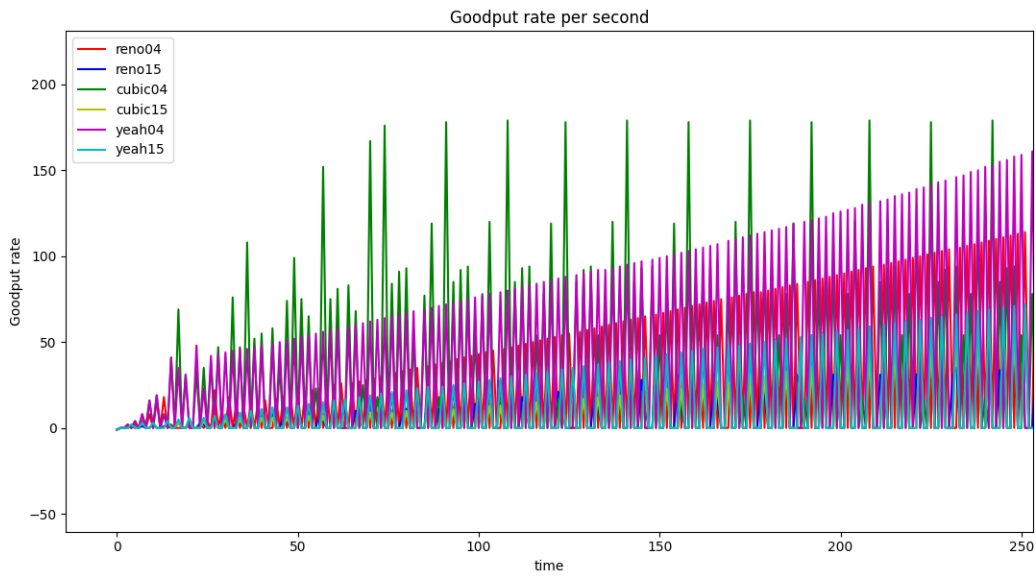
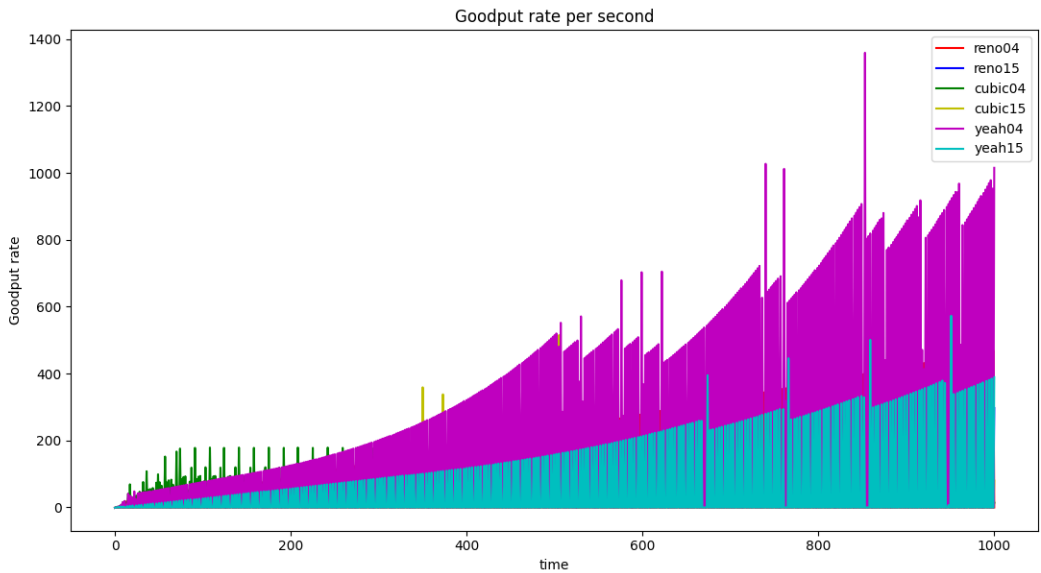
تغییرات اندازه پنجره ازدحام







سه تصویر فوق نمودارهای مربوط به تغییرات اندازه پنجره است. (تصاویر دوم و سوم زوم شده تصویر اول هستند.) همانطور که درباره‌ی tcp های مختلف توضیح داده شد، واضح است تغییرات اندازه پنجره‌ها دقیقاً متناسب با الگوریتمی که از تبعیت می‌کنند قابل انطباق است. توجه داریم که الگوریتم TCP Yeah شیب بیشتری دارد و تغییرات سریع‌تری داشته و از این منظر مناسب‌تر است و الگوریتم بهتری نسبت به سایرین محسوب می‌شود چون آستانه بزرگتری دارد.



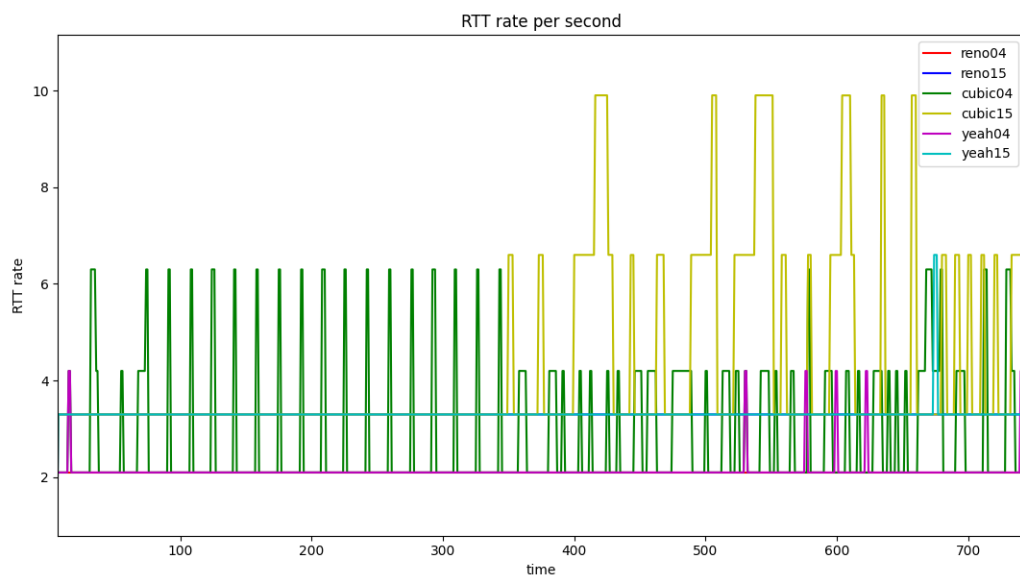
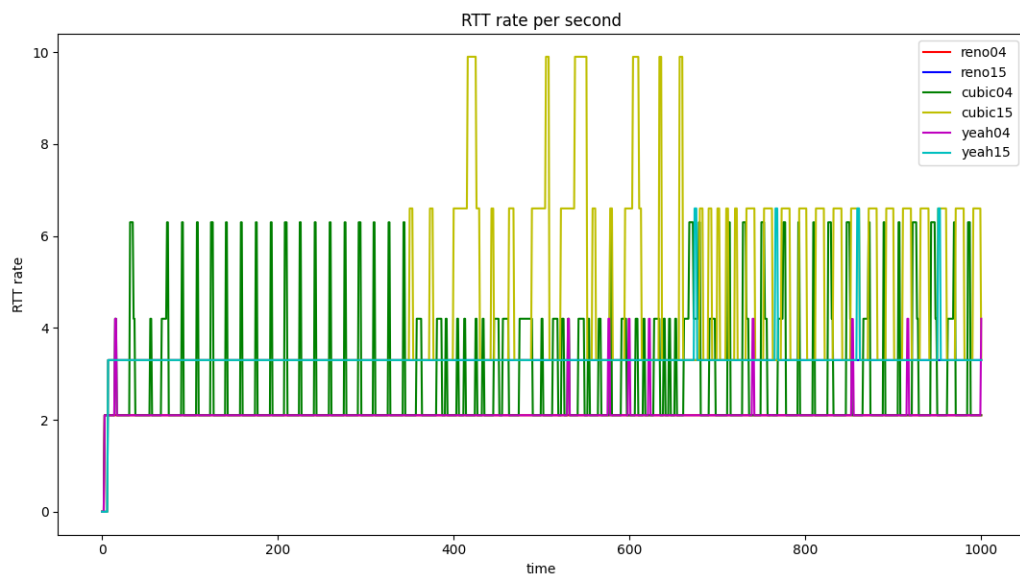



سه تصویر فوق نمودارهای مربوط به نرخ Goodput است. (تصاویر دوم و سوم زوم شده تصویر اول هستند.) همانطور که درباره‌ی tcp های مختلف توضیح داده شد، و نیز با توجه به نمودار می‌بینیم که این نرخ برای الگوریتم TCP Cubic نسبت به سایر الگوریتم‌ها بیشتر است. پس از این نظر به نظر می‌رسد که از الگوریتم‌های دیگر مناسب‌تر است. از طرفی دیگر الگوریتم TCP YeAH نیز از TCP Reno کم‌تر است پس TCP Reno الگوریتم بهتری نسب به آن است و با توجه به فرمول‌های بیان شده در بخش قبل این موضوع قابل استنباط است.

نرخ Goodput یک نسبت بین مقدار تحویل داده شده و کل زمان تحویل است. این زمان تحویل شامل موارد زیر است:

- فاصله زمانی بین بسته ای ناشی از زمان پردازش تولید بسته (منبعی که از ظرفیت کامل شبکه استفاده نمی کند) یا زمان پروتکل (به عنوان مثال جلوگیری از برخورد)
- تأخیر انتقال داده و سرپار (مقدار داده تقسیم بر نرخ بیت)
- تأخیر انتشار (فاصله تقسیم بر سرعت انتشار موج)
- تأخیر در صف بندی بسته ها
- تأخیر ترجمه NAT
- تأخیر پردازش ذخیره و انتقال گره متوسط
- زمان انتقال مجدد بسته (در صورت پاک شدن بسته های موجود در روترهای متراکم ، یا خطاهای بیتی شناسایی شده)
- تأخیر تأیید به دلیل کنترل جریان ، جلوگیری از ازدحام و تأخیر در پردازش

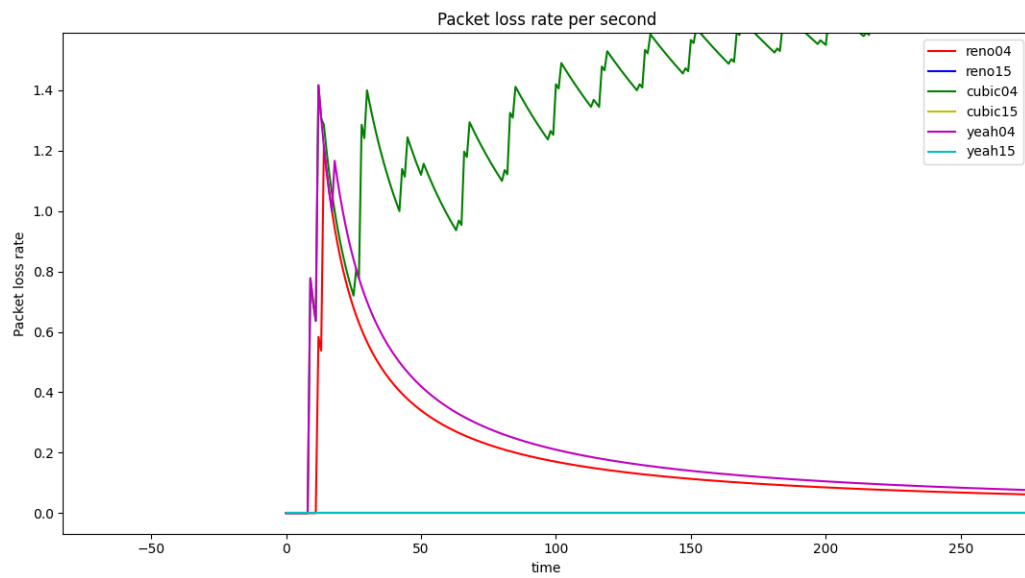
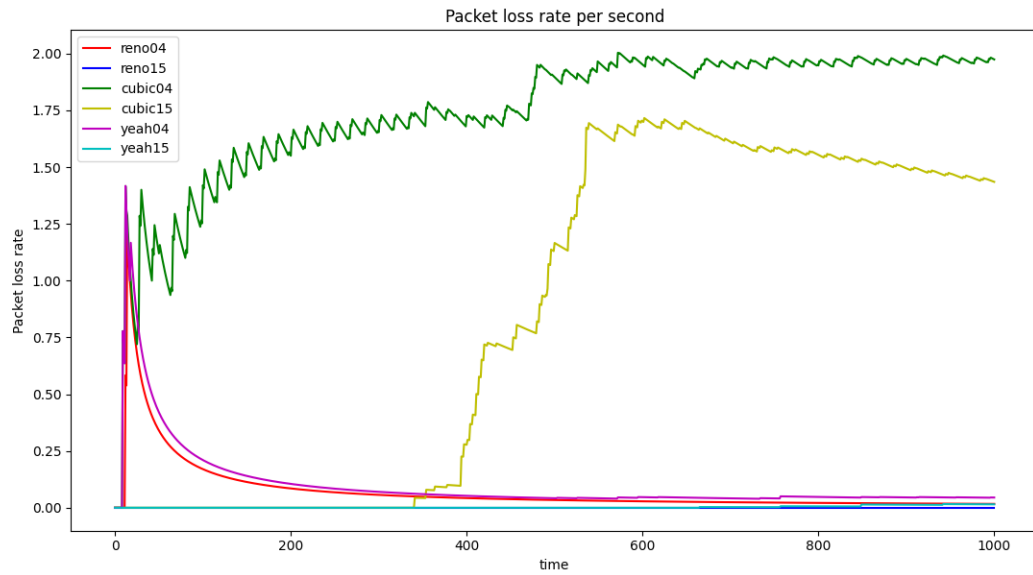
نرخ RTT





دو تصویر فوق نمودارهای مربوط به نرخ RTT است. (تصویر دوم زوم شده تصویر اول است.) به طور کلی هر سه tcp با شروع شبیه‌سازی، واضح است که این نرخ برای الگوریتم TCP Cubic بالاتر است و از این منظر چندان مناسب نیست. و در کل می‌توان گفت که TCP YeAh از سایر الگوریتم‌ها در این زمینه مناسب‌تر بوده است.

نرخ Packet Loss



دو تصویر فوق نمودارهای مربوط به نرخ Packet Loss است. (تصویر دوم زوم شده تصویر اول است.) همانطور که درباره‌ی tcp های مختلف توضیح داده شد و با توجه به نمودار واضح است که نرخ از دست رفته بسته ها در الگوریتم TCP Cubic نسبت به سایر الگوریتم‌ها بیشتر است و در طول زمان نیز افزایش می‌یابد. پس از این نظر به نظر می‌رسد که الگوریتم‌های دیگر مناسب‌تر هستند. از طرفی دیگر الگوریتم TCP Yeah نیز از TCP Reno مناسب‌تر است و نرخ از دست رفتن بسته‌ها در آن کمتر است.

نرخ Packet Loss نسبت تعداد بسته های گمشده به تعداد بسته های ارسالی را نشان می دهد. هر بسته مهلتی دارد که قبل از آن باید اجرا شود و در صورت عدم امکان ، برنامه ریز سعی می کند تعداد بسته های گمشده را به دلیل انقضای مهلت به حداقل برساند. تضمین های مربوط به رفع این محدودیت های زمان بندی و نحوه برخورد سیستم با بسته هایی که نمی توانند مهلت خود را برآورده کنند ، اهداف الگوریتم برنامه ریزی است. کسری از بسته های رها شده ، به دلیل نقض مهلت ، برای ارزیابی عملکرد از دست دادن یک برنامه زمان بندی استفاده می شود. کسری از بسته های افتاده ، به دلیل نقض مهلت ، برای ارزیابی عملکرد از دست دادن یک برنامه زمان بندی استفاده می شود. هر زمان که این کسر کوچک باشد ، زمانبند برای زمان بندی ترافیک در زمان واقعی مناسب است.

نتیجه گیری

بر اساس شبیه سازی انجام شده و نمودارهای به دست آمده، این نتیجه حاصل می شود که در مدیریت ازدحام شبکه همواره هر چهار عامل اندازه پنجره، نرخ 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 TCP های موجود در این پروژه، این نتیجه حاصل می شود که TCP Yeah به نسبت از همه ی TCP ها مفیدتر بوده و سیاست های بهتری برای مواجهه با network congestion دارد که منجر به عملکرد بهتر آن شده است.

مراجع

- [1] Baiocchi, A., Castellani, A.P. and Vacirca, F., 2007, February. YeAH-TCP: yet another highspeed TCP. In Proc. PFLDnet (Vol. 7, pp. 37-42).
- [2] https://en.wikipedia.org/wiki/CUBIC_TCP
- [3] <https://tools.ietf.org/html/rfc8312>

