گزارش پروژه چهارم درس شبکه های کامپیوتری

حمیدرضا خدادادی 810197499

محمدعلی زارع 810197626

در این تمرین به شبیه‌سازی یک شبکه با ابزار ns2 و مقایسه سه الگوریتم کنترل ازدحام پرداخته شد.

روش اجرا

با اجرای اسکریپت پایتون run.py ابتدا فایل های tcl با استفاده از ns اجرا می‌شوند و اطلاعات خروجی (traceها) در دیرکتوری output/ ذخیره می‌شوند. این اسکریپت در ادامه برای رسم نمودار اطلاعات مورد نیاز را از فایل‌های خروجی استخراج می‌کند و به کمک کتابخانه matplotlib آن‌هارا رسم می‌کند.

```
ali@ali-PC: ~/Desktop/university/curr_sem/cn/cas/ca4
File  Edit  View  Search  Terminal  Help
ali@ali-PC$ python3 run.py
tracefiles will be saved in ./output/
running reno.tcl ...
running cubic.tcl ...
cmd select_ca cubic
cmd select_ca cubic
running yeah.tcl ...
cmd select_ca yeah
cmd select_ca yeah
0x5647540c3a30 resizing timestamp table
0x5647540b88f0 resizing timestamp table
0x5647540c3a30 resizing timestamp table
0x5647540b88f0 resizing timestamp table
0x5647540b88f0 resizing timestamp table
0x5647540c3a30 resizing timestamp table
0x5647540b88f0 resizing timestamp table

(run.py:27363): Gtk-WARNING **: 17:17:36.587: Theme parsing error: gtk.css:4069:23: Junk at end of value for color
plotting goodput...
plotting cwnd...
plotting rtt...
plotting drop...
plots are saved in ./plots/
ali@ali-PC$
```

کنترل ازدحام در TCP

Reno

الگوریتم‌های کنترل ازدحام و همچون Reno با روش‌هایی شامل تغییر اندازه پنجره‌ها و slow start و ... قصد کنترل ازدحام را دارند. در Reno پکت‌های ACKای که تکثیر شده‌اند و همچنین RTO یا retransmission timeout به عنوان رخداد packet loss حساب می‌شوند. در هنگامی که سه پکت ACK تکراری دریافت شود، Reno دیگر slow start انجام نمی‌دهد و fast transmit انجام می‌دهد. همچنین اندازه پنجره ازدحام را نصف می‌کند. سپس حد آستانه slow start را برابر اندازه پنجره جدید قرار می‌دهد و سپس وارد فاز fast recovery می‌شود. اگر یک ACK تایم‌اوت شود، این الگوریتم اندازه پنجره‌را برابر یک MSS قرار می‌دهد.

Cubic

این الگوریتم از نسخه ۲.۶.۱۹ وارد کرنل لینوکس شد و تا نسخه ۳.۲ به صورت پیش‌فرض استفاده می‌شود.

در این الگوریتم اندازه پنجره یک تابع مکعبی از زمان گذشته از آخرین رخداد ازدحام است. یکی از خصوصیت‌های این روش وابسته نبودن آن به RTT برای رشد اندازه پنجره است و سریع یا کند دریافت کردن ACK تاثیری در آن ندارد که این باعث منصفانه‌تر بودن رفتار Cubic نسبت به الگوریتم‌های دیگر مانند Reno می‌شود.

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

- β: ضریب کاهش
- $w_{\max}$: اندازه پنجره قبل از آخرین کاهش
- T: زمان گذشته از آخرین کاهش
- C: مقدار ثابت
- $cwnd$: اندازه فعلی پنجره

YeAH[1]

الگوریتم YeAH با چند هدف اصلی طراحی شد. این اهداف شامل این موارد است: استفاده از بهینه از ظرفیت شبکه، فشاری که به شبکه وارد می‌کند کمتر یا برابر Reno داشته باشد، سازگاری با ترافیک‌های Reno، داخلی باشد و از نظر RTT منصفانه باشد، کارایی نباید وابستگی زیادی به پکت‌های از دسته رفته رندوم داشته باشد، اندازه کم بافر در لینک‌های نباید از کارایی بالا جلوگیری کند.

این الگوریتم در دو حالت سریع و کند عمل می‌کند. در حالت سریع الگوریتم اندازه پنجره را به صورت تهاجمی (از قانون STCP برای سادگی پیاده‌سازی آن استفاده شده) زیاد می‌کند. در حالت کند نیز این الگوریتم همانند Reno عمل می‌کند. برای اینکه الگوریتم بداند در کدام حالت است، با استفاده از یک فرمول وابسته به RTT و اندازه پنجره و goodput، تعداد پکت‌های موجود در صف باتل‌نک را تخمین می‌زند و بر اساس آن تصمیم می‌گیرد که در وضعیت سریع یا کند عمل کند.

توضیحات کد tcl

برای ذخیره اطلاعات مورد نیاز نمودارها نیاز به چند فایل داریم که ابتدای کد آن‌هارا باز می‌کنیم:

```
set nf [open output/reno.nam w]
$ns namtrace-all $nf
set tracefile1 [open output/renoTrace.tr w]
$ns trace-all $tracefile1
set cwndOut [open output/reno_cwnd.tr w]
set goodputOut [open output/reno_goodput.tr w]
set rttOut [open output/reno_rtt.tr w]
```

حال برای شبیه‌سازی ابتدا یک آبجکت شبیه‌سازی تعریف می‌کنیم. سپس توپولوژی شبکه با توصیف node ها و سپس اتصالات و نوع اتصالات میان آن‌ها تعریف می‌شود. در کد ابتدا ۶ گره تعریف شده و سپس اتصال آن‌ها از نوع duplex-link با خصوصیت‌های خواسته‌شده تعریف شده. سپس دو صف در دو طرف اتصال نود ۳ و ۴ قرار گرفته تا محدودیت صف ۱۰ پکت اعمال شود.

```
set ns [new Simulator]
```

[1] http://www.csc.lsu.edu/~sjpark/cs7601/4-YeAH_TCP.pdf

```
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$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

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

حال دو اتصال TCP باید برای دو اتصال خواسته شده بسازیم. در نوع اتصال الگوریتم کنترل ازدحام و خصوصیت‌های دیگر خواسته شده‌را مشخص می‌کنیم. یک اتصال به شکل زیر ساخته می‌شود:

```
set tcp1 [new Agent/TCP/Reno]
$tcp1 set class_ 2
$tcp1 set ttl_ 64
$tcp1 set windowInit_ 8   # 8 * MSS =  8 * 1000
$tcp1 set window_ 8000 # change max window size

$ns attach-agent $n1 $tcp1
set dst1 [new Agent/TCPSink]
$ns attach-agent $n5 $dst1
$ns connect $tcp1 $dst1
$tcp1 set fid_ 1
```

اتصال ساخته‌شده دوم نیز به همین شکل ساخته خواهد شد و به نودهای n2 و n6 نسبت داده می‌شوند.

اگر بخواهیم جای Reno از Cubic یا YeAH استفاده کنیم ابتدای کد به این شکل تغییر می‌کند:

```
set tcp1 [new Agent/TCP/Linux]
```

```
$ns at 0 "$tcp1 select_ca cubic" # or yeah
```

حال برای ایجاد ترافیک روی شبکه و این اتصال‌ها از یک اپلیکیشن با پروتکل FTP استفاده می‌کنیم:

```
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
```

در نهایت نیز شبیه‌سازی را با زمان خواسته شده شروع می‌کنیم و procedure های مربوط به ذخیره اطلاعات (بخش بعد توضیح داده شده است) را آغاز می‌کنیم:

```
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 0.0  "cwndTrace $tcp1 $tcp2 $cwndOut"
$ns at 0.0  "goodputTrace $tcp1 $tcp2 $goodputOut"
$ns at 0.0  "rttTrace $tcp1 $tcp2 $rttOut"

$ns at 1000.0 "finish"

$ns run
```

در finish فایل‌های باز شده را می‌بندیم و سپس خارج می‌شویم:

```
proc finish {} {
     global ns nf cwndOut goodputOut rttOut
     $ns flush-trace
     close $nf
     close $cwndOut
     close $goodputOut
     close $rttOut
     exit 0
}
```

ذخیره اطلاعات برای نمودارها

برای رسم هر نمودار نیاز به اطلاعاتی داریم که باید حین شبیه‌سازی ذخیره کنیم.

اطلاعات مورد نیاز نمودار نرخ از دست رفتن بسته

برای نمودار نرخ از دست رفته، باید بدانیم در هر ثانیه چند پکت دراپ شده است. با دستور زیر اطلاعاتی در فایل گفته شده ذخیره می‌شود:

```
set tracefile1 [open output/renoTrace.tr w]
$ns trace-all $tracefile1
```

هر خط این اطلاعات مانند خط زیر است:

```
r 0.55 2 3 tcp 40 ------- 1 0.0 4.0 0 0
```

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)
- : dequeue (at queue)
d : drop    (at queue)
```

در نتیجه ما باید دنبال خطوطی باشیم که با d آغاز می‌شوند و از fid می‌فهمیم که برای کدام جریان است. برای این کار در اسکریپت پایتون کل این فایل را می‌خوانیم و خطوطی که با d آغاز می‌شوند را جدا می‌کنیم و سپس از این خطهای جدا شده تعداد پکت‌های دراپ شده در هر ثانیه را استخراج می‌کنیم و از آن در رسم نمودار استفاده می‌کنیم.

```
def read_drop_data(tcp_type):
    f = open(f'output/{tcp_type}Trace.tr', 'r')
    drops = []
    for line in f.readlines():
        line = line.split()
        if line[0] == 'd':
            drops.append(line)

    data = {}
    for d in drops:
        t = int(float(d[1]))
```

```
        fid = int(d[7])
        if t not in data:
            data[t] = [t,0,0]
        data[t][fid] += 1

    data = [v for v in data.values()]

    return np.array(data)
```

که در نهایت آرایه دو بعدی‌ای خواهیم داشت که هر سطر آن به شکل زیر است:

```
Second fid1_drops fid2_drops
```

اطلاعات مورد نیاز نمودار goodput

برای این نمودار نیاز داریم بدانیم هر ثانیه چند ack دریافت می‌شوند. برای این کار در کد tcl یک procedure تعریف می‌کنیم که هرثانیه تعداد ack های جدید را در فایلی بنویسد:

```
set prevAck1 -1
set prevAck2 -1
proc goodputTrace {tcp1 tcp2 outfile} {
     global ns
     global prevAck1
     global prevAck2
     set now [$ns now]
     set ack1 [$tcp1 set ack_]
     set ack2 [$tcp2 set ack_]
     puts  $outfile "$now [expr ($ack1-$prevAck1)] [expr
($ack2-$prevAck2)]"
     set prevAck1 $ack1
     set prevAck2 $ack2
     $ns at [expr $now+1] "goodputTrace $tcp1 $tcp2 $outfile"
}
```

می‌بینیم که در این procedure به صورت بازگشتی هر ثانیه دوباره صدا زده می‌شوند و تعداد ack فعلی منهای تعداد قبلی می‌شود و در فایل ذخیره می‌شود.

اطلاعات مورد نیاز نمودار RTT

برای این اطلاعات هم یک procedure تعریف می‌کنیم تا در پایان هر ثانیه، ثانیه مقدار rtt هر دو جریان را در یک فایل بنویسد:

```
proc rttTrace {tcp1 tcp2 outfile} {
     global ns
     set now [$ns now]
     set rtt1 [$tcp1 set rtt_]
     set rtt2 [$tcp2 set rtt_]
     puts  $outfile  "$now $rtt1 $rtt2"
     $ns at [expr $now+1] "rttTrace $tcp1 $tcp2 $outfile"
}
```

اطلاعات مورد نیاز نمودار CWND

مانند RTT یک procedure تعریف می‌کنیم:

```
proc cwndTrace {tcp1 tcp2 outfile} {
     global ns
     set now [$ns now]
     set cwnd1 [$tcp1 set cwnd_]
     set cwnd2 [$tcp2 set cwnd_]
     puts  $outfile  "$now $cwnd1 $cwnd2"
     $ns at [expr $now+1] "cwndTrace $tcp1 $tcp2 $outfile"
}
```

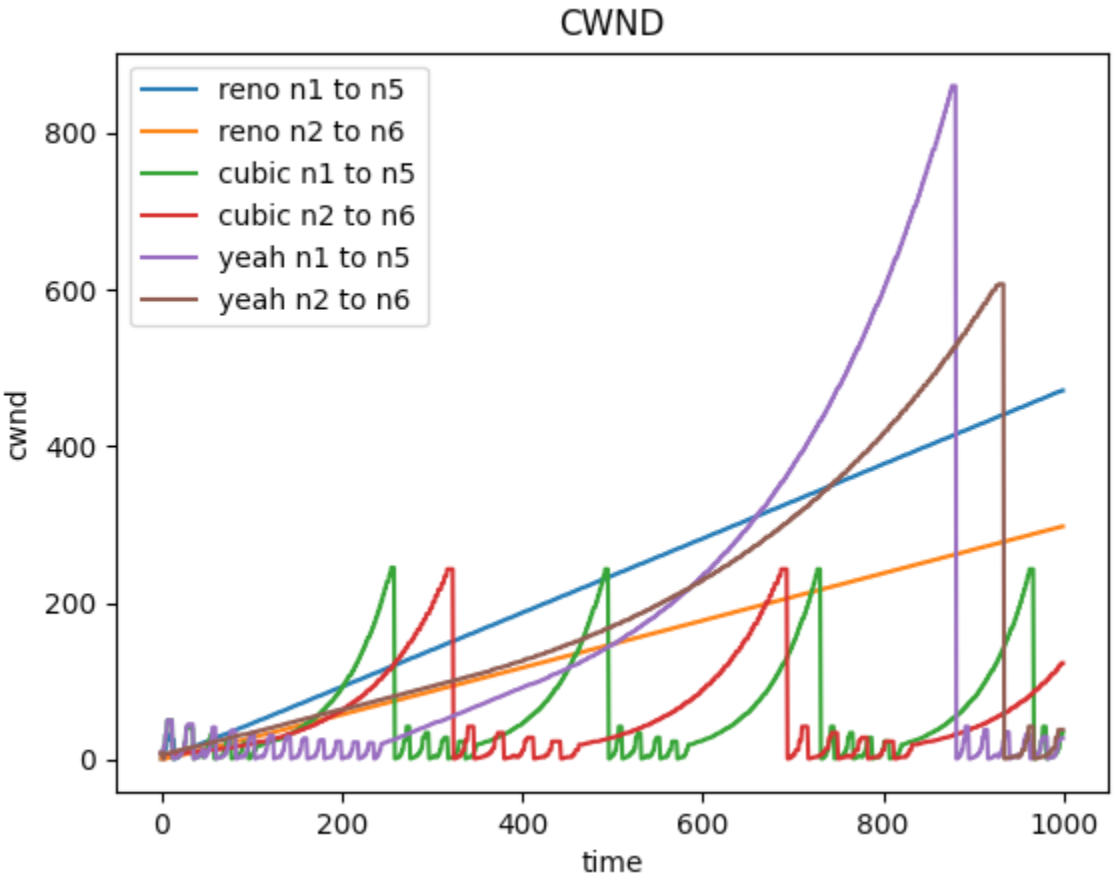
نمودارها

روش رسم نمودارها

در بخش قبل گزارش توضیح داده شد که اطلاعات چگونه ذخیره شده‌اند. همه اطلاعات در نهایت به تعدادی خط که در هر خط ۳ مقدار وجود دارد در آمدند. مقدار اول برابر ثانیه، مقدار دوم عدد مربوط به جریان گره۱ به گره ۵ و مقدار سوم عدد مربوط به جریان گره ۲ به گره ۶ است.

برای رسم نمودارها به کمک کتابخانه matplotlib در پایتون این اعداد را نشان می‌دهیم. ثانیه (مقدار اول در هر خط) نقطه در محور x و هر کدام از دو عدد دیگر نقطه در محور y را نشان می‌دهند.

نمودار CWND

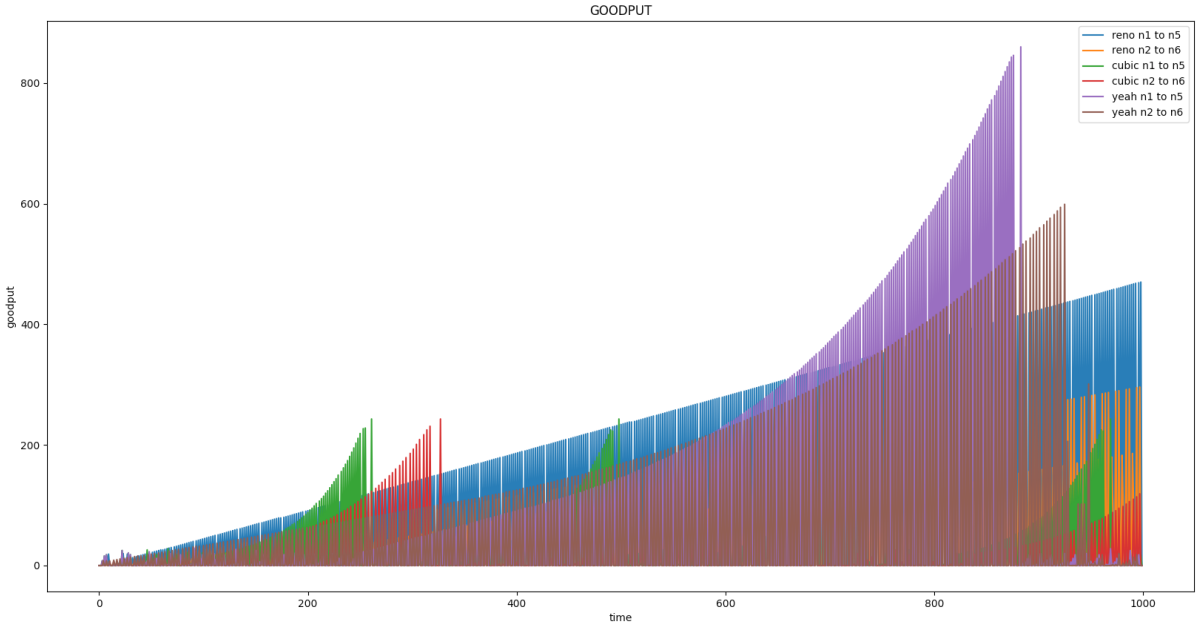


می‌بینیم که reno در ابتدا نمودار کمی بالا رفته و دوباره پایین آمده (نصف شده) و سپس به شکل تقریبا خطی افزایش پیدا کرده است.

برای cubic همانطور که می‌دانیم با تابعی مکعبی بدون وابستگی به rtt رشد می‌کند که در نمودار هم مشخص است که با گذشت زمان از آخرین کاهش سریع‌تر رشد می‌کند.

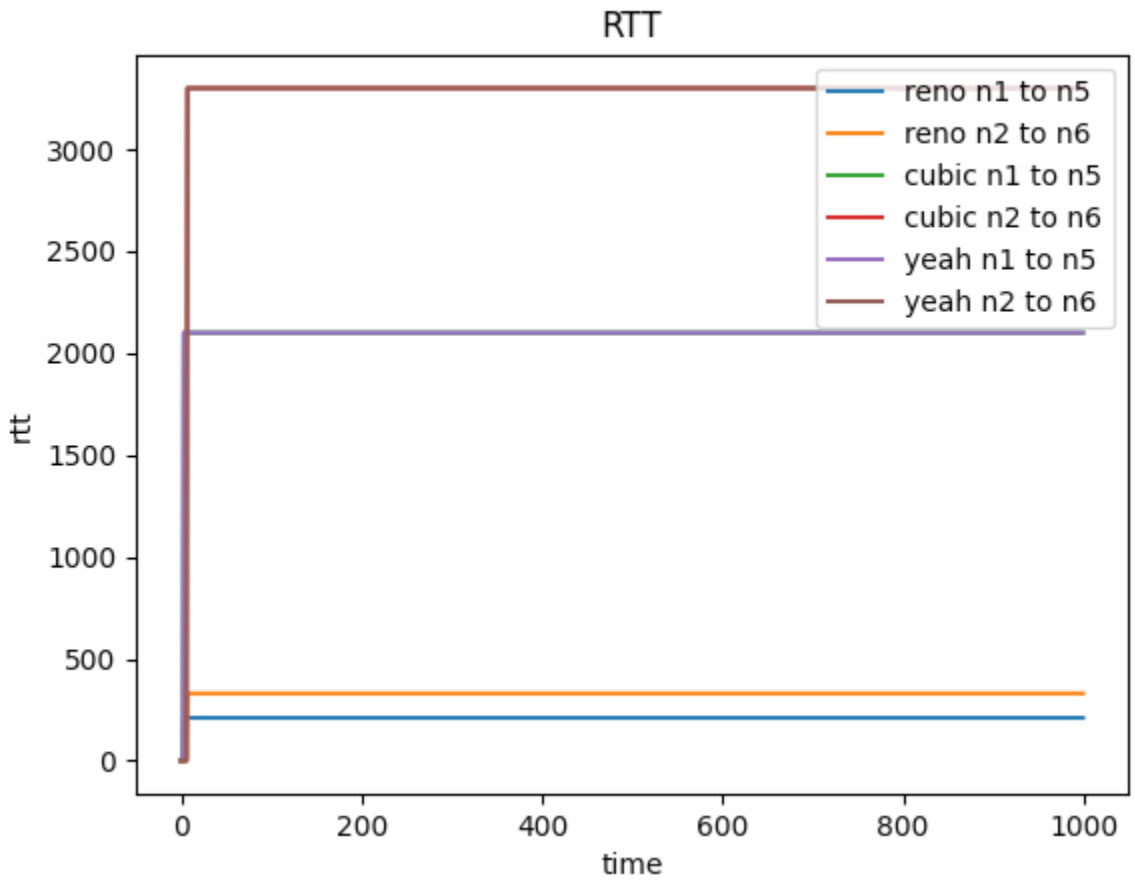
برای YeAH هم نیز می‌بینیم که سعی می‌کند اندازه پنجره را افزایش دهد و از بیشترین ظرفیت شبکه استفاده کند. در جاهایی که وارد حالت کند می‌شود مانند reno عمل می‌کند. اول خط بنفش و انتهای خط بنفش و قهوه‌ای میبینیم که اندازه پنجره مانند reno تصف می‌شود.

نمودار goodput

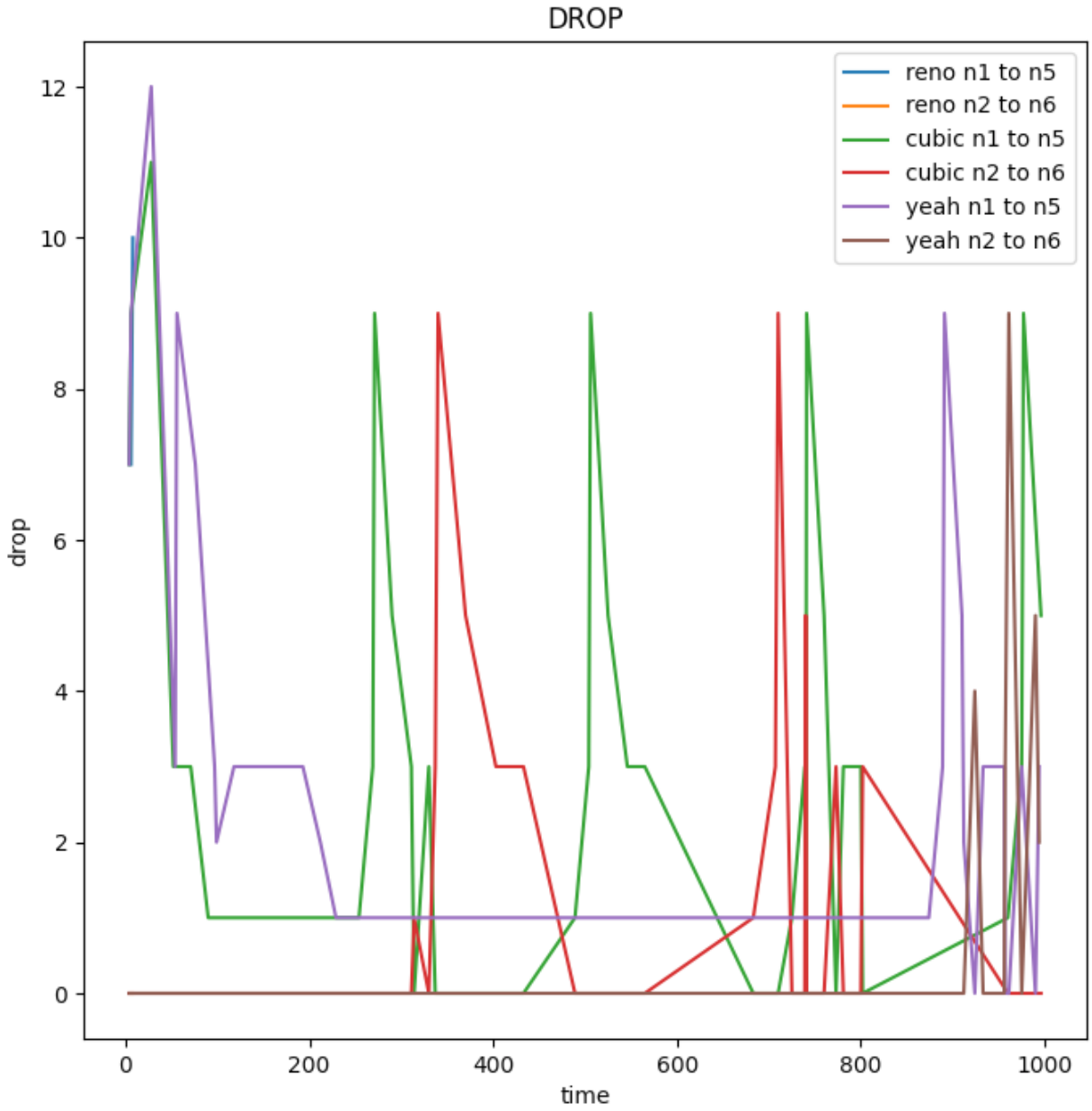


مقادیر goodput را از طریق تعداد ack های دریافتی درست در هر ثانیه به دست آوردیم. در نمودارها می‌بینیم با افزایش اندازه پنجره (نمودار قبلی) مقدار goodput برای آن جریان نیز زیاد شده است. این امر در هر سه الگوریتم کنترل ازدحام قابل دیدن است.

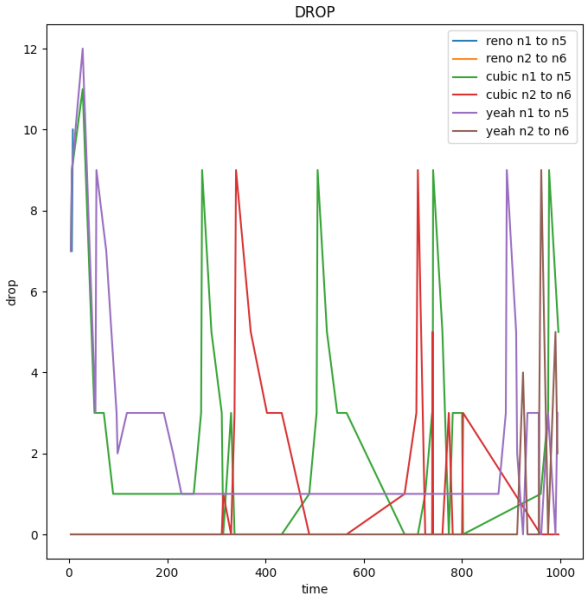
نمودار RTT

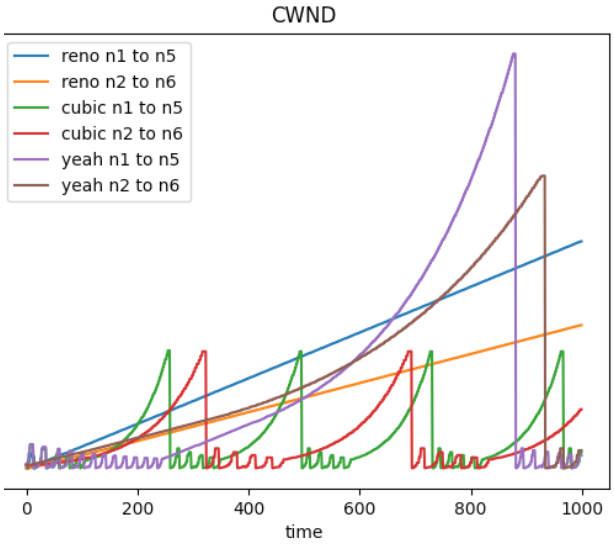

RTT
reno n1 to n5
reno n2 to n6
cubic n1 to n5
cubic n2 to n6
yeah n1 to n5
yeah n2 to n6
rtt
time
0
500
1000
1500
2000
2500
3000
0
200
400
600
800
1000

نمودار نرخ از دست رفتن بسته



می‌بینیم که همه الگوریتم‌ها در ابتدا دراپ زیادی دارند که به دلیل slow start است. سپس کم و زیاد شدن نرخ از دست رفتن بسته‌ها به نست تغییر اندازه پنجره‌ها اتفاق می‌افتد. اگر در نمودار cwnd اندازه پنجره کاهش یافته باشد نرخ از دست رفتن بسته‌ها زیاد شده است و بالعکس. اگر در کنار هم این دو نمودار را نگاه کنیم این موضوع مشهودتر است:





نتیجه‌گیری کلی

با توجه به نمودار ها در می‌یابیم که تغییرات اندازه پنجره و goodput و دیگر متغیرهای مطرح شده همگی به هم وابسته هستند تغییرات آن‌ها در نمودارها در زمان‌های یکسان مشهود است. شکل تغییرات آن‌ها با توجه به الگوریتم کنترل ازدحام انتخاب شده متفاوت است. همانطور که از نمودارها دیدیم به نظر می‌رسد که الگوریتم YeAH عملکرد بهتری نسبت به دو الگوریتم دیگر داشت.