	به نام خدا	

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه‌های کامپیوتری

پروژه‌ی برنامه‌نویسی 2

نام و نام‌خانوادگی	**نرگس غلامی - ریحانه احمدپور**
شماره دانشجویی	**810198494 - 810198447**
تاریخ ارسال گزارش	**1401/2/10 - شنبه ۱۰ اردیبهشت**

فهرست گزارش سؤالات:

توضیحات راجع به کد tcl

در ابتدای گزارش کار به توضیح فایل tcl می پردازیم.

```
set ns [new Simulator]

$ns use-newtrace
set nf [open out.nam w]
$ns namtrace-all-wireless $nf $opt(x) $opt(y)
set tf [open out.tr w]
$ns trace-all $tf

proc finish {} {
    global ns tf nf
    $ns flush-trace
    close $tf
    close $nf
    exec nam out.nam &
    exit 0
}
```

در این قسمت از کد، simulator ست می‌شود. فایل‌های خروجی مورد نیاز ساخته می‌شود که به طور ویژه در این پروژه ما با trace file سر و کار داریم.

یک تابع finish نیز نوشته می‌شود که بتوانیم در نهایت کار شبیه سازی را پایان دهیم.

```
# A
set rownode(0) [$ns node]
$rownode(0) set X_ 200
$rownode(0) set Y_ 350
$rownode(0) set Z_ 0
$rownode(0) label "A"

$ns at 0.0 "$rownode(0) color red"
$rownode(0) color red
# B
set rownode(1) [$ns node]
$rownode(1) set X_ 50
$rownode(1) set Y_ 200
$rownode(1) set Z_ 0
$rownode(1) label "B"
# C
```

در این قسمت شروع به تعریف توپولوژی شبکه می‌کنیم که قرار است شبیه سازی روی آن انجام شود.

از آن جایی که در شبکه خود 9 نود داریم این 9 نود را در شبکه قرار می‌دهیم. برای هر نود موقعیت مکانی آن را، یعنی x و y و z آن را مشخص می‌کنیم. سعی می‌کنیم نودهای همسایه فاصله کمتری داشته باشند و نودهایی که همسایه نیستند فاصله قابل توجهی از هم داشته باشند. نودهای مبدا با رنگ قرمز و نودهای مبدا با رنگ بنفش مشخص شده اند.

ما در هنگام ران کردن برنامه چند آرگومان ورودی داریم. آرگومان اول bandwidth است که به صورت زیر ست می‌شود:

```
if { $argc != 2 } {
   puts "The CA2.tcl script requires bandwidth and packet size. \n For example, 'ns CA2.tcl 1.5 1000' \n Please try again!"
    return 0;
} else {

   if { [lindex $argv 0] == "1.5" } {
        set Bandwidth 1.5Mb
    } else {
      if { [lindex $argv 0] == "55" } {
        set Bandwidth 55Mb
        } else {
            if { [lindex $argv 0] == "155" } {
            set Bandwidth 155Mb
        } else {
            puts "The CA2.tcl script requires bandwidth. \n For example, 'ns CA2.tcl 1.5' \n Please try again."
            return 0;
        }
    }
  }
}
```

سپس با تکه کد زیر در شبکه ست می‌شود.

```
Mac/802_11 set dataRate_ $Bandwidth
```

آرگومان دوم، packet size است که با استفاده از تکه کد زیر در شبکه ست می‌شود:

```
set arg2 [lindex $argv 1]
set packet_size [expr $arg2]
```

مورد بعدی که در شبکه ما اهمیت دارد نرخ ارسال است:

با استفاده از فانکشن زیر و قرار دادن آن در شبکه میتوان مقدار نرخ خطای ارسال را تعیین نمود

```
proc ErrRate {} {
    set err [new ErrorModel]
    $err set rate_ 0.000001
    $err unit packet
    return $err
}

$ns node-config -adhocRouting $opt(adhocRouting) \
                -llType       $opt(ll) \
                -macType      $opt(mac) \
                -ifqType      $opt(ifq) \
                -ifqLen       $opt(ifqlen) \
                -antType      $opt(ant) \
                -propType     $opt(prop) \
                -phyType      $opt(netif) \
                -channel      $chan1 \
                -topoInstance $topo \
                -wiredRouting OFF \
                -agentTrace   ON \
                -IncomingErrProc ErrRate \
                -OutgoingErrProc ErrRate \
                -routerTrace  ON \
                -macTrace     OFF
```

در این قسمت 4 شبکه tcp با 4 سینک مختلف ساخته می‌شود.

که دو تا از این تعاریف شبکه را در روبرو مشاهده می‌نمایید.

به ازای هر مبدا دو tcp و به ازای هر مقصد دو sink تعریف می‌شود

و این‌ها به یکدیگر متصل می‌شوند.

سپس شبکه شروع به کار می‌کند و در سیمولیشن مشاهده می‌شود که

نود A و D به نود L و H پکت ارسال می‌کنند.

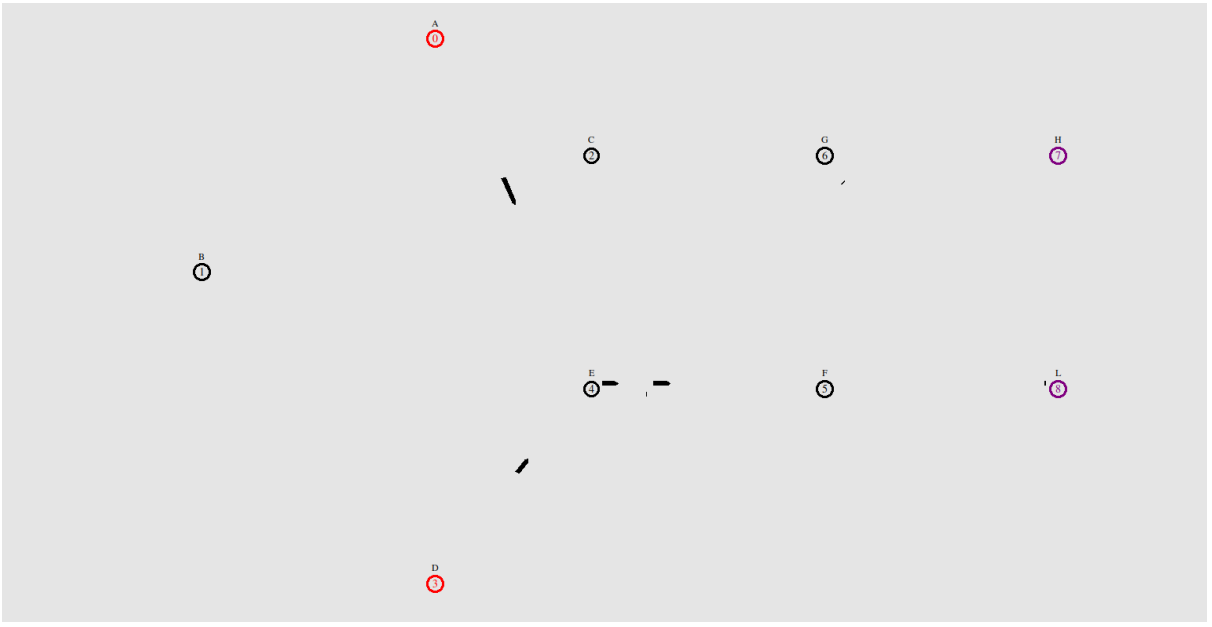
```
set tcp1 [new Agent/TCP]
$tcp1 set class_ 2
$tcp1 set packetSize_ $packet_size
set sink1 [new Agent/TCPSink]
$ns attach-agent $rownode(0) $tcp1
$ns attach-agent $rownode(8) $sink1
$ns connect $tcp1 $sink1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 0 "$ftp1 start"

set tcp2 [new Agent/TCP]
$tcp2 set class_ 2
$tcp2 set packetSize_ $packet_size
set sink2 [new Agent/TCPSink]
$ns attach-agent $rownode(3) $tcp2
$ns attach-agent $rownode(8) $sink2
$ns connect $tcp2 $sink2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ns at 0 "$ftp2 start"
```

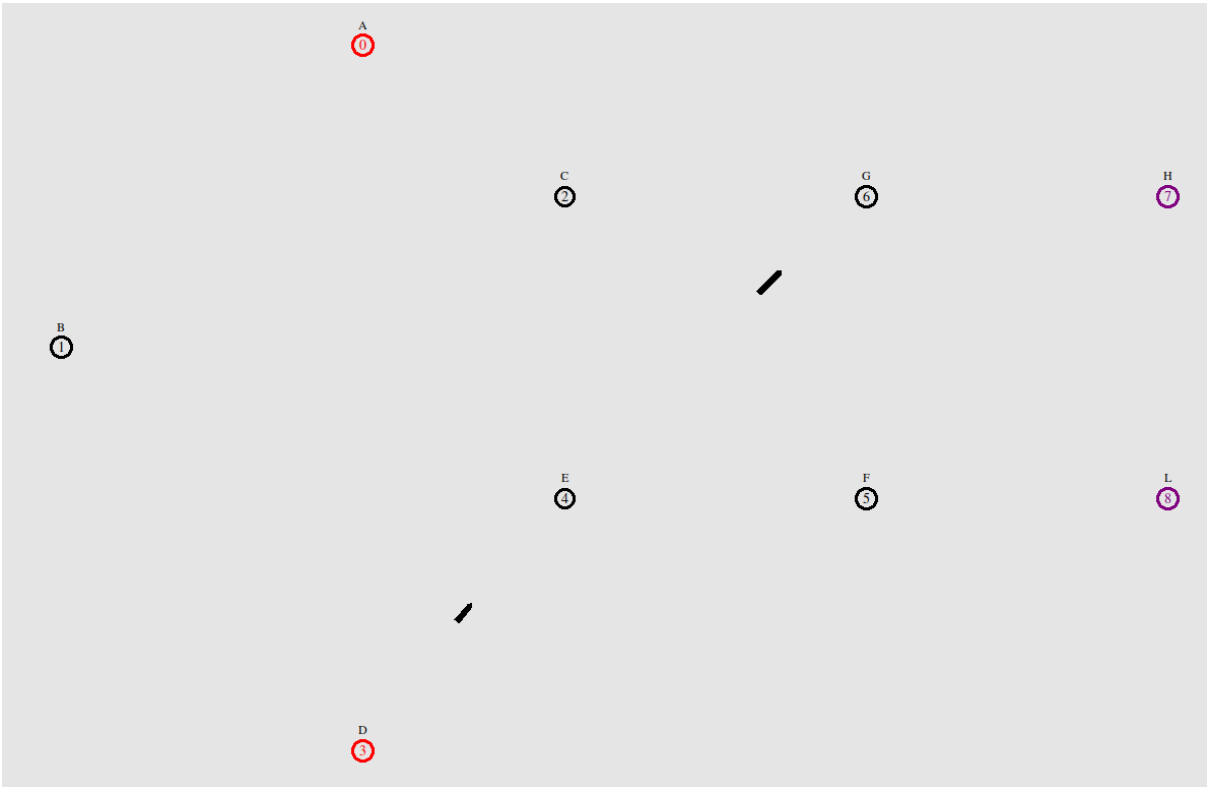
در نهایت شبیه سازی ران شده و بعد از 100 ثانیه به پایان می‌رسد.

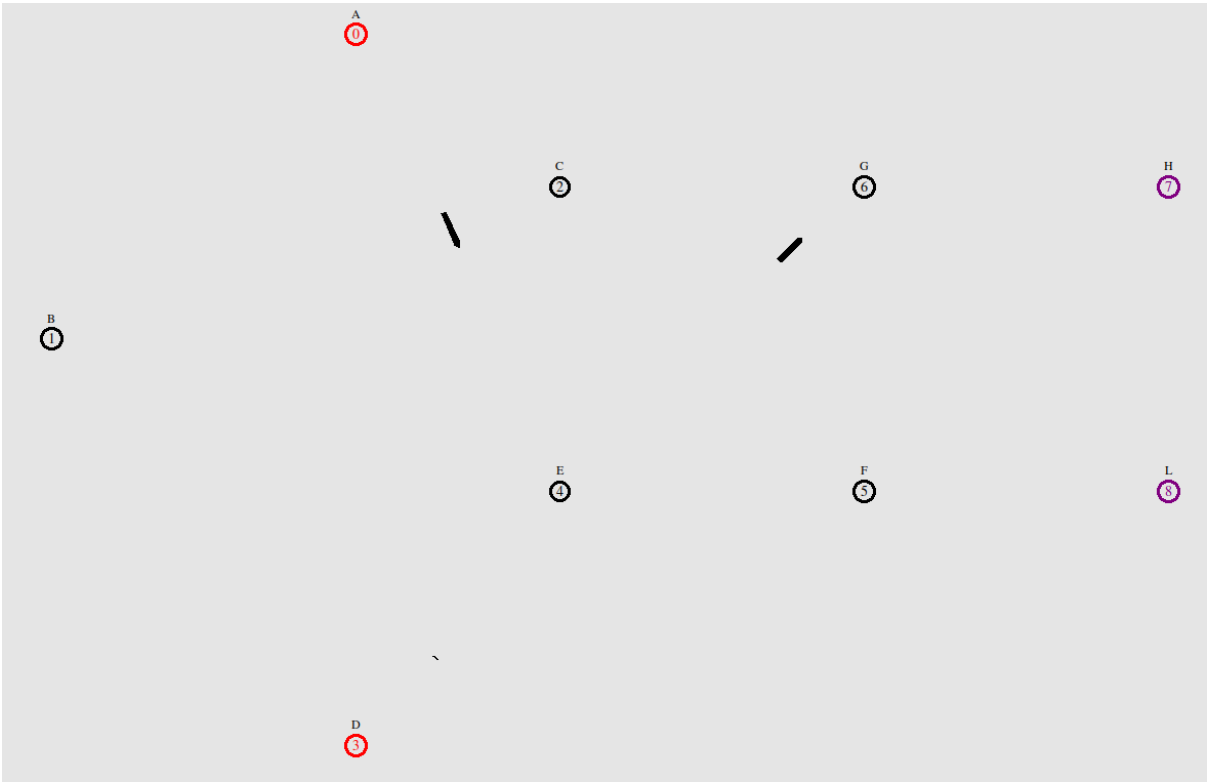
```
$ns  at 100.0 "finish"
# begin simulation
$ns run
```

به این صورت می‌توان شبکه را با متغیرهای مختلف آن تعریف و شبیه سازی کرد.

نمونه‌ای از سیمولیشن:






A
0
C
2
G
6
H
7
B
1
E
4
F
5
L
8
D
3

توضیحات کد پایتون:

```
1
2   import numpy as np
3   import matplotlib.pyplot as plt
```

تصویر بالا کتابخانه‌های مورد نیازمان برای پیاده سازی پروژه به زبان پایتون را نشان می‌دهد. خط 2 به دلیل این است که برای بخش throughput بتوانیم لیست محور xها را آماده کنیم. خط۳۹ هم برای این است که بتوانیم نمودار بخش throughput را رسم کنیم.

در ابتدا اسکوپ main را تعریف می‌کنیم تا گام‌های تجزیه و تحلیل را به ترتیب قرار دهیم:

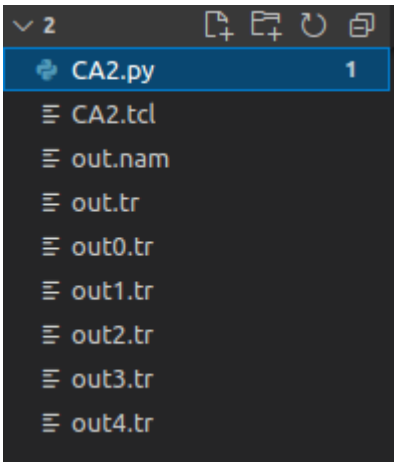
CA2.py > ...

```
70  def main():
71
72      throughput_array = [0.0]*5
73      throughput_array[0] = throughput("out0.tr", 100)
74      throughput_array[1] = throughput("out1.tr", 500)
75      throughput_array[2] = throughput("out2.tr", 1000)
76      throughput_array[3] = throughput("out3.tr", 2000)
77      throughput_array[4] = throughput("out4.tr", 5000)
78      print(throughput_array)
79
80      xpoints = np.array([100, 500, 1000, 2000, 5000])
81      plt.plot(xpoints, throughput_array, 'o')
82      plt.show()
83
84      print(packet_transfer_rate("out.tr"))
85      print(end_to_end_delay("out.tr"))
86
87  main()
```

بخش throughput:

```python
def throughput(filename):
    trace = open(filename ,  'r')
    recvdSize = 0
    startTime = 1e6
    stopTime = 0
    for line in trace:
        words = line.split(' ')
        packet_size = int(words[36])
        time = float(words[2])
        if line.startswith('s') and 'AGT' in line and packet_size > 512:
            if  time < startTime:
                startTime = time
        if line.startswith('r') and 'AGT' in line and packet_size > 512:
            if time > stopTime:
                stopTime = time
            packet_size -= packet_size % 512
            recvdSize += packet_size
    throughput = recvdSize / (stopTime - startTime)*(8/1000)
    trace.close()
    return throughput
```

برای اینکه بتوانیم throughput را به ازای packet sizeهای مختلف بدست آوریم، tcl را برای هر packet size ران می‌کنیم و خروجی‌های آن را تولید می‌کنیم که در محیط پروژه چنین وضعیتی را دارد:



حال تابع throughput را به ازای تمام tr خروجی‌ها که برای packet sizeهای مختلف بدست آورده بودیم اجرا می‌کنیم، نتیجه‌ی هر اجرا که خروجی throughput باشد را در لیستی از خروجی‌های throughput ذخیره می‌کنیم تا به ازای این لیست بتوانیم نمودار خروجی را رسم(plot) و تحلیل کنیم.

پیاده‌سازی الگوریتم throughput: ابتدا در خط ۱۲ فایل tr output را که خروجی ns است که توسط زبان tcl تولید شده‌است و وضعیت شبکه را خلاصه کرده است، می‌خوانیم و در متغیر trace ذخیره می‌کنیم. خط ۱۳ که متغیر recvdSize را مشاهده می‌کنیم برای این است که تعداد کل بایت‌های فرستاده شده را جمع کنیم. خط ۱۴ startTime متغیری است که زمان شروع عملیات شبکه را ذخیره می‌کند در ابتدا دیده می‌شود که مقدار 1e6 دارد تا ماکزیمم مقدار باشد.

در صورتی که شبکه agent باشد (یعنی AgentTraces are marked with AGT) و در حال ارسال باشد (شروع شدن با کاراکتر 's' که به معنی ارسال است) و حداقل طول پکت در new wireless متد را داشته باشد، از شرط گذر می‌کند و بسته را ارسال می‌کند. بنابراین زمان شروع کل کار شبکه را ذخیره می‌کنیم.

خط ۱۵ stopTime متغیری است که زمان آخرین پکت که وارد لینک شده و حداقل سایز ورودی را دارا است و به مقصد فرستاده می‌شود(شروع شدن با کاراکتر 'r' که به معنی دریافت است) را ذخیره می‌کند.

در خط ۱۶ تا ۲۷ اینگونه عمل می‌کنیم که کل خط های فایل tr output ورودی را می‌خوانیم، اگر این خط نشانه‌ای از وارد شدن پکت(چه ارسال چه دریافت) و در مرحله‌ی agt بودن و حداقل سایز پکت ورودی را داشت(خط ۲۰و ۲۳ کد) آنگاه یعنی پکت جدیدی منتقل می‌شود، پس باید طبق خط ۲۲و ۲۵ کد زمانی که این پکت فرستاده می‌شود را ذخیره کنیم(اگر طبق ۲۲ زمان آغازین بود برای اولین و آخرین بار ذخیره می‌کنیم و اگر طبق ۲۵ زمان نهایی شبکه بود آن را همواره با توجه به خطوط بعدی tr output آپدیت می‌کنیم).

در خط ۱۸ طبق ساختار tr output داده ۳۶ام سایز پکت را دارد.در خط ۱۹ هم داده‌ی دوم خط که نشان دهنده‌ی زمان است را ذخیره می‌کنیم. در خط ۲۰تا۲۲ همان‌طور که گفته شد بررسی می‌شود اگر این اولین پکت بود زمان شروع مقداردهی شود، سپس این عملیات تکرار می‌شود و تعداد بایت‌های کل پکت‌های لینک بدست می‌آید و در همان زمان آخرین پکت ارسالی در خطوط ۲۳تا۲۵ به‌روزرسانی می‌شود. در هر خط بعد از تشخیص دریافت پکت طبق خط ۲۶و۲۷ کد، ما سایز بسته‌های ارسالی در کل عملیات جمع می‌کنیم، اما چون یک ویژگی در متد new wireless وجود دارد که بعد از هر ۵۱۲ بیت داده‌ای که می‌خواند داده‌ای اضافه می‌کند که این متد Routing Information Protocol نامیده می‌شود باید این داده اضافه که جزو بیت‌های پکت نیست را کم کنیم که این هدف در خط ۲۶ام عملی شده و سپس در خط ۲۷ام سایز پکت جدید را به جمع‌مان اضافه می‌کنیم. در نهایت زمان شروع و پایان قرارگیری پکت داده‌ها با سایز ورودی را داریم و طبق فرمول throughput آن را حساب می‌کنیم و در خط ۲۹ فایل tr output ورودی را می‌بندیم و در خط ۳۰ام نتیجه‌ی محاسبات را ارسال می‌کنیم.

فرمول throughput:

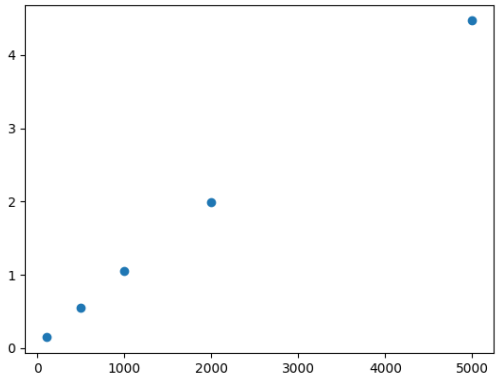
average throughput = (recvdSize/(stopTime-startTime))*(8/1000)

recvdSize = Store received packet's size

stopTime = Simulation stop time

startTime = Simulation start time

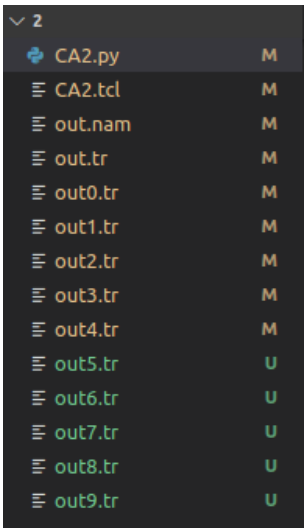
********** خروجی throughput ها **********



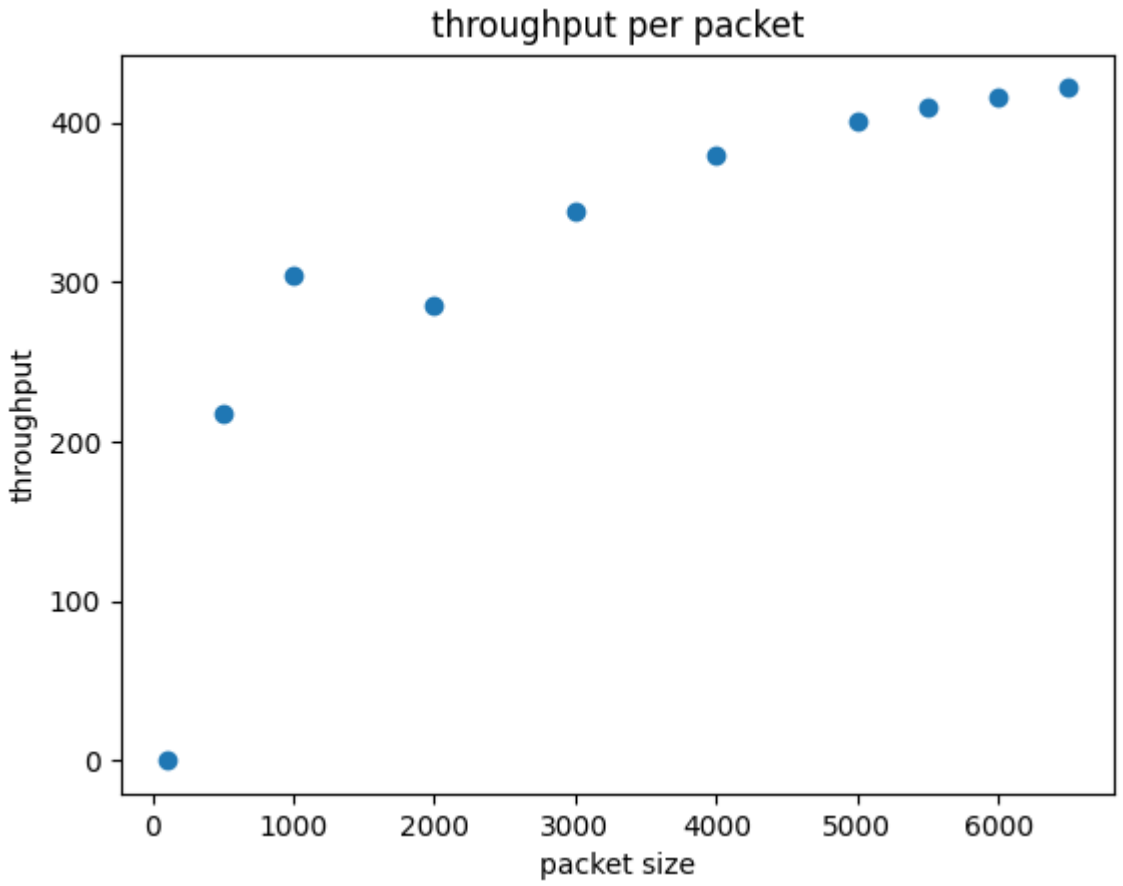
پلاتی که مشاهده می‌شود برای پکت سایزهای 100 500 1000 2000 5000 می‌باشد.

می‌بینیم که هر چه مقدار پکت سایز بزرگتر شود مقدار throuput نیز افزایش می‌یابد.

همین‌طور برای پکت سایزهای متفاوت تر [100, 500, 1000, 2000, 3000, 4000, 5000, 5500, 6000, 6500] امتحان می‌کنیم:



```
ryhn@ryhnap:~/CNca/2$ ns CA2.tcl 1.5 100
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
ryhn@ryhnap:~/CNca/2$ ns CA2.tcl 1.5 1000
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
ryhn@ryhnap:~/CNca/2$ ns CA2.tcl 1.5 2000
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
ryhn@ryhnap:~/CNca/2$ ns CA2.tcl 1.5 3000
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
ryhn@ryhnap:~/CNca/2$ ns CA2.tcl 1.5 4000
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
ryhn@ryhnap:~/CNca/2$ ns CA2.tcl 1.5 5000
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
ryhn@ryhnap:~/CNca/2$ ns CA2.tcl 1.5 5500
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
ryhn@ryhnap:~/CNca/2$ ns CA2.tcl 1.5 6000
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
ryhn@ryhnap:~/CNca/2$ ns CA2.tcl 1.5 6500
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
ryhn@ryhnap:~/CNca/2$ python3 CA2.py
```


throughput per packet
throughput
packet size
0
100
200
300
400
0
1000
2000
3000
4000
5000
6000

بخش packet_transfer_rate:

```python
def packet_transfer_rate(filename):
    trace = open(filename ,  'r')
    max = 0
    for line in trace:
        words = line.split(' ')
        index = line.find("-Ps")
        if index != -1:
            seq_num_str = line[index+4:]
            seq_words = seq_num_str.split(' ')
            seq_num = seq_words[0]
            if line.startswith('f') and 'tcp' in line and int(seq_num) > max:
                max = int(seq_num)
                time = float(words[2])
    trace.close()
    return max/time
```

packet transfer ration به این معناست که در هر ثانیه چند بسته ارسال می‌شود. برای پیدا کردن این مقدار طبق دستورالعمل زیر عمل می‌کنیم:

در ابتدا در خط ۳۳ trace file مورد نظر را باز می‌کنیم.

در نظر داریم برای پیدا کردن packet_transfer_ratio خطی را که با f شروع می‌شود و packet type آن برابر با tcp است و شامل شماره بیشترین packet idای که در این شبیه سازی ارسال شده است(یعنی ماکزیمم و نهایت تعداد پکتی که ارسال شده) را بیابیم. برای این کار یک حلقه بر روی تمامی خطوط فایل می‌زنیم(خطوط ۳۵ تا ۴۴)، در هر بار پیدا کردن خط با این مشخصات time و sequence number که در اینجا برای خوانایی max گذاشتیم را ذخیره می‌کنیم.

روش هم اینگونه است که flag ای که برای sequence number وجود دارد را پیدا می‌کنیم و موقعیت پشت آن را پیدا می‌کنیم(خط ۳۷) و سپس اگر این نشانه یافت شد به اندازه‌ی نشانه جلو می‌رویم "-Ps " و از آن پس داده‌ها را جدا می‌کنیم و زمان و شماره پکت را ذخیره می‌کنیم.

سپس در انتها آخرین sequence number ذخیره شده در max را تقسیم بر time می‌کنیم. مقدار بدست آمده مقدار packet transfer ratio است.

بخش average end-to-end-delay:

فرمول مورد نظر برای محاسبه average end-to-end-delay:

$$D = \frac{1}{n} \sum_{i=1}^{n} (Tr_i - Ts_i) * 1000 \text{ [ms]}$$----------

Where

D = Average E2E Delay

i = packet identifier

Tr_i = Reception time

Ts_i = Send time

n = Number of packets successfully delivered

کد محاسبه average end-to-end-delay:

CA2.py > ...

```python
end_to_end_delay(filename):
trace = open(filename ,  'r')
_max = 0

for line in trace:
    index = line.find("-Ps")
    if index != -1:
        seq_num_str = line[index+4:]
        seq_words = seq_num_str.split(' ')
        seq_num = seq_words[0]
        if int(seq_num) > _max:
            _max = int(seq_num)

start_time = [-1]*_max
end_time = [-1]*_max

trace1 = open(filename ,  'r')
for line in trace1:
    word = line.split(' ')
    index = line.find("-Ps")
    if index != -1:
        seq_num_str = line[index+4:]
        seq_words = seq_num_str.split(' ')
        seq_num = int(seq_words[0])
        if line.startswith('s') and "AGT" in line and start_time[seq_num-1] == -1:
            start_time[seq_num-1] = float(word[2])
        if line.startswith('r') and 'tcp' in line:
            end_time[seq_num-1] = float(word[2])

packet_duration = 0
for packet_id in range(_max):
    start = start_time[packet_id]
    end = end_time[packet_id]
    packet_duration += end - start
trace.close()

return packet_duration/_max
```

در ابتدا trace file مورد نظر را در خط ۴۹ باز می‌کنیم. در خط ۵۰ متغیر _max وجود دارد که وظیفه‌اش پیدا کردن ماکزیمم تعداد پکت منتقل شده است که مشابه بخش قبل در خط ۵۲ تا ۵۹ این خواسته عملی می‌شود. در خط ۶۱و۶۲ دو آرایه زمان شروع و اتمام انتقال پکت‌ها درست می‌شود که مقدار پیش‌فرض 1- دارند و طول این دو آرایه به اندازه‌ی تعداد پکت‌هاست.

سپس در خط ۶۵ تا ۷۵ تمام سطرهای فایل را دوباره می‌خوانیم و در حین خواندن سطرها داده‌های هر سطر را در words ذخیره می‌کنیم(خط ۶۶) سپس مشابه بخش‌های قبل مقدار sequence number را نیز از خط ۶۷تا۷۱ می‌یابیم، حال در خط ۷۲و۷۳ با توجه به agent بودن گره(یعنی فرستنده بودنش AgentTraces are marked with AGT) و وضعیت ارسالش (شروع شدن با کاراکتر 's' که به معنی ارسال است) و بار اول بودن مقداردهی زمانش(start_time[seq_num-1] == -1) زمان شروع این پکت را ذخیره می‌کنیم، همین‌طور در خط ۷۴و۷۵ با توجه به وضعیت ارسال که از نوع receive و دریافت است و با توجه به tcp بودن نوع packet زمان نهایی دریافت پکت را همواره به‌روزرسانی می‌کنیم.

در خط ۷۷ متغیر packet_duration را تعریف می‌کنیم و در خطوط ۷۸تا۸۱ زمان انتقال تمام پکت‌ها را در این متغیر جمع می‌زنیم.در نهایت این عدد صورت و تعداد پکت‌ها _max مخرج کسر average end-to-end delay ما خواهد بود.

و پس از تقسیم مقدار نهایی را در خط ۸۴ برمی‌گردانیم.

خروجی برنامه پایتون به ازای نرخ خطای ارسال‌های مختلف

نرخ خطا = 0.000001

```
throughput is:  918.4188083107147
packet transfer ratio is:  33.007593990143896
end to end delay is:  13.017956025975767
```

نرخ خطا = 0.000002

```
throughput is:  918.4188083107147
packet transfer ratio is:  33.007593990143896
end to end delay is:  13.017956025975767
```

نرخ خطا = 0.000003

```
throughput is:  918.4188083107147
packet transfer ratio is:  33.007593990143896
end to end delay is:  13.017956025975767
```

نرخ خطا = 0.000004

```
throughput is:  918.4188083107147
packet transfer ratio is:  33.007593990143896
end to end delay is:  13.017956025975767
```

مشاهده می‌شود که در نرخ خطاهای با فاصله کم تفاوتی در خروجی ایجاد نمی‌شود.

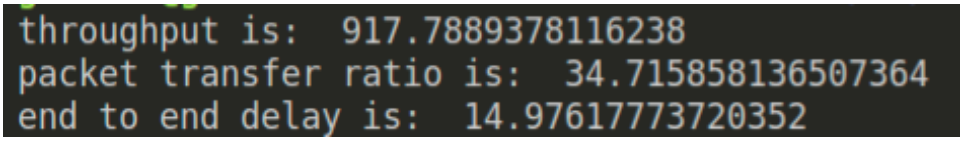
نرخ خطا = 0.000005

```
throughput is:  916.1681504488
packet transfer ratio is:  33.12733691284116
end to end delay is:  12.583437217986146
```

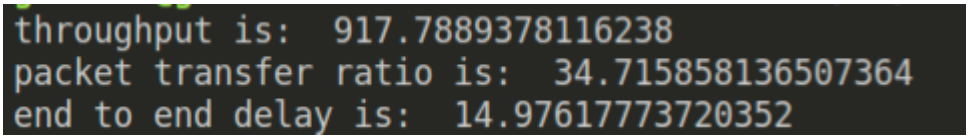
نرخ خطا = 0.000006

```
throughput is:  916.5860590878228
packet transfer ratio is:  33.40039468948422
end to end delay is:  13.999430813518702
```

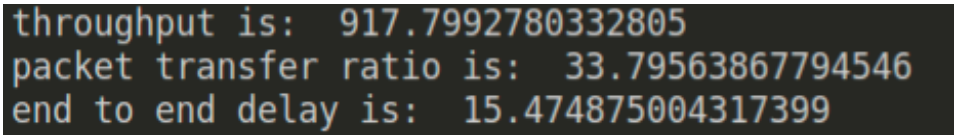
نرخ خطا = 0.000007



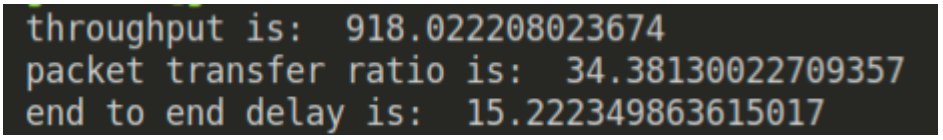
نرخ خطا = 0.000008



نرخ خطا = 0.000009



نرخ خطا = 0.00001



حال برای سه bandwidth شبیه سازی را انجام میدهیم و نتیجه را نشان می‌دهیم.

bandwidth = 1.5

```
throughput is:  304.2309931334358
packet transfer ratio is:  11.352477318053143
end to end delay is:  20.186036944370688
```

bandwidth = 55

```
throughput is:  918.022208023674
packet transfer ratio is:  34.38130022709357
end to end delay is:  15.222349863615017
```

bandwidth = 155

```
throughput is:  952.0764925942793
packet transfer ratio is:  35.077818955763114
end to end delay is:  15.778255522169493
```

هر چه bandwidth افزایش می‌یابد packet transfer ratio ، average end-to-end delay ، throuhput نیز افزایش می‌یابد.