

به نام خدا

امیرحسین محمد امیری

9527113

گزارش پروژه شماره 3

پیاده سازی پروتکل ارسال داده **go_back_N**

در این پروژه قصد داریم توسط پروتکل go_back_N یک کانال ارسال داده غیر مطمئن (udt) را به یک کانال ارسال داده مطمئن تبدیل کنیم ؛ و همچنین تاثیر پارامتر های مختلف همچون :

✚ احتمال دیسکارد (discard) شدن بسته ها

✚ طول پنجره (N)

✚ بیشینه مقدار اندازه سگمنت ها (MSS)

را در ارسال داده ها بررسی کنیم.

توجه : برای زیبایی، در کد سمت سرور از کتابخانه preattytable و از command line های lolcat و figlet

استفاده کرده ام که در صورت استفاده از محیط لینوکس پازیتوری های این دو command line را می توان به راحتی با دستور های زیر نصب کرد و در صورت عدم استفاده از محیط لینوکس می توان خط فلان کد سمت سرور را پاک کرد و خط فلان را از حالت کامنت خارج نمود تا بدود ارور کد اجرا شود.

```
Sudo apt-get install lolcat
```

```
Sudo apt-get install figlet
```

```

1) #!/usr/bin/python
2) import sys
3) import time
4) import socket
5) import numpy as np
6) import threading
7) from socket import *
8)
9) # *****
10) # ***** *this is the client code* *****
11) # *****
12)
13) *** in this part we check the number of input arguments *****
14) if len(sys.argv) != 6:
15)     print "please check your input arguments"
16)     print("1) servers ip")
17)     print("2) servers port")
18)     print("3) the file_name that you want to send")
19)     print("4) the window size")
20)     print("5) max of segment size")
21)     exit(2)
22) -----
23)
24) ***** initializing the arguments *****
25) ip = sys.argv[1] #-----this is the sever ip
26) sock = int(sys.argv[2] )#-----this is the sever port
27) file_name =sys.argv[3]
28) N = int(sys.argv[4])
29) Mss = int(sys.argv[5])
30) s=socket(AF_INET,SOCK_DGRAM)#--creating the socket
31) header_1 = ' 01010101010101'
32) data = []
33) base = 1
34) nextseqnum = np.uint32(1)
35) timer = [time.time(),True]
36) stopflg = True

```

همان‌طور که در شکل مشخص است در خطوط ۲ تا ۷ کتاب‌خانه های مورد نیاز اضافه شده اند. لازم به ذکر است که کتاب‌خانه numpy برای جمع کردن به صورت ۳۲ بیت اضافه شده است.

■ در خطوط بین ۱۴ تا ۲۱ تعداد آرگومان های ورودی چک می‌شوند که در صورت اشتباه بودن در خروجی جمله (please check your input arguments) را چاپ میکند و از برنامه خارج می‌شود.

■ در خط های ۲۵ و ۲۶ مقادیر آدرس و شماره پورت سرور مشخص شده اند.

■ در باکس (initializing the arguments) متغیر های اولیه تعریف و مقدار دهی شده اند. در خط ۳۱ متغیر header_1 با توجه به خواسته های پروژه تعریف شده است؛ توجه شود که این هدر به صورت رشته در کد استفاده شده است اما اگر هم بخواهیم به صورت عددی استفاده کنیم تفاوتی نمی‌کند و فقط نیاز است به جای خط ۳۱ ، header_1 = 0b0101010101010101 نوشته شود و در خط ۸۳ عبارت header-1 را به صورت str(header_1) نوشت .

```
37)sample_Rtt = ()
38)st1 = 0
39)st2 = 0
40)estimated_Rtt = 0
41)DevRTT = 0
42)stopflg2 =0
43)ss=True
44)#-----
45)
46)*** in this part each Mss bytes of data is appended to a list *****
47)text=[]
48)a = 1
49)with open(file_name,'rb') as ob:
50)while a :
51)    a = ob.read(Mss)
52)    text.append(a)
53)counter = 0
54)print('the number of packet is {}'.format(len(text)))
55)#-----
56)
57)***** time function *****
58)def start_timer():
59)    global timer
60)    timer = [time.time(),1]
61)#-----
```

از دو متغیر st1 و st2 که در خطوط ۳۸ و ۳۹ تعریف شده اند در ادامه برای اندازه گیری زمان کل ارسال استفاده شده است. همچنین از متغیر stopflg برای زمانی که کل فایل ارسال شده است استفاده می کنیم تا از حلقه while مربوطه خارج شویم.

در خطوط ۴۹ تا ۵۴ فایل مورد نظر برای ارسال را به قسمت های به اندازه ی MSS تقسیم کرده ام و هر قسمت در یک متغیر لیست به نام text قرار داده ام.


کارکرد تابع های start_timer() و make_pkt() کاملاً واضح است فقط باید توجه شود timer در کد زمانی روشن محسوب می شود که timer[1] برابر یک باشد و اگر برابر صفر باشد خاموش محسوب می شود.


توجه شود که در خط ۵۹ تایمر به صورت global در تابع make_pkt() تعریف شده است، این کار باعث می شود که این متغیر که در اصل در خارج از این تابع تعریف شده است درون تابع قابل شناسایی باشد.

```

62)
63)#####the function takes the data as input and returns the checksum in string #####
64)def check_sum_maker(arg):
65)    f = []
66)    temp = []
67)    checksum = 0
68)    a = bytearray(arg)
69)    for i in range(0,len(a)):
70)        f.append(a[i])
71)    for z in range(0,len(f)-1,2):
72)        temp.append((f[z]<<8) + f[z+1])
73)    for i in range(0,len(temp)):
74)        checksum = checksum +temp[i]
75)        checksum2 = checksum - 65535
76)        if(checksum2 >= 0):
77)            checksum = -65535 + checksum
78)            checksum = 65535 - checksum
79)    return str(checksum)
80)#####
81)
82)##### a fucntion for making packets ready for sending #####
83)def make_pkt(arg,arg2):#####arg stands for main data
84)    global header_1    #####arg2 stands for nextseqnum
85)    t = check_sum_maker(arg) + str(arg2) + header_1 + arg
86)    return t
87)#####

```

 در خطوط ۶۴ تا ۷۹ تابعی برای محاسبه checksum ایجاد شده است. این تابع یک ورودی گرفته و آن ورودی را توسط تابع **bytearray()** به بایت های آن تقسیم می کند. چون checksum استفاده شده در udp ۱۶ بیتی است پس باید داده ها را ابتدا از ۸ بیت به ۱۶ بیت تبدیل و سپس عمل جمع هر ۱۶ بیت انجام می شود. توجه شود که عدد ۶۵۵۳۵ همان $2^{10} - 1$ است که برای چک کردن سرریز هنگام جمع هر دو ۱۶ بیت استفاده می شود. در صورت سرریز شده عدد یک هنگام جمع دو عدد باید آن یک باز هم با حاصل جمع این دو عدد جمع شود که این اتفاق در خطوط ۷۳ تا ۷۹ صورت می گیرد. و در خطوط ۶۹ تا ۷۰ نیز داده به صورت ۱۶ بایت کنار هم قرار می گیرد.

 در خطوط ۸۱ تا ۸۴ تابعی برای درست کردن پکت نوشته شده است ترتیب هدر ها و داده به صورت زیر در

checksum

Next_seq_num

0b01010101010101

Main data

این تابع کنار هم قرار می گیرند.

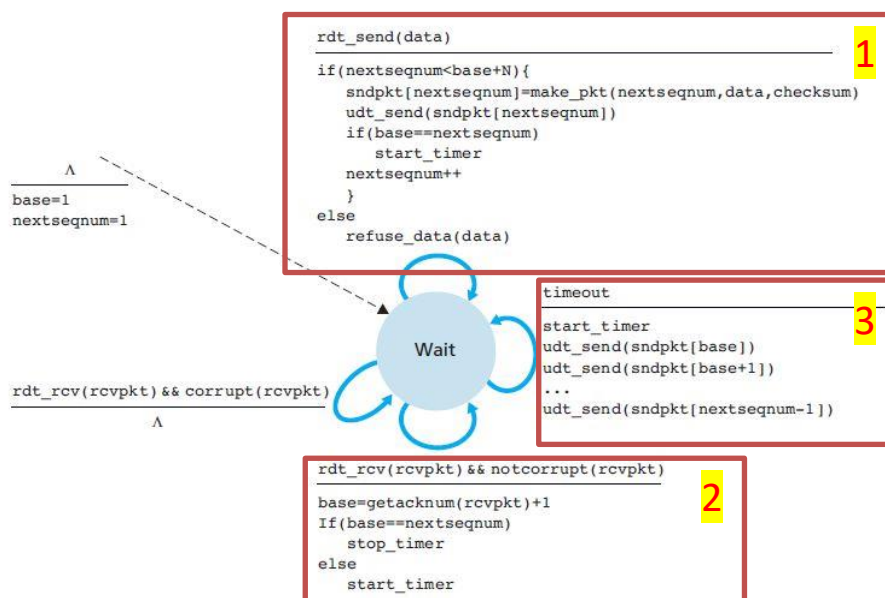
```

88)
89)***** fuction for estimating the RTT *****
90)def Rtt(sample_rtt):
91)    global estimated_Rtt , DevRTT
92)    alpha = 0.125
93)    beta = 0.25
94)    estimated_Rtt = (1 - alpha) * estimated_Rtt + alpha * sample_rtt
95)    DevRTT = (1 - beta) * DevRTT + beta * abs(sample_rtt - estimated_Rtt)
96)    return (estimated_Rtt + 4 * DevRTT)
97)#-----
98)
99)# ***** the function for reliable data transfer*****
100) def rdt_send(content_of_file):
101)     global nextseqnum , N , base , sample_Rtt,st1,ss
102)     if nextseqnum < (base + N) and ss:
103)         snd_pkt = make_pkt(content_of_file,nextseqnum)
104)         s.sendto(snd_pkt,(ip,sock))
105)         st1 = time.time()
106)         if base == nextseqnum :
107)             start_timer()
108)             nextseqnum = np.uint32( nextseqnum + 1 )
109)             return True
110)         else :
111)             return False
112) # -----
  
```

■ در خطوط ۹۰ تا ۹۷ تابعی برای محاسبه تخمین **Rtt** نوشته شده که در آن از فرمول های موجود در کتاب استفاده شده است که این فرمول ها را می توان در صفحات ۲۳۹، ۲۴۰ و ۲۴۱ کتاب یافت.

■ در تابع **rdt_send()** بخشی از پروتکل gbn پیداده سازی شده این قسمت در تصویر شماره (۱) که از کتاب آورده شده است با شماره ی ۱ مشخص شده است. توجه شود که این تابع داده را ارسال می کند و در صورت ارسال مقدار True را برمی گرداند و در غیر این صورت مقدار False را برمی گرداند که به این معنی است در ایجاد یا اتصال سوکت مشکلی وجود داشته و یا شماره next sequence number پکت مورد نظر خارج از پنجره ی ارسال است.

■ در خط ۱۰۵ زمان ارسال پکت را در متغیر st1 قرار می گیرد و در ادامه با احتساب زمان دریافت تصدیق این پکت (در خط ۱۳۴) مقدار rtt را برای این پکت استفاده می کنیم و از این مقدار برای محاسبه ی تخمین rtt بعدی توسط تابع **Rtt()** استفاده می کنیم.



تصویر شماره ی ۱

```

88)##### main function for sending the fil#####
89)def gbn_send():
90)    time_for_hole_packet = time.time()
91)    global data ,stopflg,text,counter
92)    while True :
93)        try:
94)            if rdt_send(text[counter]):
95)                counter = counter + 1
96)        except:
97)            if acknum==len(text)-1 or acknum==len(text):
98)                stopflg = False
99)                s.sendto("close",(ip,sock))
100)                time_for_hole_packet = time_for_hole_packet - time.time()
101)                print(' sending time is : %d ')%(-time_for_hole_packet)
102)                break
103)
104)    if counter > (len(text)) and stopflg2==len(text)-1:
105)        stopflg = False
106)        s.sendto("close",(ip,sock))
107)        time_for_hole_packet = time_for_hole_packet - time.time()
108)        print('the hole time for sending the file is : %d ')%(-time_for_hole_packet)
109)    break
110)# -----
111)
112)##### function for recieving the acknum #####
113)def gbn_rcv():
114)    global nextseqnum , base ,timer,st2,stopflg2,text
115)    while True:
116)        if stopflg:
117)            try :
118)                ack = s.recv(1024)
119)                acknum=int(ack.partition('0000'+str(0b1010101010101010))[0])
120)                st2 = time.time()
121)                s.settimeout(10)
122)            except :
123)                pass
124)            elif not stopflg :
125)                break
126)        base = np.uint32(base + 1)
127)        if(base == nextseqnum):
128)            timer[1] = False
129)        else:
130)            timer = [time.time(), True]
131)# -----

```


■ ■ تابع **gbn_send()** در یک حلقه‌ی همواره درست پکت‌ها را ارسال می‌کنیم و در هر بار اجرای حلقه دو مورد بررسی می‌شود:

1) این که آیا تابع **rdt_send()** مقدار صحیحی را برگشت داده یا خیر (اگر این تابع مقدار صحیحی بدهد به این معنی است که پکت ارسال شده و ممکن است پنجره‌ی آن هنوز جا برای ارسال دارد).

2) با یک متغیر به نام **counter** بررسی می‌کنیم که آیا به انتهای فایل رسیده ایم یا خیر و آیا تصدیق آخرین پکت هم دریافت شده یا خیر، در صورت اتفاق افتادن این دو مورد ابتدا عبارت **close** را ارسال می‌کنیم تا سرور متوجه به انتها رسیدن فایل شود و سپس مقدار متغیر **stopflg** را False قرار می‌دهیم تا در تابع‌های دیگر نیز متوجه به پایان رسیدن فایل شویم. (توجه شود که عبارت **close** بدون هدرها و به صورت ساده فرستاده شده است که در اصل باید توسط هدرها ارسال می‌شد)

■ ■ در تابع **gbn_send()** ابتدا بررسی شده که در صورت تمام شدن ارسال فایل

■ ■ در خطوط ۱۱۳ تا ۱۳۰ تابعی برای دریافت تصدیق‌ها نوشته شده است که مطابق الگوریتم نوشته شده در کتاب می‌باشد. به تصویر یک شماره ۲ نگاه کنید. در این تابع ابتدا در خط ۱۱۶ بررسی شده که آیا به انتهای فایل رسیدیم یا خیر در صورتی که به انتهای فایل رسیده باشیم از حلقه با دستور **break** خارج می‌شویم.

■ ■ در خط ۱۱۸ پکت تصدیق را دریافت کرده ایم و در خط ۱۱۹ با استفاده از تابع **partition** (که در پروژه قبلی به صورت کامل توضیح داده شده) پکت را تجزیه و عدد موجود در پکت را استخراج کردیم.

```

132)
133)##### function for considering the time out #####
134)def time_out():
135)    global timer , nextseqnum , stopflg,ss,counter
136)    while 1 :
137)        if not stopflg:
138)            print("+++++++=====+++++++=====")
139)            break
140)        if ((time.time()-timer[0]) > 0.1 ) and (stopflg) and timer[1]:
141)            print("Timeout -----> sequence_number = %d "%(nextseqnum))
142)            ss=False
143)            for q in range(base,nextseqnum):
144)                snd_pkt = make_pkt(text[q-1],q)
145)                s.sendto(snd_pkt,(ip,sock))
146)            ss=True
147)            timer = [time.time(),1]
148)#-----
149)
150)thread1 = threading.Thread(target=gbn_send)
151)thread2 = threading.Thread(target=gbn_rcv)
152)thread3 = threading.Thread(target=time_out)
153)
154)thread1.start()
155)thread2.start()
156)thread3.start()
157)thread1.join()
158)thread2.join()
159)thread3.join()

```

در ادامه `time_out()` وظیفه دارد که بررسی کند که آیا تایمر منقضی شده یا خیر و در صورت منقضی شده تایمر پکت ها مطابق الگوی مشخص شده در شکل ۱ شماره ۳ مجددا ارسال کند.

توجه شود که متغیر `ss` ارسال مجدد پکت ها مقدار `false` دارد تا توسط آن مانع ارسال پکت ها توسط `Rdt_send()` در این بازه شویم.

```
#!/usr/bin/python
import socket
import random
import numpy as np
from socket import *
import sys
import os
from prettytable import PrettyTable
if len(sys.argv) != 3:
    print "check your input argument \n you shoud print : "
    print "1) a name that you want to save the file with that name"
    print "2) the probability of discard"
    exit(2)

# *****
#***** * this is server code *****
# *****
serverSock = socket(AF_INET, SOCK_DGRAM)
serverIp = '127.0.0.1'
serverPort = int(7735)
serverSock.bind((serverIp, serverPort))
acknum = np.uint32(1)
header_1 = '01010101010101'
header_2 = str(0b10101010101010)
filename =sys.argv[1]
pro =float(sys.argv[2])
tabl = PrettyTable(["event ", "seqnum"])
def check_sum_maker(arg):
    f = []
    temp = []
    checksum = 0
    a = bytearray(arg)
    for i in range(0,len(a)):
        f.append(a[i])
    for z in range(0,len(f)-1,2):
        temp.append((f[z]<<8) + f[z+1])
    for i in range(0,len(temp)):
        checksum = checksum +temp[i]
        checksum2 = checksum - 65535
        if(checksum2 >= 0):
            checksum = -65535 + checksum
        checksum = 65535 - checksum
    return str(checksum)

#
```

در تصویر بالا نکته خاصی برای توضیح وجود ندارد، و صرفاً متغیرها ایجاد و مقدار دهی شده‌اند.

همچنین تابع `checksum` کاملاً مشابه کد سمت کلاینت می‌باشد.

کتابخانه `random` برای در نظر گرفتن احتمال دیسکار اضافه شده است (تابع `random.random()` یک عدد تصادفی با احتمال یکنواخت بین صفر و یک برمی‌گرداند)

```
46)with open(filename,'wb') as f :
47)    i = 0
48)    while True:
49)        discard = random.random()
50)        msg, addr = serverSock.recvfrom(100000)
51)        tem = msg.partition(header_1)
52)        recieving_data = tem[2]
53)        checksum_recieving = check_sum_maker(recieving_data)
54)        tem = tem[0].partition(checksum_recieving)
55)#we could find checksum it means that the chechsum is True
56)        recieving_seqnum = tem[2]
57)        if discard >= pro:
58)            if recieving_seqnum == str(acknum) + " ":
59)                f.write(recieving_data)
60)                serverSock.sendto((str(acknum) + '0000' + header_2 ), addr)
61)                acknum = np.uint32(acknum + 1)
62)        else :
63)            tabl.add_row(['Packet loss',acknum])
64)        if msg == "close":
65)            f.close()
66)            break
67)print(tabl)
68)os.system("figlet -f slant d o n e !!lolcat -a -d 2")
```

در خط ۶۴ فایل با نام خواسته شده باز شده است.

در خط ۵۰ پکت و آدرسی که پکت از آن آمده را دریافت می‌کنیم.

می‌دانیم که پکت دریافتی محتویات درون آن به شکل زیر کنار هم قرار گرفته‌اند.

checksum

Next_seq_num

Header_1

Main data

پس `msg.partition(header_1)` آن را به سه قسمت زیر تبدیل می‌کند.

checksum

Next_seq_num

Header_1

Main data

پس داده اصلی را بدست از پکت جدا کردیم. با علم به این که در سمت فرستنده checksum فقط روی داده اصلی گرفته شده با بدست آوردن مقدار چکسام روی داده تجزیه شده (خط ۵۳) و پیدا کردن این مقدار در پکت دریافت شده توسط تابع partition می‌توانیم next_seq_num ایی که در پکت وجود دارد را نیز بدست آوریم.

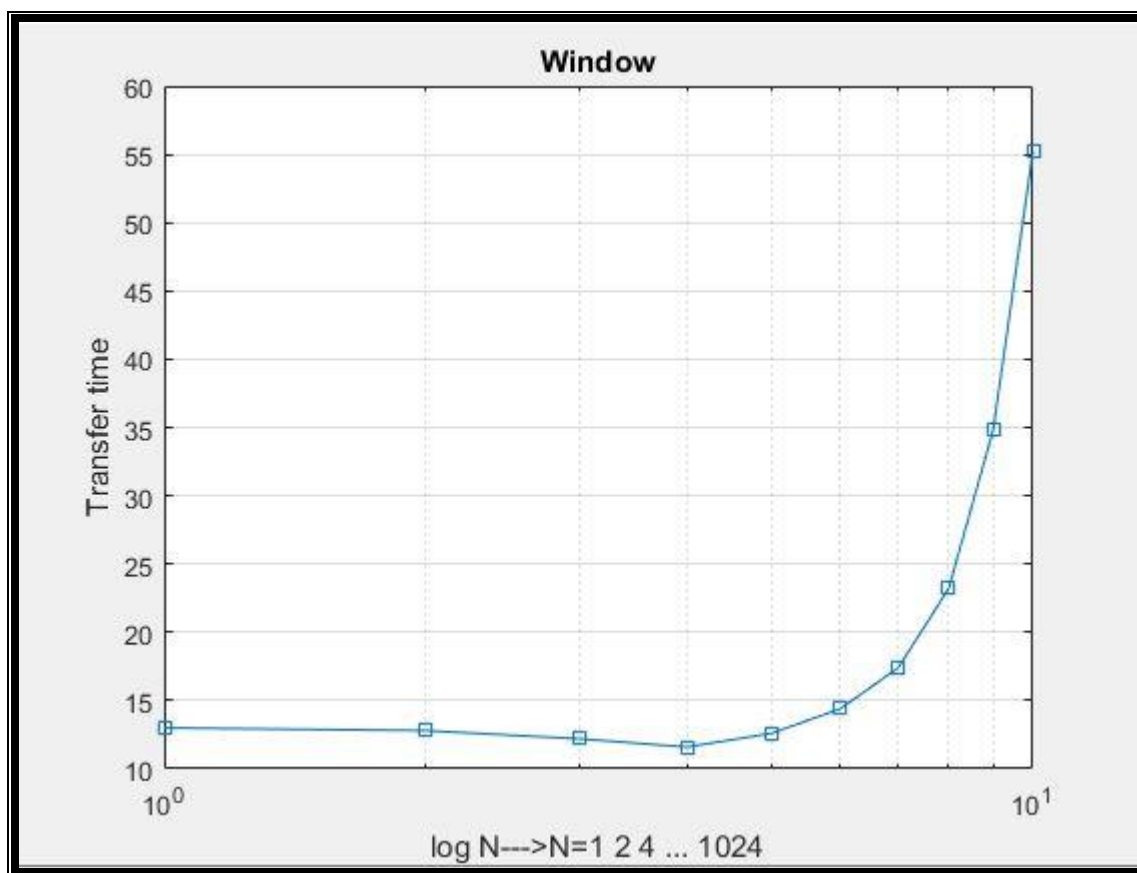
بعد از این که nextseqnum را بدست آوریدم آن را با مقدار مورد انتظار مقایسه می‌کنیم (خط ۵۸) و اگر برابر بودن داده را در فایل می‌نویسیم و تصدیق آن را انطور که در صورت پروژه گفته شده ایجاد و سپس ارسال می‌کنیم. (خط ۶۰)

در انتها می‌خواهیم ببینیم که تغییر پنجره ارسال، احتمال دیسکارد شدن و MMS چه تاثیری روی زمان ارسال فایل دارد. نتایج را در جدول ها و تصاویر زیر می‌بینید.

✓ تغییرات طول پنجره

N	1	2	4	8	16	32	64	128	256	512	1024
1	14	13	11	12	13	11	15	18	21	32	56
2	16	13	13	14	10	12	14	14	24	31	58
3	14	13	14	13	11	14	14	15	21	39	54
4	14	13	13	12	11	13	15	20	25	39	51
5	14	13	13	11	13	13	14	20	25	33	57
میانگین	14.4	13	12.8	12.2	11.6	12.6	14.4	17.4	23.2	34.8	55.2

جدول (۱) $P=0.05$, $MSS=500$



شکل ۲

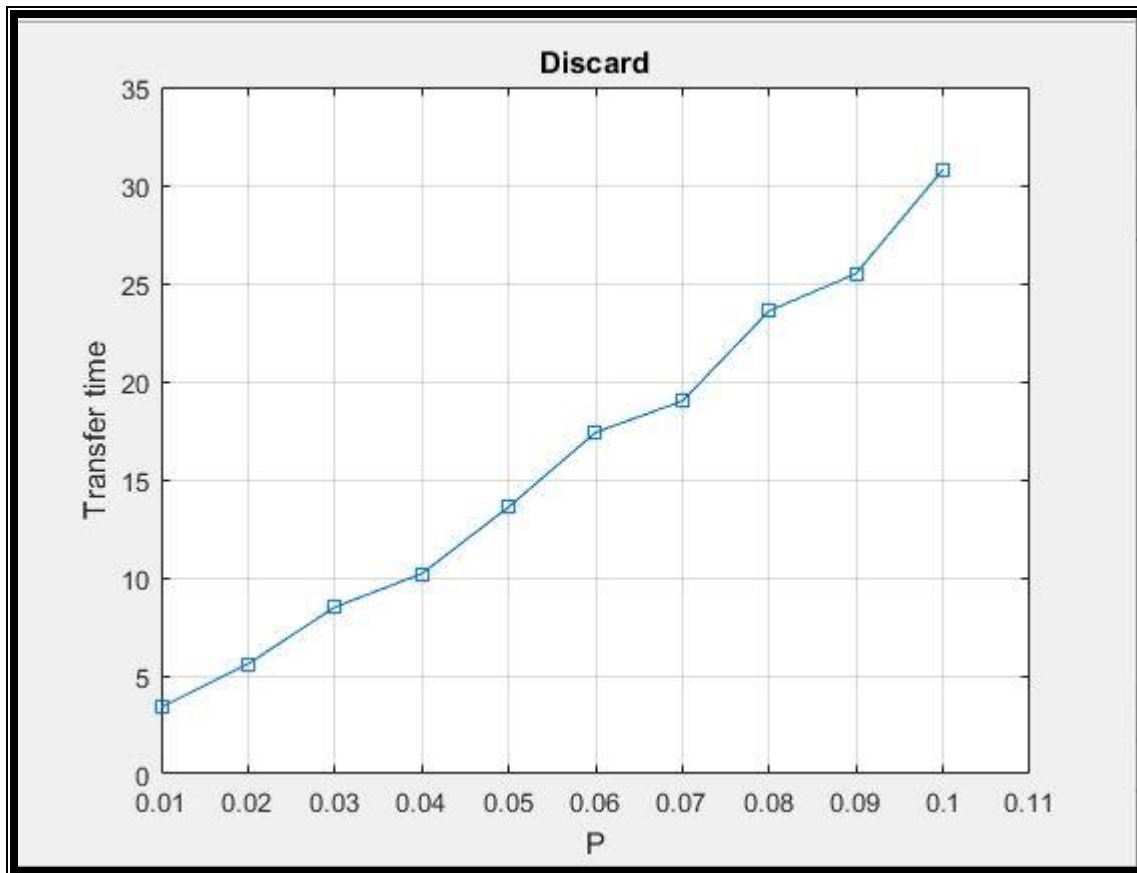
هنگامی که طول پنجره مقدار کمی دارد، به طور مثال برابر ۱ یا ۲ است کل پهنای باند ما صرف ارسال همین ۱ یا ۲ عدد پکت می‌شود و در بسیاری از مواقع از ظرفیت کانال استفاده نمی‌شود بنابراین انتظار داریم که با افزایش طول پنجره استفاده از ظرفیت کانال بهبود یابد و زمان ارسال فایل کاهش یابد اما پروتکل gbn به نحوی عمل می‌کند که اگر پکتی از دست برود دوباره کل پکت‌های پنجره‌ی ارسال را باز ارسال می‌کند در صورتی که ممکن است خیلی از این پکت‌ها قبلاً به دست گیرنده رسیده باشند و این یعنی هر چه طول پنجره بیشتر باشد در کانال‌هایی که دارای packet loss هستیم بخش بیشتری از ظرفیت کانال هدر می‌رود به گونه‌ای که اگر طول پنجره خیلی بزرگ باشد درصد کوچکی از گم شدن پکت‌ها باعث می‌شود که ظرفیت کانال به شدت افت کند. در نتیجه توقع داریم با افزایش طول پنجره از مقدار ۱ ابتدا شاهد کاهش زمان ارسال فایل شویم و سپس افزایش افزایش این زمان را شاهد باشیم.

این موضوع در شکل ۲ دیده می‌شود. در پنجره با طول ۴ کمترین زمان ارسال فایل را بدست آورده‌ایم.

✓ تغییرات احتمال دیسکارد

P	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
1	4	7	9	9	16	17	18	24	23	29
2	3	6	7	12	11	18	21	26	27	30
3	3	5	9	9	14	16	20	21	27	32
4	4	5	8	11	12	18	18	23	24	31
5	3	5	9	10	15	18	18	24	26	32
میانگین	3.4	5.6	8.5	10.2	13.6	17.4	19	23.6	25.5	30.8

MSS=500 , N=64



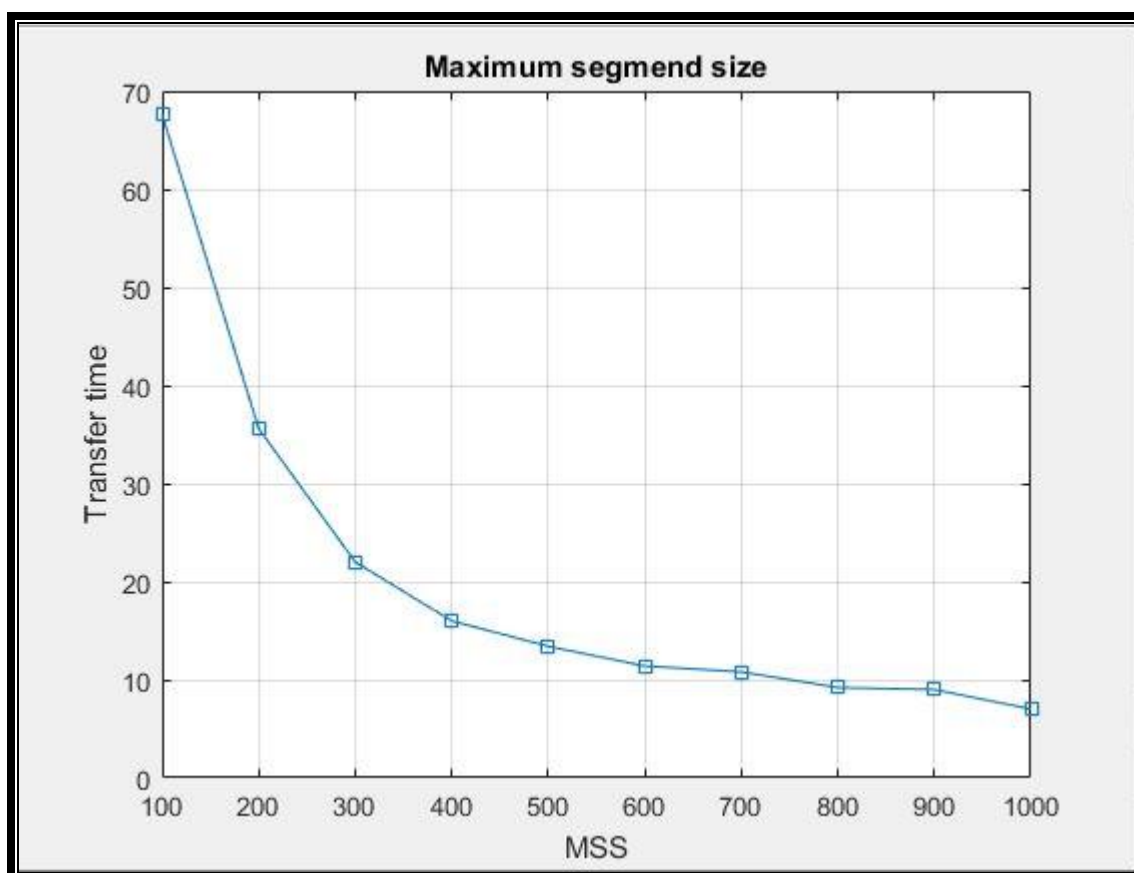
شکل ۳

با افزایش احتمال دیسکار شدن زمان مورد نیاز برای باز ارسال پکت ها افزایش می یابد بنابراین این شاهد کاهش ظرفیت کانال و افزایش زمان ارسال داده هستیم. شکل ۳ نیز این موضوع را نشان می دهد. همچنین با توجه به این شکل متوجه می شود که زمان ارسال داده با احتمال دیسکار شدن رابطه تقریباً خطی دارد.

✓ تغییرات MSS

MMS	100	200	300	400	500	600	700	800	900	1000
1	67	34	24	17	14	13	10	9	8	7
2	69	39	21	18	14	10	12	10	10	7
3	63	36	22	15	11	11	10	8	11	7
4	72	35	22	16	16	13	10	10	7	7
5	67	34	21	14	12	10	12	10	9	7
میانگین	67.6	35.6	22	16	13.4	11.4	10.8	9.2	9	7

جدول ۳) $P=0.05$, $N=64$



شکل ۴

می‌دانیم که هرچه نسبت $\frac{frame\ size}{bandwidth}$ بیشتر باشد یعنی از ظرفیت بیشتری از کانال استفاده کرده‌ایم. پس توقع داریم با افزایش MSS به پهنای باند بهتر و در نتیجه زمان ارسال کمتر برای داده دست پیدا کنیم. شکل ۴ نیز همین موضوع را نشان می‌دهد. با توجه به شکل ۴ متوجه می‌شویم که این کاهش زمان ارسال به صورت نمایی رخ می‌دهد.

باید توجه شود که مقدار MSS باید با توجه به پهنای باند انتخاب شود و نباید مقدار آن به گونه‌ای انتخاب شود که برای ارسال آن باید به ظرفیت بیشتری از ظرفیت کانال نیاز داشته باشیم.

فایلی که توسط آن کد ها اجرا شده‌اند در پوشه گزارش قرار داده شده است.

تصویر زیر نمونه‌ای از خروجی کد می‌باشد.

(به دلیل این که نتوانستم در هنگام اجرای این پروژه مشکلات gns را برطرف کنم از ip داخلی برای اجرای پروژه استفاده کردم به همین دلیل برای مشهود بودن مقدار ها برای time out از مقدار ثابت ۰.۱ ثانیه استفاده کردم).

Ubuntu - VMware Workstation

File Edit View VM Tabs Help

Mon 03:45

amiramri0200@ubuntu: ~/Desktop

amiramri0200@ubuntu: ~/Desktop

amiramri0200@ubuntu: ~/Desktop

```
Timeout -----> sequence_number = 65
Timeout -----> sequence_number = 65
Timeout -----> sequence_number = 65
^Timeout -----> sequence_number = 65
Timeout -----> sequence_number = 65
Timeout -----> sequence_number = 65
AZ
[47]+ Stopped python client_gbn.py 127.0.0.1 7735 a.png 64 700
amiramri0200@ubuntu:~/Desktop$ python client_gbn.py 127.0.0.1 7735 a.png 64 700
the number of packet is 1439.
Timeout -----> sequence_number = 102
Timeout -----> sequence_number = 103
Timeout -----> sequence_number = 177
Timeout -----> sequence_number = 274
Timeout -----> sequence_number = 295
Timeout -----> sequence_number = 491
Timeout -----> sequence_number = 786
Timeout -----> sequence_number = 809
Timeout -----> sequence_number = 857
Timeout -----> sequence_number = 874
Timeout -----> sequence_number = 918
Timeout -----> sequence_number = 1108
Timeout -----> sequence_number = 1137
Timeout -----> sequence_number = 1440
Timeout -----> sequence_number = 1440
the hole time for sending the file is : 2
+++++
amiramri0200@ubuntu:~/Desktop$
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Type here to search

3:15 PM 2/10/2020

Activities

File Edit View VM Tabs Help

amiramri0200@ubuntu: ~/Desktop

amiramri0200@ubuntu: ~/Desktop

amiramri0200@ubuntu: ~/Desktop

```
amiramri0200@ubuntu:~/Desktop$ python server_gbn.py ssjdf1.png 0.01
-----+-----+
event | seqnum
-----+-----+
Packet loss | 38
Packet loss | 38
Packet loss | 39
Packet loss | 113
Packet loss | 210
Packet loss | 210
Packet loss | 231
Packet loss | 231
Packet loss | 231
Packet loss | 427
Packet loss | 722
Packet loss | 745
Packet loss | 745
Packet loss | 793
Packet loss | 793
Packet loss | 810
Packet loss | 854
Packet loss | 854
Packet loss | 854
Packet loss | 1044
Packet loss | 1073
Packet loss | 1073
Packet loss | 1402
Packet loss | 1415
Packet loss | 1416
-----+-----+
done!
amiramri0200@ubuntu:~/Desktop$
```