به نام خدا

حدیث غفوری 9825413

توضیح کد پایتون

کد شامل تعداد بسته های دریافتی و تعداد باز ارسال ها

در این کد اول یک فایل را به حالت write باز کردم بعد که در حالت live برنامه شروع به گرفتن پکت ها میکند این فایل هم نوشته میشود و بعد از 10 ثانیه هم دیگه بسته ای گرفته نمیشود. سپس فایل را میبندم و تعداد بسته هایی که ارسال شده را با len(capture) حساب میکنم.

برای حساب کردن تعداد باز ارسال های tcp هم اون فایلی که قبلش نوشته بودم را به حالت tcp.analysis.retransmission میخونم و با یک حلقه ی فور تعداد ابجکت هاش رو حساب میکنم.

و بعد هم چاپ میکنم.

اخر برنامه هم برای جلوگیری از crash برنامه از os._exit(1) استفاده کردم.

```
import pyshark
import os

totPkt=0
totData=0
network_interface = 'lo'
file = "./pysharkOutput.pcap"
output = open(file, "w")
#save liveCapture to a .pcap file and use it later
capture = pyshark.LiveCapture(interface=network_interface , bpf
_filter='port 4040',output_file=file)
capture.sniff(timeout=10)
# try:
```

```
#     for pkt in capture:
#         try:
#             totData = totData+ int(pkt.tcp.len)
#             print(totData)
#             totPkt= totPkt + 1
#             print(totPkt)
#         except:
#             pass
# except:
#     pass


# totData = totData*0.000008
# print('whole throughput: ',totData/10)

output.close()

print("number of packets: ",len(capture))

#count number of retransmissions with .pcap file and use displa
y_filter methods:
cap = pyshark.FileCapture('pysharkOutput.pcap', display_filter=
'tcp.analysis.retransmission')

counter = 0
try:
    for packet in cap:
        try:
            counter+=1
        except:
            print(counter)

except:
    pass
print("number of retransmissions: ",counter)
```

```
os._exit(1)
```

کد شامل میزان گذردهی کل

برای این قسمت هم از فایل کپچر شده استفاده کردم و یک حلقه برای طی کردن همه ی ابجکت های این فایل گذاشتم و اطلاعات پکت های tcp , udp و header len , .. را حساب کردم.

برای جلوگیری از Crash کردن برنامه هم از try,except استفاده کردم.

دراخر هم چون طول بسته هایی که دریافت کردم برحسب بایت بود و من مگابیت میخاستم در 0.000008 ضرب کردم.

زمان را هم به صورت دقیق از فایل کپچر شده به کمک متد sniff_timestamp برحسب ثانیه محاسبه کردم.

در اخر هم برای گذردهی کل:

طول کل بسته ها رو به زمان محاسبه شده تقسیم کردم.

برای رند شدن اعداد هم از تابع round استفاده کردم.

اخر برنامه هم برای جلوگیری از crash برنامه از os._exit(1) استفاده کردم.

```
import os
import pyshark


cap = pyshark.FileCapture('pysharkOutput.pcap')

totPkt=0
totData_tcp=0

ref=float(cap[0].sniff_timestamp)
timediff =0
capturedDatas= 0
packets_length =0
packets_headers =0
try:
    for pkt in cap:
        try:
            # print("packet len: ",pkt.tcp.len)
            # print("captured: ",pkt.captured_length)
            # print("length: ",pkt.length)
            # print(pkt.tcp.hdr_len)
            capturedDatas +=  int(pkt.captured_length)
            packets_length += int(pkt.length)
            #if was tcp:
            try:
                packets_headers += int(pkt.tcp.hdr_len)
                totData_tcp = totData_tcp+ int(pkt.tcp.len)
                # print("header len: ",pkt.tcp.hdr_len)

            except :
                pass
```

```
            # print("captured Datas: ",capturedDatas)
            # print("packets length: ",packets_length)
            totPkt= totPkt + 1
            timediff=float(pkt.sniff_timestamp)-ref
            # print('time: ',timediff)
        except:
            pass

except:
    pass


capturedDatas = capturedDatas *0.000008
packets_length = packets_length * 0.000008
packets_headers = packets_headers * 0.000008
print("total packets: ",totPkt)
print("total packet headers size (Megabit): for TCP",packets_he
aders)
# print("total Data size(byte) for TCP: ",totData_tcp)
print("all captured packets size:(headers + datas)",capturedDat
as)
print("packets_length (headers + datas)",packets_length)
#convert to Megabit
totData_tcp = totData_tcp*0.000008
print("time (s): ",timediff)
print("data size(Mbit): for TCP ",totData_tcp)
print('whole throughput(Mbit/s) for TCP: ',totData_tcp/timediff
)
print('whole throughput(round) for TCP: ',round(totData_tcp/tim
ediff))

# print('whole throughput(Mbit/s) for UDP: ',capturedDatas/time
diff)
# print('whole throughput(round) for UDP: ',round(capturedDatas
/timediff))
```

```
os._exit(1)
```

در فولدر pictures همه ی اطلاعات و عکس های مربوط به بخش های مختلف و کانال های مختلف برای tcp , udp را قرار دادم :)

TCP

	کانال ایده ال	کانال با تاخیر ثابت	کانال با تاخیر تصادفی	کانال با گم شدگی کم	کانال با گم شدگی زیاد
میانگین گذردهی فرستنده	23	5	1	33	16
تعداد بسته دریافتی	728	176	87	1250	568
تعداد بازارسال ها	0	0	0	93	61

TCP is Impacted by Retransmission and Packet Loss

۱.افزایش تاخیر و گم شدن بسته ها چه تاثیری روی میانگین گذرده فرستنده و گیرنده در پروتکل TCP دارد؟چرا؟

همان طور که در نمودار های زیر مشخص است هم سمت کلاینت هم سرور گذردهی tcp با گم شدن بسته ها خیلی تغییر میکند.

برای گم شدگی گذردهی بین 0 تا 3500Mbps تغییر میکند.

برای کانال با تاخیر ثابت بین 0 تا 60Mbps تغییر میکند.

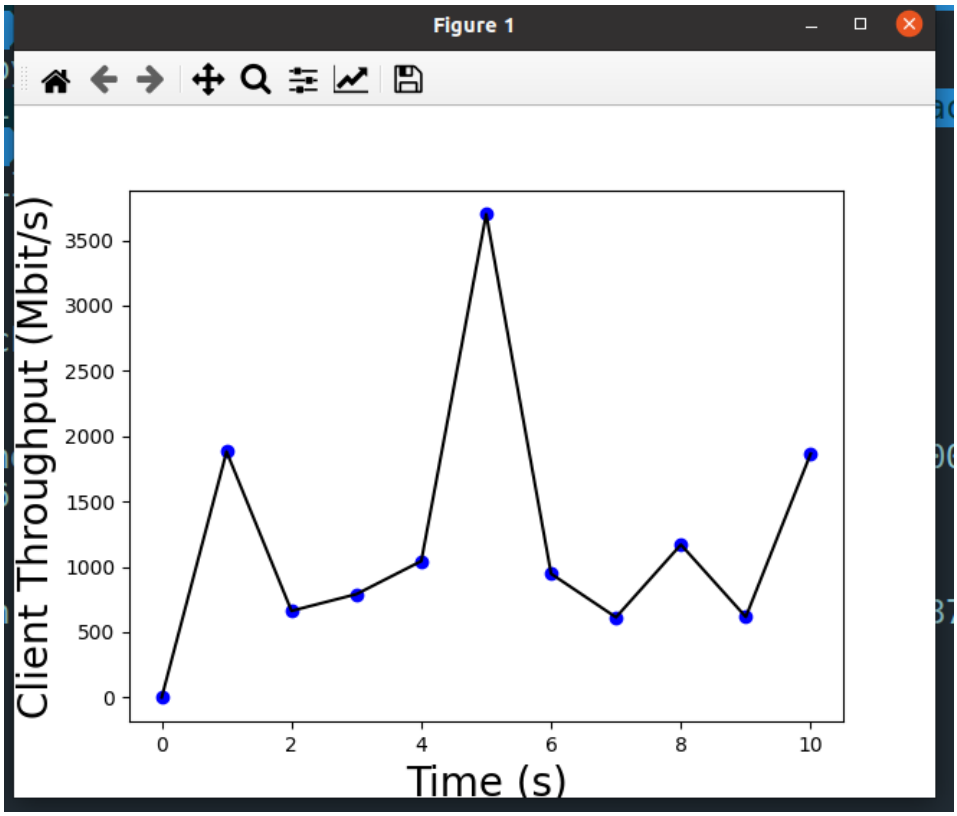
پس میفهمیم اثر تاخیر ثابت در کانال خیلی بیشتر از گم شدن بسته ها است و خیلی بیشتر بر گذردهی بسته ها تاثیر میگذارد.

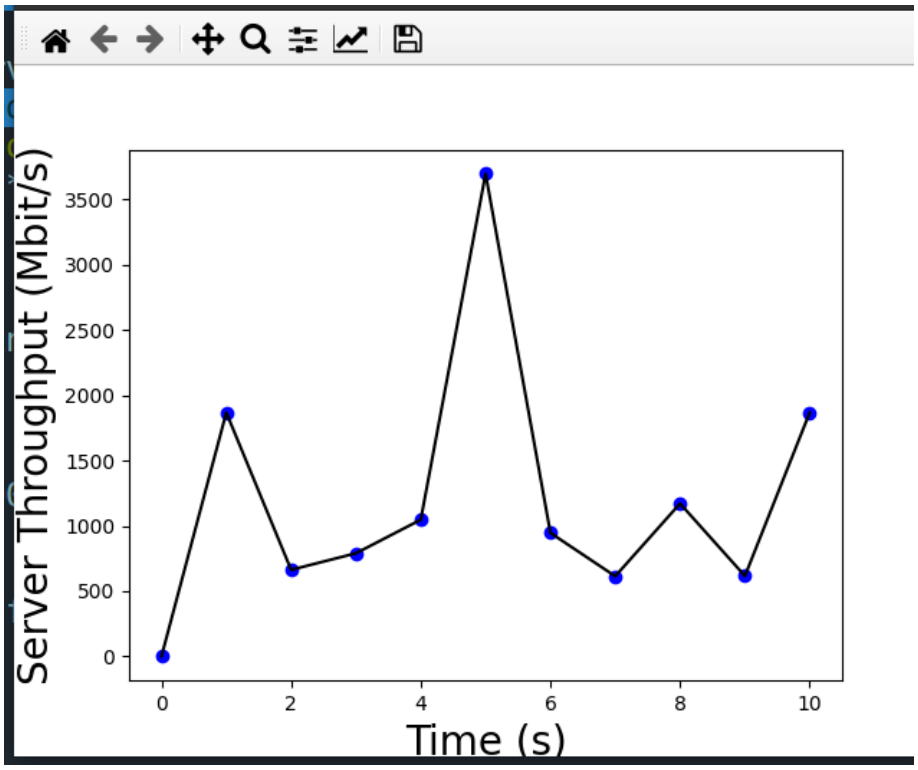
How the TCP congestion window impacts throughput

If we make the hypothesis that no packet gets lost; the sender will send the first quota of packets (corresponding to the TCP congestion window) and when it will receive the acknowledgment packet, it will increase the TCP congestion window; progressively the number of packets that can be sent in a given period of time will increase (throughput). The delay before acknowledgment packets are received (= latency) will have an impact on how fast the TCP congestion window increases (hence the throughput).

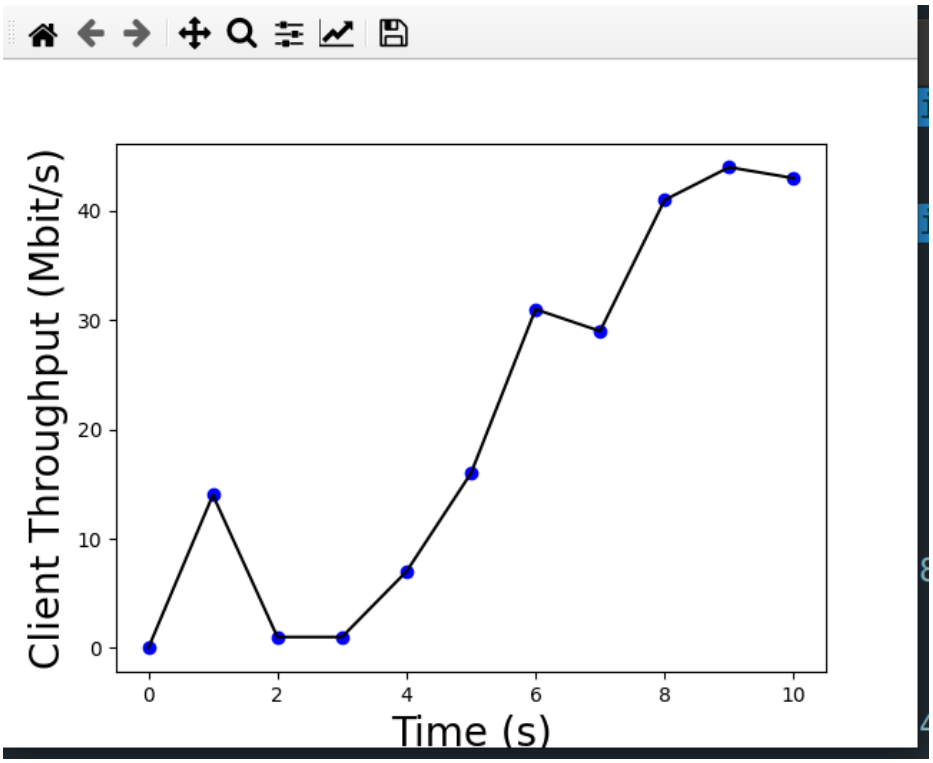
When latency is high, it means that the sender spends more time idle (not sending any new packets), which reduces how fast throughput grows.

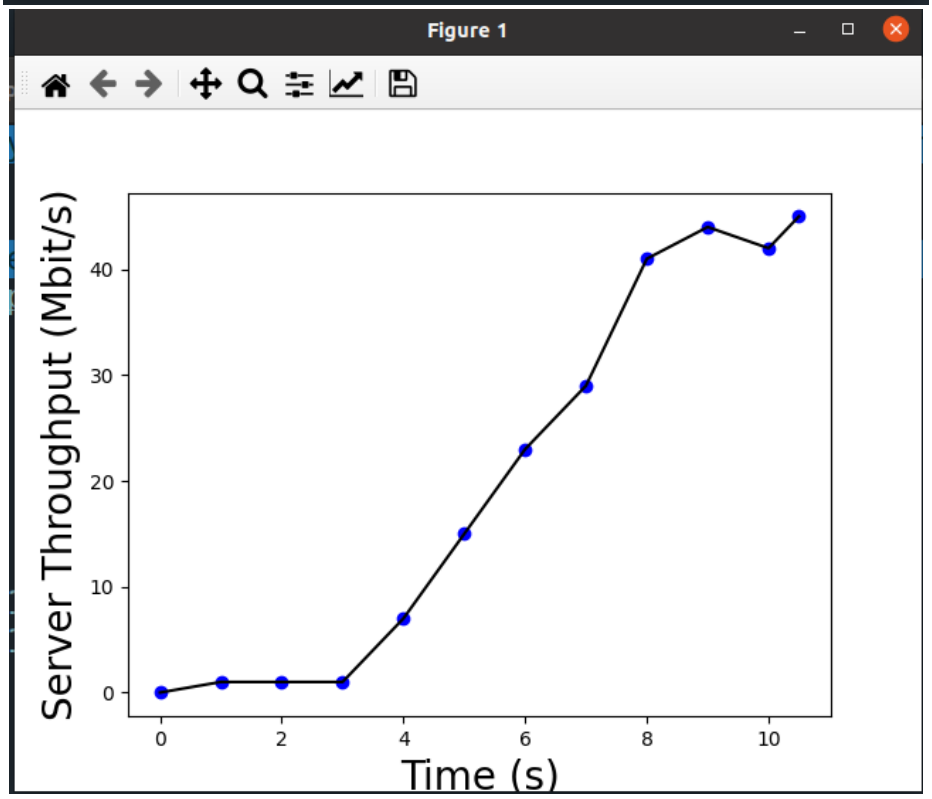
کانال با گم شدگی





کانال با تاخیر ثابت





۲. آیا تعداد بسته های دریافتی در سناریو های مختلف تفاوت دارد؟ چرا؟

قاعدتا زمانی که تاخیر یا گم شدن بسته ها را داریم تعداد بسته ها کمتر میشود البته باز هم بستگی به تعداد پکت های ارسالی از سمت کلاینت دارد.(که در اینجا از iperf بسته ها ارسال شده اند.)

طبق congestion control

TCP congestion control provides a function to detect and recover packet losses by retransmissions, which is called loss recovery in short. If the loss recovery is successful, packet transmission continues without a retransmission timeout (RTO).

TCP Retransmissions

Each byte of data sent in a TCP connection has an associated sequence number. This is indicated on the sequence number field of the TCP header.

When the receiving socket detects an incoming segment of data, it uses the acknowledgement number in the TCP header to indicate receipt. After sending a packet of data, the sender will start a retransmission timer of variable length. If it does not receive an acknowledgment before the timer expires, the sender will assume the segment has been lost and will retransmit it.

۳. با افزایش گم شدگی بسته ها تعداد باز ارسال ها چه تغییری میکند؟چرا؟

First off, retransmissions are essential for assuring reliable end-to-end communication in networks. Retransmissions are a sure sign that the self-healing powers of the TCP protocol are working

با افزایش گم شدن بسته ها تعداد باز ارسال ها هم زیاد میشود(با درصد گرفتن مشخص است)

چون پکت های بیشتری گم شده باید تعداد بیشتری بسته ارسال شود.

۴. با تصادفی کردن تاخیر لینک تعداد باز ارسال ها چه تغییری میکند؟چرا؟

Sending TCP sockets usually transmit data in a series. Rather than sending one segment of data at a time and waiting for an acknowledgement, transmitting stations will send several packets in succession. If one of these packets in the stream goes missing, the receiving socket can indicate which packet was lost using selective acknowledgments.

These allow the receiver to continue to acknowledge incoming data while informing the sender of the missing packet(s) in the stream.

selective acknowledgements will use the ACK number in the TCP header to indicate which packet was lost. At the same time, in these ACK packets, the receiver can use the SACK option in the TCP header to show which packets have been successfully received after the point of loss.

The SACK option is a function that is advertised by each station at the beginning of the TCP connection. Most network analyzers will flag these packets as duplicate acknowledgements because the ACK number will stay the same until the missing packet is retransmitted, filling the gap in the sequence.

Typically, duplicate acknowledgements mean that one or more packets have been lost in the stream and the connection is attempting to recover. They are a common symptom of packet loss. In most cases, once the sender receives three duplicate acknowledgments, it will immediately retransmit the missing packet instead of waiting for a timer to expire. These are called fast retransmissions.

Connections with more latency between client and server will typically have more duplicate acknowledgement packets when a segment is lost. In high latency

connections, it is possible to observe several hundred duplicate acknowledgements for a single lost packet.

۵. با توجه به این که تعداد بسته های ارسال iperf مشخص نیست و صرفا در مدت زمان گفته شده با گذردهی ممکن اقدام به ارسال بسته ها میکند، آیا تعداد باز ارسال ها می تواند معیار خوبی برای مقایسه عملکرد شبکه باشد؟

خیر باز ارسال ها معیاری خوبی نیست چون ممکن است دقیق نباشند و تعداد بسته های ارسالی هم مشخص نیست.

همچنین پروتکل udp این امکان را ندارد.

تعداد بازارسال ها فقط برای tcp است و وقتی گم شدن اتفاق میفتد باز ارسال داریم. و خب تعداد باز ارسال ها هم به اندازه ی درصدی است که در کانال مشخص کردیم و به نسبت تعداد پکت ها با افزایش گم شدن زیاد میشود.

۶. پروتکل TCP در هنگام مواجهه با گم شدگی بسته ها چه تغییری در گذردهی ارسال بسته ها ایجاد می کند؟ این رفتار به دلیل کدام خاصیت TCPرخ میدهد؟

Congestion control

How TCP Congestion Handles Missing Acknowledgment Packets

The TCP congestion window mechanism deals with missing acknowledgment packets as follows: **if an acknowledgement packet is missing after a period of time, the packet is considered as lost and the TCP congestion window is reduced by half** (hence the throughput too – which corresponds to the perception of limited capacity on the route by the sender); the TCP congestion window size can then restart increasing if acknowledgment packets are received properly.

Packet loss will have two effects on the speed of transmission of data:

1. **Packets will need to be retransmitted** (even if only the acknowledgment packet got lost and the packets got delivered)
2. **The TCP congestion window size will not permit an optimal throughput**

۷. در هنگام مواجهه با گم شدگی بسته ها چه تغییری در گذردهی دریافت بسته ها ایجاد میشود؟ چرا؟

رفتارش مثل گذردهی ارسال بسته ها است.

افزایش گم شدن بسته ها درtcp باعث میشود گذردهی کم شود.

طبق نمودار های بالا هم از 0تا 3500Mbps تغییر میکند.

چون تعداد بسته های دریافتی کمتر میشود.

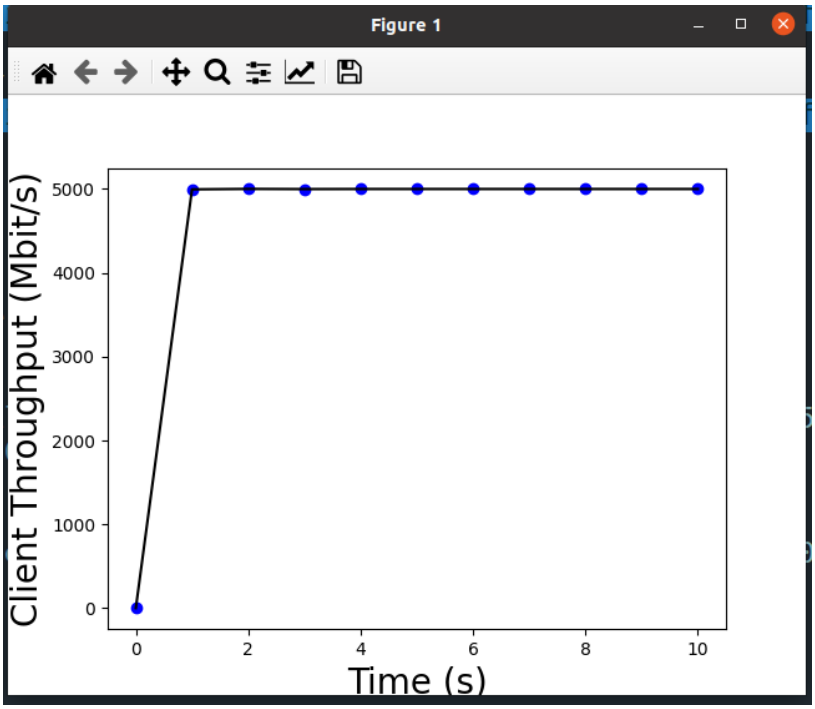
UDP

	کانال ایده ال	کانال با تاخیر ثابت	کانال با تاخیر تصادفی	کانال با گم شدگی کم	کانال با گم شدگی زیاد
میانگین گذردهی فرستنده	63	13	47	59	62
تعداد بسته دریافتی	2188	450	1534	2056	2094
تعداد بازارسال ها	0	0	0	0	0

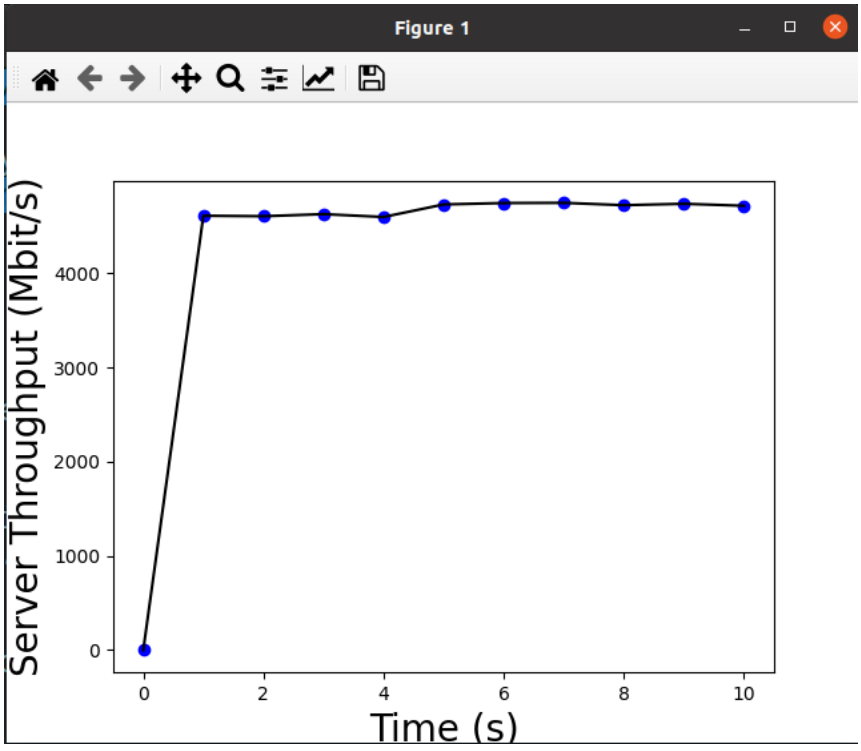
۱. پروتکل UDP در هنگام مواجهه با گم شدگی بسته ها چه تغییری در گذردهی ارسال بسته ها ایجاد می کند؟چرا؟

در هنگام گم شدن بسته ها در این پروتکل تغییر خاصی در گذردهی اتفاق نیفتاده است.

گذردهی udp طبق نمودار های زیر با گم شدگی تغییر زیادی نمیکند و این عملکرد خوبی است.



۲. در هنگام مواجهه با گم شدگی بسته ها چه تغییری در گذردهی دریافت بسته ها ایجاد می شود؟چرا؟ تغییر خاصی ندارد به دلیل خاصیت های udp



۳. با مقایسه ی نمودار های این قسمت و قسمت قبل، به نظر شما کدام یک از پروتکل های لایه انتقال در دنیای واقعی کارآمدتر اند؟ کدام یک عادلانه تر عمل میکنند؟

به نظر من هردو کارامد هستند و هردو در زمان و جای مناسب استفاده میشوند. نمیشود مقایسه کرد که کدام بهتر است.

از udp زمانی استفاده میکنیم که سرعت برایمان مهم تر است و دقت اهمیت کمتری دارد. چون تعداد پکت های دریافتی در این پروتوکل بیشتر از tcp است.

قطعا tcp برای پکت هایی مثل فایل ها که نیاز به دقت بالایی در ارسال دارند بهتر است.

در این پروتکل باز ارسال هم نداریم چون مخصوص tcp است.

و خب برای گم شدن بسته هایی که دریافتشون مهم است این یک عیب محسوب میشود اما اگر دقت دریافت بسته ها مهم نباشد این یک مزیت است. پس در نهایت بستگی به پکت هایی دارد که قصد ارسالشان را داریم.

اگر بخواهیم بازارسال ها را درنظر نگیریم udp از نظر سرعت خیلی بهتر عمل میکند.

- **Latency** is the time required to transmit a packet across a network:
  - Latency may be measured in many different ways: round trip, one way, etc.
  - Latency may be impacted by any element in the chain which is used to transmit data: workstation, WAN links, routers, local area network (LAN), server,… and ultimately it may be limited, in the case of very large networks, by the speed of light.

- **Throughput** is defined as the quantity of data being sent/received by unit of time
- **Packet loss** reflects the number of packets lost per 100 packets sent by a host

UDP Throughput is Not Impacted by Latency

UDP is a protocol used to carry data over IP networks. One of the principles of UDP is that we assume that all packets

sent are received by the other party (or such kind of controls is executed at a different layer, for example by the application itself).

In theory or for some specific protocols (where no control is undertaken at a different layer; e.g., one-way transmissions), the rate at which packets can be sent by the sender is not impacted by the time required to deliver the packets to the other party (= latency). Whatever that time is, the sender will send a given number of packets per second, which depends on other factors (application, operating system, resources, …).

TCP is directly impacted by latency

TCP is a more complex protocol as it integrates a mechanism which checks that all packets are correctly delivered. This mechanism is called acknowledgment: it consists of having the receiver transmit a specific packet or flag to the sender to confirm the proper reception of a packet.