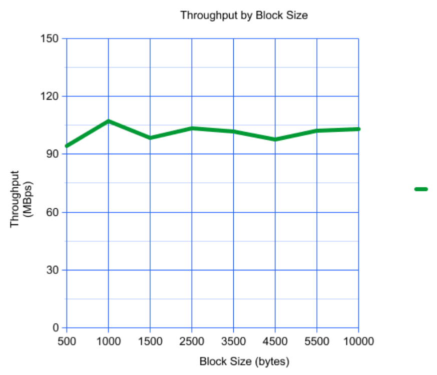
CS 536 Fall 2016: Lab3

Maria L. Pacheco

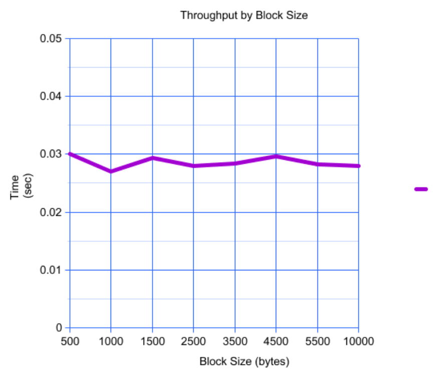
October 17, 2016

1 Problem 1

In figures 1a and 1b we can observe the reliable throughput and transfer time by making variations to the block size when transferring files. These experiments were run using `sslab22.cs.purdue.edu` as a server and `sslab01.cs.purdue.edu` as a client.



(a) Throughput by Block Size



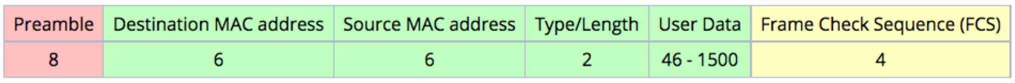
(b) Transfer time by Block Size

There doesn't seem to be a significant impact in the reliable throughput or the transfer time when we variate the block size between 500 and 10,000 bytes. Reliable throughput fluctuates between 90 and 110 MBps, reaching a small peak at 1,000 bytes. However, continuous transfers using the same block size also fluctuate in this range. Time stays very close to 0.03 seconds, dropping to 0.027 at 1000 bytes, which explains the throughput result.

We can conclude that the system overhead of making subsequent calls to write and read and the overhead of sending more or less packets are minimal and speed is not affected by them in the studied block size range.

2 Problem 2

In table 1 we can observe the completion time, bit rate in Mbps and packets per second when sending a total of 1,000,000 bytes sent in 1,000 packets. To include UDP header and Ethernet header/trailer in the Mbps calculation constants were defined. We know the UDP header contains 8 bytes and the Ethernet frame in figure 2 was used as reference for defining the Ethernet header and trailer.



Preamble	Destination MAC address	Source MAC address	Type/Length	User Data	Frame Check Sequence (FCS)
8	6	6	2	46 - 1500	4

Figure 2: A physical Ethernet packet

Where the header is of size 22 bytes and the trailer of 4 bytes.

In table 1 we can observe that bit rate both in Mbps and pps reported by the sender is consistently close to the one reported by the receiver. The receiver reports, in all cases, a slightly better performance. Additionally, as the interval decreases, time spent decreases and the bit rate increases. This is an expected result as the system won't wait before sending the next packet, it will send it as soon as it has sent the other one. On the receiver side, all packets were received in all experiments: 1,034,000 bytes including overhead and 1,000 packets.

It seems the smallest interval is not small enough to cause packets to queue on the receiver side. However, it could come a time where making it even smaller would stabilize the performance for this reason.

	Sender			Receiver		
Interval	time	Mbps	pps	time	Mbps	pps
50 ms	0.1061	65.94	9,420.80	0.1059	66.06	9,437.27
10 ms	0.0670	104.40	14,914.72	0.0669	104.50	14,929.53
5 ms	0.0592	118.14	16,878.48	0.0590	118.46	16,923.43
1 ms	0.0578	120.94	17,277.14	0.0577	121.28	17,326.82
0.5 ms	0.0551	126.98	18,140.59	0.054942	127.40	18,201.04
0.1 ms	0.0545	128.28	18,326.38	0.0543	128.74	18,392.76

Table 1: Varying the interval with payload of 1,000 bytes and 1,000 packet count

In table 2 we can observe the results of fixing the packet count to 10,000 and the intervals 1 msec and varying the payload of the packets from 50 bytes to 8,000 bytes.

	Sender			Receiver		
Payload	time	Mbps	pps	time	Mbps	pps
50 bytes	0.5399	11.11	18,519.06	0.5397	11.11	18,525.61
500 bytes	0.5429	73.66	18,416.38	0.5428	73.68	18,421.77
1,000 bytes	0.5418	143.96	18,456.78	0.5416	144.01	18,463.60
2,000 bytes	0.5486	282.52	18,227.55	0.5481	282.79	18,244.60
4,000 bytes	0.5575	550.58	17,934.32	0.5573	550.80	17,941.65
8,000 bytes	0.6732	909.07	14,854.09	0.6736	908.49	14,844.66

Table 2: Varying the payload of 10,000 packets in an interval of 1 msec

In this case we can observe how the bit rate measured in Mbps increases as the payload increases while the bit rate measured in packets per second decreases. The increase in Mbps performance can be explained in terms of efficiency. The time required to send bits vs. the interval between transmissions. Initially, the bottleneck is in the interval. The transfer pipe is underutilized while we wait to send the next package. If we continued to increase our payload further, there will come a moment where we would use it to a full capacity and the interval we wait wouldn't be idle time as it amount to transfer time between sender and receiver. At this moment, the Mbps rate would stabilize.

On the other hand, the bit rate of packets per second decreases as payload is incremented. As each packet will be much heavier. However, we can see that pps decrease is much more smooth than Mbps increase, which is much more pronounced.

For all these experiments, the receiver ran on `sslab22.cs.purdue.edu` and the sender ran on `sslab01.cs.purdue.edu`.

3 Problem 3

10 packets of 10 bytes were generated to port 10000 using the traffic generator. In listing 1 inspecting the 10 captured Ethernet frames we can see the following headers.

Listing 1: Ethernet Headers

```
borg23 105 $ tcpdump -r mylogfile -e
reading from file mylogfile, link-type EN10MB (Ethernet)
22:47:40.745700 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
    length 52: remote.43148 > host.10000: UDP, length 10
22:47:40.745788 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
    length 52: remote.43148 > host.10000: UDP, length 10
22:47:40.745858 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
    length 52: remote.43148 > host.10000: UDP, length 10
22:47:40.745926 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
    length 52: remote.43148 > host.10000: UDP, length 10
22:47:40.745994 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
    length 52: remote.43148 > host.10000: UDP, length 10
22:47:40.746062 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
    length 52: remote.43148 > host.10000: UDP, length 10
22:47:40.746129 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
    length 52: remote.43148 > host.10000: UDP, length 10
22:47:40.746194 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
    length 52: remote.43148 > host.10000: UDP, length 10
22:47:40.746258 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
    length 52: remote.43148 > host.10000: UDP, length 10
22:47:40.746322 6a:9e:a9:09:e3:05 (oui Unknown) > b6:de
    :25:ac:9d:df (oui Unknown), ethertype IPv4 (0x0800),
length 52: remote.43148 > host.10000: UDP, length 10
```

We can take the first packet in listing 2 and take a look into its contents directly. Which be obtained by running `tcpdump -r mylogfile -XX`.

Listing 2: Ethernet Packet 1

```
tcpdump −r mylogfile −XX
reading from file mylogfile, link−type EN10MB (Ethernet)
22:47:40.745700 IP remote.43148 > host.10000: UDP, length
    10
  0x0000:  b6de 25ac 9ddf 6a9e a909 e305 0800 4500
  0x0010:  0026 f1dc 4000 4011 c596 c0a8 0102 c0a8
  0x0020:  0101 a88c 2710 0012 8377 5050 5050 5050
 0x0030:  5050 5050                                   ...
```

We can see that the source and destination addresses correspond to what we saw above: source address is `b6:de:25:ac:9d:df` and destination address is `6a:9e:a9:09:e3:05`. Then directly we have the type `0800` which corresponds to the summary above as well where verbosely we know that `0x0800` corresponds to ethertype IPv4. Since we have the type after the addresses, we can conclude that captured frames are DIX.

Then we go directly into the Ethernet payload, whose 4 first bits are the part of the IP header corresponding to the version, again we see a that the version is "4", since each number in the hexadecimal representation corresponds to 4 bits. The next 4 bits will tell us the IP header size, we see a five, which represents the default IP header size size: 20 bytes.

Now, we want to go to the UDP header, since the UDP packet is the payload of the IP packet, we want to start looking after 20 bytes. The source port are the first 2 bytes of the UDP header right after the 20 bytes: `0xa88c` which means the source port is 43148, which corresponds to the verbose summary above. Then, the destination port are the consecutive 2 bytes `0x2710`, so the destination port is 10000, which is the one set as a parameter to the traffic sender program. Then we have two blocks of 8 bytes for the length of the UDP packet and the checksum, followed by the data: a consecutive sequence of 10 bytes where the number 50 represents the letter "P", my last name initial.