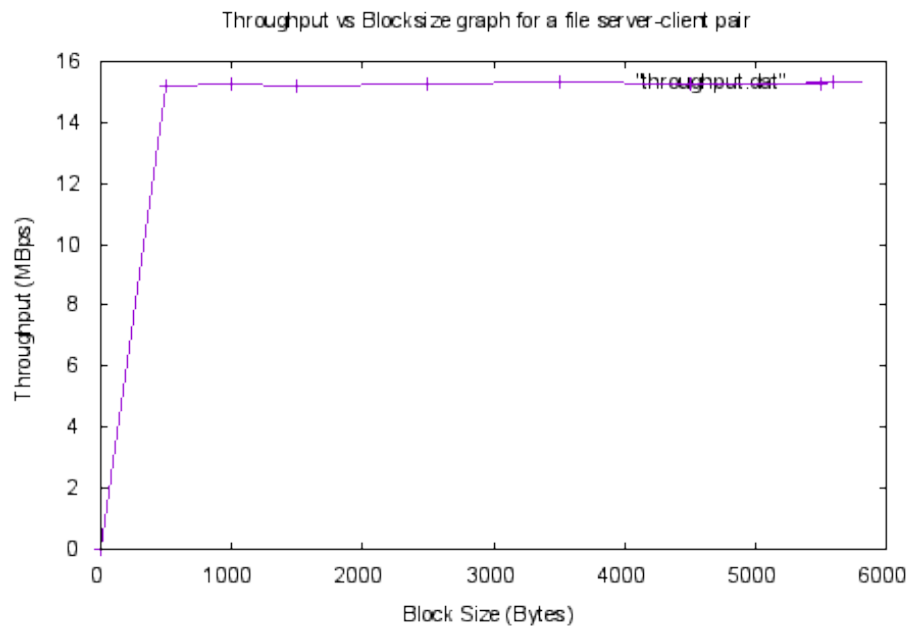


CS 536 Lab Answers 3

Prashant Ravi — ravi18@purdue.edu

October 17, 2016

Problem 1



In my graph the peak is obtained at 1500 bytes which makes sense because on executing `ifconfig` i noticed that we get MTU 1500. The maximum transmission unit is 1500 bytes beyond which I don't see an increase in throughput by increasing the data size.

To test for multiple client correctness, I wrote a script `start.sh` that downloads the files correctly on each client.

Problem 2

1 Number Listings

Number listings for variable delay:

Sender

50 msec

Completion time: 50.040000 seconds
Application bitrate: 19.984013 pps
Application bitrate: 167226.219025 bps

10 msec

Completion time: 10.079000 seconds
Application bitrate: 99.216192 pps
Application bitrate: 830241.095347 bps

5 msec

Completion time: 5.087000 seconds
Application bitrate: 196.579516 pps
Application bitrate: 1644977.393356 bps

0.5 msec

Completion time: 0.579000 seconds
Application bitrate: 1727.115717 pps
Application bitrate: 14452504.317789 bps

0.1 msec

Completion time: 0.160000 seconds
Application bitrate: 6250.000000 pps
Application bitrate: 52300000.000000 bps

Receiver

50 msec

Completion time: 50.040000 seconds
Application bitrate: 20.003997 pps

Application bitrate: 167226.219025 bps

10 msec

Completion time: 10.079000 seconds

Application bitrate: 99.315408 pps

Application bitrate: 830241.095347 bps

5 msec

Completion time: 5.087000 seconds

Application bitrate: 196.776096 pps

Application bitrate: 1644977.393356 bps

0.5 msec

Completion time: 0.579000 seconds

Application bitrate: 1728.842832 pps

Application bitrate: 14452504.317789 bps

0.1 msec

Completion time: 0.160000 seconds

Application bitrate: 6256.250000 pps

Application bitrate: 52300000.000000 bps

Number listings for variable payload size:

Sender

50 bytes

Completion time: 10.952000 seconds

Application bitrate: 913.075237 pps

Application bitrate: 701241.782323 bps

500 bytes

Completion time: 10.756000 seconds

Application bitrate: 929.713648 pps

Application bitrate: 4060989.215322 bps

1000 bytes

Completion time: 10.934000 seconds

Application bitrate: 914.578379 pps

Application bitrate: 7653191.878544 bps

2000 bytes

Completion time: 10.804000 seconds

Application bitrate: 925.583117 pps

Application bitrate: 15149944.465013 bps

4000 bytes

Completion time: 11.006000 seconds

Application bitrate: 908.595312 pps

Application bitrate: 29409413.047429 bps

8000 bytes

Completion time: 10.857000 seconds

Application bitrate: 921.064751 pps

Application bitrate: 59287095.882841 bps

Receiver

50 bytes

Completion time: 10.952000 seconds

Application bitrate: 913.166545 pps

Application bitrate: 701241.782323 bps

500 bytes

Completion time: 10.756000 seconds
Application bitrate: 929.806620 pps
Application bitrate: 4060989.215322 bps

1000 bytes

Completion time: 10.934000 seconds
Application bitrate: 914.669837 pps
Application bitrate: 7653191.878544 bps

2000 bytes

Completion time: 10.804000 seconds
Application bitrate: 925.675676 pps
Application bitrate: 15149944.465013 bps

4000 bytes

Completion time: 11.006000 seconds
Application bitrate: 908.686171 pps
Application bitrate: 29409413.047429 bps

8000 bytes

Completion time: 10.857000 seconds
Application bitrate: 921.156857 pps
Application bitrate: 59287095.882841 bps

Analysis:

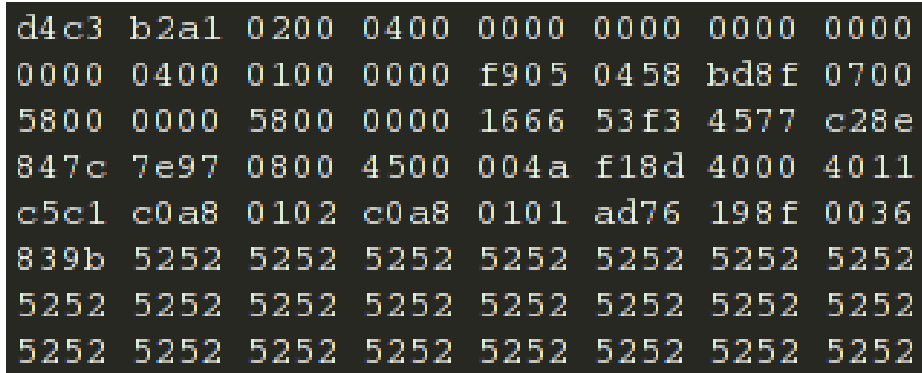
The throughput measured at sender and receiver are approximately equal.

On decreasing the packet spacing we are bombarding bits at greater frequency so there is an obvious increase in throughput. However, the throughput starts leveling out as we keep decreasing the packet spacing because the scheduler will place a lower bound on packet spacing. By reducing packet spacing we are also sleeping less causing us to get smaller time quantum which cause us to get smaller quantum. This very tradeoff places the lower bound on packet spacing.

By increasing packet size we see a linear increase in throughput. This is true because we know the sender and receivers windows sizes are the same so we can expect this linear increase in throughput, as the throughput is measured as the $\min(\text{sender_frame_size}, \text{rec_frame_size}) / \text{RTT}$.

Yes! we met the theoretical expectations .

Problem 3



d4c3	b2a1	0200	0400	0000	0000	0000	0000
0000	0400	0100	0000	f905	0458	bd8f	0700
5800	0000	5800	0000	1666	53f3	4577	c28e
847c	7e97	0800	4500	004a	f18d	4000	4011
c5c1	c0a8	0102	c0a8	0101	ad76	198f	0036
839b	5252	5252	5252	5252	5252	5252	5252
5252	5252	5252	5252	5252	5252	5252	5252
5252	5252	5252	5252	5252	5252	5252	5252

Deep Inspection :

The 48 bit source MAC address is : 1666 53f3 4577

The 48 bit destination MAC address is : c28e 847c 7e97

The next two bytes are 0800 which signified IPv4 in DIX. It can't be 802.3 because that length is too large. So we have DIX ethernet format.

IP version is 4. Source port of udp header is 0xad76

Destination port of udp header is 0x198f

First letter of last name is R which is 0x52 and I ran the code with argument 46 as packet size which make sense because there are 46 , 0x52 in the image above. Checksum is 839b. I went about finding the field values by executing the tcp-dump command.