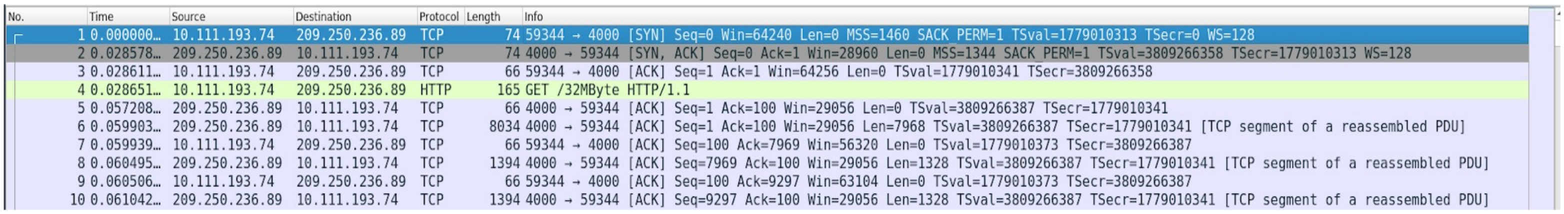
Summative Assignment – Networks

Part 3 (30%): Repeat part 1 using ports 4011 through 4019, which will lose the acknowledgements your machine sends. describe the strategy the server and the client use to maintain (some) throughput.

Exercise 1:

Answer:

For each port, the server and client initially connect to each other with the TCP handshake initiation, then a get request is sent via HTTP (the curl command via terminal asking for the 32Mbyte file) and the file data starts to be transported from server to client with continous acknowledgements. The ports, especially as the numbers get higher, seem to be designed as an attempt to create an unreliable network and to show how the system will deal with lost packets.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000…	10.111.193.74	209.250.236.89	TCP	74	59344 → 4000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1779010313 TSecr=0 WS=128
2	0.028578…	209.250.236.89	10.111.193.74	TCP	74	4000 → 59344 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1344 SACK_PERM=1 TSval=3809266358 TSecr=1779010313 WS=128
3	0.028611…	10.111.193.74	209.250.236.89	TCP	66	59344 → 4000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1779010341 TSecr=3809266358
4	0.028651…	10.111.193.74	209.250.236.89	HTTP	165	GET /32MByte HTTP/1.1
5	0.057208…	209.250.236.89	10.111.193.74	TCP	66	4000 → 59344 [ACK] Seq=1 Ack=100 Win=29056 Len=0 TSval=3809266387 TSecr=1779010341
6	0.059903…	209.250.236.89	10.111.193.74	TCP	8034	4000 → 59344 [ACK] Seq=1 Ack=100 Win=29056 Len=7968 TSval=3809266387 TSecr=1779010341 [TCP segment of a reassembled PDU]
7	0.059939…	10.111.193.74	209.250.236.89	TCP	66	59344 → 4000 [ACK] Seq=100 Ack=7969 Win=56320 Len=0 TSval=1779010373 TSecr=3809266387
8	0.060495…	209.250.236.89	10.111.193.74	TCP	1394	4000 → 59344 [ACK] Seq=7969 Ack=100 Win=29056 Len=1328 TSval=3809266387 TSecr=1779010341 [TCP segment of a reassembled PDU]
9	0.060506…	10.111.193.74	209.250.236.89	TCP	66	59344 → 4000 [ACK] Seq=100 Ack=9297 Win=63104 Len=0 TSval=1779010373 TSecr=3809266387
10	0.061042…	209.250.236.89	10.111.193.74	TCP	1394	4000 → 59344 [ACK] Seq=9297 Ack=100 Win=29056 Len=1328 TSval=3809266387 TSecr=1779010341 [TCP segment of a reassembled PDU]

As with most modern networks, pipelining is in place within the simulation to ensure a quicker connection; if only one packet was sent at a time before acknowledgement, the data would be always end up arriving but the efficiency would be incredibly low. In this case the protocol go-back-N is used for the transfer, we know this because when duplicate acknowledgements or lost packets occur (discussed in detail later), the sequence number of the most recent packet received is sent alone, rather than both this number and the numbers of all the other packets received after the lost packet which is the method used for specific retransmission (The other protocol for pipelining taught in this module).

There are five notable outputs from the pcap files that are not the usually packet acknowledgements:

- TCP Window update - The size of the window is relatively small initially but one of these is sent after a period of time (when the sender knows the reciever can/can’t handle the connection) to increase or decrease the window size.
- TCP Dup ACK – This occurs when an acknowlegement has not been received from the sender within the timeout, can end up triggering a fast retransmission.
- TCP (Fast) Retransmission – A key output, if the packet is lost (No acknowledement has been recieved) the system will attempt to resend it. The difference between retransmission and fast retransmission is that retransmission involves the server just not receiving the recent acknowledgement before the timeout while with fast retransmission there is a packet known to be lost, this is linked with the duplicate ack as once one or more is received the packet the system knows a specific packet

need retransmission and sends it straight away without waiting for a timeout

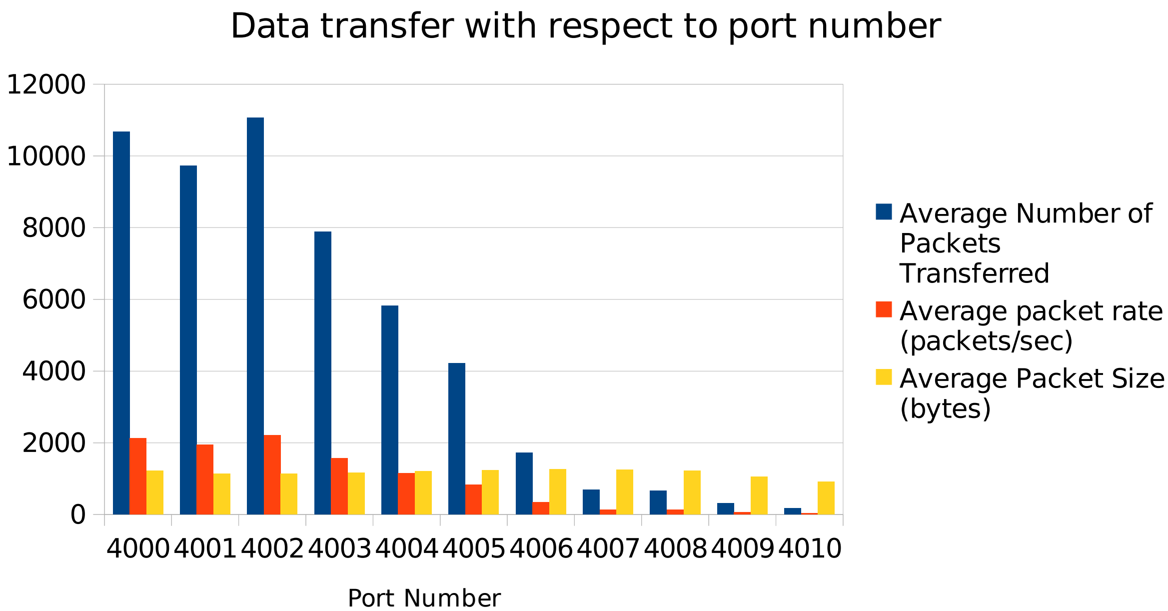
- TCP Previous segment not captured – This is the indicator that a packet was lost and can be used to understand the data further
- TCP Out-Of Order – This simply means that at some point, for whatever reason, a later packet has arrived quicker than a older one, this is resolved by the reciever once all packets are received or on the fly.

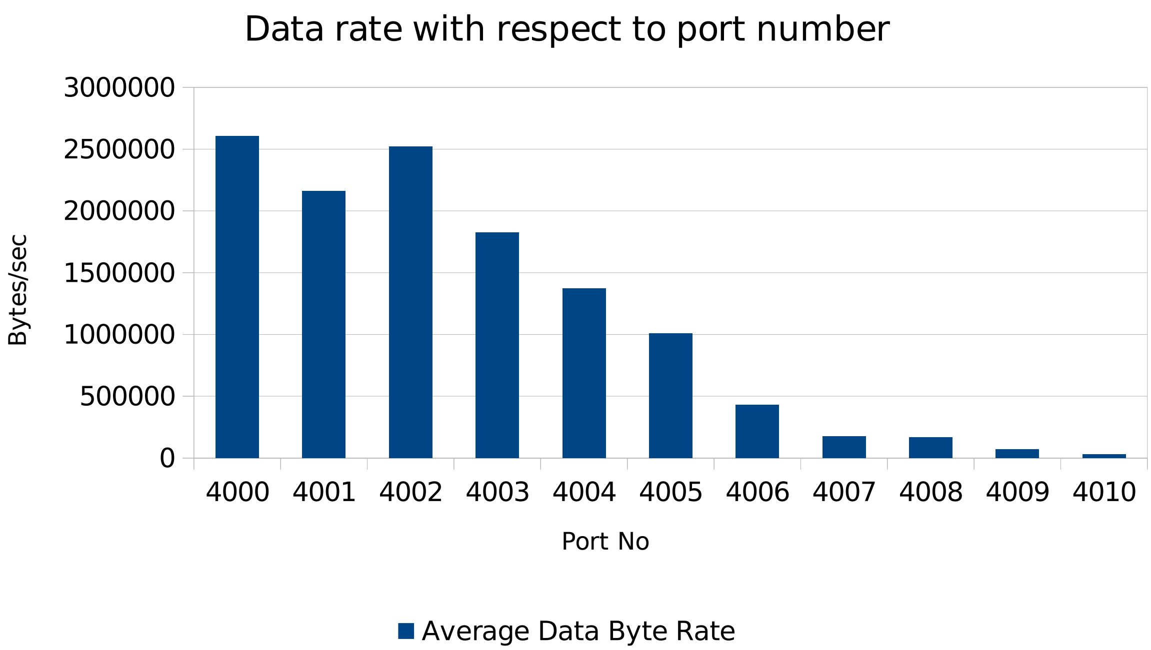
Code Implemented: None
Further Explanation: None

<u>Exercise 2:</u>
Answer:
To gain detailed answers that were not tainted by a poor internet connection or other interference, 10 tests were done on each port (spending five seconds on eache test) and an average of the results is given below:

Port No	Average Number of Packets Transferred	Average packet rate (packets/ sec)	Average Data Byte Rate (bytes/ sec)	Average Packet Size (bytes)	Average Capture Duration (seconds)
4000	10680	2132	2605309	1223	4
4001	9733	1940	2158345	1142	4
4002	11069	2207	2522117	1141	5
4003	7890	1572	1822718	1170	5
4004	5823	1157	1374067	1211	4
4005	4212	837	1009010	1232	5
4006	1723	342	430146	1262	4
4007	693	137	174037	1246	5
4008	662	131	166072	1215	4
4009	320	62	71553	1060	5
4010	177	31	31924	912	5

This table clearly shows that there is a downwards trend in ability to transfer data as the port number increases, the number of packets drops by a factor of 100 from port 4000 to 4010 and this occurs similarly for the other data transfer columns. This data shows that at around port 4006 is where the connection starts to breakdown, this correlates with 30% packet loss. The data is also shown in a graph below:





This all seems to be due to the continous retransmissions of data and how the structure of go-back-N works. As the number of lost packets increases, the network ends up sending a larger and larger numbers of packets that had already been received because they would have arrived but the packet before them was lost.

Another interesting thing to note is that the average packet size increases (if only marginally) as the ability of the connection worsened until the final two ports when it started to decrease. Although maybe a coincidence or a fluke of the network connection, this may have been due to an awareness by the system of the dropping packets and an attempt to send larger packets to compensate.

Code Implemented (Bash):

```
server='http://offsite10.batten.eu.org'
filecol="/32MByte"
filename="SA2"
#curl -v -4 -o /dev/null http://offsite10.batten.eu.org:4003/32MByte

for ports in {4000..4010} #Iterates through all ports needed
	do
	for iter in {1..10} #Iterates 10 tens so an average can be taken
		do
		loc="" #Directory where all files will be uploaded
		fullfile="${filename}-${ports}-${iter}.pcap" #Pcap file location
		fullloc="${loc}${fullfile}" #where to store and how to name the pcap files
		echo "Port Number: ${ports}"
		echo "Attempt: ${iter}"
		#for 10 seconds the tshark will watch for data from the specific port used
		tshark -i 1 -a duration:10 -w  $fullloc -f "port ${ports}" &
		sleep 2s #Sleep on either side as a buffer so edge packets aren’t missed
		#approx 5 seconds of connection to each port
		timeout 5s curl -s -4 -o /dev/null "${server}:${ports}${filecol}"
		sleep 2s
		#following is the data collected about the pcaps, stored in a text file
		fullloctwo="${loc}${filename}-${ports}-info.txt"
		if [[ iter -eq 1 ]]; then
			capinfos -c -u -a -e -x -z -y -M $fullloc > $fullloctwo
		else
			capinfos -c -u -a -e -x -z -y -M $fullloc >> $fullloctwo
		fi
		#The merging of the pcap files happens below
		locall="${loc}${filename}-all.pcap"
		loctemp="${loc}{filename}-temp.pcap"
		if [[ $ports -eq 4000 && iter -eq 1 ]]; then
			mergecap  -w $locall $fullloc
		else
			mergecap -w $loctemp $locall $fullloc
```

```
                    mergecap -w $locall $loctemp
                    rm $loctemp
                fi
        done
done
```

Further Explanation:
Due to wanting to be more accurate with the data given, a hundred tests would have to have been carried out, ten on each port, which was too large a workload to do manually. I researched Tshark, a console implimentation of wireshark (that is supposedly similar to how tcpdump works) to automate the process. Using the code above allowed me to iterate through the ports, carrying out the curl command for a particular time (5 seconds in this case) and gain info about each test using capinfos. There is also a final part that merges all the files into one, this was in an attempt to collate the data into one full graph, this was useful for some of the results displayed above. A seperate script was created to grep the results of this automation and display the averages and output them into a csv file for use in graph making. The variable $loc was cleared to retain security by not displaying how my filesystem works but if you wish to use this script in future simply enter the location you want the packets to output to.

Exercise 3:

Answer:
These ports lose received packets at various percentages,on earlier ports when the connection is reliable a continous stream of acknowledgements is sent back and forth similar to in task 1. Looking at a capture for port 4016 (discussed later as point of collapse) however, there are a large number of duplicate acknowledgements sent to the server by the reciever but very few confirmation acknowledgements. This seems to be a situation in which cumulative ACK and pipelining are used in conjunction to attempt to keep a semblance of efficiency by sending these large quantities of packets and only needing an ack for retransmission.

	Time	Source	Destination	Protocol	Length	Info
64	2.613344…	209.250.236.89	10.111.193.74	TCP	1394	4016 → 35844 [ACK] Seq=46481 Ack=100 Win=29056 Len=1328 TSval=3898479084 TSecr=1825527361 [TCP segment of a reassembled PDU]
65	2.614054…	10.111.193.74	209.250.236.89	TCP	78	[TCP Dup ACK 22#20] 35844 → 4016 [ACK] Seq=100 Ack=7969 Win=63104 Len=0 TSval=1825527397 TSecr=3898478197 SLE=15937 SRE=47809
66	2.619758…	209.250.236.89	10.111.193.74	TCP	1394	4016 → 35844 [ACK] Seq=47809 Ack=100 Win=29056 Len=1328 TSval=3898479088 TSecr=1825527362 [TCP segment of a reassembled PDU]
67	2.620319…	209.250.236.89	10.111.193.74	TCP	2722	4016 → 35844 [ACK] Seq=49137 Ack=100 Win=29056 Len=2656 TSval=3898479092 TSecr=1825527367 [TCP segment of a reassembled PDU]
68	2.620780…	10.111.193.74	209.250.236.89	TCP	78	[TCP Dup ACK 22#21] 35844 → 4016 [ACK] Seq=100 Ack=7969 Win=63104 Len=0 TSval=1825527404 TSecr=3898478197 SLE=15937 SRE=51793
69	2.622663…	209.250.236.89	10.111.193.74	TCP	1394	4016 → 35844 [ACK] Seq=51793 Ack=100 Win=29056 Len=1328 TSval=3898479098 TSecr=1825527375 [TCP segment of a reassembled PDU]
70	2.622888…	209.250.236.89	10.111.193.74	TCP	1394	4016 → 35844 [ACK] Seq=53121 Ack=100 Win=29056 Len=1328 TSval=3898479098 TSecr=1825527375 [TCP segment of a reassembled PDU]
71	2.623681…	10.111.193.74	209.250.236.89	TCP	78	[TCP Dup ACK 22#22] 35844 → 4016 [ACK] Seq=100 Ack=7969 Win=63104 Len=0 TSval=1825527407 TSecr=3898478197 SLE=15937 SRE=54449
72	2.628007…	209.250.236.89	10.111.193.74	TCP	2722	4016 → 35844 [ACK] Seq=54449 Ack=100 Win=29056 Len=2656 TSval=3898479099 TSecr=1825527375 [TCP segment of a reassembled PDU]
73	2.629027…	10.111.193.74	209.250.236.89	TCP	78	[TCP Dup ACK 22#23] 35844 → 4016 [ACK] Seq=100 Ack=7969 Win=63104 Len=0 TSval=1825527412 TSecr=3898478197 SLE=15937 SRE=57105
74	2.636937…	209.250.236.89	10.111.193.74	TCP	2722	4016 → 35844 [ACK] Seq=57105 Ack=100 Win=29056 Len=2656 TSval=3898479116 TSecr=1825527393 [TCP segment of a reassembled PDU]
75	2.637962…	10.111.193.74	209.250.236.89	TCP	78	[TCP Dup ACK 22#24] 35844 → 4016 [ACK] Seq=100 Ack=7969 Win=63104 Len=0 TSval=1825527421 TSecr=3898478197 SLE=15937 SRE=59761
76	2.638693…	209.250.236.89	10.111.193.74	TCP	2722	4016 → 35844 [ACK] Seq=59761 Ack=100 Win=29056 Len=2656 TSval=3898479116 TSecr=1825527393 [TCP segment of a reassembled PDU]
77	2.639711…	10.111.193.74	209.250.236.89	TCP	78	[TCP Dup ACK 22#25] 35844 → 4016 [ACK] Seq=100 Ack=7969 Win=63104 Len=0 TSval=1825527423 TSecr=3898478197 SLE=15937 SRE=62417
78	2.640841…	209.250.236.89	10.111.193.74	TCP	2722	4016 → 35844 [ACK] Seq=62417 Ack=100 Win=29056 Len=2656 TSval=3898479116 TSecr=1825527393 [TCP segment of a reassembled PDU]
79	2.641886…	10.111.193.74	209.250.236.89	TCP	78	[TCP Dup ACK 22#26] 35844 → 4016 [ACK] Seq=100 Ack=7969 Win=63104 Len=0 TSval=1825527425 TSecr=3898478197 SLE=15937 SRE=65073
80	2.643021…	209.250.236.89	10.111.193.74	TCP	1394	4016 → 35844 [ACK] Seq=65073 Ack=100 Win=29056 Len=1328 TSval=3898479118 TSecr=1825527393 [TCP segment of a reassembled PDU]
81	2.643306…	209.250.236.89	10.111.193.74	TCP	1394	4016 → 35844 [ACK] Seq=66401 Ack=100 Win=29056 Len=1328 TSval=3898479118 TSecr=1825527393 [TCP segment of a reassembled PDU]
82	2.643319…	209.250.236.89	10.111.193.74	TCP	2722	4016 → 35844 [ACK] Seq=67729 Ack=100 Win=29056 Len=2656 TSval=3898479120 TSecr=1825527393 [TCP segment of a reassembled PDU]
83	2.644034…	10.111.193.74	209.250.236.89	TCP	78	[TCP Dup ACK 22#27] 35844 → 4016 [ACK] Seq=100 Ack=7969 Win=63104 Len=0 TSval=1825527427 TSecr=3898478197 SLE=15937 SRE=70385
84	3.849590…	209.250.236.89	10.111.193.74	TCP	1394	[TCP Retransmission] 4016 → 35844 [ACK] Seq=7969 Ack=100 Win=29056 Len=1328 TSval=3898480315 TSecr=1825527423
85	3.849607…	10.111.193.74	209.250.236.89	TCP	78	35844 → 4016 [ACK] Seq=100 Ack=9297 Win=61824 Len=0 TSval=1825528634 TSecr=3898480315 SLE=15937 SRE=70385
86	4.975096…	10.111.193.74	209.250.236.89	TCP	78	35844 → 4016 [FIN, ACK] Seq=100 Ack=9297 Win=61824 Len=0 TSval=1825529759 TSecr=3898480315 SLE=15937 SRE=70385
87	5.237288…	10.111.193.74	209.250.236.89	TCP	78	[TCP Retransmission] 35844 → 4016 [FIN, ACK] Seq=100 Ack=9297 Win=61824 Len=0 TSval=1825530021 TSecr=3898480315 SLE=15937 SRE=
88	5.562819…	209.250.236.89	10.111.193.74	TCP	1394	[TCP Retransmission] 4016 → 35844 [ACK] Seq=9297 Ack=101 Win=29056 Len=1328 TSval=3898482043 TSecr=1825530021
89	5.562874…	10.111.193.74	209.250.236.89	TCP	54	35844 → 4016 [RST] Seq=101 Win=0 Len=0

Port No	Average Number of Packets Transferred	Average packet rate (packets/ sec)	Average Data Byte Rate (bytes/ sec)	Average Packet Size (bytes)	Average Capture Duration (seconds)
4011	11870	2359	2368216	1007	5
4012	11426	2277	2311357	1047	4
4013	11627	2314	2323527	1007	5
4014	10669	2214	2124566	906	4
4015	9713	1935	1961059	919	4
4016	4815	960	951129	770	4
4017	1471	292	311067	493	4
4018	163	28	35487	310	4
4019	7	1	159	93	4

As the percentage failure goes up by 10% for each port, the network seems to only start to degrade at reaching higher than 50% (port 4016), which is much higher than the 30% percieved in task 1. This shows that for the reason discussed the network is more able to bounce back from large numbers of failures.

Code Implemented: None

Further Explanation: None