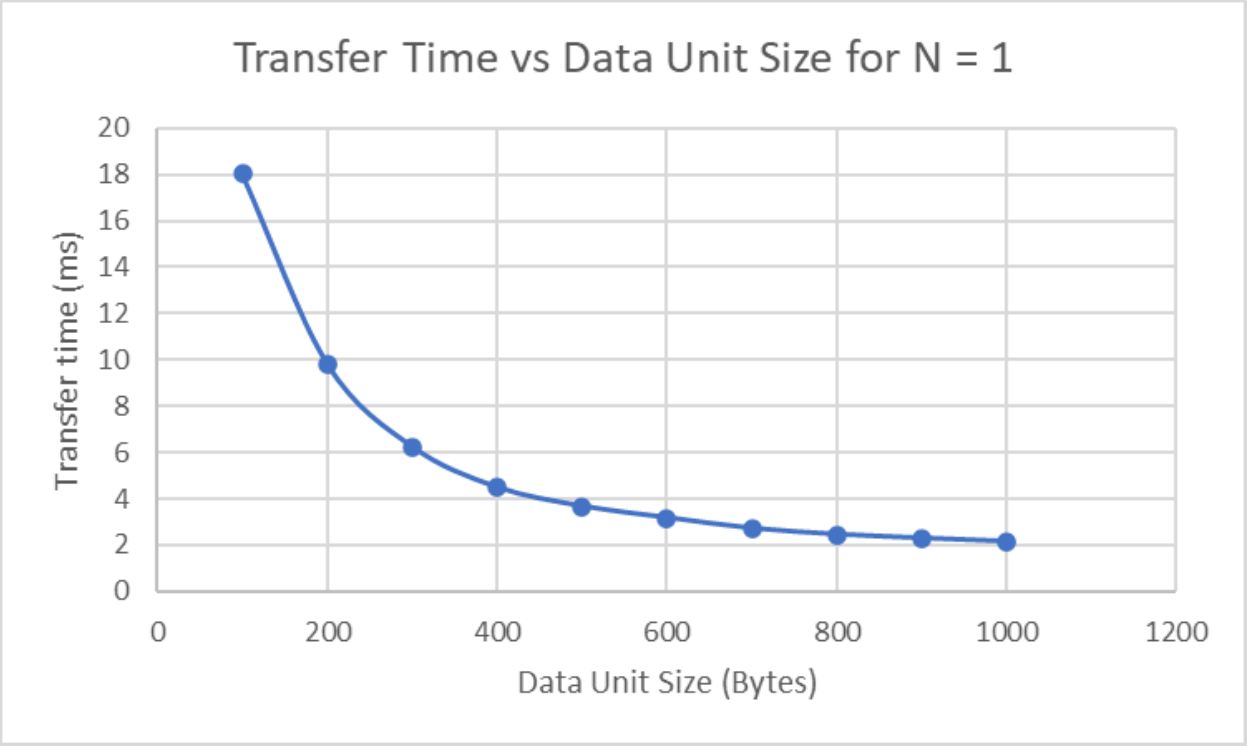
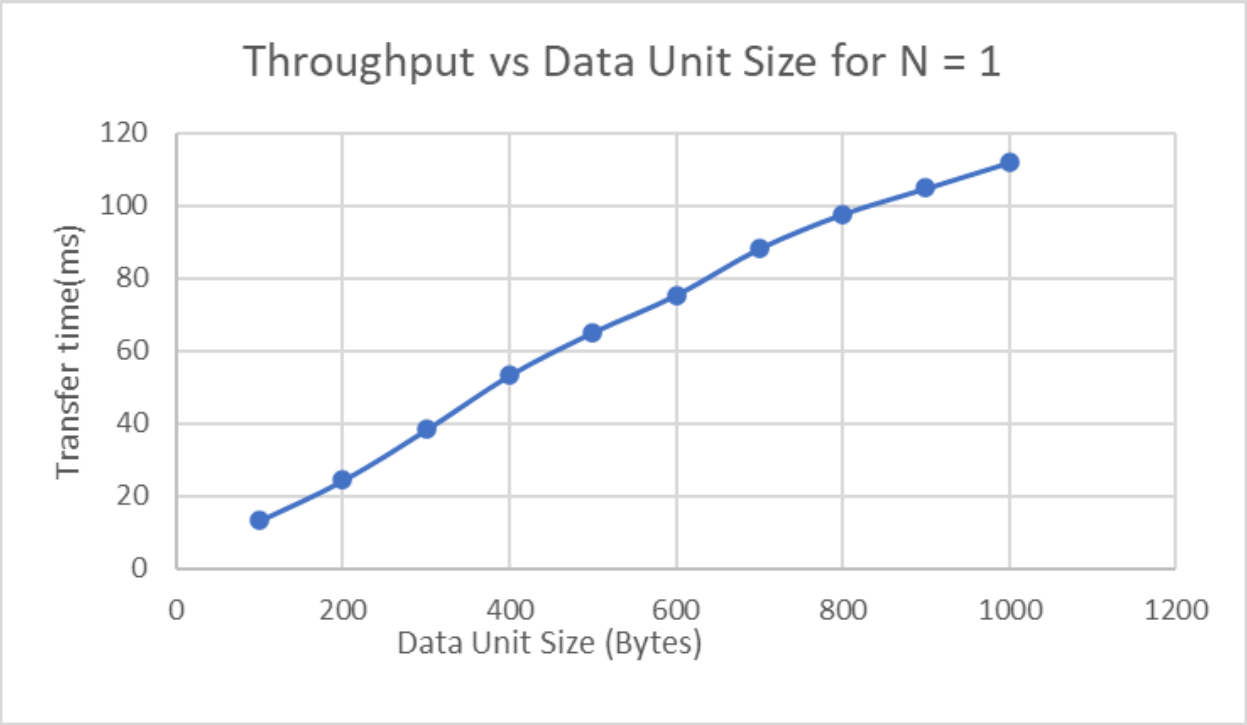
EE4204 Lab Assignment Report

Zhuang Jianning
A0214561M

N = 1

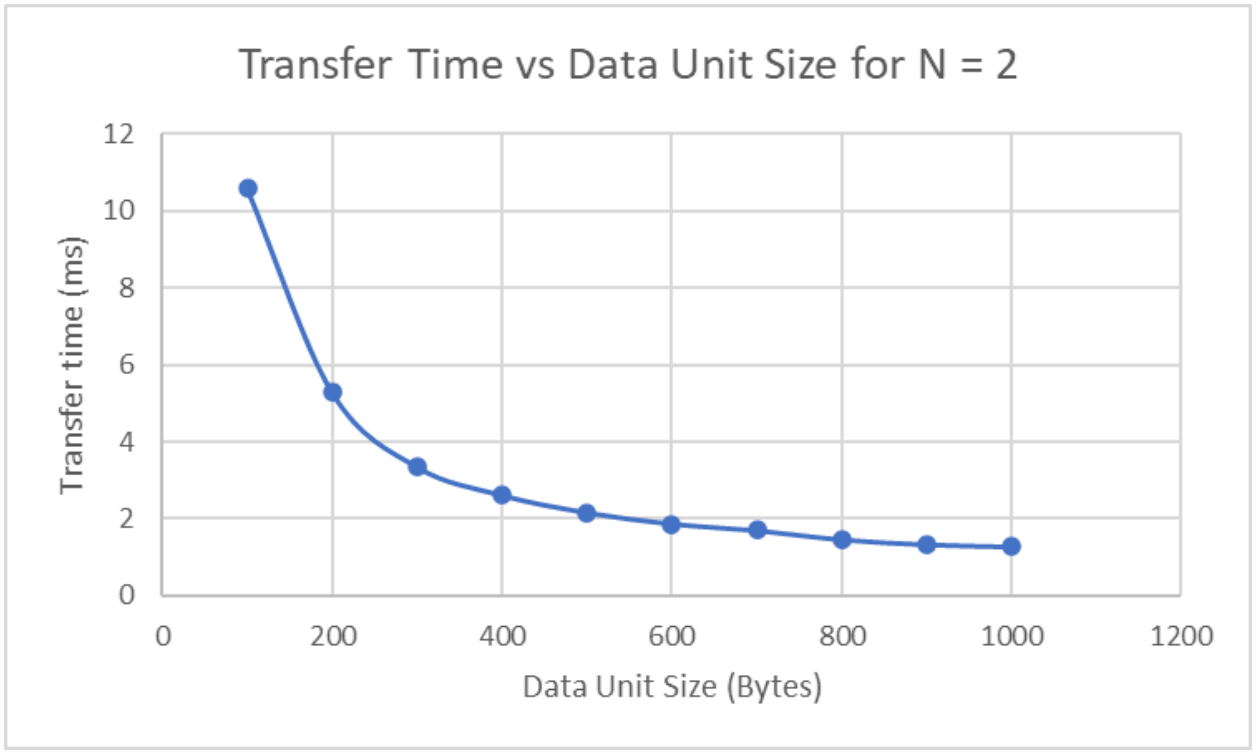
Data Unit Size (Bytes)	Transfer Time (ms)	Throughput (Mbps)
100	18.047	13.253
200	9.824	24.346
300	6.241	38.323
400	4.488	53.292
500	3.674	65.099
600	3.174	75.354
700	2.713	88.159
800	2.450	97.623
900	2.284	104.718
1000	2.139	111.817

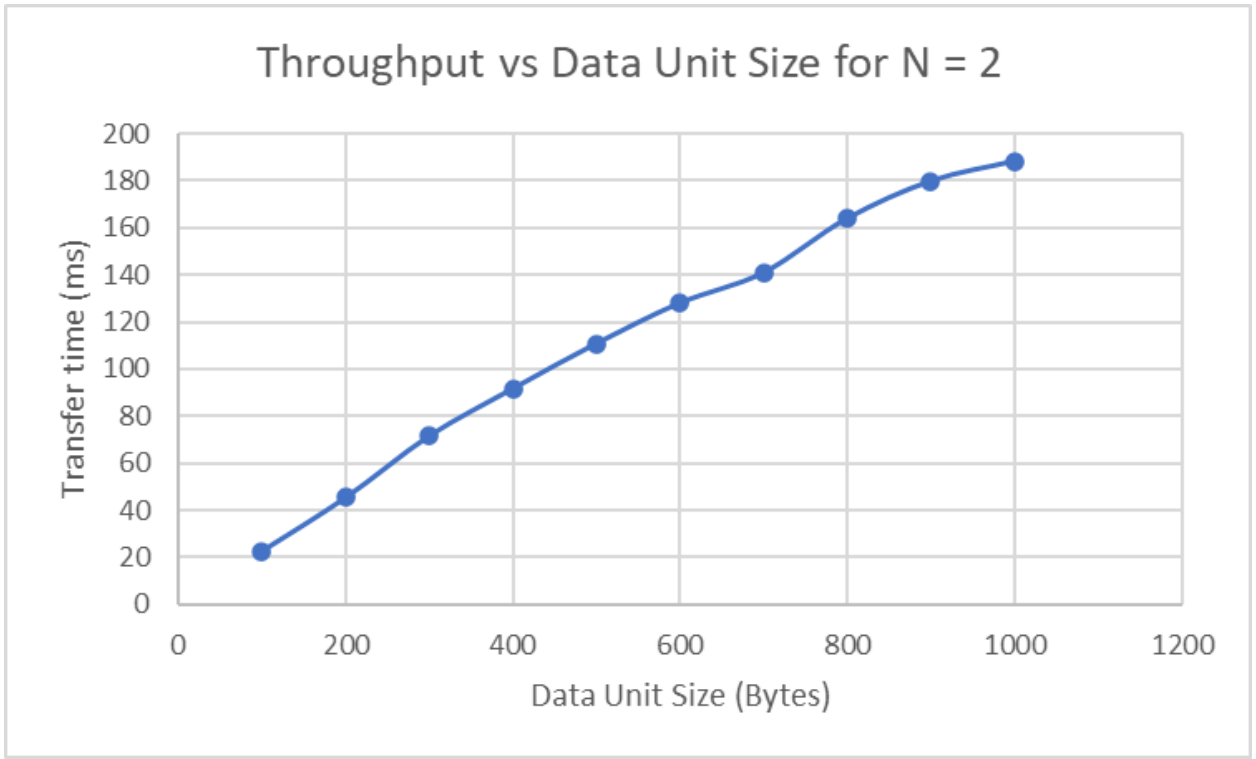




N = 2

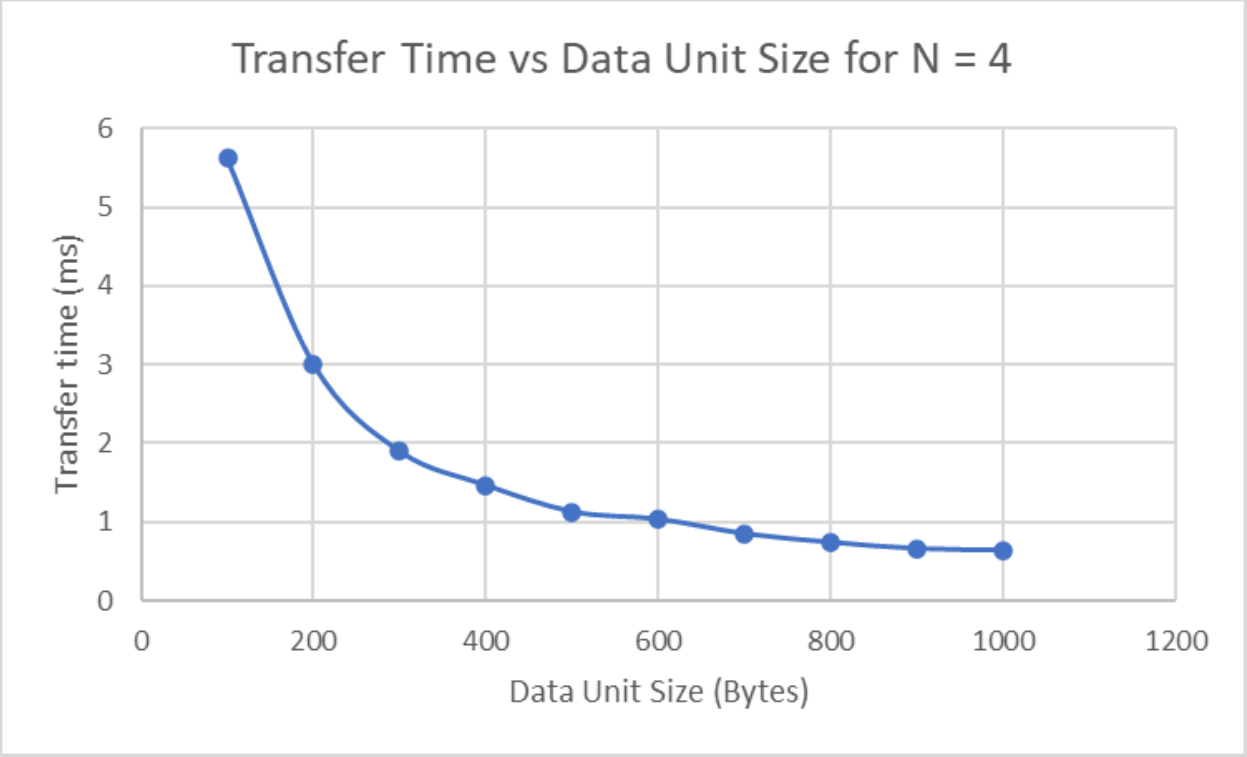
Data Unit Size (Bytes)	Transfer Time (ms)	Throughput (Mbps)
100	10.574	22.619
200	5.276	45.332
300	3.343	71.545
400	2.611	91.603
500	2.158	110.832
600	1.868	128.039
700	1.697	140.940
800	1.459	163.931
900	1.330	179.832
1000	1.271	188.327

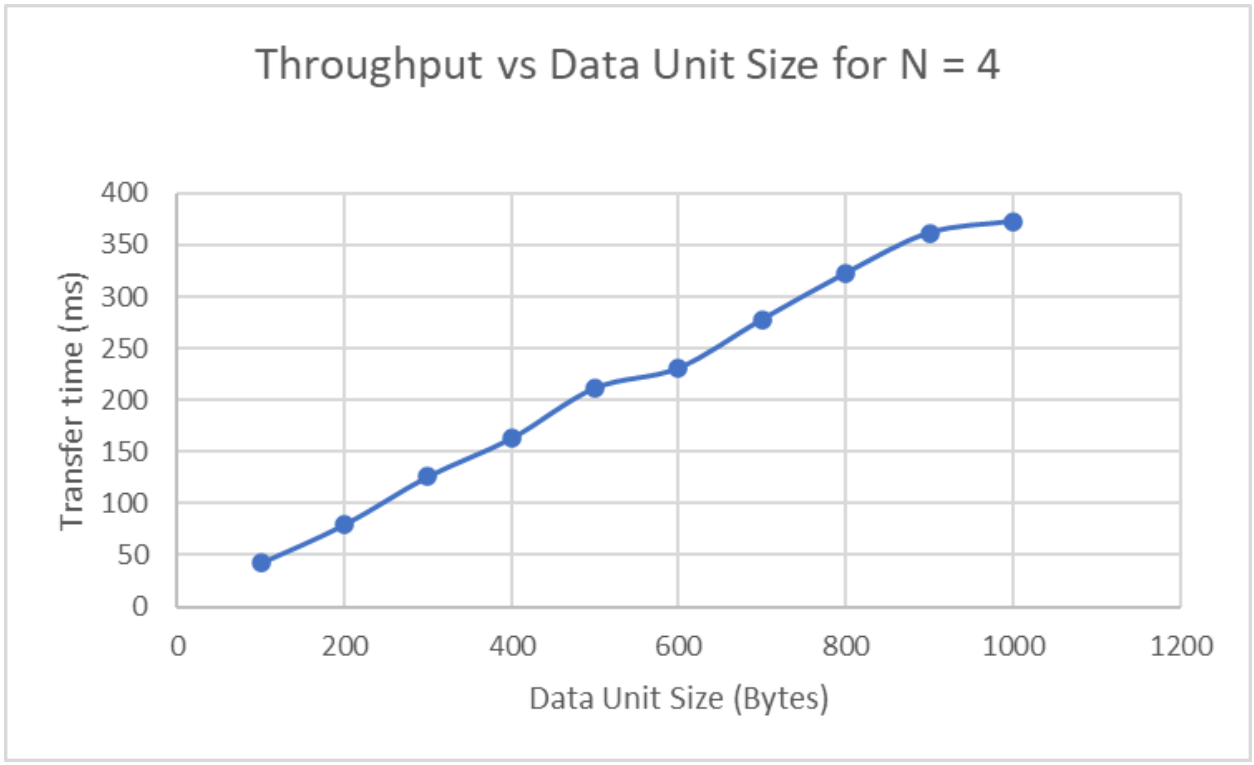

Transfer Time vs Data Unit Size for N = 2
Transfer time (ms)
0
2
4
6
8
10
12
0
200
400
600
800
1000
1200
Data Unit Size (Bytes)

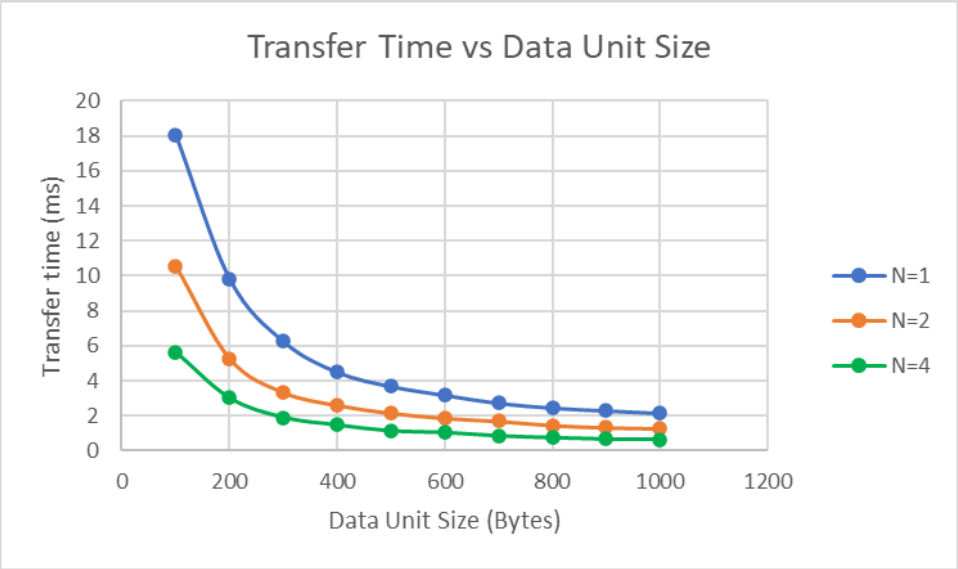

Throughput vs Data Unit Size for N = 2
Transfer time (ms)
0
20
40
60
80
100
120
140
160
180
200
0
200
400
600
800
1000
1200
Data Unit Size (Bytes)

N = 4

Data Unit Size (Bytes)	Transfer Time (ms)	Throughput (Mbps)
100	5.626	42.513
200	3.008	79.513
300	1.898	126.015
400	1.469	162.815
500	1.131	211.473
600	1.036	230.865
700	0.851	277.789
800	0.742	322.339
900	0.662	361.293
1000	0.642	372.548

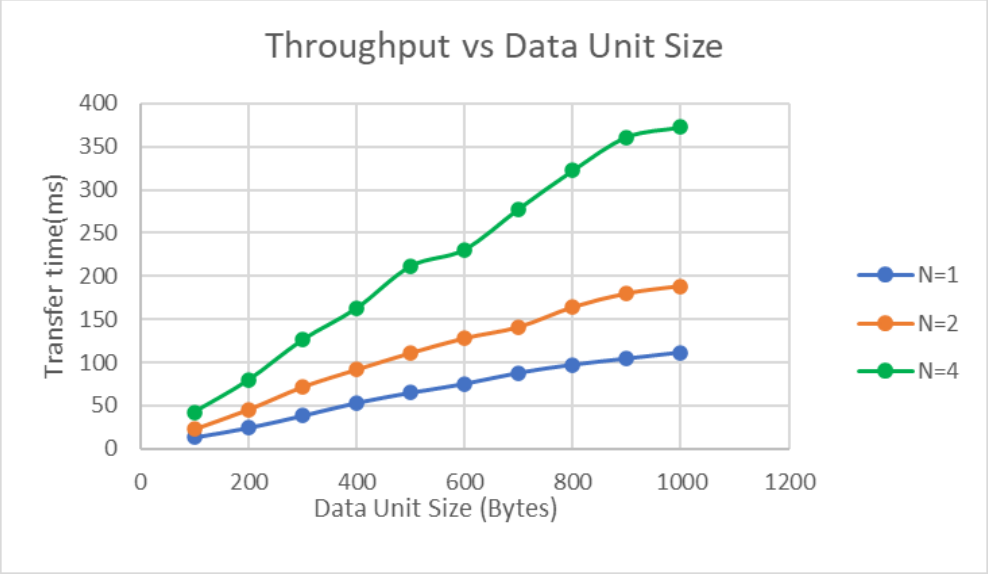



Throughput vs Data Unit Size for N = 4
Transfer time (ms)
400
350
300
250
200
150
100
50
0
0
200
400
600
800
1000
1200
Data Unit Size (Bytes)



Transfer time is inversely proportional to data unit size. As data unit size increases, more data is sent per packet, resulting in fewer packets sent overall. There is less overhead and fewer RTTs required for acknowledgements, hence transfer time decreases.

As N increases, fewer RTTs are required for acknowledgements, hence transfer time decreases proportionally. More discussion in the later section.



Throughput is proportional to data unit size. As throughput is inversely proportional to transfer time with fixed message size, transfer time being inversely proportional to data unit size means throughput is directly proportional to data unit size. More details in discussion.

Discussion

Throughput = message size/transfer time (Mbps)
Throughput is inversely proportional to transfer time with fixed message size

Let message size be M (around 30000 Bytes for myfile.txt)
Let data unit size be D (ranging from 100 Bytes to 1000 Bytes)

When N = 1, this jumping window protocol becomes similar to stop and wait ARQ
Transfer time = M/D * RTT

E.g M = 30000,
D = 100, Transfer Time = 300RTT
D = 200, Transfer Time = 150RTT
…
D = 1000, Transfer Time = 30 RTT

Thus, transfer time is dependent on number of RTT which is inversely proportional to D(see graphs)
Since transfer time is inversely proportional to data unit and throughput is inversely proportional to transfer time, throughput is directly proportional to data unit size (see graphs)

When N = 2
Transfer time = M/ND * RTT = M/2D * RTT

E.g M = 30000,
D = 100, Transfer Time = 150RTT
D = 200, Transfer Time = 75RTT
…
D = 1000, Transfer Time = 15 RTT

Number of RTTs halved as compared to N = 1, roughly halves transfer time as shown in data
E.g 18ms -> 10ms

When N = 4
Transfer time = M/ND * RTT = M/4D * RTT

E.g M = 30000,
D = 100, Transfer Time = 75RTT
D = 200, Transfer Time = 38RTT (round up since last batch still has RTT)
…
D = 1000, Transfer Time = 8 RTT (round up since last batch still has RTT)

Number of RTTs halved as compared to N = 2, roughly halves transfer time as shown in data
E.g 18ms -> 10ms -> 5.6

Determine range of data unit sizes to use:

After testing with various data unit sizes for N = 1, 2 and 4, upper limit of 1000 Bytes chosen since transfer time becomes quite small (<1ms) and number of RTTs also becomes low (8 ACKS) when N = 4.

Lower limit of 100 Bytes chosen since throughput when N = 1 is around 10Mbps.
Range of 100 Byte to 1000 Byte also allows for increments of 100 Bytes to show proportional relationship between throughput and data size

UDP functions

CLIENT

```
//create socket
sockfd = socket(AF_INET, SOCK_DGRAM, 0)

//set server address, sin is short for sockaddr_in
ser_addr.sin_family = AF_INET;
ser_addr.sin_port = htons(MYUDP_PORT);
memcpy(&(ser_addr.sin_addr.s_addr), *addrs, sizeof(struct in_addr));
bzero(&(ser_addr.sin_zero), 8);

//send data to a specific socket
sendto(sockfd, &sends, strlen(sends), 0, addr, addrlen)
```

addr is &ser_addr, address of server to send data to
addrlen is sizeof(struct sockaddr_in)

SERVER

```
//create socket
sockfd = socket(AF_INET, SOCK_DGRAM, 0)

//bind socket to associate socket with the given local address, bind() is usually used when the process intends to use a specific network address and port number
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(MYUDP_PORT);
my_addr.sin_addr.s_addr = INADDR_ANY; //in case system has multiple IP addresses
bzero(&(my_addr.sin_zero), 8);
bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr))

//receive data from client
struct sockaddr_in addr;
len = sizeof (struct sockaddr_in);
recvfrom(sockfd, &recvs, MAXSIZE, 0, (struct sockaddr *)&addr, &len))
```

&addr is client address which is set when data is received
this address can then be used to return ack

//Assignment Implementation
The message is split into short data-units (DUs) which are sent and acknowledged in batches of size n.
The sender sends n DUs and then waits for an ACK before sending the next batch of n DUs.
It repeats the above procedure until the entire file is sent and the acknowledgement for the last batch is received.
The receiver sends an ACK after receiving n DUs.
It repeats the above procedure, until the acknowledgement for the last batch is sent.
Note that the last batch may have less than n DUs.

DATALEN is size of each Data Unit/Packet
BATCHSIZE is number of Data Units sent before ACK required

To send BATCHSIZE Data Units before requiring ACK from server,
a for loop is added within the while loop which keeps track of the current index <= filesize

In each iteration of inner for loop for (int i = 1; i <= BATCHSIZE; i++)
copy data of size DATALEN from buffer to sends array

In the case of sending the last batch with less than BATCHSIZE Data Units, a flag will break out of the for loop (lsize+1-ci) <= DATALEN

After sending BATCHSIZE Data Units, client waits for ACK from server before continuing while loop

Similar for server side, server waits to receive BATCHSIZE Data Units before sending ACK
If last Data Unit received, if (recvs[n-1] == '\0')
set end to 1 and break out of for loop and while loop

//Appropriate values for DATALEN

https://www.geeksforgeeks.org/udp-server-client-implementation-c/