RELIABLE BLAST USER DATA GRAM PROTOCOL

August 23, 2019

Gerrit Burger - 21261687
Jacques Huysamen - 20023669
Computer Science 354 - Networking

Contents

1 Project Description

We were instructed to build a system for transferring file over a networking using the RBUDP protocol. We also measured the performance of this protocol in comparison with TCP, another file transfer protocol. We used Python as our language of choice because of its ease of use and the helpful socket libraries.

1.1 RBUDP protocol

RBUDP aims to provide a solution where guaranteed packet delivery is desirable, but TCP adds too much overhead. The protocol uses UDP to send packets, as UDP sends much faster than TCP. Because UDP is used packets can be dropped and thus a check has to be done to guarantee that the receiver receives all the data of the file before it writes the file. When the sender has sent all of the data packets over UDP, a signal will be sent to the receiver over TCP telling the receiver to check which packets are still missing. The receiver will find all the packets that are missing, by checking which packet ids are still missing, and send the packets ids to the sender via TCP. When the sender receives these missing packet ids it will fetch the corresponding data and again send these packets to the receiver. This loop will continue until the receiver has received all the file data. When the receiver has no more missing data it will signal the sender to say that it has received everything

2 Project Features

2.1 Sender Features

Features included for the sender:

- Simple GUI to select file to be sent, quit the program, show data captured about the file transfer and select the protocol to be used.
- File transferring using the TCP protocol.
- Makes use of data gram packets for data transfer and may use a TCP connection for signalling only.
- File transferring using the RBUDP protocol.
- Creates unique sequence numbers for data gram packets.

2.2 Receiver Features

Features included for the receiver:

- GUI that shows information and progress about the file being transferred.
- Ability to receive files from the sender using TCP or RBUDP protocols.
- Shows the progress of incoming files.
- Handles dropped packets appropriately.
- Handles out of order packets appropriately.

2.3 Features not included

As this project was designed and implemented with the idea of only sending smaller files, there is a limitation to amount of packets that can be sent. A 4-byte number is used to store the packer ID's and this can thus overflow if too many packets are sent. If this

project had to be expanded to send larger files a wrap around implementation would have to be used. This could be done by sending the file's data in a burst, stopping to re transmit the missing packets of that burst, and then resetting the packet ID's used in the first burst to continue sending the next burst of file data. With this implementation much larger files can be sent, but it could possibly increase the amount of time needed to send a file.

3 Algorithms and Data Structures

3.1 Algorithms

We did not use any algorithms in this project

3.2 Data Structures

3.2.1 Sender Data Structures

The python socket library was used in this project to create UDP and TCP sockets. These sockets are connected and bound to the receiver's IP address to be used to send the data and signals. When using the RBUDP protocol the sender sets a buffer size that will be used to determine the amount of bytes to be read in before it is sent over the UDP socket. Packets are numbered by a 2 byte integer starting at zero for the first packet that will be sent. A dictionary to store the data is created where the packet ID is used as a key and the 1024 bytes of data is used as the value, as the data is read and sent it is inserted into the dictionary for later use. When the sender has to signal the receiver over the TCP socket a list containing one or more messages is created and the python pickle library is used to serialize this list to be sent over the socket. The receiver can then receive this list and de-serialize the messages to perform the required actions. When the TCP protocol is used 1024 bytes of data is read and sent to the receiver before again reading another 1024 bytes and sending the bytes. As TCP can not drop packets we do not need any data structures to store the data.

3.2.2 Receiver Data Structures

As with the sender the python socket library was used to create the sockets and bind them to the sender's IP address. When a new file is sent the sender sends the name and size of the file as a array using the pickle library over the TCP socket. These values are stored and used in later calculations. For the TCP protocol the receiver will receive a set amount of bytes from the sender and then write these bytes to a file, this is repeated until the amount of received bytes are equal to the size of the file, the file is thus then fully received. For the RBUDP protocol the receiver will receive packets from the sender containing the packet ID number and a set buffer size of file data, the first four bytes correspond to the packet ID and the rest of the bytes are data. A list is used to store all of the received packet ID's as they are received, the packet ID is spliced from the received bytes and inserted into the list. This list is used to calculate which packets are still missing to be sent to the sender to be re transmitted. A dictionary is used to store all of the received data, the data is spliced from the received bytes and inserted into the dictionary by using the packet ID as the key value. When all of the packets have successfully been received a function is called to assemble and write the data of the file. This function retrieves the file data from the dictionary in the correct order by using the packet ID numbers which are the dictionary keys. The data from the dictionary is then written to a file in the correct order until all of the data has been written and the file transfer is complete.

4 Experiments

During the development of the project a number of experiments was identified to test the differences in perfomance of the TCP and RBUDP protocols and the performance of the RBUDP protocol itself.

4.1 We used the following experiments:

4.1.1 TCP and RBUDP throughput

Hypothesis

The throughput of the TCP protocol is faster than the throughout of the RBUDP protocol.

Method

The program was run 12 times to get the throughput of the TCP protocol and RBUDP protocol respectively. The file sizes used was 1MB, 10MB, 100MB and 1GB. The TCP protocol send was used first to get all of the results. Afterwards the RBUDP protocol was run 8 times to test the throughput.

Results

File size	Sender/Receiver	Time taken (s)	Throughput (MB/s)
1MB	Sender Receiver	0.025 0.0276	39.9743
10MB	Sender Receiver	0.172 0.2376	57.85
100MB	Sender Receiver	2.279 2.3417	43.866
1GB	Sender Receiver	23.25 23.3462	43.02

From the results shown above it is seen that the throughput is between 40 MB/s and 60 MB/s. The TCP protocol throttles itself and as the size of the files get larger the throughput tends to a value between 40MB/s and 45MB/s.

The same files were used and the results for the RBUDP protocol was as follows:

File size	Sender/Receiver	Time taken (s)	Throughput (MB/s)
1MB	Sender	0.0137	121.88
	Receiver	0.1028	
	Sender	0.015	130.2
	Receiver	0.0673	
10MB	Sender	0.0917	117.695
	Receiver	0.1407	
	Sender	0.11	116.9
	Receiver	0.2055	
100MB	Sender	0.8981	116.29
	Receiver	0.9474	
	Sender	0.89	116.33
	Receiver	1.0516	
1GB	Sender	8.974	115.67
	Receiver	9.0701	
	Sender	8.968	115.7
	Receiver	9.0571	

From the results above it is seen that the RBUDP protocol's throughput also declines as the file size gets larger and tends to the range between 115MB/s and 120MB/s.

Conclusion

From the results it is apparent that the RBUDP protocol's throughput is quite a lot faster than that of the TCP protocol. It is also apparent that the RBUDP protocol sends data faster than TCP protocol as the time taken for the RBUDP protocol is lower than that of the TCP protocol. Hence, the hypothesis is declined and it is proven that the RBUDP protocol has a faster throughput than the TCP protocol.

4.1.2 Packet size used with RBUDP and TCP

Hypothesis

As packet sizes decrease the speed of the protocols will increase.

Method

For this experiment the program was run 12 times, 4 times for each specified packet size, one time each for each file size (1MB, 10MB, 100MB and 1GB). The time taken as well

as the packet loss (will be used in the next experiment) was measured for each run. The packet loss was measured with two formulas, one for an under estimation and one for an overestimation (See the following experiment for more on that).

Results

Packet size	**File size**	**Sender/Receiver**	**Time taken (s)**	**Packet loss (%)**	
				Under estimate	Over estimate
500 bytes	1MB	Sender Receiver	0.017 0.1	0	10
	10MB	Sender Receiver	0.12 0.32	5	6
	100MB	Sender Receiver	1.1 1.01	3	12
	1GB	Sender Receiver	10.14 10.23	2	1

Packet size	**File size**	**Sender/Receiver**	**Time taken (s)**	**Packet loss (%)**	
				Under estimate	Over estimate
1024 bytes	1MB	Sender Receiver	0.0137 0.1028	8	21
	10MB	Sender Receiver	0.0917 0.1407	0	1
	100MB	Sender Receiver	0.8981 0.9474	0	0.396
	1GB	Sender Receiver	8.974 9.0701	0	0.6459

Packet size	**File size**	**Sender/Receiver**	**Time taken (s)**	**Packet loss (%)**	
				Under estimate	Over estimate
5120 bytes	1MB	Sender Receiver	0.01 0.0994	0	0
	10MB	Sender Receiver	0.08 0.1372	0	0
	100MB	Sender Receiver	0.85 0.9	0	0.031
	1GB	Sender Receiver	8.54 8.6	0	0.16

Conclusion

From the results it is clear to see that as the packet sizes increase so does the speed of the transmission, it is easiest to see on the 1GB files. When the packet size was 500 bytes the transfer time was +/- 10.2 seconds and for a packet size of 1024 bytes the transfer

time was +/- 9 seconds, this was decrease ever further to +/- 8.6 seconds with a packet size of 5120 bytes. It is also apparent that the packet loss decreased a lot (more on that in the next experiment). Hence the hypothesis is declined and it is established that as the packet size increases so does the speed of the protocol.

4.1.3 Packet loss with RBUDP on varying packet sizes

Hypothesis

As the packet sizes increase the packet loss will increase.

Method

This experiment method is linked to that of the previous. The packet loss is calculated in three different ways. Initially it was calculated with the following formula:

$$packetloss = (\frac{missing\ packets}{amountofpackets * transmissioncount}) * 100,$$

where the transmission count is the amount of times packets were sent until all of the packets are received.
This gave an under estimation. If a minuscule amount of packets are dropped the transmission count also increases and does not scale correctly, giving a less accurate measurement for the packet loss percentage if small amount of packages are dropped many times.

The formula was modified as follows and ran again,

$$packetloss = (\frac{missingpackets}{amountofpackets})$$

This gave an over estimation. If a large amount of packet is sent many times this could result in the value of missing packets being more than the value of the amount of packets, resulting in a value like 105%, 113%, etc. which is obviously not possible. Also it does not account for packets that are missed twice or more.

A good solution to the problem of both over and under estimation would be to create a dictionary to store previously missed packets, if a packet has been dropped before increase the size of the missing packets as well as the size of the amount of packets, hence scaling the amount of packets to be able to give a more accurate result.

Results

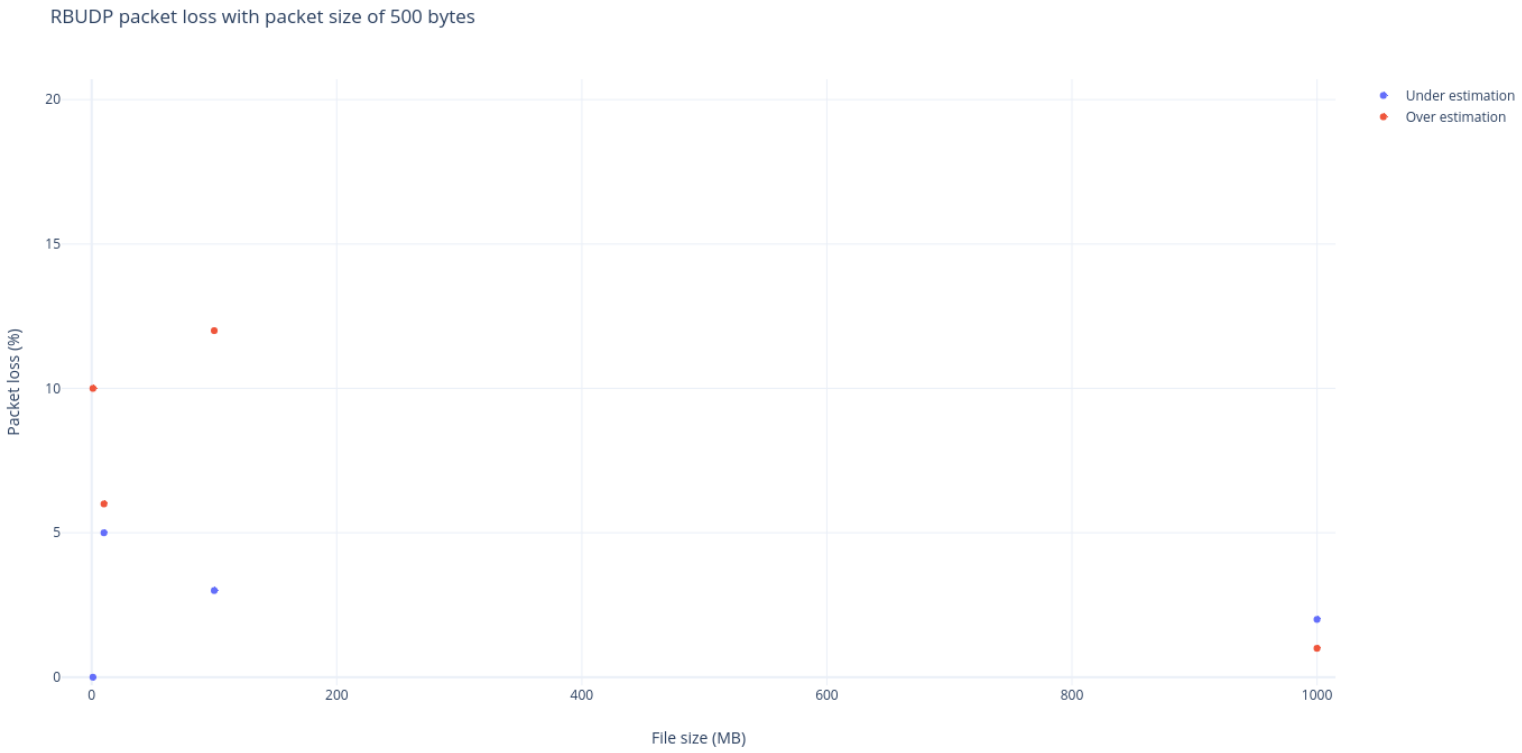

Figure 4.1: Packet loss for packet size 500 bytes

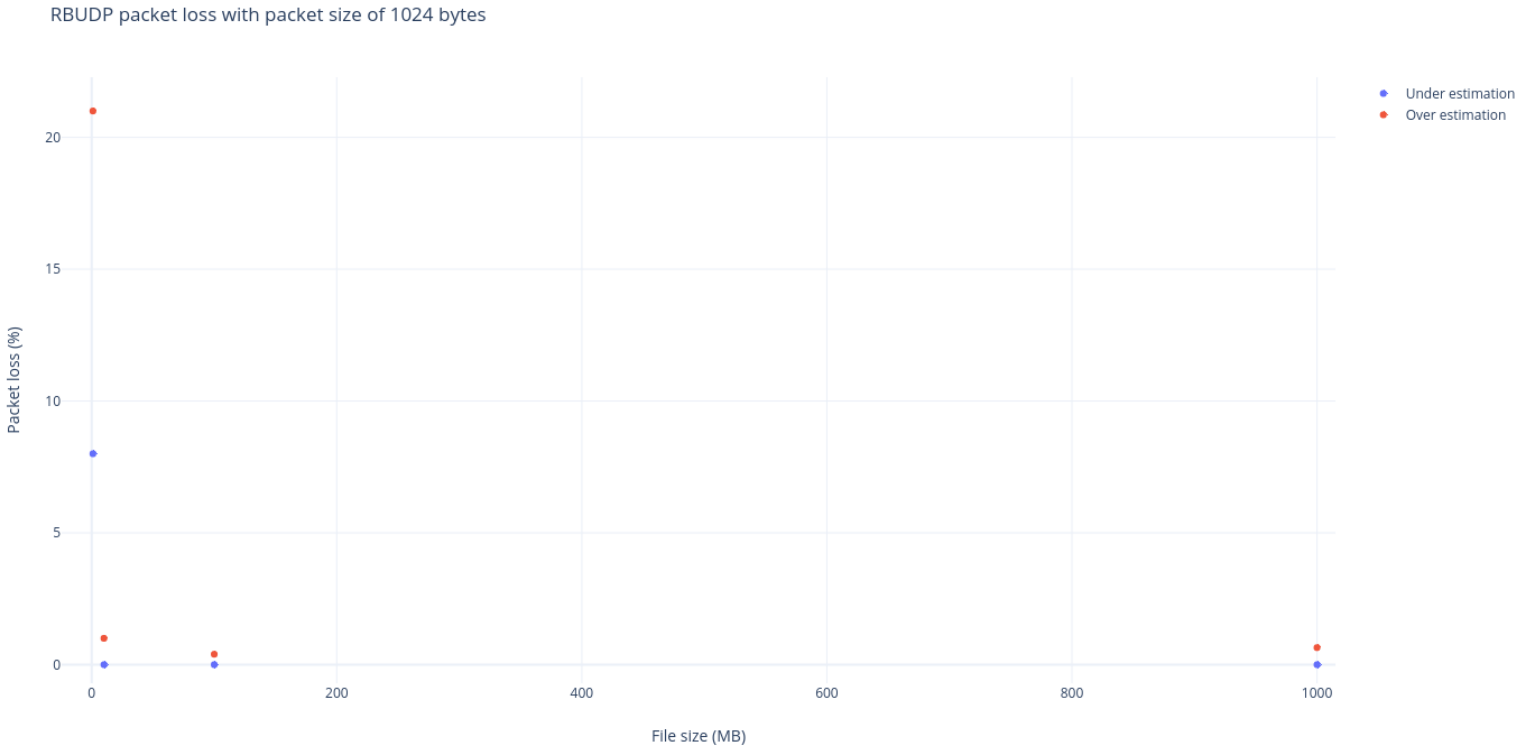

Figure 4.2: Packet loss for packet size 1024 bytes

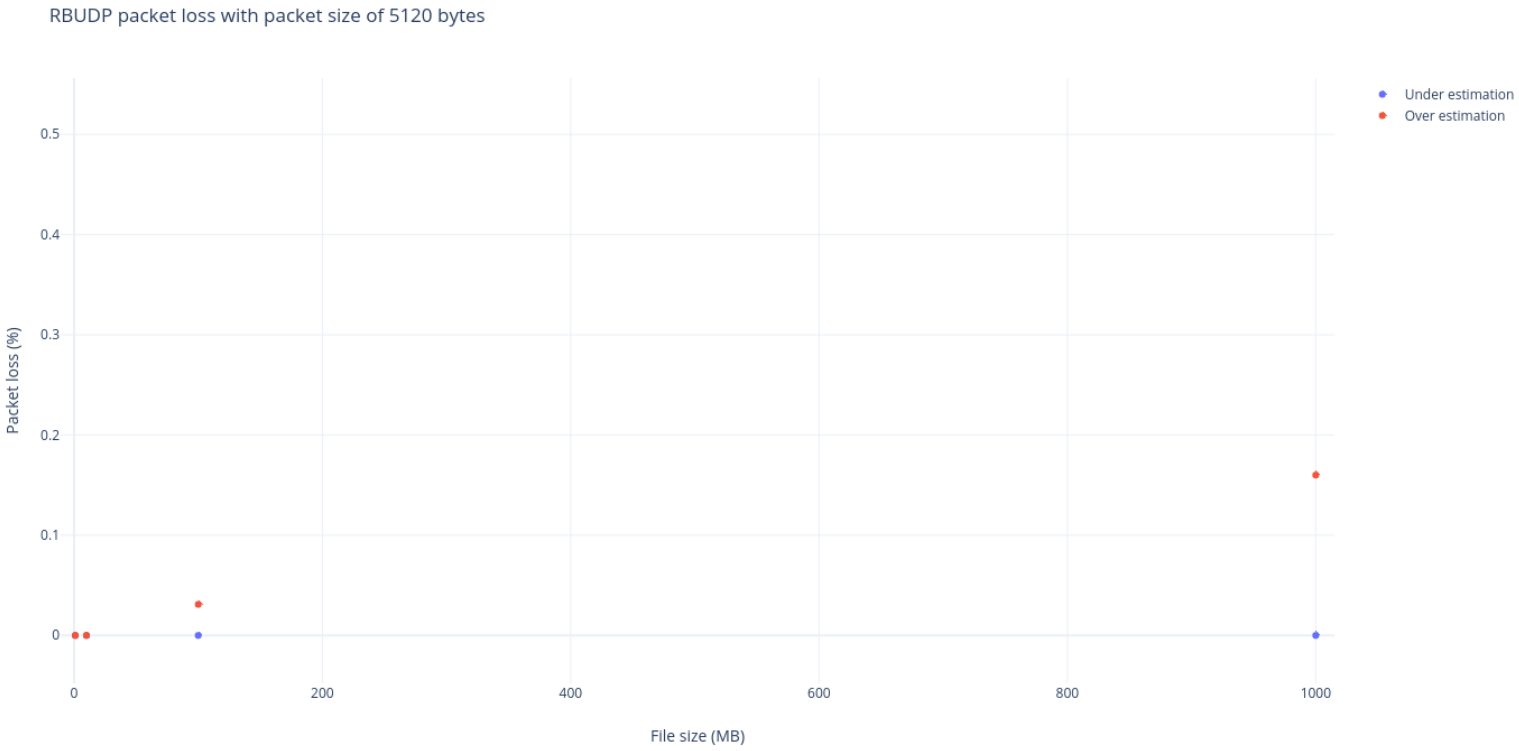

Figure 4.3: Packet loss for packet size 5120 bytes

Conclusion

From the data collected it is easy to see that as the packet size increases the packet loss percentage decreases drastically. Hence the hypothesis must be declined.

4.1.4 Influence of packet drop rate on RBUDP protocol

Hypothesis

The time taken will increase severely as the packet drop rate is increased.

Method

To replicate varying packet loss rates random packet drops were enforced for the sender. For a packet drop rate of 10% only 90% of the file's data is sent to the receiver originally to replicate the packet drop rate. Three different packet drop rates were used and for each 3 different files were used with increasing file size. The time to send and receive was recorded to come to a conclusion.

Results

Packet drop rate	File size	Sender/Receiver	Time taken (s)	Packet loss (%)	
				Under estimate	Over estimate
10%	1MB	Sender Receiver	0.0133 0.1027	8.547	17.093
	10MB	Sender Receiver	0.1111 0.1611	8.688	17.377
	100MB	Sender Receiver	1.0743 1.1275	5.174	10.347

Packet drop rate	File size	Sender/Receiver	Time taken (s)	Packet loss (%)	
				Under estimate	Over estimate
20%	1MB	Sender Receiver	0.0181 0.1077	12.112	36.336
	10MB	Sender Receiver	0.097 0.1865	9.999	19.998
	100MB	Sender Receiver	0.9958 1.0857	6.786	20.357

Packet drop rate	File size	Sender/Receiver	Time taken (s)	Packet loss (%)	
				Under estimate	Over estimate
50%	1MB	Sender Receiver	0.0153 0.1048	29.974	49.949
	10MB	Sender Receiver	0.1135 0.2031	24.995	49.99
	100MB	Sender Receiver	1.1859 1.2383	16.785	50.356

Conclusion

As seen from the tables above we can come to the conclusion that the packet drop has a very small influence on the time taken to send and receive files. As the drop rate is increased there is a small increase in the time taken to send a file of the same size, as the test were run on a large network with a large amount of connected clients these small time differences can also be influenced by other network activity.A conclusion can be made that the packet drop rate has a very small influence on the speed of the file transfer. We reject the hypothesis.

4.1.5 Sha256sum testing with RBUDP and TCP

Hypothesis

The RBUDP and TCP protocols both will received files without being corrupted and without data integrity lost.

Method

The program was run for both TCP and RBUDP for the file sizes 1MB, 10MB, 100MB and 1GB. After the receiver has saved the file the command "sha256sum <file name>" was run on the sender pc and the receiver pc and then the hash returned was compared to see whether they match.

Results

Hash for the 1MB file:
RBUDP protocol:

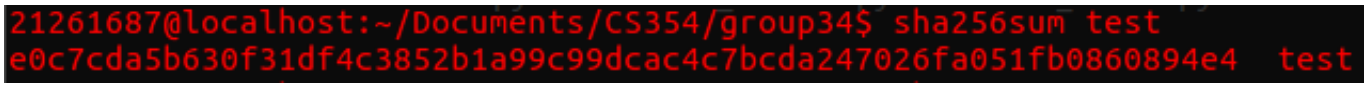

Figure 4.4: Receiver sha256sum RBUDP

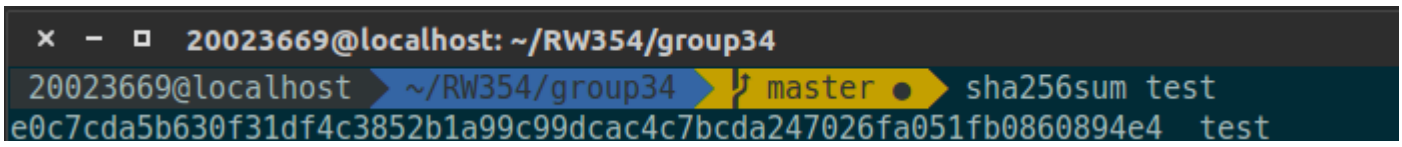

Figure 4.5: Sender sha256sum RBUDP

TCP protocol:

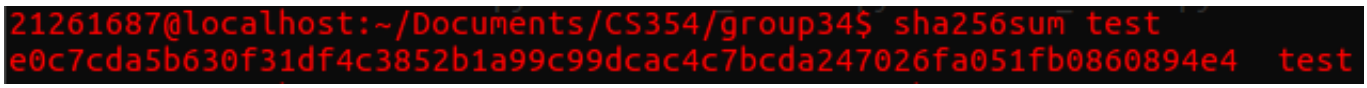

Figure 4.6: Receiver sha256sum TCP

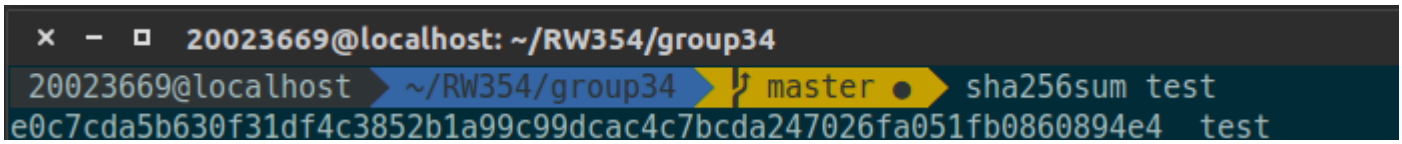

Figure 4.7: Sender sha256sum TCP

Hash for the 10MB file:
RBUDP protocol:

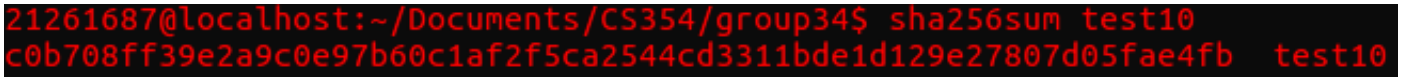

Figure 4.8: Receiver sha256sum RBUDP

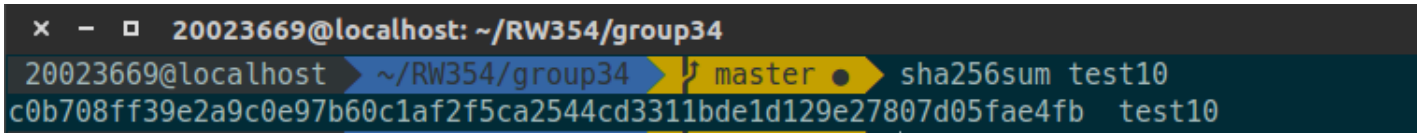

Figure 4.9: Sender sha256sum RBUDP

TCP protocol:

```
21261687@localhost:~/Documents/CS354/group34$ sha256sum test10
c0b708ff39e2a9c0e97b60c1af2f5ca2544cd3311bde1d129e27807d05fae4fb  test10
```

Figure 4.10: Receiver sha256sum TCP

```
20023669@localhost: ~/RW354/group34
20023669@localhost  ~/RW354/group34  master ●  sha256sum test10
c0b708ff39e2a9c0e97b60c1af2f5ca2544cd3311bde1d129e27807d05fae4fb  test10
```

Figure 4.11: Sender sha256sum TCP

Hash for the 100MB file:

RBUDP protocol:

```
21261687@localhost:~/Documents/CS354/group34$ sha256sum test100
bf9c4ba0b0bb7cb66bf98e51f578e2ef7f59c3f43eb98dbf46312b8f7d73c04c  test100
```

Figure 4.12: Receiver sha256sum RBUDP

```
20023669@localhost: ~/RW354/group34
20023669@localhost  ~/RW354/group34  master ●  sha256sum test100
bf9c4ba0b0bb7cb66bf98e51f578e2ef7f59c3f43eb98dbf46312b8f7d73c04c  test100
```

Figure 4.13: Sender sha256sum RBUDP

TCP protocol:

```
21261687@localhost:~/Documents/CS354/group34$ sha256sum test100
bf9c4ba0b0bb7cb66bf98e51f578e2ef7f59c3f43eb98dbf46312b8f7d73c04c  test100
```

Figure 4.14: Receiver sha256sum TCP

```
20023669@localhost: ~/RW354/group34
20023669@localhost  ~/RW354/group34  master ●  sha256sum test100
bf9c4ba0b0bb7cb66bf98e51f578e2ef7f59c3f43eb98dbf46312b8f7d73c04c  test100
```

Figure 4.15: Sender sha256sum TCP

Hash for the 1GB file:

RBUDP protocol:

```
21261687@localhost:~/Documents/CS354/group34$ sha256sum testGB
5b47f5ecd2ecac1207f793ce56432a9d1dd703a63438494a156b616d4b8da87c  testGB
```

Figure 4.16: Receiver sha256sum RBUDP

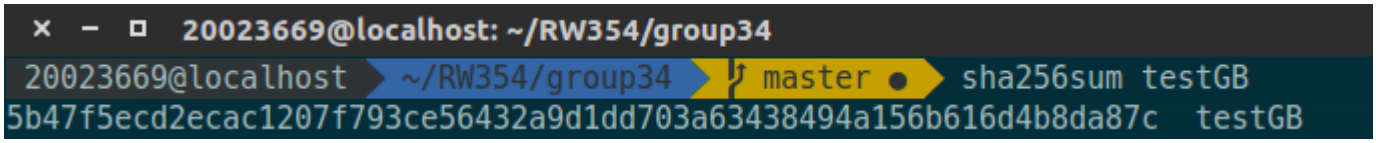

Figure 4.17: Sender sha256sum RBUDP

TCP protocol:

Figure 4.18: Receiver sha256sum TCP

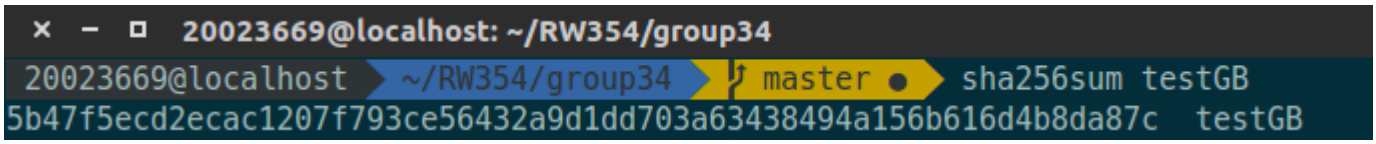

Figure 4.19: Sender sha256sum TCP

Conclusion

From the results it is apparent that the hypothesis can be accepted, no data integrity was lost in the process of sending data with the TCP or RBUDP protocols.

5 Conclusion

After completion of the project we found that the following was apparent:

- As TCP protocol throttles itself and RBUDP protocol does not, the throughput of the RBUDP protocol is much higher than that of the TCP protocol.
- The speed of the RBUDP protocol is increased when the packet sizes are increased.
- The packet loss of the RBUDP protocol is significantly lower with larger packet sizes.
- No data integrity is lost in the process of sending files via either the TCP or RBUDP protocol.
- When using RBUDP the packet loss rate has a small infleunce on the time needed to transfer a file.

We can come to the conclusion that RBUDP is a good solution when sending large amounts of data. It is much faster than TCP when larger files are being used. A possible bottleneck in regards to RBUDP could be the amount of memory the protocol uses because of the complexity and many data structures needed to store missing data that could be needed for re transmission.