MAY 2, 2023

PORTFOLIO 2: DATA2410

RELIABLE TRANSPORT PROTOCOL (DRTP)

SERINA SERIFE ERZENGIN S364561
SILJE FLAGLIEN S364579
VU QUANG NGUYEN 364540

1 Introduction

In today's hyper-connected world, reliable data transfer has become increasingly crucial. Real-time data processing and the emergence of new technologies like the Internet of Things (IoT) have made the need for reliable data transfer more crucial than ever. Reliable data transfer is critical because of the devastating consequences that data loss or corruption can cause. This is particularly true in industries such as healthcare, where a loss or corruption of data can have life-threatening consequences.

Additionally, there are numerous applications such as electronic mail and web document transfer, where reliable data transfer is essential to ensure that the data transmitted from one end of the application is received correctly and completely at the other end. When a protocol provides such a guaranteed data delivery service it is known as reliable data transfer (Kurose & Ross, 2021, 8.edition).

In this project, we have implemented a simple transport protocol – DATA2410 Reliable Transport Protocol (DRTP) - that provides reliable data delivery on top of the User Datagram Protocol (UDP). By adding acknowledgment, timers, sequence numbers, and retransmission mechanisms, we guarantee that data sent by one process will arrive intact at the destination process. Along with this we also implemented a file transfer application, consisting of a client and a server communicating using the DRTP protocol. Lastly, the application was evaluated through a series of test cases, conducted on a virtual network managed by Mininet inside a virtual machine using UTM. Various tests have been conducted to evaluate the reliability functionality of the protocol under different modes and RTT values. Furthermore, we have created fictitious test cases to show how well our technology handles various network scenarios.

The report's remaining sections are arranged as follows: The background of the project is presented in Section 2, and the building blocks of the DRTP and the client-server communication are discussed in Section 3. Section 4 describes the experimental setup, and finally, Section 5 presents the discussion of the different test cases.

2 Background

UDP (User Datagram Protocol) and TCP (transmission Control Protocol) are the two commonly used transport layer protocols in computer networking. For applications like streaming media, DNS lookups and online gaming, UDP is widely used as it is a connectionless protocol that provides

unreliable, but fast communication with minimal overhead (Ashtari, 2022). On the other hand, TCP is a connection-oriented protocol that ensures reliable and ordered delivery of data (Kurose & Ross, 2020), which makes it appropriate for web browsing and file transfer applications.

As we have been working with TCP (which already provides reliable delivery of data through its sliding window mechanism (Kurose & Ross, 2020)) previously, this project will mainly focus on recreating the three simple reliability functions: Stop-and-wait, Go-Back-N (GBN) and Selective-Repeat (SR) for UDP.

Stop-and-wait is a simple flow control mechanism used in communication systems. A single packet in this protocol is sent, and the client will intentionally wait for the receiver to confirm that packet and acknowledge it before sending a new packet. If the sender has not received any message from receiver or packets are being sent in wrong order, it will ask for retransmission. Stop-and-wait protocol is quite simple, but it can be very slow and inefficient, especially for high-bandwidths or high-latency networks.

Sliding windows methods, such as GBN and SR, are used to improve the efficiency of transferring data through unreliable networks. When a single packet is lost or corrupted, the GBN protocol retransmits all unacknowledged packets in a window. However, SR is more effective than GBN since it just retransmits the lost packet. Moreover, these protocols allow multiple packets to be in transits at the same time by using a sliding window to keep track of the sent and received data. For that reason, it consequently increases network throughput.

3 Reliable Transport Protocol (DRTP)

Our implementation of DRTP is divided into many sections: header handling, bonus and other, arguments parsing, establish/close connections between client and server, and the three main reliable functions.

The header handling section consists of “create_packet”, “parse_header” and “parse_flags” methods. “Create_packet” is used for creating packets for transferring, “parse_header” extracts the upper part of each packet, and “parse_flags” method will return SYN, ACK, FIN flag of each packet.

Furthermore, in “bonus and other” section, we have created a roundtrip time function for calculating RTT and set it as timeout value when bonus mode is activated. Otherwise, the timeout will be set as 500ms as default. Besides that, a “throughput” function has also been added to calculate throughput of transferring process.
At the end of application.py is the building block of arguments parsing. This block includes three separate groups containing arguments for server, client, and both. For server, it contains arguments like “server” and “bind”. Nevertheless, client has multiple arguments such as “client”, “serverip”, “window” (for specifying the window size in GBN and SR modes), and “bonus” (by writing -B in terminal, the bonus exercise will be activated). Both client and server have these common arguments: “port”, “modus”, “file” and “test”. By specifying “file” argument as client/sender, user must include a file that needs to be sent. Contrarily, server/receiver will name the received file after this “file” flag. For testing purposes, client and server must use “loss” or “skipack” followed by “test” argument respectively to ask for retransmission.

In order to establish connections between client and server, we have used two functions “connection_establishment_client” and “connection_establishment_server” to achieve this. Ideally, the connection between them is established due to the three-way handshake principle. Once the transferring process has been completed, the corresponding functions will call “close_connection_client” for client and “close_connection_server” for server to gracefully close the connection. Additionally, we also implement a function called “file_splitting”. This function will take in a file as argument, divide it into smaller parts and put them in a list. By doing this method, we can keep track of which packets in the list the protocols are working on. Now, let’s move to the three reliability functions.

Firstly, SAW_client will split the file to a list to handle packets sending. Each item in the list equally contains 1460 bytes (except the last item because it is only rests). We will have a sequence id variable representing sequence number of each packet. If there are still unsent packets in the list, the sender will continuously send packets until all data has been sent. After sending a packet, the client will set a timeout and wait for packet’s ACK. Server will wait for packets from client and check if it yet has received the right packet. If the right packet has arrived, the server will send a ACK message with the related ACK_number to the client for confirmation. When the client receives the right ACK message, the sequence id will increase by 1 and the next packet will be sent. If something is wrong with packet (timeout while waiting for ACK; or packet loss test makes the packets sent in wrong

order), the server will send a DUPACK (ACK number of the last ACK packet) and ask the client to resend the right packet.

Secondly, GBN_client will split the file into smaller parts in a list and send the number of packets based on the sliding window size. In this function, we will have variables “base” and “next_to_send”. “Base” refers to the “first” packet in sliding window, and “next_to_send” represents the next packet that will gradually be sent. GBN_client will always send packets until the window is full if there are still unsent packets in the list. By sending packet, “next_to_send” will increase by 1 as a result to keep track of the next packet. Once the window is full, it will wait for ACK message from server before moving the sliding window. At the time the client receives the right ACK message, “base” will increase by 1 meaning the sliding windows has moved one step to the right side of the data list. GBN_server will always check if packets arrive in the right order in case of packet loss or ACK message is dropped.
If these test cases take place, the server will notify the client about this, discard all the packets that have not previously been acknowledged and ask for retransmission.
Lastly, SR_client will also split the file as the other two functions. However, to keep track of packet status, we have created another dictionary containing all information of the packets. Each packet will therefore have three key items: sequence number aka “seq_num”, “data” and “acked”. “acked” in this case store a Boolean value illustrating whether the packets have been acknowledged. The sending process is almost the same as GBN where packets are being sent until the window is full (“next_to_end” will also increase by 1 as soon as a packet is sent). Once the number of packets sent has reached the window size, the client will wait for the corresponding ACK message from server. However, the server will receive all the packets from the client, increase last_ack_sent, put them in receiver buffer even though packets might arrive in the wrong order and send ACK messages to client. When the client receives the right ACK messages, it will go in the list and mark packets as “ACKed”. After marking ACKed, there is a loop going through the list and constantly increasing the value of “base”. In cases of packet loss/ACK skipping that leads to timeout, the client will again go through the list from index of “base” to “base” + “WINDOW_SIZE”. At this slice of the list, the client will resend all the packets that have not been marked as “acked”, and immediately wait for ACK messages of the newly sent packets before relocating the sliding window. The sequence number of the received packet is checked to see if it is less than "last_ack_sent" on the server side since "last_ack_sent" constantly increases despite the incorrect order. A sequence number that is less than "last_ack_sent" indicates that the incorrect order packet has finally been delivered. Eventually, the

server will use the list's built-in function - "insert" - to place the data of this packet in the appropriate location.

4 Experimental setup

In this project, we will use a simple topology to experiment and evaluate the three reliable functions. The use of virtual networks allows us to test the application without the need for real hardware.

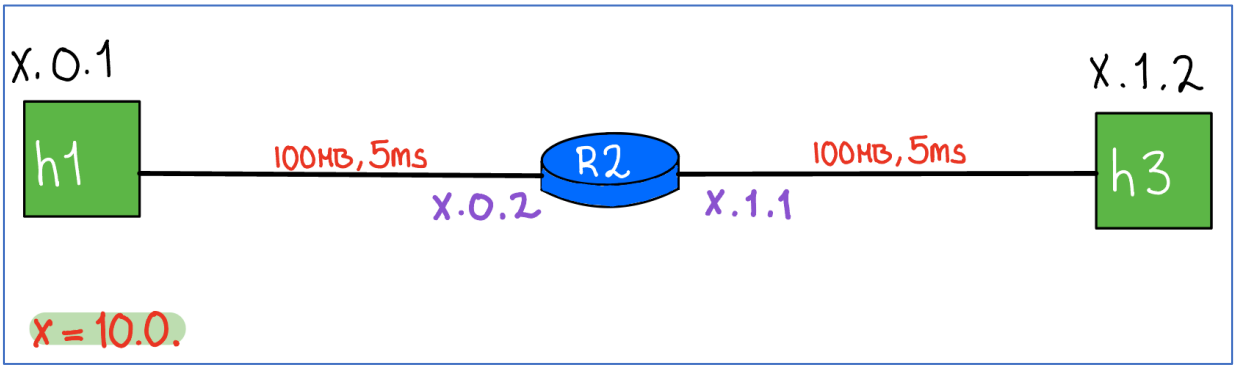

Figure 1. Network Topology

The virtual network is created with the use of the Python library “Mininet”, and the topology consists of 2 hosts (h1 and h3) and 1 router (R1). The two hosts h1 and h3 have IP addresses 10.0.0.1 and 10.0.1.2 respectively. They will connect to the router by addresses 10.0.0.2 and 10.0.1.1. The hosts and routers are connected with links. Each link has a pre-set bandwidth and delay. In Figure 1 above you can see the topology’s delay and bandwidth values, and the IP-addresses. To sum up, this topology will be used for testing our application.

5 Discussion

5.1 Network tools

In the context of this project, we have used a network tool: Ping. Ping measures the round-trip time (RTT), which is the time it takes a packet to travel from the sender to the destination and all the way back again. The RTT can be useful for evaluating the speed and reliability of the network (Own work, 2023, Simpleperf-portfolio 1). In this project, the software tool Ping is utilized in the bonus task for the purpose of measuring the RTT between the host in the network topology. With the measured RTT, the socket timeout will have an optimized timeout based on the RTT. The optimized RTT minimizes idle waiting caused by timeout, which in total increases the throughput and efficiency of the protocol.

Besides that, we also use “tc-netem” to provide a network emulation which allows us to add delay, packet loss, duplication, and other conditions to simulate different network scenarios. With this tool,

we can observe how well our code could handle the tests as well as the efficiency of the three network protocols.

5.2 Performance metrics

In order to assess our application, we have, among other things, used a performance metric: throughput. Throughput refers to the rate at which bits are transmitted from sender to receiver. It describes the amount of data that can be transferred through a network in a specific amount of time. A high throughput indicates that the system can process data quickly, which is why throughput is crucial for performance evaluation. Low throughput delays may arise because of the system's inability to handle all requests quickly (Own work, 2023, Simpleperf-portfolio 1). By analyzing throughput, we can evaluate the efficiency of the three protocols SAW, GBN and SR. Additionally it gives us the opportunity to compare them, to determine which protocol is the most effective in different scenarios.

5.3 Test case 1: Performance Comparison of Stop-and-Wait and Go-Back-N Protocols with varying window sizes and RTTs.

The application can use 3 different modus to ensure that data packets are delivered to the server without loss or corruption: SAW, GBN and SR. In this test case we experimented with different round-trip times (RTTs) of 25, 50 and 100ms for each mode and calculated throughput for each test. Moreover, we performed additional tests for SR and GBN with different window sizes of 5, 10 and 15. As a result, we got three main tests to assess the performance of the modes.

5.3.1 Results

SAW:

RTT	Throughput
25ms	435.88 Mbps
50ms	223.76 Mbps
100ms	111.57 Mbps

GBN:

Throughput at:	Window 5	Window 10	Window 15
25 RTT	2097.99 Mbps	3985.05 Mbps	5611.09 Mbps
50 RTT	1077.30 Mbps	2010.94 Mbps	2878.98 Mbps
100 RTT	533.16 Mbps	1016.37 Mbps	1421.98 Mbps

SR:

Throughput at:	Window 5	Window 10	Window 15
25 RTT	2106.78 Mbps	3978.85 Mbps	5609.92 Mbps
50 RTT	1076.30 Mbps	2049.39 Mbps	2907.86 Mbps
100 RTT	532.66 Mbps	1022.06 Mbps	1412.77 Mbps

5.3.2 Discussion

As mentioned, in the SAW protocol the sender waits for the server to acknowledge the transmitted data packet, before sending another. This means that the protocol only sends one packet at a time. Thus, we expected the SAW protocol to have the slowest data transfer rate of the 3 modes. On the other hand, both GBN and SR are “sliding window” protocols, which means that several data packets are sent before waiting for confirmation from the receiver. Therefore, we anticipated that their throughput would be significantly higher than the Stop-and-wait protocol. The GBN and SR protocols are similar in the way they send data packets, but as explained earlier, they differ in how they handle the loss of data packets and missing acknowledgments. Since no packet loss is generated in this experiment, we therefore expected that GBN and SR would have approximately the same throughput.

Regarding the various tests performed with different RTTs, we hypothesized that the efficiency of the data transfer would be affected by increasing RTT values. We anticipated that higher RTT values would lead to a decrease in throughput. The reason for this is that when there is more delay on the link, it takes longer for a packet to reach the end host, and longer for an ack to be returned to the sender. As a result, the final throughput will be less since packet transmissions take longer. As predicted, our results showed that increasing RTT values led to lower throughput. Additionally, SAW had the lowest throughput in all RTT cases, while SAW and GBN had throughputs that were substantially higher but relatively equivalent. Therefore, the result supported our hypothesis.

Together with this we also anticipated that GBN and SR would have higher throughput when the window size gradually increased. This is explained by the fact that a bigger window size enables the client to transmit more packets all at once. This was also in line with what we expected. The efficiency of the data transfer increased as the window size expanded.

In conclusion, this test case showed that Stop-and-wait (SAW) had the lowest throughput due to its "one packet at a time" sending approach. In contrast, GBN and SR had outstandingly higher throughput on account of their sliding window approach, but they were relatively equivalent, with no significant difference between the two. The result also confirmed that increasing RTT values decreases throughput, while bigger window values lead to higher data transfer.

5.4 Test case 2: Evaluating efficacy of the modes for handling retransmission and out of order delivery.

In test case 2, we want to assess how well the three approaches handle retransmission and out of order delivery. As mentioned, we have implemented two test cases – one to skip an ack so that we trigger retransmission, and another test to skip a sequence number to demonstrate out-of-order delivery effect. This allows us to show our solutions ability to handle scenarios that may arise during data transmission.

SAW skip ACK:

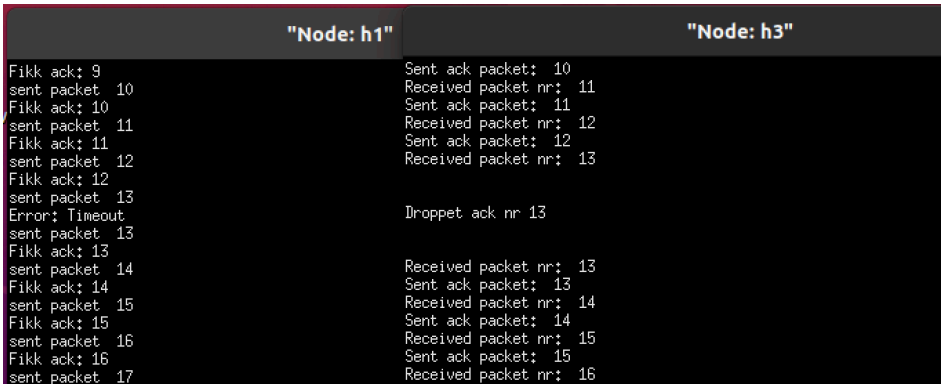

Figure 2. Stop-and-wait with skip ACK.

Figure 2 shows how our code handles drop ACK test case. In this circumstance, the sender constantly sends data packets until it reaches packet 13. We intentionally drop ACK message number 13 from the receiver side causing ACK message loss. Before sending the next packet 14, the sender needs to receive ACK message 13 before the time runs out. Without a doubt, the sender resends the packet 13 to the receiver because of timeout.

GBN Skip ACK:

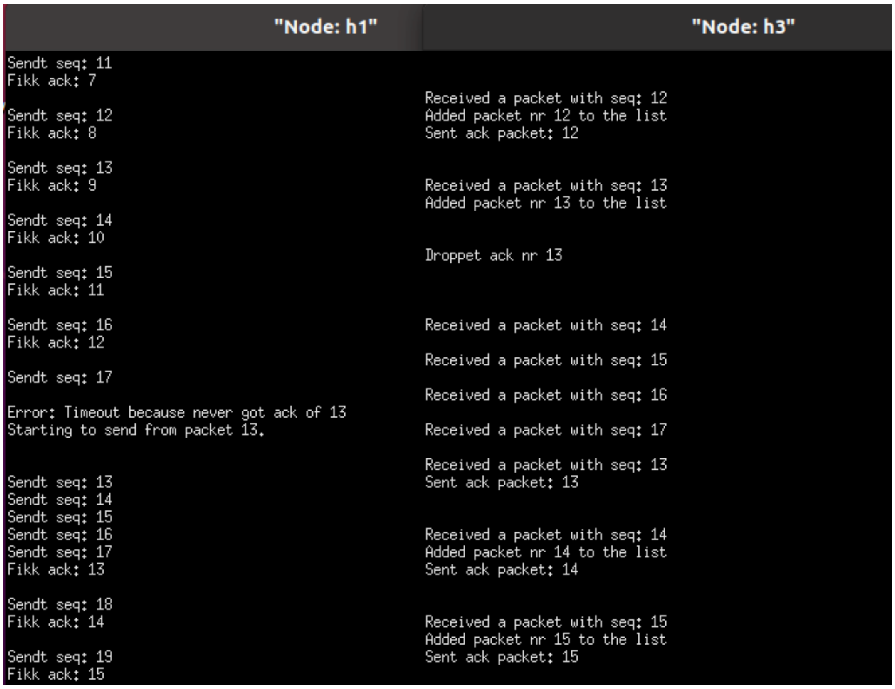

Figure 3. Go-Back-N skip ACK.

GBN loss:

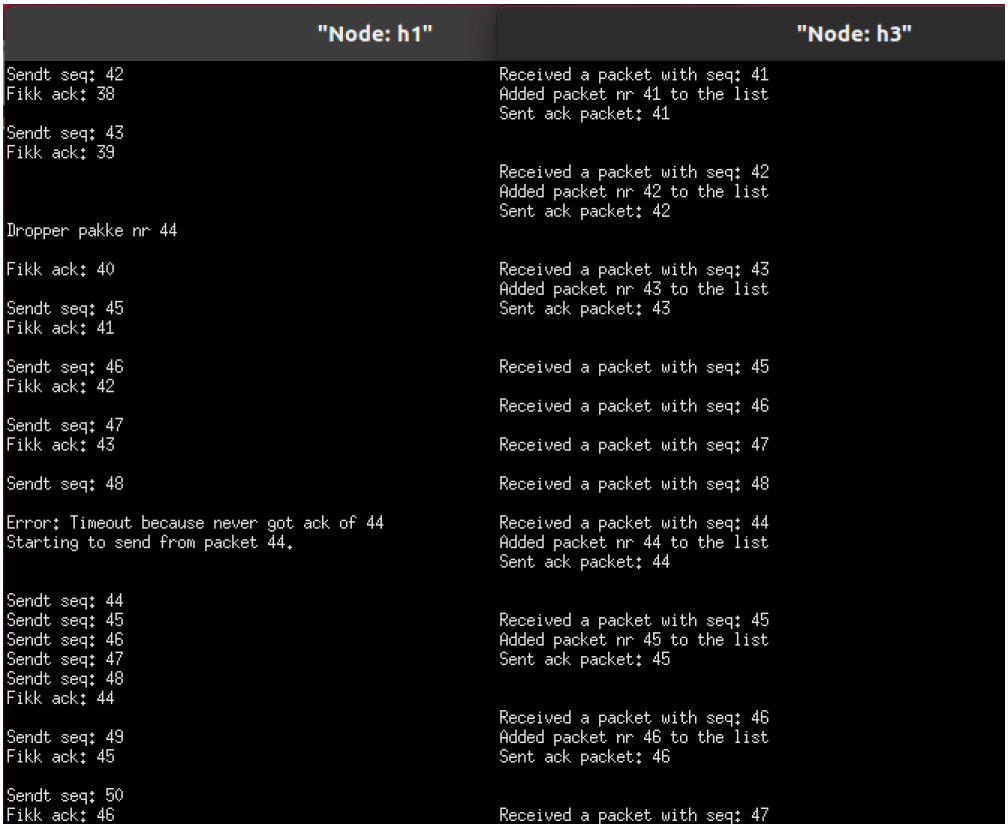

Figure 4. Go-Back-N packet loss.

In figure 3, we simulate the ACK message skipping by dropping ACK message 13. Before timeout, we can observe that the sender has already sent packets 13, 14, 15, 16 and 17. Consequently, packet with sequence number 13 has been added to the list, however, the sender has never gotten the ACK message 13. For that reason, the sender retransmits all packets 13, 14, 15, 16 and 17 since all the packets are not previously acknowledged. Similarly, in figure 4 (packet loss), the packet 44 is dropped causing packet 45, 46, 47, 48 to become discarded even though the receiver has received them. Then, after timeout, the protocol triggers retransmission asking the sender to retransmit the packets 44, 45, 46, 47 and 48.

SR SKIP ACK:

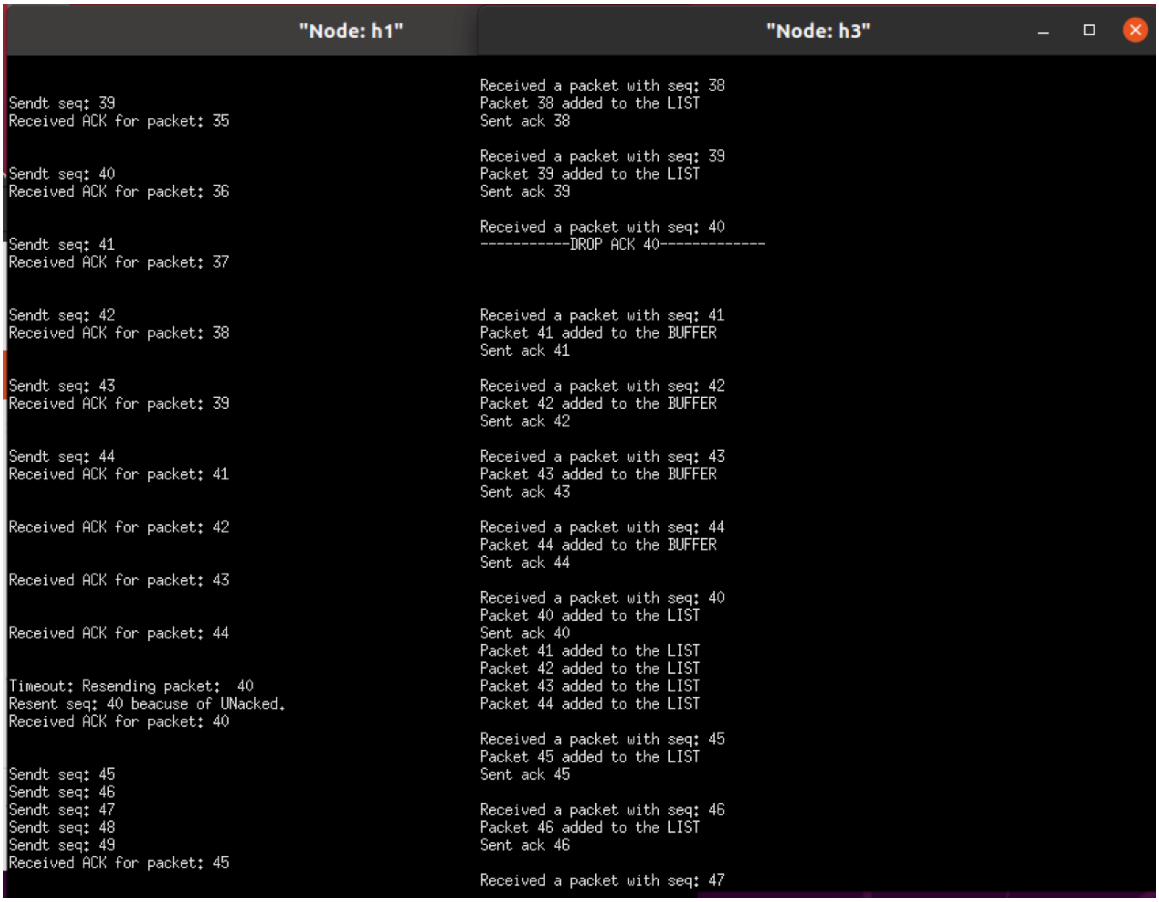

Figure 5. Selective Repeat with Skip ACK

SR loss:

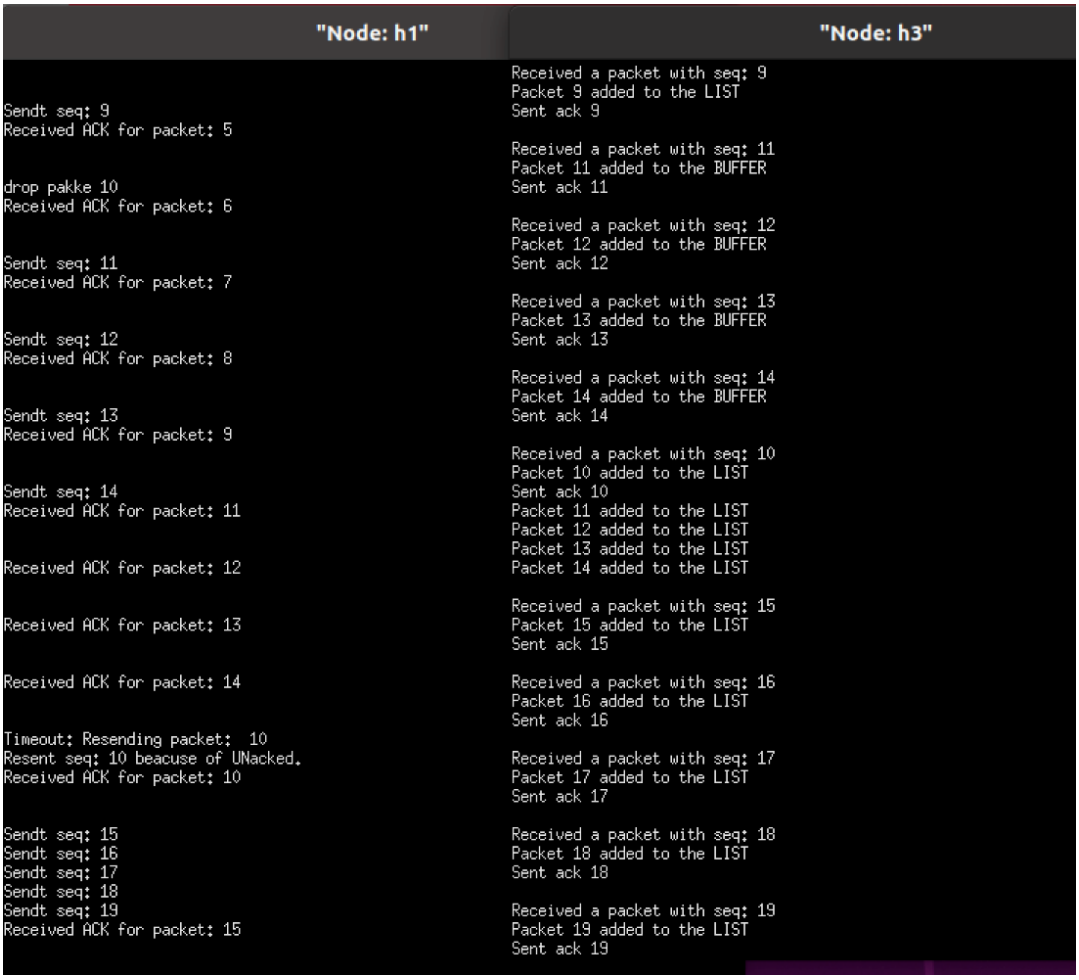

Figure 6. Selective Repeat with packet loss

In figure 5, by skipping ACK message 40, we can easily observe that packets 41, 42, 43 and 44 in this sliding window are still added to the receiver's buffer rather than discarding them. After adding the packets in buffer, the receiver sends the corresponding ACK messages to the sender to confirm packets arrival. Then, the sender will acknowledge them and get ready for sending the next packets. However, these packets are not yet added to the list because the sender is still waiting for the right ACK message to arrive in the correct order. After timeout, the sender resends packet 40 and patiently waits for ACK message 40. When the sender receives ACK message 40, the receiver adds all packets in this window to the list.

Likewise, in figure 6, packet 10 is dropped causing packets 11, 12, 13, and 14 to be added to the receiver's buffer. The sender will acknowledge these packets and confirm their arrival. When the time is running out and the sender has not gotten ACK message 10, it resends packet 10. All packets in this sliding window will be added to the list after the receiver obtains packet 10 and the sender receives the ACK message 10.

	-t flag			Bonus netem		
	Skip Ack	Loss	Skip Ack + Loss	Packet loss	Reordering	Duplicate
SAW	364.77	435.11	361.76	89.77	445.35	264.97
GBN	1055.30	1054.88	689.05	74.98	247.89	2009.96
SR	1032.02	1005.29	655.51	169.19	2072.12	2003.32

5.4.1. Testcases with -t flag

Also in this test case, we expect that SAW should have the lowest throughput in contrast to the other two modes. This is still explained by the "one packet at a time" concept. Since no new packet will be sent until the client receives an acknowledgement for the lost/delayed packet, this will have lower throughput then GBN and SR that handles several packets at once.

Although SR and GBN are faster, the loss and skip tests in this case have reduced their throughputs to about half as much compared to the throughput from test case 1. SAW on the other hand, is not affected in the same way. There are several factors to why the throughputs from SR and GBN is affected this severely and not SAW. Firstly, both SR and GBN is experiencing a timeout in both scenarios: loss and skip acknowledgment, while SAW only experience it while having a skipped acknowledgement. Since SAW does not experience timeout after packet loss due to it being handled with a duplicate ack, the throughput is barely affected. Secondly, SR and GBN are incredibly fast when not facing any problems, but SAW is noticeable slower. Therefore a 0.5 second timeout is a lot more noticeable and has a greater impact on SR and GBN, which in this case doubles their time, decreasing their throughput to half. While a timeout for 0.5 seconds only makes a slow SAW a little bit slower and is less noticeable on the throughput.

Also, SR in theory should be more effective and faster than GBN, when it comes to handling out-of-order effects. This is because the receiver in the protocol can selectively request the transmission of only the lost packets, rather than the entire window of packets as in GBN. Therefore, SR protocol reduces the number of retransmitted packets and increases the efficiency of network performance. But as our test cases only simulate low loss rates, it is expected that GBN exceeds SR. This can be explained by the fact that SR generally contains complex code making it more inefficient than GBN in low loss rates scenarios. Contrarily, GBN tends to waste bandwidth, which makes SR code more effective in cases of high loss rates. After conducting the test cases, the results illustrate that throughputs of GBN in both test cases are slightly higher than SR's throughputs.

5.4.2 Testcases with Netem (Bonus)

To be able to show the efficacy of the protocols, the network emulation functionality Netem will be used to simulate high-error rates. Netem introduces various network impairments and conditions to simulate different network scenarios. Scenarios for testing the efficacy are packet loss, reordering and packet duplication. In each scenario, the emulation was set to 10%. The next sections will deal with each protocol in the three different scenarios.

Packet loss in Netem

All three protocols give highly inconsistent outcomes, and their performance reaches its lowest point when they confront with 10% packet loss. Stop And Wait (SAW) reaches a throughput of 90 Mbps caused by the need to retransmit a significant number of packets and encountering multiple timeouts and duplicate ACKs.

Despite SAW having a massive decrease in throughput, Go-Back-N (GBN) still falls below the performance level of SAW when encountering packet loss. A direct explanation for this is that GBN is compelled to wait for timeouts like SAW, but on top of that, it must retransmit all the packets within its window. As a consequence, this leads to an even lower throughput due to the additional transmission of packets.

In scenarios with high packet loss rates, the benefits of Selective Repeat (SR) become evident. Despite encountering packet loss which leads to a severe reduced throughput, this protocol still delivers twice as high throughput as SAW and GBN. The occurrence of timeouts, like the other protocols, is the primary reason for the significantly lower throughput in this protocol. What sets it apart from the other two protocols and allows for doubling the speed is its capability to buffer packets and store acknowledgments when a packet is lost. This does not affect any of the packets within the window since only the lost packet will undergo retransmission.

Reordering

Reordering has no impact whatsoever on Stop-and-wait. In fact, it is impossible to experience reordering with SAW because the client always waits for an acknowledgment from the previous packet before sending the next. Because it waits for an acknowledgment, there cannot be any out-of-order delivery, and the protocol works as it normally would.

On the other hand, reordering significantly impacts GBN. When the receiver receives out-of-order packets, it discards all the other packets when waiting for the right one. This will trigger retransmission of the discarded packets. In addition, any acknowledgments that are out-of-order will initiate retransmission of the packets within the window. These factors significantly decrease the throughput of GBN when facing reordering.

Like SAW, Reordering hardly affects Selective Repeat (SR). As mentioned, the protocol's capability to buffer packets and store acknowledgments enables it to handle out-of-order packet arrival. It simply waits for the correct packet and then arranges them in the correct order. This eliminates unnecessary transmission of the other packets, allowing the transmission to proceed almost normally.

Duplicate

SAW encounters certain challenges when duplicates are introduced, which revolves around receiving duplicate ACKs. When sending a duplicate packet, it will similarly be treated as a new packet at the receiver, which will send an acknowledgment back. This causes the client to send a packet twice, causing unnecessary wait for a packet that’s already sent and acknowledged. Consequently, for each duplicate sent, the throughput will be reduced.

In contrast to SAW, duplicates have less impact on GBN and SR. These two protocols have got the same approach in this scenario. When the receiver receives a duplicate packet, the packet is considered as a new packet and an acknowledgment is sent. As a result, the sender only needs to resend that specific duplicate packet, without affecting the window or triggering retransmission of packets within the window. Therefore, the only factor that reduces the throughput is the additional time it takes to transmit the duplicate packet. Since both protocols handle this exceptional case in the same manner, they transmit all packets in a similar fashion, resulting in equal throughput for both protocols.

5.5 Bonus: calculate the per-packet roundtrip time and set the timeout to 4RTTs

5.5.1 Results

	Loss		Skipack	
	Bonus	Without	Bonus	Without
SAW	435.45	435.11	418.30	364.77
GBN	1747.46	1054.88	1731.04	1055.30
SR	1646.08	1005.29	1644.71	1032.02

5.5.2 Discussion

In the application, the default socket timeout is set to 0.5 seconds. The purpose of timeout is to make sure the packet is lost and not just slow. As a result, timeout becomes a necessity. But a timeout set for 0.5 seconds may not be necessary. In general, the needed timeout is no more than four times the RTT. This gives the packet more than enough time to be received, buffered, handled and for an ack to be returned. In cases where the RTT is low, optimized timeout will be severely lower than the default RTT of 0.5 seconds. Depending on the RTT, an optimized timeout could save more time, and especially in cases of high-error rates, it could save a few seconds.

The network tool ping is needed in this calculation because it makes it possible to calculate the round-trip time (RTT) between the hosts. This RTT is then multiplied with 4, giving the transmitted package enough time, unless it is lost during the transfer. To show the effect from the optimized timeout, the artificial test cases are used to simulate packet loss and skip acknowledgment which will force a timeout.

Packet loss in Stop-and-Wait commonly does not trigger a timeout, but it will trigger when an acknowledgment is skipped. Even with just one skipped ACK, the throughput has increased by 20%, which is a lot since SAW is generally considered slow and is not affected by time as much as GBN and SR are.

As we have mentioned earlier, Stop-and-wait only experiences timeout when an acknowledgment is skipped. When using the new timeout, the throughput is increased by 20% just by skipping one ACK message. That is a lot taking into consideration SAW is slower making the throughput less vulnerable when experiencing small time differences.

Using the optimized timeout with GBN and SR has a huge impact on their throughputs. Both increase their throughput by over 50%, leaving a massive impact after just one packet loss or skipped acknowledgment. If several timeouts occur, a lot of unnecessary waiting will be avoided. This will save plenty of time under high-error rates for all protocols.

6 Conclusions

In conclusion, Stop-and-Wait, Selective Repeat (SR), and Go-Back-N (GBN) are the three widely used protocols for reliable data transfer in computer networks.

SAW is a simple protocol where the sender waits for an acknowledgement from the receiver before sending the next packets. This protocol works fine, but it is unfortunately ineffective in transferring a huge number of data due to the wait time for acknowledgement.

SR and GBN are more advanced protocols where sender can send multiple data packets before receiving acknowledgements. In SR, the receiver keeps track of the received packets and only retransmits the lost packets while GBN only accepts packets in subsequential order and retransmits all unacknowledged packets in the sliding window. Thanks to the sliding window concept, GBN and SR are more effective in transferring a huge amount of data over time.

In summary, the choice of protocols depends on specific requirements, network conditions and different purposes. Each network protocol has its own advantages and disadvantages, so choosing the appropriate reliability is extremely important to effectively optimize the network quality and its performance.

7. References

Ashtari, H. (2022). *What Is User Datagram Protocol (UDP)? Definition, Working, Applications, and Best Practices for 2022*. Retrieved from Spiceworks: https://www.spiceworks.com/tech/networking/articles/user-datagram-protocol-udp/#:~:text=User%20datagram%20protocol%20(UDP)%20is%20used%20for%20time%2Dcritical,online%20gaming%2C%20and%20video%20streaming.

Kurose, J. F., & Ross, W. K. (2020). *Computer Networking, 8th edition.* Amherst: Pearson.

Kurose, J., & Ross, K. (2021, 8.edition). *Computer Networking: A Top-Down Approach, Global Edition.* Pearson Education Limited.

Serina Erzengin, Own work (2023) *simpleperf-portfolio 1* [upublisert semesteroppgave]. Oslo Met