Universitatea Tehnică "Gheorghe Asachi" din Iași
Facultatea de Automatică și Calculatoare
Rețele de Calculatoare - Proiect

File transfer - Sliding window protocol

Echipa: 6
Beldiman Vladislav
Grupa 1305A
Hârțan Mihai-Silviu
Grupa 1305B
Timofti Gabriel
Grupa 1305B

December 14, 2020

Contents

1 Introduction

Sliding window protocols are used for reliable in-order delivery of packets is required, such as in the Transmission Control Protocol (TCP). They are also used to improve efficiency when the channel may include high latency. Our application goal is to reliably transfer a file over UDP using the Go-Back-N algorithm.

2 Algorithms

2.1 Sliding window protocol

The Sliding Window Algorithm

According to [1] the sliding window algorithm works as follows:

First, the sender assigns a sequence number, denoted **SeqNum**, to each frame. The sender maintains three variables: The send window size, denoted **SWS**, gives the upperbound on the number of outstanding (unacknowledged) frames that the sender can transmit; **LAR** denotes the sequence number of the last acknowledgment received; and **LFS** denotes the sequence number of the last frame sent. The sender also maintains the following invariant: LFS - LAR $\leq$ SWS.

When an acknowledgment arrives, the sender moves LAR to the right, thereby allowing the sender to transmit another frame. Also, the sender associates a timer with each frame it transmits, and it retransmits the frame should the timer expire before an ACK is received.

The receiver maintains the following three variables: The receive window size, denoted **RWS**, gives the upper bound on the number of out-of-order frames that the receiver is willing to accept; **LAF** denotes the sequence number of the largest acceptable frame; and **LFR** denotes the sequence number of the last frame received. The receiver also maintains the following invariant: LAF - LFR $\leq$ RWS.

When a frame with sequence number SeqNum arrives, the receiver takes the following action. If SeqNum $\leq$ LFR or SeqNum > LAF, then the frame is outside the receiver's window and it is discarded. If LFR < SeqNum $\leq$ LAF, then the frame is within the receiver's window and it is accepted. Now the receiver needs to decide whether or not to send an ACK. Let **SeqNumToAck** denote the largest sequence number not yet acknowledged, such that all frames with sequence numbers less than or equal to SeqNumToAck have been received. The receiver acknowledges the receipt of SeqNumToAck, even if higher numbered packets have

been received. This acknowledgment is said to be cumulative. It then sets LFR = SeqNumToAck and adjusts LAF = LFR + RWS.

The same notations as in [1] will be used in further sections. In addition, ***MaxSeqNum*** *will denote the number of available sequence numbers, and* ***NextSeqNum*** *will track the next packet to send.*

If the sender receives a duplicate ACK message, an untreated case in [1], it simply ignores the message.

2.2 Go-Back-N

The Go-Back-N implementation of the sliding window protocol uses a SWS > 1, but has a fixed RWS = 1, thus the receiver refuses to accept any other packet but the next one in sequence. As RWS = 1, the sender only needs one timer for the entire window, and, when the timer expires it will resend the entier window. Furthermore, RWS = 1 means that MaxSeqNum $\geq$ SWS + 1 is sufficient. [1]

If a packet is lost in transit or arrives but is corrupted, all following packets are discarded until the missing packet is retransmitted which implies a minimum delay of a round-trip time and a timer timeout. Consequently, it is not efficient on connections that suffer frequent packet loss and/or from noise.

In the case that the receiver is sent a duplicate packet, it sends an ACK message of SeqNumToAck - 1.

2.3 Block diagrams

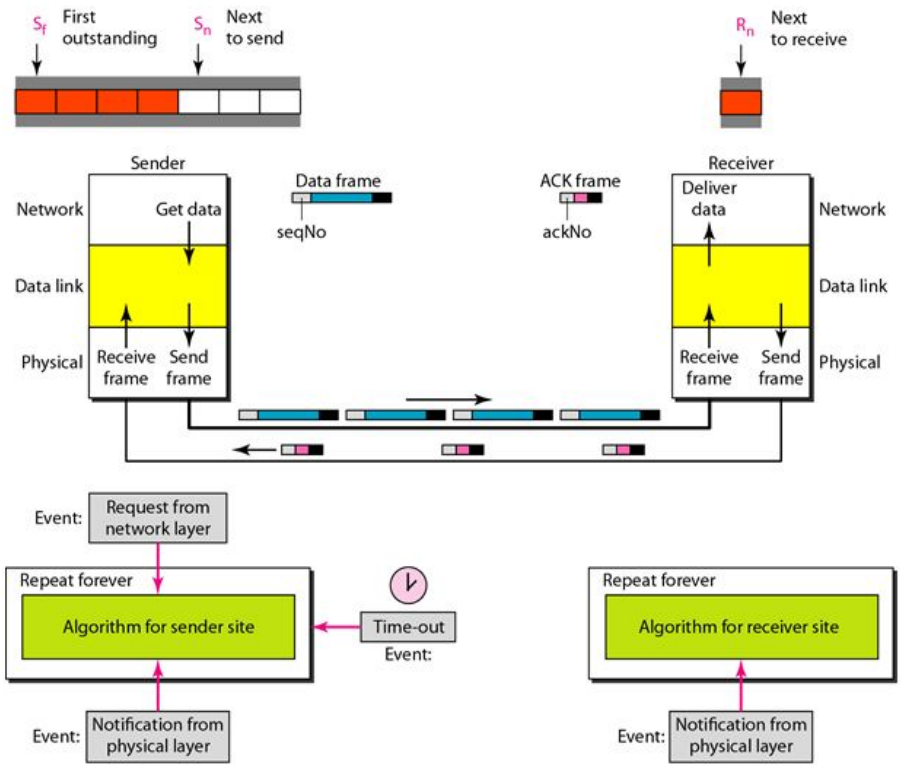

Figure 1: Block diagram of Go-Back-N algorithm. [Source]

3 Classes

3.1 Sender

The Sender (Figure 2) begins by creating and binding a socket to ('0.0.0.0', SENDER_PORT) in order to listen on any ip, but on one specific port built-in the application and waits for a receiver to send a request message a certain amount of times. Once the first request message is received, it sets the destination address in its Udp member object so that it's receive function returns data only from that address only, and the send-related methods send only to said address.

After that, it tries to send a "handshake" message in which it transmits parameters selected in the sender instance that are useful for the receiver and waits for an acknowledgement message with the sequence number set to -1 a set amount of times.

Next, it then reads the frames from the chosen files into memory. Then, Go-Back-N is used to send all of the frames as such: it sends the entire window, then listens for acknowledgements. If an acknowledgement that would slide the window is received, the timer is restarted, the window is moved, the rest of the window is sent, and then it goes back to listening. Otherwise, if there is a timeout - it resets the timer, resends the window, and then it starts listening again. After a certain amount of timeouts for the same window it gives up with an error.

After the last acknowledgement is received it starts sending finish messages and listening for a finish response a certain amount of times. If the expected answer is received, the socket is closed and the sender ends errorlessly. Otherwise, an error message is printed as it could not be confirmed that the receiver was closed.

3.2 Receiver

The Receiver (Figure 2) begins by creating a socket and sending a request to the user-specified ip on the built-in port and updates its Udp member object with the socket ip and port generated by the first send. Next, it continues to send requests and listening for handshake messages a certain amount of times. If a handshake message is received, it starts sending ack messages with the sequence number -1 and listening for the first data packet a set amount of times.

After the first data packet is received, it proceeds with the Go-Back-N algorithm in order to receive the file. It listens for data packets one is received, a finish message is received or its timer gives a timeout. If the first event happens, it restarts the timer, sends the corresponding acknowledgement and writes the data to memory. It gives up with an error in case of the latter.

If a finish message is received, it writes the data from memory to the user-chosen folder. Next, it sends a certain number of finish messages and shuts down errorlessly.

3.3 Udp

The Udp class abstracts packet sending and receiving, along with all abstractions provided by the PacketHandler member. It does so using AF_INET family,

non-blocking sockets of DGRAM type.

3.4 Packet Handler

This class hides packet structure, packet conversion from data to bytes and vice versa, and packet integrity checks. The packet structure is as described in figure 3.

3.5 Timer

A simple class used by both the Sender and Receiver to keep track of the window timeout and the maximum time between the receival of two legitimate frames, respectively.

3.6 Logger

This class is used for logging. It will use the signal for logging if it is passed one or print to console otherwise.

3.7 Enums

Three enums are used.

One for packet types, with their corresponding: DATA = 1, ACKnowledgement = 2, REQuest = 3, PaRaMeTers = 4 and FINish = 5.

The next for log types: Start/End of Transmission, SeNT, ReCeiVed, ERRor, WaRNing, INFormative and OTHer.

And, finally, one for finish types: NORMAL, FORCED, and ERROR.

4 Other specifications

4.1 Flow

A typical theoretical flow is presented in figure 2, and a real example (over the same network, with a window size of 2, 2 different sized packets to send, a low corruption chance and very low loss chance) in figure 3 along with the corresponding logs (Sender 4 and Receiver 5). If an error occurs at either end, or it does not connect, it will close the connection. The other end will repeat a certain

amount of times the operation it is currently trying to complete before giving up with an error and shutting down, or display an error message and shutdown immediately if the connection is made over the loopback address.

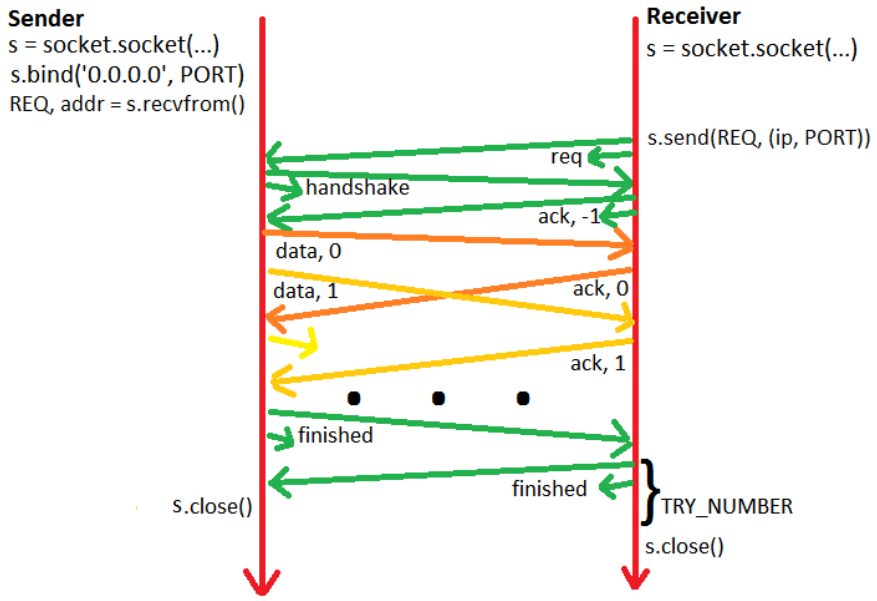

Figure 2: Typical Theoretical Sender-Receiver Flow.

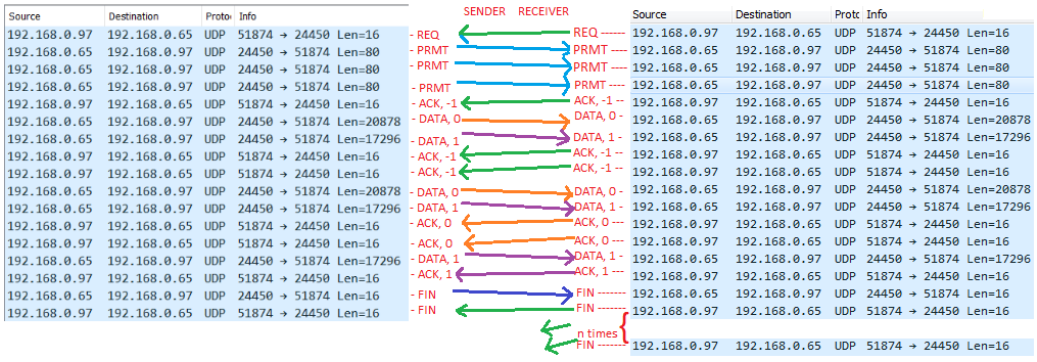

Figure 3: Typical Practical Sender-Receiver Flow.

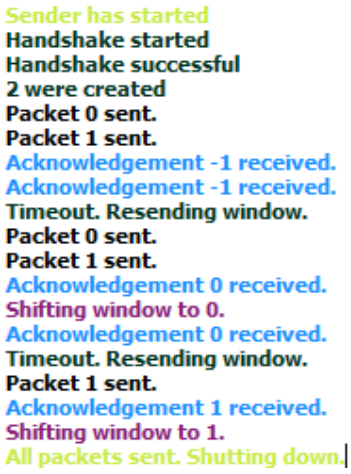

Figure 4: Typical Practical Sender Logs.

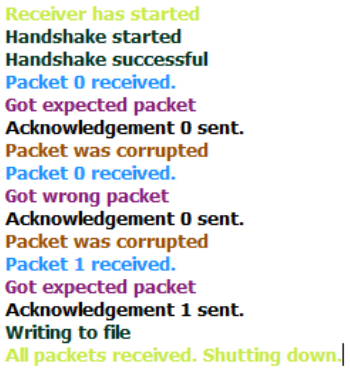

Figure 5: Typical Practical Receiver Logs.

4.2 Sender-Receiver Launch Order

Do note that it does not matter which of the Receiver or Sender is launched first as long as the other delivers the first message successfully before the timeout.

4.3 Cross-network transmissions

When same device process or same network communication (Figure 6) is needed - simply introducing the loopback address (127.0.0.1) or the address of the sender, respectively, should be enough for successful transmission, unless a firewall prevents that.

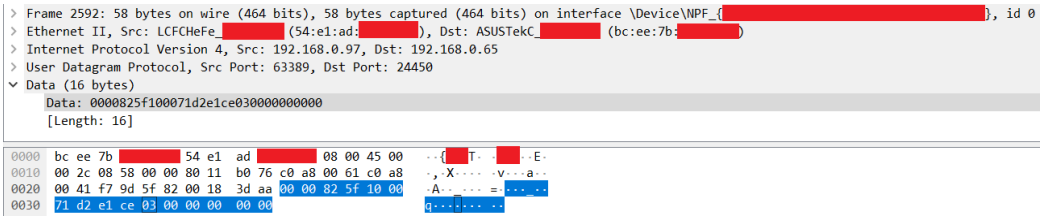

Figure 6: Same network sent packet example.

If the Sender and Receiver are on different networks and we have access to all of the switches and routers that connect them, appropriate switching and routing

should be configured, such that the Receiver is able to find the device with the specified ip.

In the case that the traffic has to go through your ISP first, the Receiver has to use the public ip of the Sender (Figure 7 and 8), then, if the isp-server connection is not direct (but through a router) there must be a port forwarding rule set on the router with the interface with the public ip address that forwards traffic on the SENDER_PORT = 24450 to the device running the Sender (Figure 9).

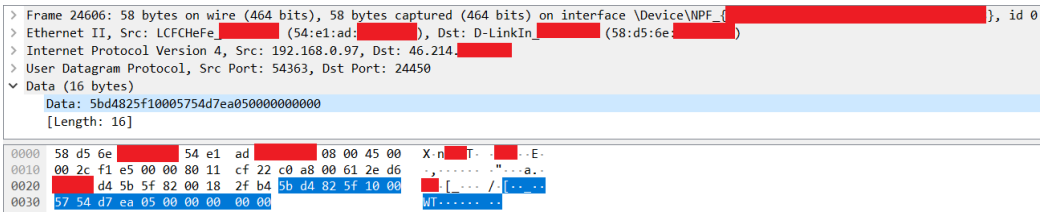

Figure 7: Public ip sent packet example.

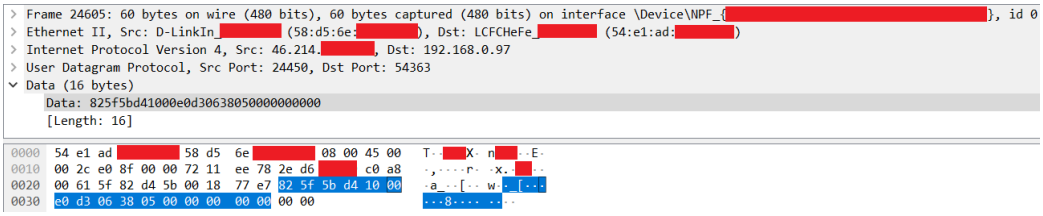

Figure 8: Public ip received packet example.

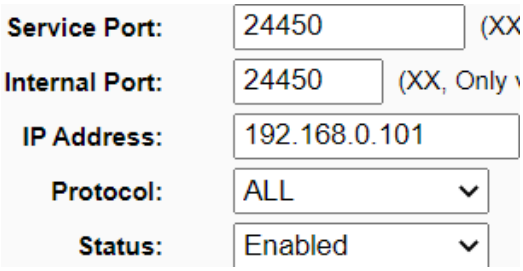

Figure 9: Port forwarding configuration example.

4.4 Packet Structure

The structure of the packet that is sent through the sockets is shown in figure 10.

DATA packets have at least one byte inside the data field. For ACK, REQ and FIN packets, that field is empty. And PRMT packets containt data with the structure displayed in figure 11.

DATA and ACK packets have the sequence number field set according to the sliding window protocol. All other packets have it set to 0.

An ACK packet with the sequence number to -1 means both that no frame has been received yet (as the first sequence number is 0) and a confimation for PRMT receival (which has the same meaning as the former).

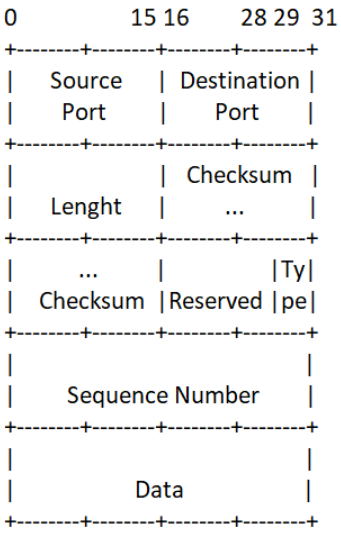

Figure 10: Packet Structure (Values are in bits).

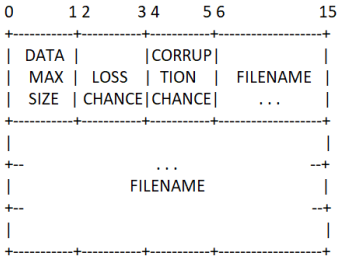

Figure 11: Parameters Data (Values are in Bytes.

4.5 Packet loss and corruption

As during our tests the networks proved to be quite reliable, we introoduced an artificial packet loss and corruption. Both can be configured separately, even eliminated by the sender at the options page.

A packet may be arificially lost in send-related Udp functions by not sending it and reporting the loss via the logger.

A packet may be arificially corrupted in the PacketHandler get_bytes function by changing the values of a random bit.

4.6 Dependencies

This project depends on the following standard library modules: socket, random, zlib, time, enum, sys, os and threading.

And on these non standard modules:

- PyQt5 (pip install PyQt5)
- PyDispatcher (pip install PyDispatcher)

5 Git repository

https://github.com/gabitim/RC_Proiect

6 Reference

[1] Peterson L. L. and Davie B. S. Computer Networks a systems approach. (Fifth edition)