Ecole Polytechnique de Louvain

INGI2141

Report on projet 1 : Reliable transfer protocol

Author:
Groupe XX
Henri Crombé
Mallory Declercq

Supervisor:
Oliver Bonaventure
David Lebrun
Fabien Duchêne

October 31, 2014

1 Introduction

For this first project it was asked to implement in C a reliable transmission protocol based on selective repeat on top of UDP.

In order to do that we created two programs able to reliably transmit a file of any length through the network. One program is the sender of the file. The other one is the receiver. A file is transmited from the sender to the receiver by sending frames of fixed size carrying payload of the file.

2 Explaination of the implementation

2.1 General functioning

As said previously, a file is transmited through the network by sending frames carrying small part of it. So that the receiver reliably receives the entire file, the sender and the receiver has to rely on the selective repeat protocol in oder to deal correctly with packet loss and packet corruption. In addition, frames also use CRC to verify their integrety during their transit.The functioning of the communication between the sender and the receiver can be so summarized :

First, the receiver has to be launched and be waiting for incoming frames. Then the sender can start its job. It begins by opening the file to send and by configurating the network socket. When descriptors are ready, the exchange of datagrams can start.

2.2 Sender

The sender works as follow:

If there is a filename specified in the command line when we launch the program then it opens the related file. Otherwise we need to write directly on the standard input (stdin) the data we want to send to the receiver.

In order to interact with the receiver, the sender creates its own socket. Once the sender socket is ready we can start to send datagrams to the receiver (each datagram contains a portion of the file or data entered by the

user). To deal with the packet exchange between the sender and the receiver, we use three arrays.

The first array is called *Window_status*. This array is an array of 256 integer's slots where the index numbers correspond to the sequence numbers of the frame. It contains either FREE, ACK_WAIT or ACK_WAIT_LAST values. The FREE value means that there are no constrains on the frame. The ACK_WAIT value means that the frame was sent to the receiver but it was not yet acknowledged. The ACK_WAIT_LAST value is pretty the same as the ACK_WAIT value but we use it to know when the last frame has been acknowledged. At start, we initialize each slot of this array to FREE.

The second array is called *window_frames*. This array is an array of 256 pointers associated to a frame. It allows us to store a frame when we need to resent it. At start, we initialize each slot of this array to null.

The last array is called *window_timeout_list*. This array is an array of 256 timeval structure which allows us to manage the timeout of a frame.

Now, we are going to see how the sender works with those main structures to accomplish its job. Each time a portion of the file has been read by the sender, it creates a frame containing this portion of the file and sends it through a socket to the server. In order to know that the frame is send to the sender and wait for an acknowledgment, we put the ACK_WAIT value (or ACK_WAIT_LAST if it's the last frame to send) into the *window_status* array and we put the timeout associated to this frame into the *window_timeout_list array*. When a timeout occurs, we resend the last frame send but not acknowledged.when an acknowledgement is received, it gives us two informations. The first one is the next expected sequence number of the frame to be send and the second is the maximum sequence number that the sender can send to the receiver (expected_seq + window_size).With those informations, it's easy for the sender to know which frame to send.

2.3 Receiver

The receiver is implemented as follow:

First, it checks if there is a filename associated to the command line. If

not, it will display every packets coming from the sender to the standard output. If a filename is entered after the -file option then the receiver creates (or rewrites) the corresponding file.

Then, we create a socket in order to allow the receiver to listen on a given port. When it's done, the receiver waits until a client (another socket) wants to send a file to it. To handle the interaction between the client and the server, the receiver creates two arrays. Those two arrays use the sequence number of the frames as index. Here are the goals of each of those arrays:

The first array (*window_frames*) is an array of 256 pointers associated to a frame. It allows us to store a frame when needed. At start, we initialize each slot of the array to NULL.

The second array (*window_status*) is an array of 256 integer's slots. We use this array to know if a given frame is either STORED (defined as 1) or FREE (defined as 0) into the *window_frames* array. At start, we initialize each slot of the array to FREE. When a frame arrives to the receiver with a correct CRC, if its sequence number is the one expected by the receiver then we store the frame into the file and we update the *window_status* array as follow: *window_status* `[sequenceOfCurrentFrame]` = FREE.

Then we check if the next frames (*window_status*`[sequenceOfCurrentFrame+1]`) are not already stored into the *window_frames* array because the current frame can be a resent frame so it's possible that the next frames following this one are already stored inside the receiver. If it's the case then those frame are already stored inside the *window_status* array with a value eguals to STORED at the index corresponding to their sequence number. We store those frames into the file and we update the *window_status* array (*window_status*`[sequenceOfCurrentFrame+1]`=FREE).We repeat this operation until there are no more in-sequence frame following the resent frame.

When a frame arrives to the receiver and its sequence number is not the expected one (bigger than the expected sequence number and included in the range of the receiver window) then we store this frame into the *window_frames* array and we put a STORED value inside the *window_status* array at the index corresponding to the sequence number of this frame.

When a frame arrives to the receiver and its sequence number is not the

expected one and its not include among the range of the receiver window then the frame is discarded.

2.4 Interportability test