Data Link Protocol Project

Alexandru Răducanu

Emilia-Bianca Șerban

Summary

1. Introduction

The main objective of this report is to describe and detail the implementation of the Data Link Protocol that we have created. From now on, the report will be divided into multiple sections, each one presenting a different part of our project. As such, these sections will be:

* Structure - here you may find a description of the functional blocks, file structure and the interfaces of our project.
* Code structure - a detailed description of the C code used in the project files, including data structures used, APIs and functions.
* Main use cases – an explanation about how the C code works in practice, function calls sequences and dealing with exceptions or errors.
* Logical Link Protocol - functional aspects of the Link Layer Protocol, descriptions of the main functional characteristics of this layer of the project.
* Application Layer Protocol - functional aspects of the Application Layer Protocol, descriptions of the main functional characteristics of this layer of the project.
* Validation, Testing and Efficiency - description of the test performed by the Protocol;
* Conclusions – personal achievements.

1. Structure

2.1 The project folder

Text

Description automatically generatedThe project directory has the structure presented in the image to the left. Here, we can see all the files and subfolders included. In the next lines, you can find a description of each file and subfolder.

* “penguin.gif”. This is the image we want to transfer from the transmitter to the receiver. This image can be changed with any other file the user wishes to transfer using our protocol.
* “README.txt”. This file contains instructions on how to use the “Makefile” and how to test the protocol on the user’s laptop. To be able to test the protocol, the user must have a Linux Operating System.
* “main.c”. This is the “entry” point in our project code. The binary of this file can be seen in the “bin” subdirectory, alongside the binary of the “cable.c” file.
* “application\_layer.c”, “link\_layer.c” and the respective header files contain the C code developed by our team. The protocol is divided into 2 layers, one that deals with the file that needs to be transmitted (application layer) and another that deals with the proper transmission of the file through the cable (link layer);

2.2 User interface

The project does not include an user interface. The “README.txt” file contains all the necessary steps the user must follow to use the protocol.

The protocol also shows the state of the transmission after each frame is transmitted/ received. Information on the transmission state contains: the number of bytes transmitted/received, the position of the cursor in the transmitted/received file, successful/ unsuccessful receiving or sending of the frame. Also, at the end of the process, the user is informed if the connection was successfully ended.

1. Code structure

The project was developed in the C programming language. This language was chosen because it provides us with several useful libraries that deal with the transmission through a serial port. The libraries used by us are: <termios.h>, <unistd.h>, <signal.h> , <stdio.h> , <stdlib.h> , <sys/stat.h> , <sys/types.h> , <string.h> and <fcntl.h>.

Regarding data structures, we used a “struct” named LinkLayer that represents the entity that is using the code (Transmitter or Receiver). This structure contains fields for all the necessary information that relates to the transmission parameters and the transferred file (baud rate, timeout value in seconds, number of retransmission, transferred file name). The transmission parameters for the serial port are set using a “termios” structure. These are the only data structure used in our project, all other information being manipulated as unsigned char strings or variables.

As explained before our project is divided into 2 main parts, the Link Layer Protocol, and the Application Protocol. In this section, all the functions used in the project will be listed, along with the detailed explanation of each one of them. The function calling sequence in practice will be detailed in the next section of this report. We mention that all the functions can be found in the C files and header files: “application\_layer.c” and “application\_layer.h” for the Application Layer Protocol functions, and “link\_layer.c” and “link\_layer.h” for the Link Layer Protocol functions.

* 1. Functions used in the Application Layer Protocol

1. **void applicationLayer (const char \*serialPort, const char \*role, int baudRate, int nTries, int timeout, const char \*filename)**

This is the main function of the application\_layer.c file. It is called in the “main.c” and receives all the necessary information regarding the transmission parameters and transferred file name. It creates the LinkLayer structure and populates its fields with the given parameters. **llopen(), recvFile(), sendFile()** and **llclose()** functions are called from this one, so the user can see this function as a bridge between the Application Layer and Data Link Layer.

1. **int recvFile(int connection\_fd)**

Function used by the receiver, this is the application layer function that deals with creating a new file and constantly updating it with the new frames received from the link layer. The function receives the connection file descriptor in order to be able to call the **llread()** function from the Link Layer. The function returns -1 in case of error, or 1 in case of success.

1. **int sendFile(int connection\_fd, const char \*pathname, LinkLayer link\_struct)**

Function used by the transmitter, this is the application layer function that deals with opening the desired file and sending bytes of its content to the **llwrite()** function. It returns -1 in case of error, or 1 in case of success

1. **int data\_packet(FILE \*file\_fd, unsigned char \*buf, unsigned char N)**

This function receives a buffer and a file descriptor and creates a data packet with all the additional fields required by the protocol. The packet start from buffer[0]. It returns the number of characters written in the buffer.

1. **int control\_packet(FILE \*file\_fd, unsigned char \*buf, int c\_flag, const char \*pathname)**

As explained above, this function receives a buffer and creates a control packet from the buffer, starting from buffer[0]. The control packet can be of 2 types, depending on the c\_flag variable. It returns the number of characters written in the buffer.

* 1. Functions used in the Application Layer Protocol

1. **int llopen(LinkLayer connectionParameters)**

This function receives a LinkLayer structure and sets all the necessary connection parameters to the desired values. Also, depending on the “role” field of the LinkLayer structure, it calls for the **llopen\_rx()** or **llopen\_tx()** functions. These ones are the ones that send/receive the SET frames, so that each entity knows that the connection has been established successfully. In case of success, all 3 functions return 1. In case of error, the functions return 0 or -1.

1. **int llwrite(int connection\_fd, const unsigned char\* buf, int bufSize, LinkLayer link\_struct, int color)**

This function receives the file descriptor of the connection and its parameters, the data payload buffer and the color and creates a valid frame from those. The frame is sent using **write()** function from the C library. Also, this function waits for the acknowledgement and resends the frame in case of noisy channel. It also implements the timeout and retransmission functionality. It returns the number of bytes sent in case of success, or 0 otherwise.

1. **int llread(int connection\_fd, unsigned char\* packet, int expected\_color)**

This function receives a frame and sends back a positive or negative acknowledgement (RR or REJ). Thus, this functions make all the necessary verifications to see if the frame was altered during the transmission and, in case the frame is valid, this function destuffs it. If the frame is a valid one, the function copies only the data payload to the packet buffer. It returns the number of bytes of the data payload, in case of success. It returns 0 if there is any error.

1. **int stuffing(unsigned char\* buf, int buf\_size)**

This function receives a frame and its size and stuffs it. It returns the new size of the frame.

1. **int destuffing(unsigned char\* buf, int buf\_size)**

This function receives a valid frame and destuffs it. It returns the new size of the destuffed frame.

1. **int check\_received\_frame(unsigned char\* buf, int bufsize, int expected\_color)**

This function receives a frame and checks if it is a valid one. Based on the return value, you can identify what was the problem with the frame. It returns 1 if the frame is valid.

1. **void send\_rej(int color, int connection\_fd) and void send\_rr(int color, int connection\_fd)**

Those functions send the negative/positive acknowledgement from the receiver to the transmitter.

1. **int llclose(int fd, LinkLayer connectionParameters, int showStatistics)**

This function ends the connection between transmitter and receiver. It is similar to **llopen()**, as it calls for **llclose\_rx**() or **llclose\_tx()** based on the role of the LinkLayer structure.

1. Main use cases

In this section we will provide a detailed explanation of the functioning of the protocol, how the functions described above call other functions and how the information is transmitted frame by frame from the transmitter to the receiver.

4.1 Function calls

For both the transmitter and the receiver, the **main()** function calls the **applicationLayer()**. Here a LinkLayer structure is created. This structure contains transmission parameters and the name of the file to be transmitted. The **application\_layer()** calls the **llopen()** in order to establish a connection to the other entity. Based on the role, **llopen()** calls **llopen\_rx()**/**llopen\_tx()** and these functions start changing control frames between entinties (SET frames and UA frame). If everything goes well, the connection is established and we return back to **applicationLayer()**.

Now depending on the role of the entity, the function **recvFile()** or **sendFile()** is called. These functions use **llwrite()** and **llread()** in order to send information frames from transmitter to receiver and acknowledgements from receiver to transmitter.

After all the file content is sent to the receiver, the connection must be ended. Here, we are back in **applicationLayer()**,where we call **llclose().** Inside **llclose()**, based on the role, the function **llclose\_tx() or llclose\_rx()** is called. These functions exchange control frames in order to successfully end the connection (DISC frames and UA frame).

4.2 Sending a frame of information

Note: In the following paragraph, we will consider “the normal case” as the correct way of transmitting the file. In other words, there is not any noise on the cable, so the information is not modified.

After both the transmitter and the receiver have established their connection, the sending of the file is started by the transmitter by calling sendFile() . This function is responsible for the following things (which have this exact order of execution):

* Opening the file that needs to be sent
* Creating the information packet by reading 128 bytes of information from the file (by calling data\_packet() )
* Sending the packet to the LinkLayer for it to have its headers added and to be stuffed (all of this is happening in llwrite()).
* llwrite() sends the frame to the receiver and waits for an acknowledgement.
* On the other side, the receiver starts verifying if the frame is correct and sends back an acknowledgement.
* If the acknowledgement is positive, the transmitter continues to send the next 128 bytes of information. If not, the transmitter resends the last frame.

1. Link Layer Protocol

The most interesting functional aspects of the Link Layer protocol are:

* The timeout and retransmission functionality. This is implemented using the “alarm.h” C library, function alarm() and alarmhandler(). For the proper functionality of the alarm, we chose to make the read() function a non-blocking one in llopen() with the help of the termios structure.
* Stuffing and destuffing a frame. This functionality is done by the functions stuffing() and destuffing(). We use this approach to see where a frame begins and ends. The characters stuffed are 0x7e (the flag) and 0x7d (ESC).
* ACK for each frame. In order to have a robust protocol which resists noise or short disconnections, we implemented the acknowledgement of a frame in the receiver. We have 2 types of acknowledgement: a positive (RR) and a negative one (REJ). The transmitter proceeds to send the next chunk of information only if it received an RR.

1. Application Layer Protocol

The most important functional aspects of the Application Layer protocol are:

* START and END packets. These are special Information packets that notify the receiver that the transmission of information is about to begin/end. These packets also contain meta-information about the file, such as name and size (in bytes). Every detail is stored as a triplet T L V and you can customize the content of this special information packet however you want.
* Colour of the packet. To differentiate between two consecutive packets, we implemented a field named “colour” which can be either “white” (0) or “black” (1). These enabled the receiver to send specific acknowledgements: RR0/RR1 and REJ0/REJ1.

1. Validation, Testing and Efficiency

After finishing the project, we tested it multiple times using the laboratory infrastructure. The protocol resisted noise and disconnections. Depending on the baud rate, the speed of the transmission varied massively. The teacher confirmed that our project passed all tests.

We tested the time it took for the transmission of a 10 KB file on one of our laptops with a virtual serial port, using the “time.h” library in C. The result was between 2500 and 3000 clock ticks. Given the fact that there are 1 000 000 clock ticks in a second, it is almost instantaneous.

1. Conclusions

The first laboratory project was a very interesting one. It challenged our C programming skills and made us learn new things about Computer Networks. The new things that we learned were using alarms in C, modifying the serial connection parameters of the termios, and organising a big project (we wrote about 1000 lines of code). We are very proud of our project, and we are happy that we passed all our test in the practical class.