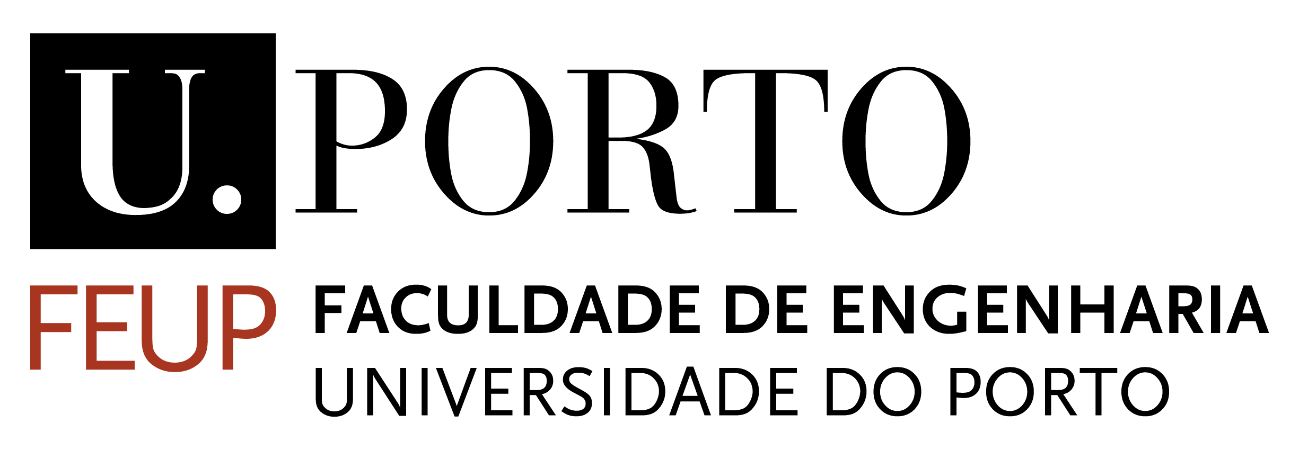
Laboratory Project 1

Computer Networks

Data Link Protocol



**Class 13**

Bernardo Ferreira Campos – up202006056

Ioan-Mircea Radu - up202202727

# Summary

This is the report for the first project of the Computer Network curricular unit of the Informatics and Computing Engineering course. The objective of this project was the development of an application capable of transferring files in between two computers using an asynchronous serial port cable, robust enough to withstand error injection and disconnection of the cables and still maintain proper functioning.

Contents

[Summary 2](#_Toc122960227)

[Introduction 3](#_Toc122960228)

[Architecture 3](#_Toc122960229)

[Code Structure 3](#_Toc122960230)

[Main use cases 5](#_Toc122960231)

[Link Layer 5](#_Toc122960232)

[Application Layer 6](#_Toc122960234)

[Validation 7](#_Toc122960235)

[Data Link protocol efficiency 7](#_Toc122960236)

[Conclusion 7](#_Toc122960237)

# 

# Introduction

The purpose of this report is to enlighten the theorical aspects and the though process behind the development of the data link protocol application.

As mentioned before, the objective of the project was implementing a data link protocol application capable of transferring files between two machines connected by an asynchronous serial port cable (RS-232), even through error injections through the form of *“noise”* and disconnection of the cable.

To do this, functions that handle the connection and disconnection, organizing data into frames, doing frame synchronization, error control, flow control, etc. The application was also written in two independent layers to guarantee the robustness, encapsulation, and transparency of the code.

As specified, the report will abide by the following structure:

* Architecture – description of the functional blocks and interfaces
* Code structure – description of the APIs, data structures, and functions used as well as their relation within the architecture
* Main use cases – identifying the main use cases and function call sequences
* Link layer – identification of the main functional aspects and implementation strategy
* Application layer - identification of the main functional aspects and implementation strategy
* Validation – description of the performed tests
* Data link protocol efficiency – statistic characterization of the protocol’s efficiency
* Conclusion – brief synthesis on the previous topics, reflection on the achieved learning goals

# Architecture

As mentioned, the data link protocol was divided into two independent layers, the application layer and the link layer.

The link layer is responsible for direct communication with the serial port. It handles the establishment and ceasing of the connection, writing and reading data, as well as processing the frames, guaranteeing their proper delimitation and transmission. It uses a system of positive and negative acknowledgments, as well as timeouts and retransmissions to do this.

The application layer resorts to the link layer to execute many of its functions. It handles the processing of different types of packets containing both data and information.

However, the application layer guarantees that, from the link layer perspective, there is no distinction between these packets to maintain layer transparency. In the same fashion, outside of the link layer there is no trace of the stuffing, acknowledgment messages, timeouts, or other error control mechanisms.

# Code Structure

There are a few relevant code files:

**main.c**

Responsible for the initialization of the protocol. Made available by the curricular unit, no changes were made to this code.

**cable.c**

Made available by the curricular unit, no changes were made to this code.

**application\_layer.c /.h**

Responsible connecting the main file to the lower level layer of the application, the link layer. Most relevant functions:

* applicationLayer – initializes the LinkLayer struct with the arguments passed to it. Identifies the role of the machine, as transmitter or recetor, and depending on that will follow a flow of different function calls.
* buildControlPacket – as the name indicates, builds the control packet, necessary before and after of the data packets transmission. Returns the file size.
* TLV – builds the TLV (type, length, value parameters) necessary for the control packet
* buildDataPacket – as the name indicates, builds the data packet, containing the data contained in the file to be transferred.

Responsible for communicating with the serial port. Most relevant functions:

* llopen – responsible for establishing the connection between the two machines.
* llopen\_tx and llopen\_rx – called through llopen, are auxiliary functions used for the transmitter and receiver respectively.
* llwrite – responsible for sending the data frames from the transmitter and processing the response from the receiver
* llread – responsible for receiving the frames from the transmitter and sending an appropriate response according to the state of that frame.
* llclose – responsible for ceasing the connection between the two machines
* llclose\_tx and llclose\_rx – called through llclose, are auxiliary functions used for the transmitter and receiver respectively
* alarmHandler – responsible for the alarm signal after being initiated. Used to control the state of the alarm and the number of times the alarm was enabled
* state\_machine – used to process the frames sent and received by the machines. It is used by several functions above to determine the appropriate acknowledgement. Also used to help process data in data frames

There are two relevant structures part of link\_layer:

LinkLayer struct contains: *char serialPort[50], LinkLayerRole role, int baudRate, int nRetransmissions, int timeout*.

State struct, used in the state machine, contains: *state state\_c, unsigned char address, unsigned char control, unsigned char bcc, unsigned char \*data, unsigned int datasize.*

There is also a definition of the possible states: *START, FLAGRCV, ARCV, CRCV, BCCOK1, DATARCV, ESCRCV, BCCOK2, REJRCV, STOP.*

# 

# Main use cases

The user can select the serial port, the role in the transfer and the file to be transferred before the execution of the program. This can be done through the command terminal or by editing the Makefile.

The sequence of function calls will vary depending on the role of each machine in the transfer. For the transmitter:

* Connection will be established with the receiver with the use of llopen and llopen\_tx.
* The file to be transferred will be opened, and the function buildControlPacket will called to build a control packet containing the size of the file.
* The control packet will be sent with llwrite.
* The program will enter a loop, where data packets are built using the buildDataPacket function and sent using the llwrite function. It will exit the loop once all the bytes of the file are sent or if the reading or sending of the file fails.
* Considering the file was sent successfully, the program will now send another control packet, built, once again, using the buildControlPacket function and the close the file.
* The connection is terminated using llclose and llclose\_tx

For the receiver:

* Connection will be established with the transmitter with the use of llopen and llopen\_rx.
* Using llread, the receiver will receive the first control packet, and use TLV to “interpret” the information received, as the size of the file.
* The program will enter a loop, where it will call llread to get the packets sent by the transmitter. It will exit the loop once it received all the bytes of the file or if there is some error in the receiving or sending process.
* The connection is terminated using llclose and llclose\_rx.

# 

# Link Layer

# 

This function is responsible for establishing the connection between the transmitter and the receiver.

It opens a serial port using the arguments passed to the function. It stores the current port settings and sets the new ones.

Depending on the role of the machine, either llopen\_tx or llopen\_rx will be called, for the transmitter and the receiver respectively.

In llopen\_tx the SET frame will be built and sent, and then the alarm will be activated. The transmitter will now wait for a response from the receiver. If it does not get an appropriate response the alarm will timeout, and the process wil repeat itself until the transmitter gets an appropriate response or the number of tries for this process is exceeded.

In llopen\_rx the receiver will start by processing the SET frame sent by the transmitter. If the SET frame was properly received, the process will continue. The receiver will now build and send an UA frame.

In both these functions, the frames received are processed using an auxiliary function, state\_machine.



This function is responsible for sending the Information frames, containing the contents of the file to be transferred.

The DATA frame is built, and the byte stuffing is made with the use of the stuffing auxiliary function. This frame will be sent, and the alarm will be set, and the transmitter will wait for a response from the transmitter, if an appropriate response is not received before a timeout, this process will be repeated until the number of tries is exceeded or an appropriate response is received. If a REJ frame is received, instead of the RR frame that is expected, the number of tries will be reset, and retransmission will begin.

If the RR frame is received, the DATA frame flag will be changed.



This function will receive the information frames.

The processing of the received frame will be made with the state\_machine auxiliary function, as well as the byte destuffing. If the information frame is received as supposed to, llread will respond by sending an RR frame. Otherwise, a REJ frame will be sent.



The function is responsible for the termination of the connection between the transmitter and the receiver.

In the same fashion as the llopen function, a different auxiliary function will be called, depending on the role of the machine.

For the transmitter, llclose\_tx will be called. Here, a DISC frame will be built and sent, and the alarm will be set. Now the transmitter will wait to receive a DISC frame back from the receiver, as acknowledgement to the disconnection. Similar to previous functions, if it does not receive said frame before the timeout occurs, a retransmission will occur until the number of tries is exceeded. Having received the DISC frame, and UA frame will be sent.

For the receiver, llclose\_rx will be called. The DISC frame will be received and processed, and a DISC frame will be sent in response. Finally, the receiver will wait for an UA frame as response. Once its received, the connection is ceased.

In both these functions, the frames received are processed using an auxiliary function, state\_machine.

# 

# Application Layer



This application will be responsible for the control of the transferring process. The settings for the transfer, the creation, reading and writing of the transferred file, as well as the call to all the functions from link layer will be done through this function.

It will start by defining the settings of the transfer, putting them into an appropriate struct, using the arguments passed though the function parameters.

Depending on the role of the machine, the flow of this function differs.

In the transmitter, the file will be opened, a control packet will be built and sent using llwrite. Then the contents of the file will be read, put into data packets, and sent using llwrite again. A final control packet is built and sent.

In the receiver, the first control packet is read using llread. A file is created, and as data packet are received using llread, information is written into it.

Finally, the connection is terminated using llclose.

Multiple system calls, used to opening, reading and writing files, as well as calls to link layer functions are prone to errors. So, every time they are used they are checked for these errors, and if any occurs the process will be interrupted.

# Validation

Due to a problem in the application we were not able to solve, no tests of the whole protocol were executed, besides the testing of the llopen and llclose functions that work as expected.

# Data Link protocol efficiency

As mentioned, due to a problem with the application, the protocol could not be tested, so no relevant statistic regarding the transfer process could be collected.

# Conclusion

Despite not being able to create a working application, this project still allowed us to grasp major concepts regarding a data link protocol using a serial cable.

We believe that the overall code that was written is correct, possibly with only a small, but critical mistake, that does not allow for the proper functioning of the application.