**PKS project 2**

**Communication through UPD protocol**

**Design**

Márk Bartalos

Seminars with**: Ing. Lukáš Mastiľak**Seminars on**: Thu 14:00-15:50**

# Introduction

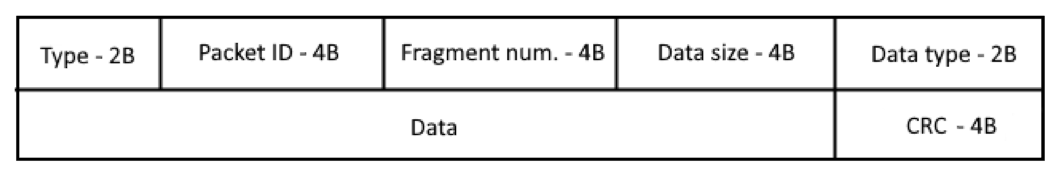
We have to design an application that uses our custom protocol under UDP. That application has to deliver text messages and files. The transfer of data will happen in binary form.

When we start the application, the application will ask if we want to use this instance as receiver or as sender. These states can be change while the program is running.

To be able to communicate and make sure the data is not corrupt we have to design a custom protocol header.

# Header

The protocol header will consist of 7 parts. Each part will have its appropriate bytes (size).



## Communication type (2B)

* 0 - request for synchronization (SYN)
* 1 – accept synchronization (SYN+ACK)
* 2 – OK response for keep-alive, response for package received and other types of requests
* 3 – end connection (FIN)
* 4 – resend data
* 5 – request for keep-alive (keep alive check)
* 6 – request for data transfer initialization
* 7 – data send
* 8 – switch tasks

## Packet ID

Will contain the unique identification number of the packet.

## Fragment num

If the data is split into fragments (almost all the time), this will be used to track the number of fragments and to be used to reconstruct the data.

## Data size

This will contain the size of the data (or the fragments size) payload.

## Data type

* 0 – Text
* 1 – File transfer

## Data

This will contain the data (complete or fragment) itself.

## CRC

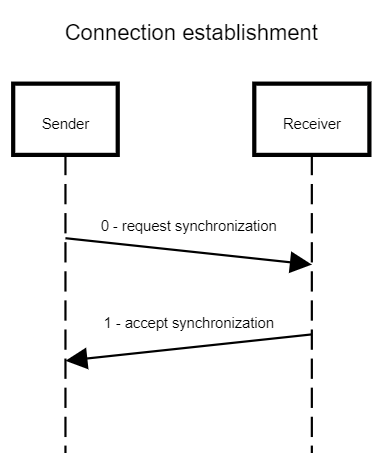
This will contain the Cyclic Redundancy Check code for the data to verify the correctness.

# Connection management

There will be different connection scenarios which will be defined in this section.

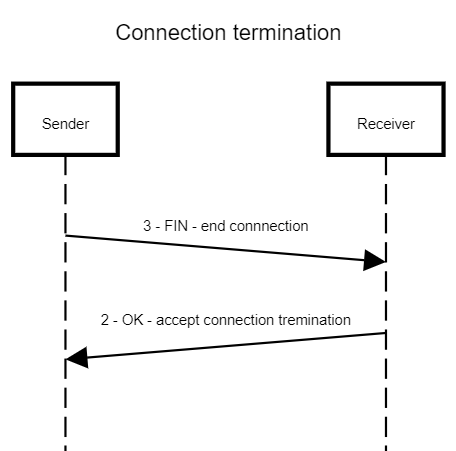
## Connection establishment

Successful connection establishment will consist of two (or maybe three, another ACK for security) requests. First the sender will send a synchronization request to the receiver, by initializing the request this will define the first device as the sender. The second device will have to reply with a SYNCHRONIZATION ACCEPT response for the connection to be successfully established.



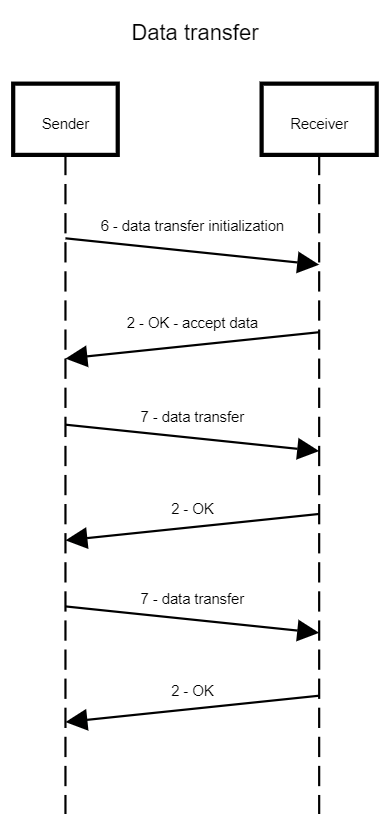
## Connection termination

Connection termination similarly to establishment will consist of two requests, but not necessarily. Here in difference to the first one either of the computers can initialize the connection termination.



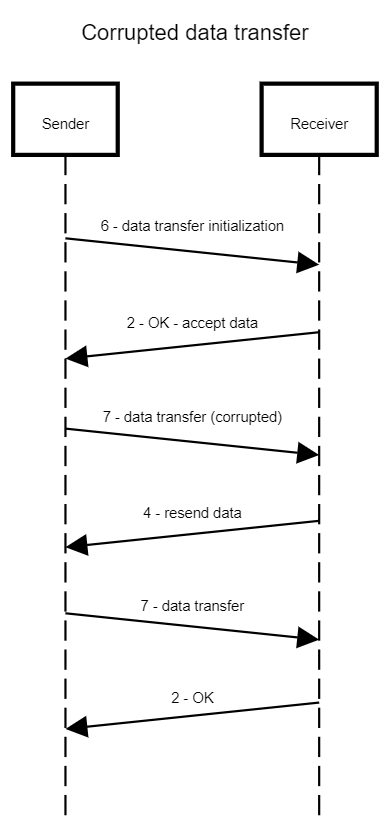
## Successful data transfer

For a successful data transfer the every transfer has to be started with an initialize data transfer request and has to be accepted with an OK request, then all data fragment responses has to be followed with an OK request that the data is received and the checksum is ok.



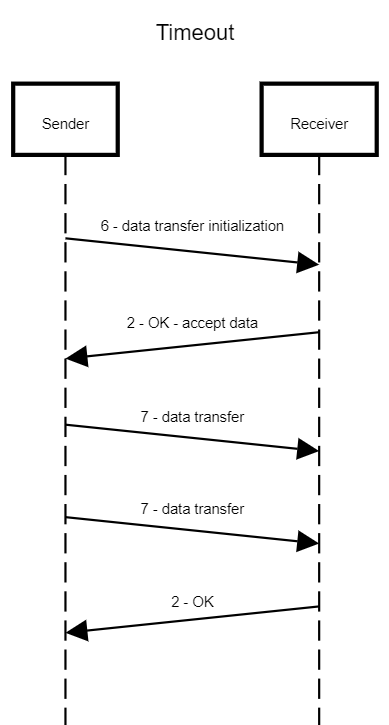
## Corrupted data transfer

If corrupted data is received, the checksum doesn’t match, than the receiver will send a RESEND DATA request back to the sender and the sender will send the data fragment again. This corruption will be simulated by randomly changing a bit in the data or by manually changing the CRC value.



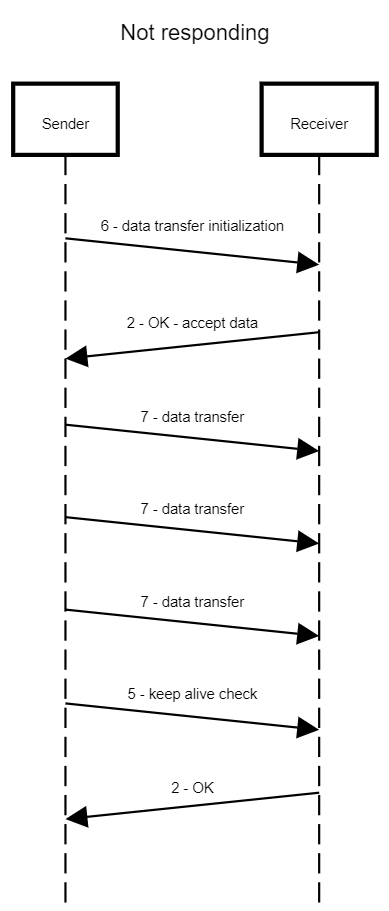
## Data transfer timeout (ARQ)

If the sender doesn’t receive OK response after the data is sent, the sender will wait some time and send the data again, this will have a limit, after the limit it will send keep alive request to check the connection state.



## The receiver is not responding

If the receiver doesn’t respond after several responses the sender will send a keep alive request to check if the receiver is alive or it just doesn’t know how to handle the data packages. The data transfer most probably will be aborted.



## Switch tasks

Switching task will start with a SWITCH TASK request which either could be sent by the receiver or the sender. The SWITCH TASK request has to be accepted with an OK response. After the tasks have been switched optionally there, we be an establishment of the connection again.

## 

# CRC method