Reliable protocol over UPD

PKS 2022

Cvičiaci: Kristián Košťál

Student: Mykhailo Sichkaruk

Table of Contents

Main

Description:

My Protocol (MRP) - is full duplex, reliable, fast, message-aware, supporting multiple connections and sends.

Header:

Header takes 8 bytes in the packet

Data part can take from 1 to 1016

Bits/ Row	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	Packet type				File id				Packet number in window							
2	Window number															
3	Checksum															
4																
5	Data															
N																

Packet types:

1. Data
2. Confirm Data
3. Open Connection
4. Confirm Open Connection
5. Init File Transfer
6. Confirm Init File Transfer

File ID:

To provide multiple connections and 2 way sending protocol should have identification of file which is currently sending. For example when you're simultaneously sending 2 files to 1 endpoint, receiver has to know what packet is responsible for which file.

Values: [0-15]

Packet number in window:

Number for packet number in window, that helps to send summarized confirm for whole window

Read more in chapter: ARQ

Window number:

Value that represents ordinal number of current window in file transfer

Values: [0-65535]

Checksum:

CRC-32 used to check packets integrity

Read more – in chapter CRC32

Connection:

Protocol is focused on sending files, so to send file, sender should open connection in this direction.

Every endpoint can only send, only receive, or receive and send. To open connection for sending, endpoint should send OpenConnection and wait for ConfirmOpenConnection packet

1-way sending (Figure 1)

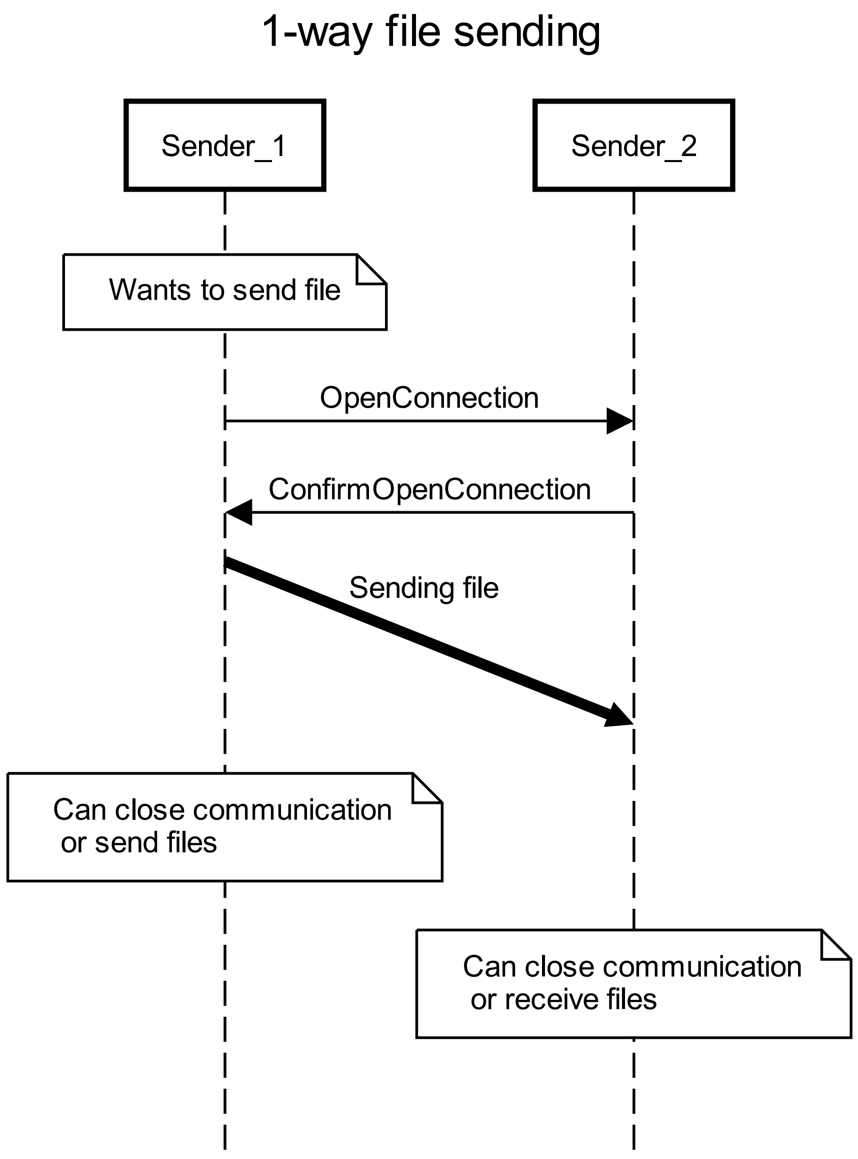


Figure 1

2-way sending (Figure 2):

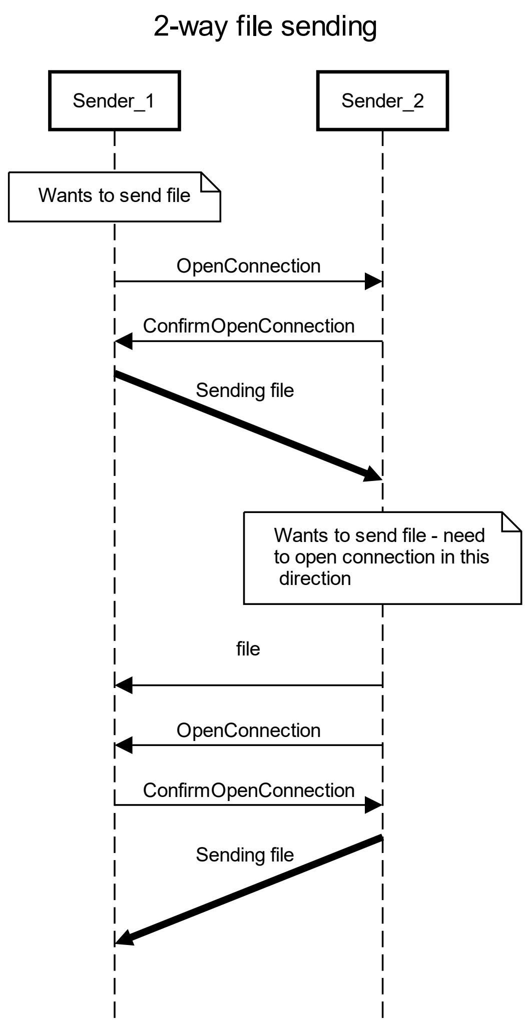


Figure 2

KeepAlive

To keep connection alive, sender should resend OpenConnection packet every 11s since last packet received, then wait for ConfirmOpenConnection, then connection is considered open.

If ConfirmOpenConnection isn't received in 11s – connection in this direction considered closed for sender

If receiver got last packet later than 21s – connection in this direction considered closed for receiver

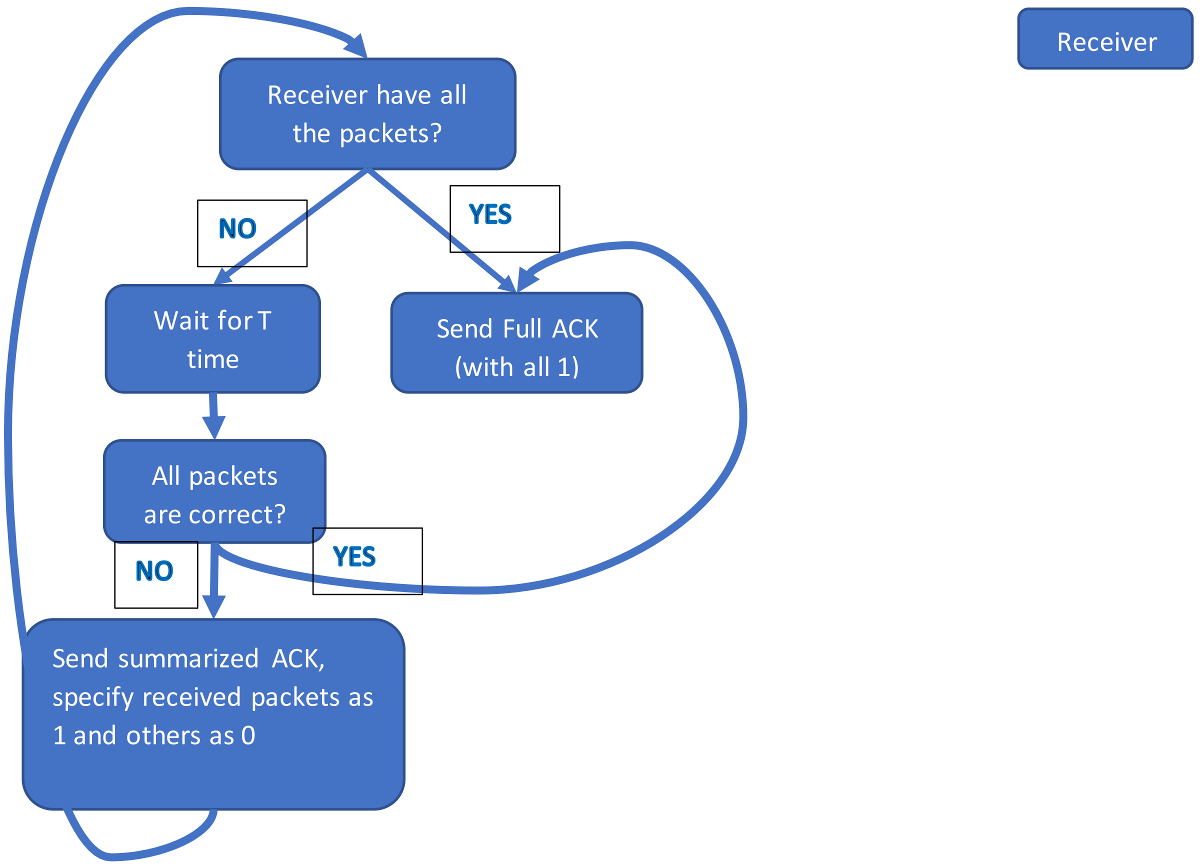
ARQ – acknowledge whole window with one packet

Sender and Receiver have window size that is divisible by 8.

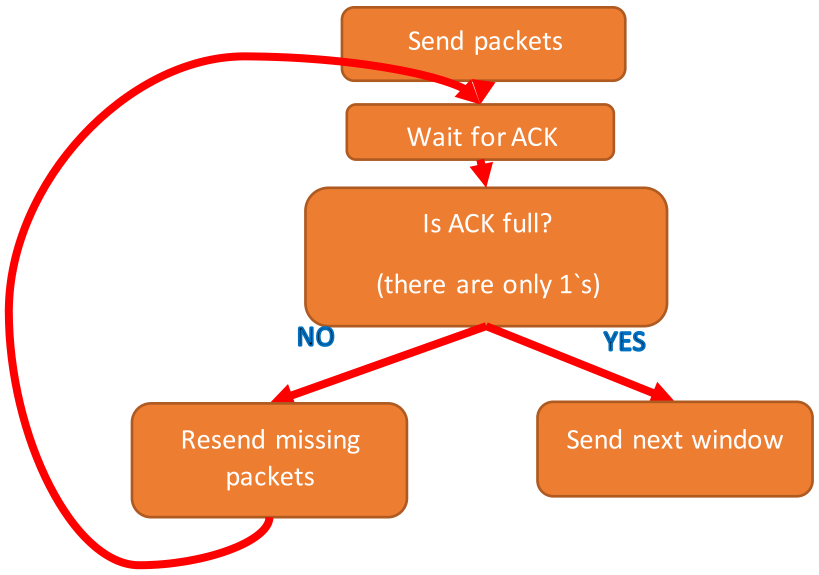
Sender pushes packets of the window and then will be waiting for answer. If everything ok it will get Full ACK (all packets is received), else it gets summarized ACK where will be specified what packets are missing, then Sender retransmits them and again waits for ACK.

Receiver gets packets and checks if there are all of them. If no, it will send summarized ACK and then would be waiting for missing packets.

Full Confirm / Summarized Confirm – packet of type "Confirm" with data that represents what packets of the window got to Receiver and what didn't


Receiver
Receiver have all the packets?
NO
YES
Wait for T time
Send Full ACK (with all 1)
All packets are correct?
NO
YES
Send summarized ACK, specify received packets as 1 and others as 0

Sender


Send packets
Wait for ACK
Is ACK full?
(there are only 1`s)
NO
YES
Resend missing packets
Send next window

Checksum

CRC23, or Cyclic Redundancy Check 23, is a type of checksum algorithm that is used to detect errors in data transmission or storage. The basic idea behind a CRC algorithm is to generate a checksum, or a small amount of error-detecting code, based on the data being transmitted. When the data is received, the receiver can use the same CRC algorithm to generate a checksum based on the received data and compare it to the original checksum. If the two checksums match, the data is assumed to be error-free. If they do not match, it indicates that there was an error in transmission and the data may be corrupted.

The specific details of how a CRC23 algorithm works can vary, but in general, it involves dividing the data being transmitted into fixed-size blocks and computing a checksum for each block using a mathematical formula. The resulting checksum values are then appended to the end of the data, forming a complete message. When the message is received, the receiver recomputes the checksums for each block of data and compares them to the original checksums. If any of the checksums do not match, it indicates that an error has occurred, and the data may be corrupted.

Why?

If we are transferring image with size < 10 MiB = 10*100,000, with packet length of 100 B, then we have 10,000 packets. Summarized error – can be 15% for crc16, but 0.0002% with crc32. Also CRC32 is good algorithm to detect errors that UPD lost

CRC Type	Undetected Errors	% Undetected
CRC-8	$1/2^{8}$	0.39
CRC-16	$1/2^{16}$	0.0015
CRC-32	$1/2^{32}$	0.00000002
CRC-64	$1/2^{64}$	5.4×10^{-20}

Message

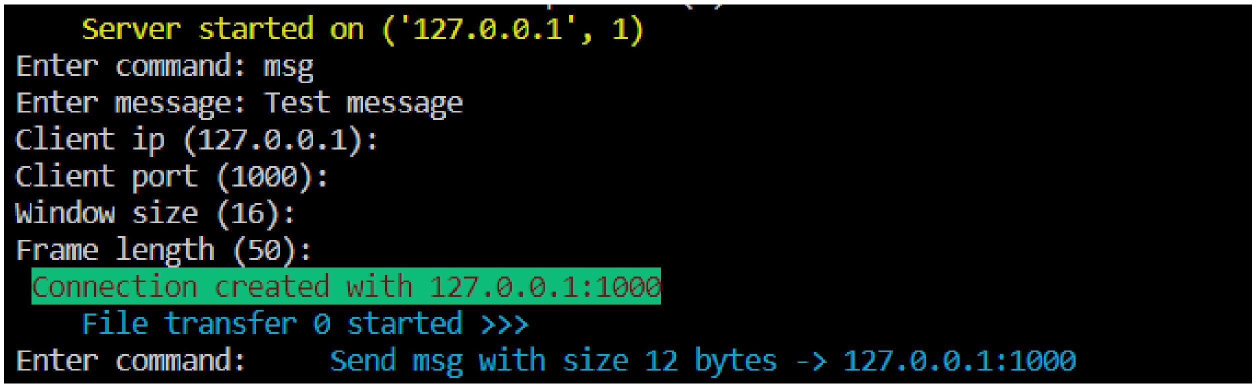
Message is a wrapper over file transfer

Under the hood message is the ".msg" text file that is received, read, and then logged as message

On sender's side it is created with message text, sent, and then deleted

On receiver's side it is received, logged as message, and then deleted

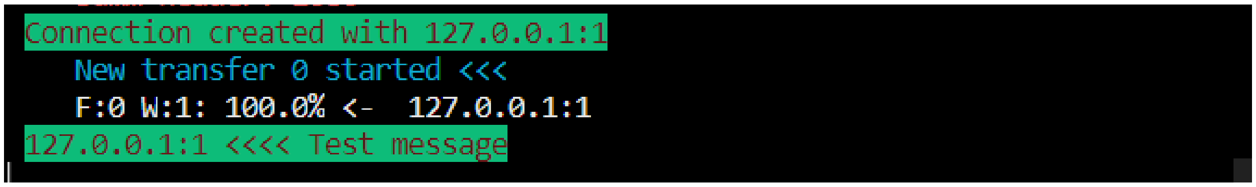
Sender's side view (Figure 3)



```
    Server started on ('127.0.0.1', 1)
Enter command: msg
Enter message: Test message
Client ip (127.0.0.1):
Client port (1000):
Window size (16):
Frame length (50):
 Connection created with 127.0.0.1:1000
    File transfer 0 started >>>
Enter command:     Send msg with size 12 bytes -> 127.0.0.1:1000
```

Figure 3

Receiver's side view



```
Connection created with 127.0.0.1:1
   New transfer 0 started <<<
   F:0 W:1: 100.0% <-  127.0.0.1:1
127.0.0.1:1 <<<< Test message
```

Figure 4

File transfer

After Connection is opened for sender, file sending can be started

First packet in file transfer is InitFileTransfer type, which has same header layout but with different meaning of fields

Init data:

1. File name
2. File length in bytes
3. MD5 hash of file

This data is converted into bytes. Then calculated number of windows to transfer this data, it depends on window size and length of a fragment.

InitFileTransfer packet:

Then InitFileTransfer packet sent with the number of windows for init data in payload

Bits/ Row	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	Packet type				File id				Window size							
2	Fragment length															
3	Checksum															
4																
5	Number of windows for init data															

Then sender waits for ConfirmInitFileTransfer packet to start sending init data and file

Receiver point of view:

When receiving is available and InitFileTransfer is received, receiver confirms with ConfirmInitFileTransfer. Than Receiver knows that first N windows will transfer init data, then file goes.

So, first N-1 windows, receiver collect data, then on N-th window receiver parse init data and opens file to write in.

From init data receiver also knows file length in bytes, and know when transfer is ended

Other

Error simulation:

Error is generated on receiver's side, by changing 1 byte inside the packet

```python
def broke_packet(self, data: bytes) -> bytes:
    # Change any byte in the packet
    random_index: int = randint(0, len(data) - 1)
    bytearray_data: bytearray = bytearray(data)
    bytearray_data[random_index] = randint(0, 255)

    return bytes(bytearray_data)
```

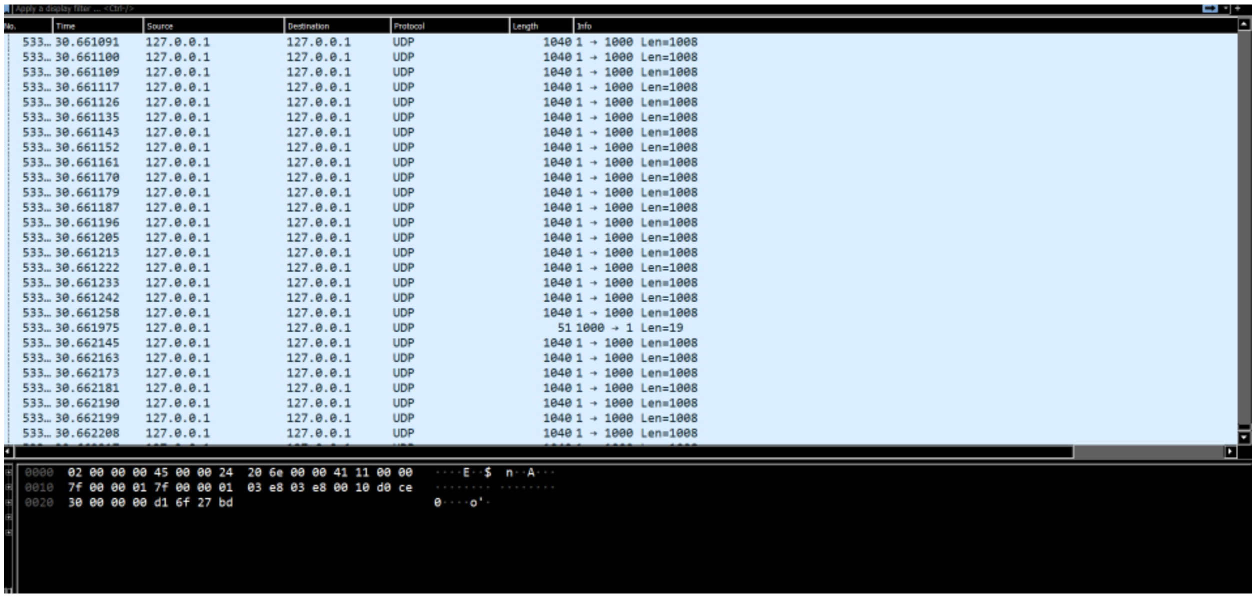
```python
        # Broke packet if error rate is set
        if self.error_rate > 0 and randint(0, self.error_rate) == 0:
            data: bytes = self.broke_packet(data)
        packet: MRP = parse_packet(data)
```

MD5 hash

Before sending, sender get MD5 of file, and send this data inside init data

After receiving, new written file also hashed and logged to compare with expected hash

Wireshark check



```
Apply a display filter ... <Ctrl-/>
No.   Time         Source       Destination  Protocol  Length Info
533…  30.661091    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661100    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661109    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661117    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661126    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661135    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661143    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661152    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661161    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661170    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661179    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661187    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661196    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661205    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661213    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661222    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661233    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661242    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661258    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.661975    127.0.0.1    127.0.0.1    UDP         51 1000 → 1 Len=19
533…  30.662145    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.662163    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.662173    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.662181    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.662190    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.662199    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008
533…  30.662208    127.0.0.1    127.0.0.1    UDP       1040 1 → 1000 Len=1008

0000  02 00 00 00 45 00 00 24  20 6e 00 00 41 11 00 00   ····E··$ n··A···
0010  7f 00 00 01 7f 00 00 01  03 e8 03 e8 00 10 d0 ce   ········ ········
0020  30 00 00 00 d1 6f 27 bd                            0····o'·
```

Sources

[1] - How to Validate Your Data with a Cyclic Redundancy Check (CRC): https://quickbirdstudios.com/blog/validate-data-with-crc/#:~:text=CRC%20is%20an%20error%20detection,stored)%20along%20with%20that%20data.&text=The%20check%20value%20is%20called,additional%20information%20to%20the%20message.