PKS, Custom UDP Protocol

Andrii Tyvodar, ID: 127301

Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava

Contents

1 Changes from previous version

1. user interface.Now i use user interface not just a command line, so i currently don't need to have another thread for sending messages, simply when user push button "send" i run a function that sends message or file
2. added fragmentation, checksum,keep-alive, Arq
3. added new type of message == 6
4. added simulation of checksum error in case of text messages

2 Packet Structure

Table 1: Custom UDP Protocol Packet Structure

Field	Size	Description
Sequence Number	4 bytes	Packet sequence number
heder Length	1 byte	(1 - there is optional fields 0 - no optional filds)
Type	1 byte	Type (0 - SYN, 1 - text, 2 - file, 3 - ACK, 4 - SYN-ACK, 5 -keep_alive, 6 - explained_aft
ACK	1 byte	Acknowledgment flag (0 - no ACK, 1 - ACK received)
Data length	2 bytes	(Optional)
Data	Variable	Actual data (text or file fragment)
checksum	2 bytes	for CRC16

type 6 is a special type for sending files. when all fragments of file sent, i send packet with type 6 to say to other side that it can save a file

3 3-Way Handshake Steps

1. **Client A Sends SYN**
   - Client A sends a SYN packet to Client B.
   - Packet Details:
     - Sequence Number
     - Type: 0 (SYN)
     - ACK: 0
     - and other data
2. **Client B Receives SYN and Sends SYN-ACK**
   - Client B receives the SYN packet and acknowledges it.
   - Packet Details:
     - Sequence Number: Unique identifier
     - Type: 4 (SYN-ACK)
     - ACK: 1
     - and other data

3. **Client A Receives SYN-ACK and Sends ACK**
   - Client A receives the SYN-ACK packet and sends an ACK to complete the handshake.
   - Packet Details:
     - Sequence Number: Incremented from original
     - Type: 3 (ACK)
     - ACK: 1
     - and other data

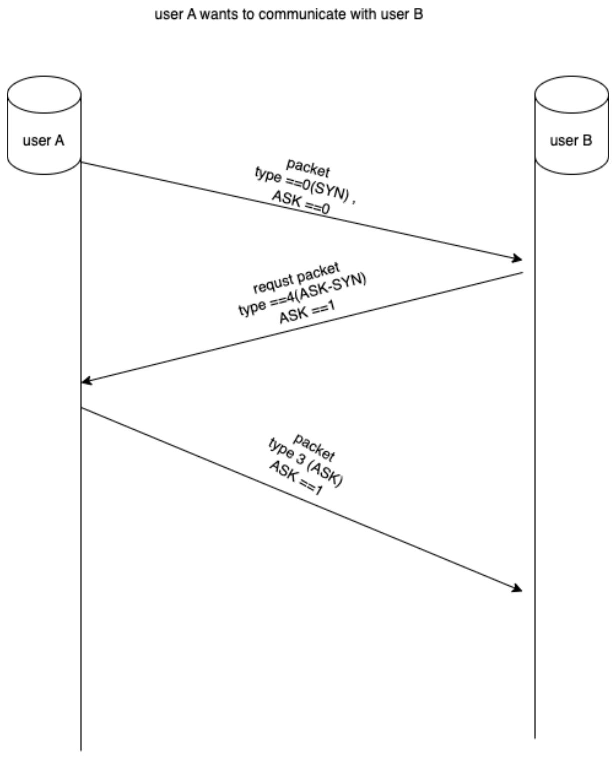


Figure 1: 3-way handshake

4 keep alive

1. **Initiation of Keep-Alive from Client A**
   - Client A decides to send a Keep-Alive packet to confirm that the connection is still active.
   - Client A constructs the packet with the following parameters:
     - Type: 5 (Keep-Alive)
     - Sequence Number: Sequence number (could be incremented)
     - ACK: 0
     - and other data
2. **Sending the Keep-Alive Packet from Client A to Client B**
3. **Receiving the Keep-Alive Packet on Client B's Side**
   - Client B receives the packet.
   - Client B checks the **Type** field:
     - If **Type** = 5, Client B understands that this is a Keep-Alive packet.
4. **Processing the Keep-Alive Packet by Client B**
   - Client B constructs an ACK packet
     - **Type**: 3 (ACK)
     - header length : 0
     - **Sequence Number**: Sequence number (could be incremented)
     - **ACK**: 1 (confirmation of receipt)
     - **other data**
5. **Sending the ACK Packet from Client B to Client A**
   - Client B sends the ACK packet back to Client A.
6. **Receiving the ACK Packet on Client A's Side**
   - Client A receives the ACK packet from Client B.
   - Client A checks the **Type** field:
     - If **Type** = 3, it indicates that Client B is alive
7. **Monitoring Activity**
   - Both clients can set timers for periodic activity checks on the connection.
   - If Client A does not receive ACK packets within a certain timeframe, it may assume that the connection is inactive.
   - Client B can do the same to check the activity of Client A.

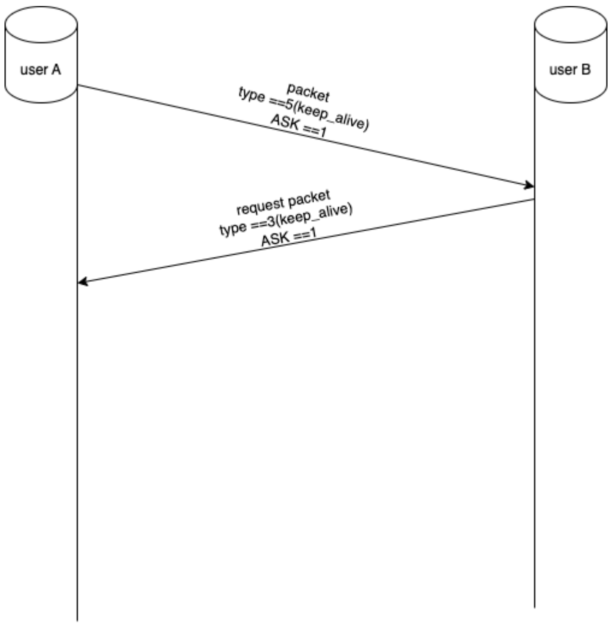


Figure 2: keep alive

5 CRC16

CRC16 value = 0xFFFF
polynomial for CRC16 = 0xA001
Checksum calculation
For the all bytes in packet:

1. XOR the current byte with the CRC value.
2. Shift the CRC value to the right by 1 bit.
3. If the least significant bit is 1, XOR the CRC value with the polynomial value.

after all calculations i riceive checksum value and send a packet to user.User than calculates checksum value.

6 ARQ(selective repeat)

If the receiver computes a checksum that doesn't match the one sent in the packet, it indicates data corruption.

1. **Send a Negative Acknowledgment (NACK):** Instead of sending an Acknowledgment (ACK), the receiver sends a Negative Acknowledgment (NACK) to the sender. This informs the sender that the packet has been corrupted or failed the integrity check.
2. **Request Packet Resend:** After receiving the NACK, the sender should retransmit the same packet with the correct sequence number to attempt a successful delivery.

I choose size of window for selective repeat == 4.
steps of ARQ:

1. **Client A Sends Fragments:**
   - Client A sends up to 4 fragments of data (without waiting for ACKs)
2. **Client A Waits for ACKs:**
   - After sending the 4 fragments, Client A waits for ACKs from Client B

3. **Client B Receives Fragments:**
   - Upon receiving a fragment from Client A, Client B checks the checksum value using CRC16 for each packet.
4. **ACK Handling by Client B:**
   - If Client B successfully verifies the integrity of a fragment:
     - It sends an ACK with the sequence number of the successfully received fragment (ACK = 1).
   - If Client B detects an error in the fragment:
     - It sends an ACK indicating an error for that specific fragment (ACK = 0).
   - **Client A checks ACKs:**
     - Client A receives ACKs from Client B.
     - For each ACK:
       - ∗ If the ACK = 0, Client A resends the specific fragment

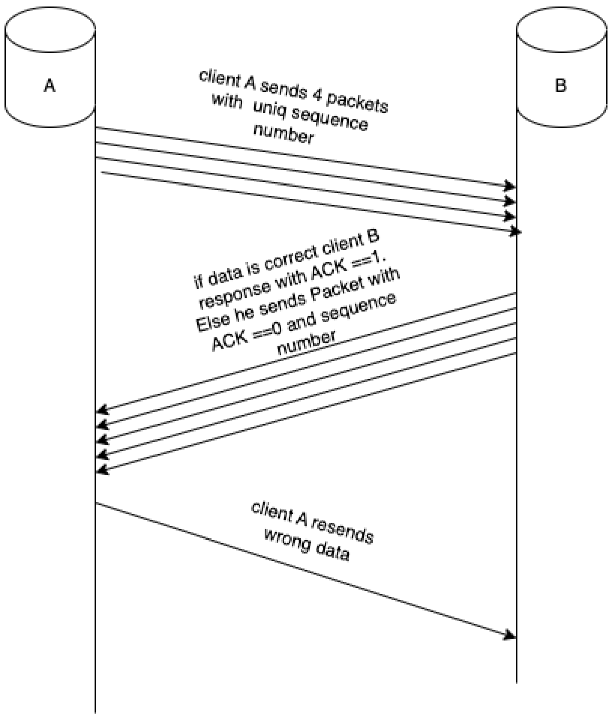

A
B
client A sends 4 packets with uniq sequence number
if data is correct client B response with ACK ==1. Else he sends Packet with ACK ==0 and sequence number
client A resends wrong data

Figure 3: ARQ

7 diagram of processes of my application

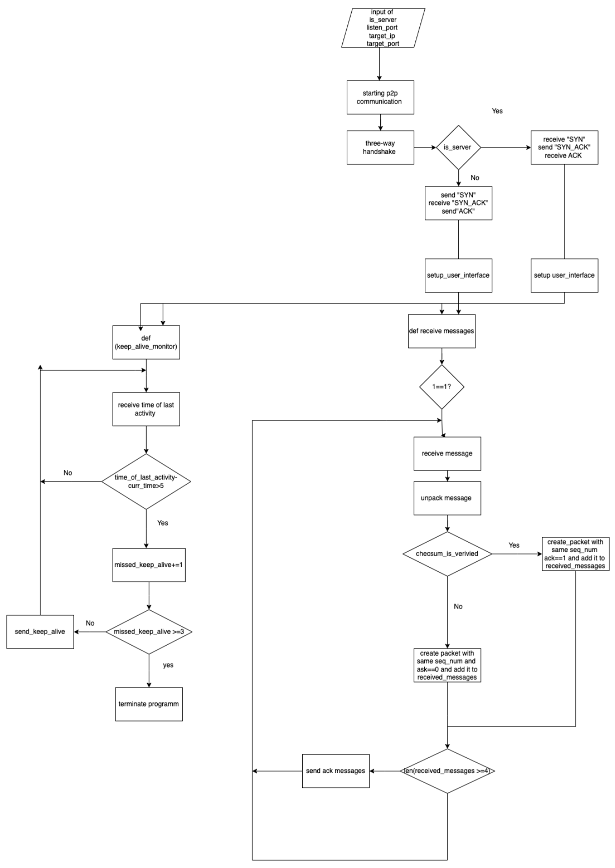

input of
is_server
listen_port
target_ip
target_port
starting p2p communication
three-way handshake
is_server
Yes
receive "SYN" send "SYN_ACK" receive ACK
No
send "SYN" receive "SYN_ACK" send"ACK"
setup_user_interface
setup user_interface
def (keep_alive_monitor)
def receive messages
1==1?
receive time of last activity
time_of_last_activity-curr_time>5
No
Yes
missed_keep_alive+=1
missed_keep_alive >=3
No
send_keep_alive
yes
terminate programm
receive message
unpack message
checsum_is_verivied
Yes
create_packet with same seq_num ack==1 and add it to received_messages
No
create packet with same seq_num and ask==0 and add it to received_messages
len(received_messages >=4)
send ack messages

Figure 4: diagram of code

8 how i will made an error for checksum

I just midify 1 byte of data before sending

9 Acceptance protocol

1. The program must be implemented in one of the following programming languages: C, C++, C#, Python. Rust can be used only if approved by the instructor.(Yes, I use Python)
2. The user must be able to set the listening port on the node.(Yes)
3. The user must be able to set the target IP address and port.(Yes)
4. The user must be able to select the maximum fragment size on a node and dynamically change it while the program is running before sending the text/file.(Yes)
5. The node must be able to display the following information about the received/sent text/file:

   the name and absolute path of the file on the node,

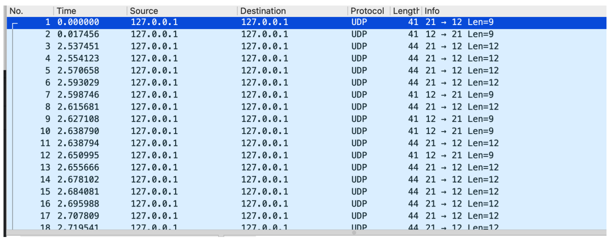
   the fragment size and number of fragments, including the total length (the last fragment may have a different size than the previous fragments). (Yes)
6. Simulate a transmission error by sending at least one erroneous fragment during the transfer of text and files (an error is deliberately introduced into the data part of the fragment or the checksum, meaning the receiving side detects the error during transmission).(Yes , but i simulate error only in text not in file)

7. The receiving side must be able to notify the sender of the correct and incorrect delivery of fragments. In the case of incorrect delivery of a fragment, it must request the resending of the corrupted data.(Yes)

8. The ability to send a 2MB file within 60 seconds and save it on the receiving node as the same file.(Yes)

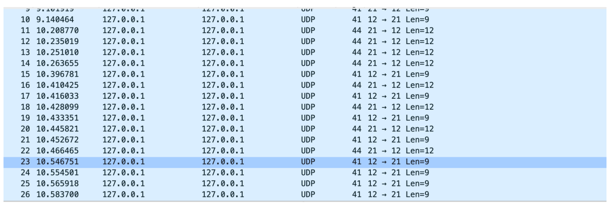
9. The ability to set the location on the node where the file will be saved upon receipt.(Yes)

10 screens from wireshark



```
No.  Time      Source     Destination  Protocol Length Info
  1  0.000000  127.0.0.1  127.0.0.1    UDP      41  21 → 12 Len=9
  2  0.017456  127.0.0.1  127.0.0.1    UDP      41  12 → 21 Len=9
  3  2.537451  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
  4  2.554123  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
  5  2.570658  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
  6  2.593029  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
  7  2.598746  127.0.0.1  127.0.0.1    UDP      41  12 → 21 Len=9
  8  2.615681  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
  9  2.627108  127.0.0.1  127.0.0.1    UDP      41  12 → 21 Len=9
 10  2.638790  127.0.0.1  127.0.0.1    UDP      41  12 → 21 Len=9
 11  2.638794  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
 12  2.650995  127.0.0.1  127.0.0.1    UDP      41  12 → 21 Len=9
 13  2.655666  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
 14  2.678102  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
 15  2.684081  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
 16  2.695988  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
 17  2.707809  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
 18  2.719541  127.0.0.1  127.0.0.1    UDP      44  21 → 12 Len=12
```

Figure 5: without corr



```
  9  9.101919   127.0.0.1  127.0.0.1  UDP  41  21 → 12 Len=9
 10  9.140464   127.0.0.1  127.0.0.1  UDP  41  12 → 21 Len=9
 11  10.208770  127.0.0.1  127.0.0.1  UDP  44  21 → 12 Len=12
 12  10.235019  127.0.0.1  127.0.0.1  UDP  44  21 → 12 Len=12
 13  10.251010  127.0.0.1  127.0.0.1  UDP  44  21 → 12 Len=12
 14  10.263655  127.0.0.1  127.0.0.1  UDP  44  21 → 12 Len=12
 15  10.396781  127.0.0.1  127.0.0.1  UDP  41  12 → 21 Len=9
 16  10.410425  127.0.0.1  127.0.0.1  UDP  44  21 → 12 Len=12
 17  10.416033  127.0.0.1  127.0.0.1  UDP  41  12 → 21 Len=9
 18  10.428099  127.0.0.1  127.0.0.1  UDP  44  21 → 12 Len=12
 19  10.433351  127.0.0.1  127.0.0.1  UDP  41  12 → 21 Len=9
 20  10.445821  127.0.0.1  127.0.0.1  UDP  44  21 → 12 Len=12
 21  10.452672  127.0.0.1  127.0.0.1  UDP  41  12 → 21 Len=9
 22  10.466465  127.0.0.1  127.0.0.1  UDP  44  21 → 12 Len=12
 23  10.546751  127.0.0.1  127.0.0.1  UDP  41  12 → 21 Len=9
 24  10.554501  127.0.0.1  127.0.0.1  UDP  41  12 → 21 Len=9
 25  10.565918  127.0.0.1  127.0.0.1  UDP  41  12 → 21 Len=9
 26  10.583700  127.0.0.1  127.0.0.1  UDP  41  12 → 21 Len=9
```

Figure 6: diagram of code

11 Sources

- SR
- Lua
- Lectures on FIIT
- diagrams i did using: draw.io