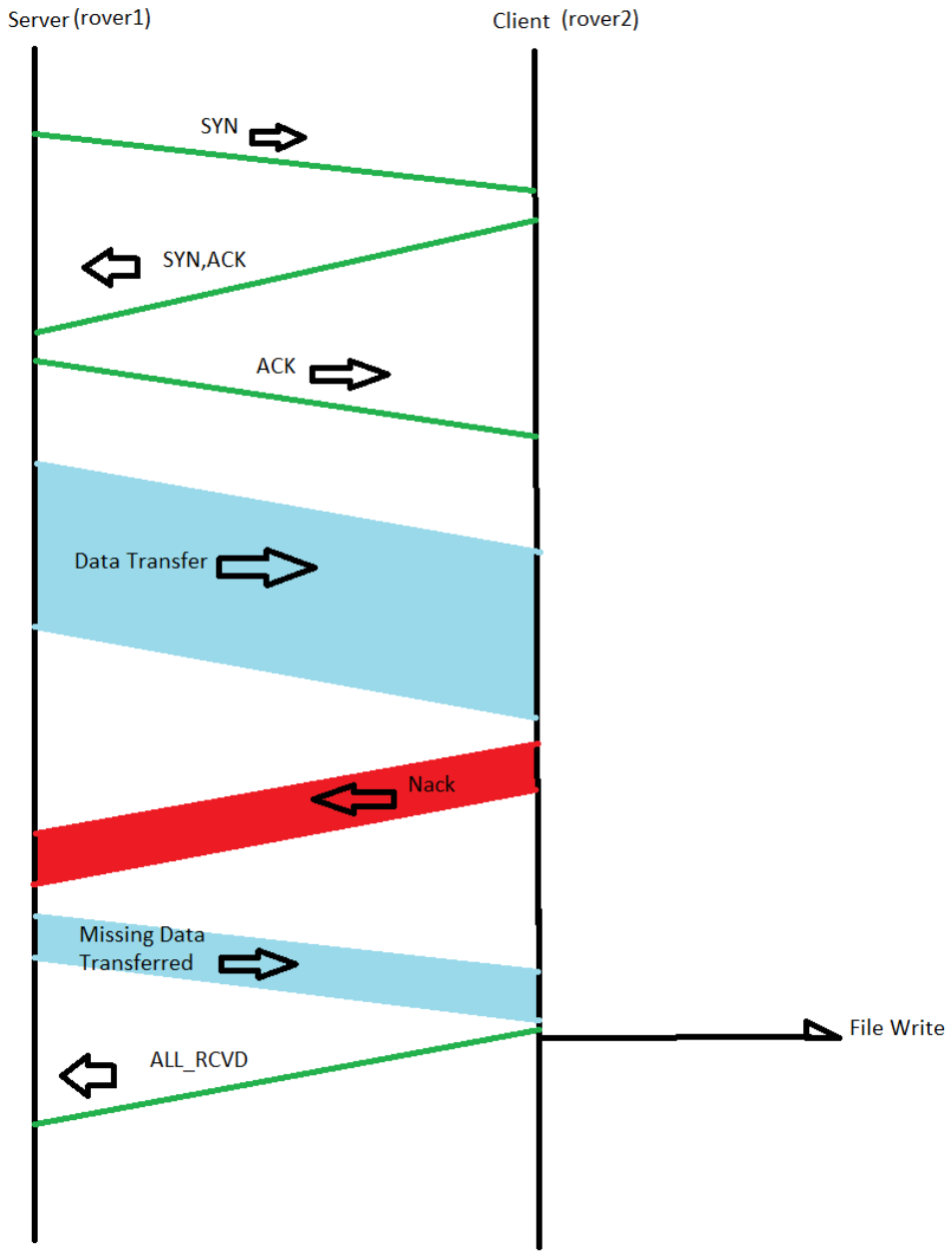
Lunar Rover Protocol

General Design

This protocol is a nack based protocol. In this protocol the client and the server exchange basic information about the file to be transferred. There is a basic three way handshake at the beginning, followed by data transfer. Unreceived segment numbers are sent by the client as nack which the server responds to by sending them. Here is an image that describes the general flow of the protocol



Handshake

The handshake process is inspired by the TCP 3 way handshake protocol. During this exchange the server sends a 'syn' (Synchronize) packet with the initial segment and last segment of the data to be transferred (packet design for each step discussed later). This is then followed by a SYN-ACK packet from the client where it acknowledges the start and end segments that the server just sent (which is the ACK) and it sends its own start and end segments (which is the SYN). If the client has no data to transfer then its start and end segments in the packet will both be 0.

Once the server receives this SYN-ACK packet from the client. It then sends an ACK packet where it acknowledges the start and end segments sent by the client.

This completes the three way handshake and it moves ahead with the next process which is data transfer.

Data transfer

Once the three way handshake is done, the server side sends all the segments to the client in one go. The server program first takes the data size to be sent (which is 1000 bytes in this program) and it creates segment numbers based on the data size and the size of the file to be sent.

For example if we have send a 10.5 Kilobyte file, considering data size per packet is 1000 bytes. The server creates segment numbers 0 to 10 (11 packets where the last packet will have 500 bytes). It then adds these segment numbers to a queue.

It then proceeds to create the data packets and sends them to the client. Once a packet is sent, it removes the segment number from the queue.

When the client receives the data packet, it creates a unique id for each segment received, this id is a 3 tuple (start segment, end segment, segment number). This helps identify each segment uniquely (incase we stream data and the segment numbers have to be reused, this helps in identifying packets that belong to different data transfer sessions).

The data and the unique id are stored in a hash map which is kept for later extraction and assembly. Once the client receives a segment number which is equal to the last segment number of the session. It begins to check for missing segments from the hashmap. When it finds a missing segment it immediately sends a nack for the same.

Nack

The client sends nacks for all the segments that it hasn't received. Once this reaches the server, the server then sends the corresponding missing data packets to the client. The client then adds data from the arriving packets to the hashmap and then checks if there are missing segments again.

All Packets Received

When the client finds all the packets have been received. It then sends an “All packets received” packet that acts as a final ACK. Once this is done, it proceeds to write to the file.

Write to file

The client extracts each unique packet from the hashmap using the 3 tuple. This is done in sequence:

(Starting segment, ending segment, 1)

(Starting segment, ending segment, 2)

…

Once the data packets are extracted they are then added to a stream and written to a file.

Packet Design

The packet design mainly follows this particular template:

```
BYTE 0                                                    BYTE 1
[SYN,DATA,NACK,ACK, ALL_RCVD,BLANK,BLANK,BLANK][HEADER LENGTH]
BYTE 2                                                    BYTE 3
[OPTIONS]                                                 [OPTIONS]
```

Byte 0 consists of flags that tell us various types of packets that are being transferred. Byte 1 tells us the header length. Byte 2 and 3 onwards are all options. We will discuss the various options below:

SYN

This is the packet used by the client to initiate the three way handshake. It sets the syn flag in the flags byte and it puts in the start and end segment numbers of the entire transfer.

```
----SYN------
BYTE 0                BYTE 1
[SYN,0,0,0, 0,0,0,0][HEADER LENGTH]
BYTE 2                BYTE 3
[Start Segment]     [start segment]
BYTE 4                BYTE 5
[End Segment]       [End Segment]
```

SYN-ACK

This is the packet sent by the client during the second step of the three way handshake

```
----SYN,ACK------
BYTE 0                        BYTE 1
[SYN,0,0,ACK, 0,0,0,0]         [HEADER LENGTH]
BYTE 2                        BYTE 3
[CLIENT Start Segment]      [CLIENT start segment]
BYTE 4                        BYTE 5
[CLIENT End Segment]        [CLIENT End Segment]
BYTE 6                        BYTE 7
[ACK SERVER Start Segment] [ACK SERVER Start segment]
BYTE 8                        BYTE 9
[ACK SERVER End Segment]    [ACK SERVER End segment]
```

ACK

This packet is sent by the client to acknowledge start and end segment numbers sent by the client. This is done as the third step of the three way handshake.

```
----ACK------
BYTE 0                 BYTE 1
[0,0,0,ACK, 0,0,0,0][HEADER LENGTH]
BYTE 2                 BYTE 3
[ACK Start Segment] [ACK start segment]
BYTE 4                 BYTE 5
[ACK End Segment]    [ACK End Segment]
```

I

NACK

When the client does not receive a segment number, it sends a NACK packet in the following format:

```
----NACK------
BYTE 0                        BYTE 1
[0,0,NACK,0, 0,0,0,0]         [HEADER LENGTH]
BYTE 2                        BYTE 3
[NACK Start Segment]           [NACK start segment]
```

DATA

Data is sent by the server in the following format:

```
---DATA---
BYTE 0                          BYTE 1
[0,DATA,0,0, 0,0,0,0]           [HEADER LENGTH]
BYTE 2                          BYTE 3
[DATA Start Segment]            [DATA start segment]
BYTE 4                          BYTE 5
[DATA End Segment]              [DATA End Segment]
BYTE 6                          BYTE 7
[Segment Number]                [Segment Number]
[DATA.....................................]
```

ALL_RCVD

This sent by the client once all the packets are received on the client's end.

```
----ALL_RCVD------
BYTE 0                              BYTE 1
[0,0,0,0, ALL_RCVD,0,0,0]           [HEADER LENGTH]
```

Timeout

During the three second handshake phase if the packet one of the three packets do not arrive at the right end, there will be a time out which ends the connection.

Congestion Control

Considering the number of packets sent back and forth are less as there are no ack packets for each segment sent. There are lesser number of packets transferred and hence there is no need for a specific congestion control mechanism.

Error Handling

This protocol tunnels packets through UDP and considering UDP and IP both have checksums, the design of the protocol takes it consideration and believes two layers performing error checks is enough to handle error corrections.