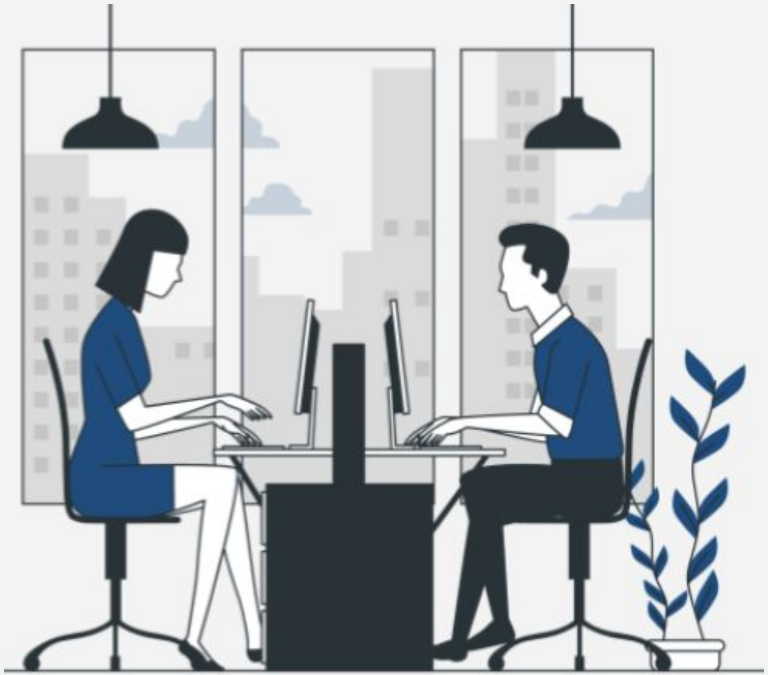
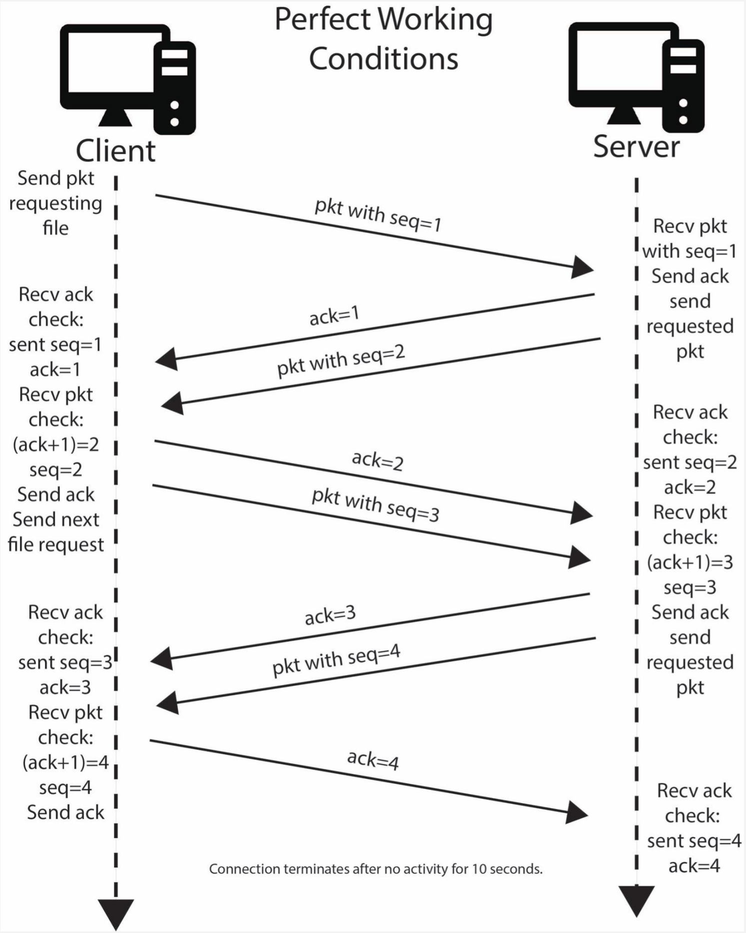
Reliable File Transfer

Livia Rose and Matt Getgen



RFT Specification Perfect Case

1. Client sends packet (with sequence number) request via port number.
   a. If no response, resend after 2 seconds
   b. If no response after 8 retries, terminate connection
2. Server receives packet request because it is always listening on the port number.
   a. If request doesn't exist, send error and terminate connection.
3. Server puts the received sequence number into an acknowledgement message, and sends the acknowledgment to the client.
4. Server also sends the requested file (with sequence number = received sequence number + 1) to the client.
5. The client receives the acknowledgment from the server, then checks to see if the previous packet sequence number is equal to the acknowledgment message.
   a. If equal, the next packet is sent.
   b. If not equal, the packet is dropped.
6. Client puts the received sequence number into an acknowledgement message, and sends the acknowledgment to the server.
7. The client also sends the next file request (with a sequence number = received sequence number + 1).
8. The process above repeats until either the client or the server terminates the connection.



Perfect Working Conditions

Client

Server

Send pkt requesting file

pkt with seq=1

Recv pkt with seq=1 Send ack send requested pkt

ack=1

pkt with seq=2

Recv ack check: sent seq=1 ack=1 Recv pkt check: (ack+1)=2 seq=2 Send ack Send next file request

ack=2

pkt with seq=3

Recv ack check: sent seq=2 ack=2 Recv pkt check: (ack+1)=3 seq=3 Send ack send requested pkt

ack=3

pkt with seq=4

Recv ack check: sent seq=3 ack=3 Recv pkt check: (ack+1)=4 seq=4 Send ack

ack=4

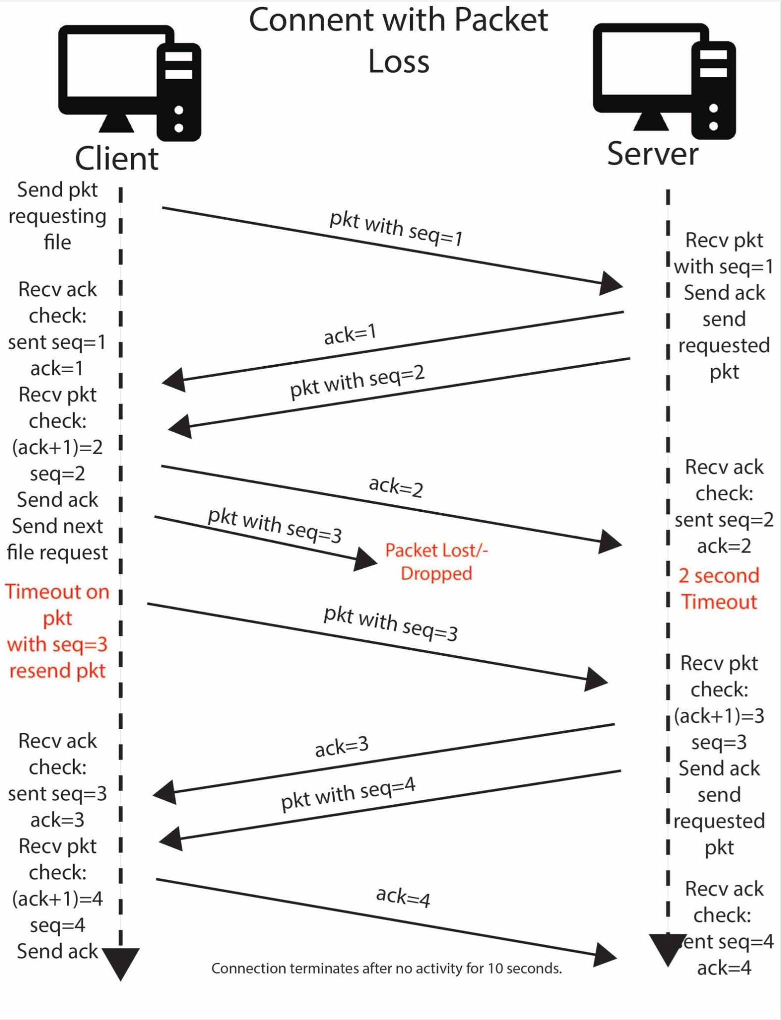
Recv ack check: sent seq=4 ack=4

Connection terminates after no activity for 10 seconds.

RFT Specification Packet Loss

To overcome packet loss we make use of a two second timer that when "timed out" will trigger the client to resend the previous packet.

If no response occurs happens more than eight times the client will assume the connection has been lost and the connection will be terminated.



Connent with Packet Loss

Client

Server

Send pkt requesting file

pkt with seq=1

Recv pkt with seq=1 Send ack send requested pkt

ack=1

pkt with seq=2

Recv ack check: sent seq=1 ack=1 Recv pkt check: (ack+1)=2 seq=2 Send ack Send next file request

ack=2

Recv ack check: sent seq=2 ack=2

pkt with seq=3

Packet Lost/-Dropped

2 second Timeout

Timeout on pkt with seq=3 resend pkt

pkt with seq=3

Recv pkt check: (ack+1)=3 seq=3 Send ack send requested pkt

ack=3

pkt with seq=4

Recv ack check: sent seq=3 ack=3 Recv pkt check: (ack+1)=4 seq=4 Send ack

ack=4

Recv ack check: sent seq=4 ack=4

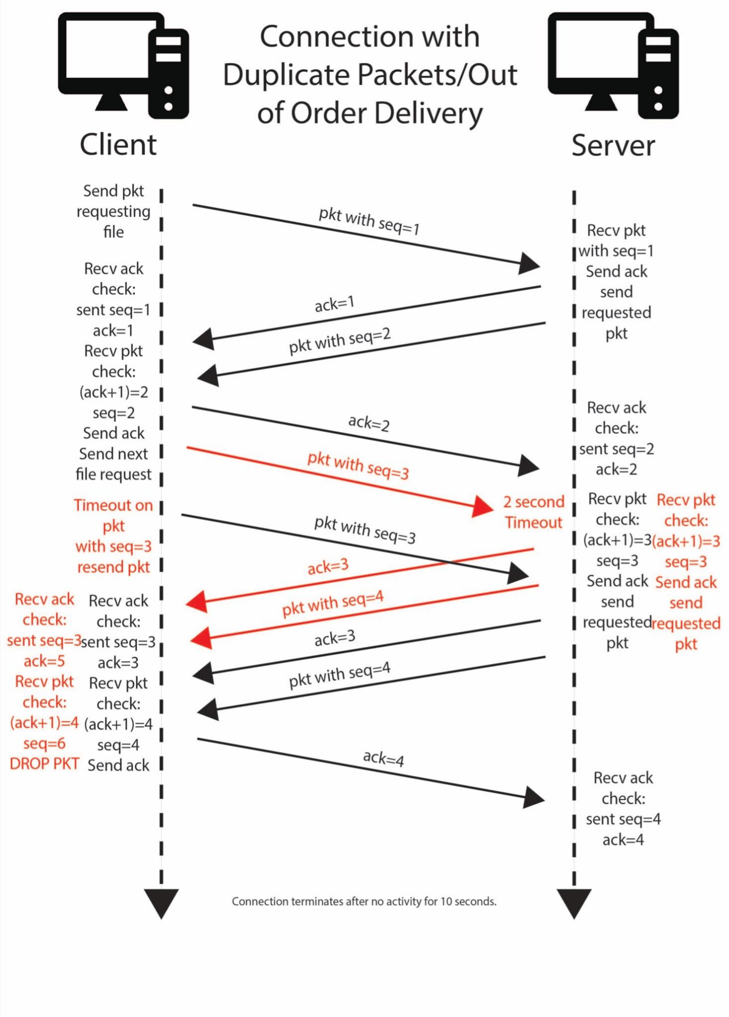
Connection terminates after no activity for 10 seconds.

RFT Specification Order Issues

To overcome loss, delay, duplicates, and other delivery issues, we make use of sequence numbers in tandem with acknowledgements.

In the case of delay, the client may send a request twice and the server will receive both requests. The server will then send file and acknowledgment twice. During the client's checking process, the duplicate packet will be found and dropped.

If a packet is dropped, the client will wait 2 seconds before resending the request to wait for the correct packet. If the correct packet does not come in 2 seconds the request is resent.


Connection with Duplicate Packets/Out of Order Delivery
Client
Server
Send pkt requesting file
pkt with seq=1
Recv pkt with seq=1 Send ack send requested pkt
Recv ack check: sent seq=1 ack=1
ack=1
pkt with seq=2
Recv pkt check: (ack+1)=2 seq=2 Send ack Send next file request
ack=2
Recv ack check: sent seq=2 ack=2
pkt with seq=3
2 second Timeout
Timeout on pkt with seq=3 resend pkt
pkt with seq=3
Recv pkt check: (ack+1)=3 seq=3 Send ack send requested pkt
Recv pkt check: (ack+1)=3 seq=3 Send ack send requested pkt
ack=3
pkt with seq=4
ack=3
pkt with seq=4
Recv ack check: sent seq=3 ack=5 Recv pkt check: (ack+1)=4 seq=6 DROP PKT
Recv ack check: sent seq=3 ack=3 Recv pkt check: (ack+1)=4 seq=4 Send ack
ack=4
Recv ack check: sent seq=4 ack=4
Connection terminates after no activity for 10 seconds.

Server Sudo Code

01. headers

Next slide.

02. send_acknowledgment

Creates and sends a ACK header.

03. Wait_for_acknowledgment

Handles the 2 second timer, 8 retries, and checks the sequence number against the acknowledgment message.

04. send_error_message

Creates and sends a ERR header.

05. send_file

Writes data into a buffer and sends the buffer.

06. handle_connection

Here is where the sequence number is stored, and where send_acknowlegment, send_file, and send_error_message are called.

07. main

Here is where the socket is opened and is actively listening.

Sudo Code Headers

SEQ Packet Header:

0 1 2 3	4 5 6 7	8 9 10 11	...
SEQ STR	SEQ NUM	DATA SIZE	DATA
4 bytes	4 bytes	4 bytes	? bytes

ERR Packet Header:

0 1 2 3	4 5 6 7	8 9 10 11	...
ERR STR	NULL NUM	DATA SIZE	ERR MSG
4 bytes	4 bytes	4 bytes	? bytes

ACK Packet Header:

0 1 2 3	4 5 6 7
ACK STR	ACK NUM
4 bytes	4 bytes

Client Sudo Code

01. headers

Headers are the same as Server Sudo Code:

02. send_acknowledgment

Send_acknowledgment is the same as Server Sudo Code:

03. Wait_for_acknowledgment

Wait_for_acknowledgment is the same as Server Sudo Code:

06. handle_connection

Here is where the sequence number is stored, file requests are sent, and acknowledgments are received and sent.

07. main

Here is where the socket is opened and the connection is handled.