Multi-threaded downloader for parallel data download based on User Datagram Protocol (UDP)

Cohort 1 Team 1

Team Members :

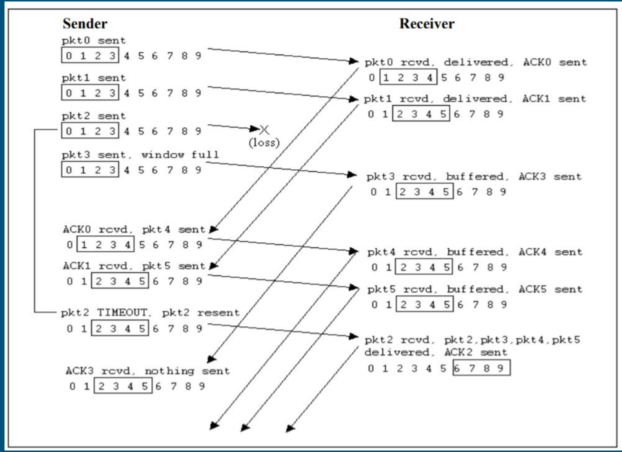
Kimberlyn Loh (1002221)
Tan Yi Long (1001566)
Chen Wenshu (1002291)
Delbert Felix Nurawan (1002374)

General Idea

- ❖ TCP comes at a costly trade off of significant latency due to connection setup and congestion control mechanism.
- ❖ UDP does a simple transmission model without implicit handshaking technique but is unreliable and datagram may arrive out of order, duplicated or missing
- ❖ Multithreaded downloading utilizes more bandwidth with more number of connections to server.
- ❖ Networking topics :
  - ➢ Service provided by transport layer,
  - ➢ Segmentation and reassembly
  - ➢ Congestion control

Reliable UDP

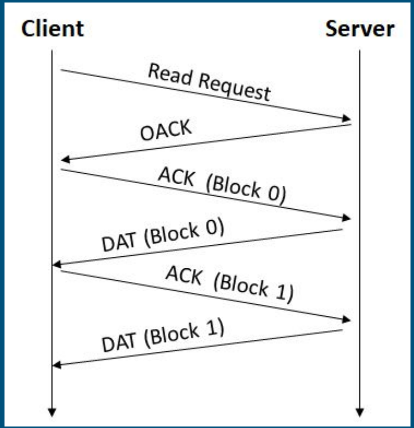
1) Reliable User Datagram Protocol (RUDP)
   - ❖ A transport layer protocol where reliability is achieved using the Selective Repeat version of the Sliding Window protocol.
   - ❖ Utilises acknowledgements and timeouts to ensure reliable data transfer



Reliable UDP

2) Trivial File Transfer Protocol (TFTP)

- ❖ A simple protocol used to transfer files.
- ❖ Each nonterminal packet is acknowledged separately. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent.

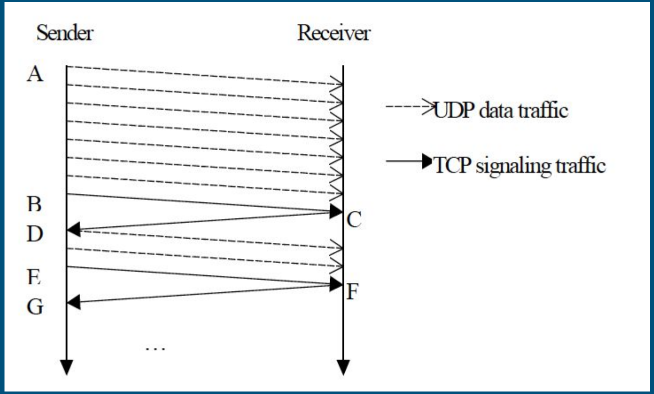


Reliable UDP

3) Reliable Blast UDP (RBUDP)

❖ Data transport tool and protocol specifically designed to move very large files over wide area high-speed networks

❖ Key features: i) keep the network pipe as full as possible during bulk data transfer and ii) avoid TCP's per-packet interaction from ACK



Efficiency (experiment procedure)

Experiment conducted on each protocol, on a single thread, to determine best performance in terms of throughput and packet loss.

1. Setup server and client end-points.
   Server contains 2 files to be downloaded by client, test.txt (1 kb) and testHuge.txt (109 kb).

2. Client requests test.txt from server and downloads it. Record time taken for download and number of packets lost.

3. Repeat download process in step 2 for testHuge.txt.
4. Record time taken for download and number of packets lost.

Efficiency (experiment result)

Protocol	Average Time Taken/ s	Packet Loss
RUDP	Small file : 0.03	No
	Large file : 3.19	No
TFTP	Small file: 0.01	No
	Large file: 1.03	No
RBUDP	Small file: 0.04	No
	Large file: 0.14	No
TCP	Small file: 0.01	No
	Large file: 0.07	No

Multi-threaded downloader

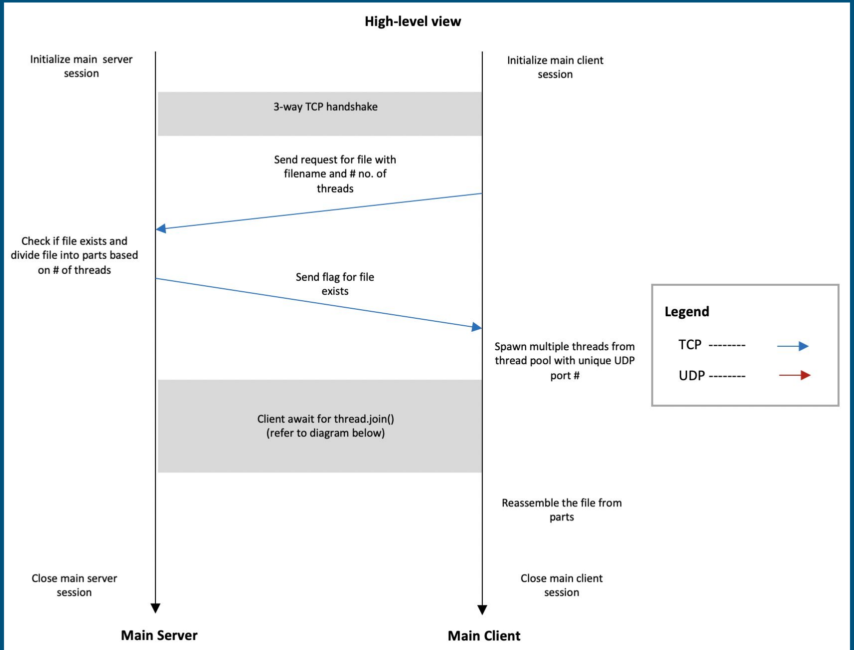
- Manage parallel download and merge file parts in proper order on the client side
- Downloader uses TCP connections to communicate between server and client for access control and UDP connections to transfer file based on RBUDP protocol

Multi-threaded downloader

1. The client first sends over the number of threads and the name of the file to download through TCP.

2. The server receives and divides the file into smaller parts accordingly based on the number of threads, as illustrated in Fig. 3.1.3 (E.g. if there are 4 threads, the file will be divided into 4 parts). It then sends over the flag that indicates whether the file exists on the server.

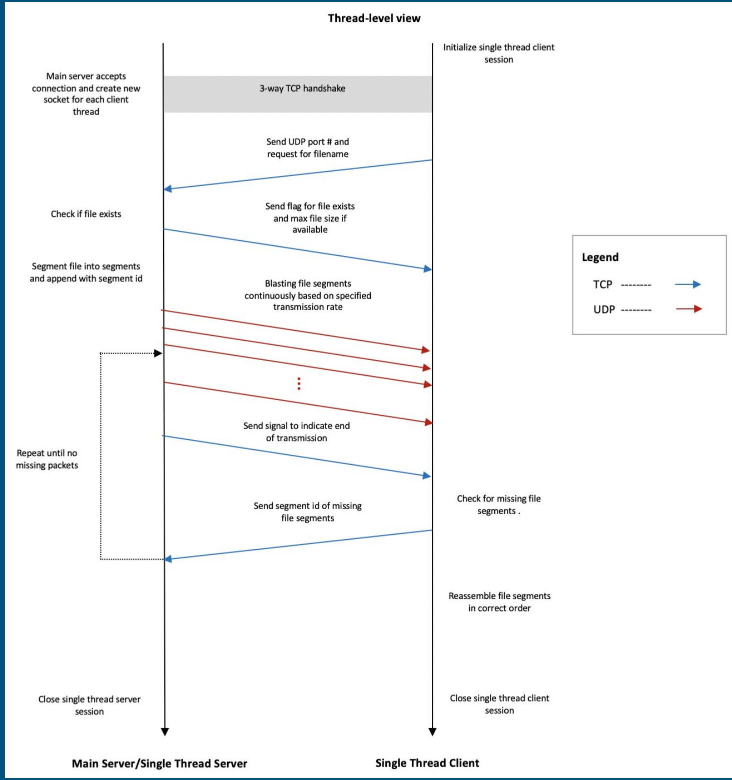
3. The client creates threads from a thread pool. Each thread creates a TCP socket that establishes a connection with the TCP socket on the server side.



Multi-threaded downloader

4. Upon receiving an incoming TCP connection, the server will create a new TCP socket and spawn a new thread to handle each client thread.

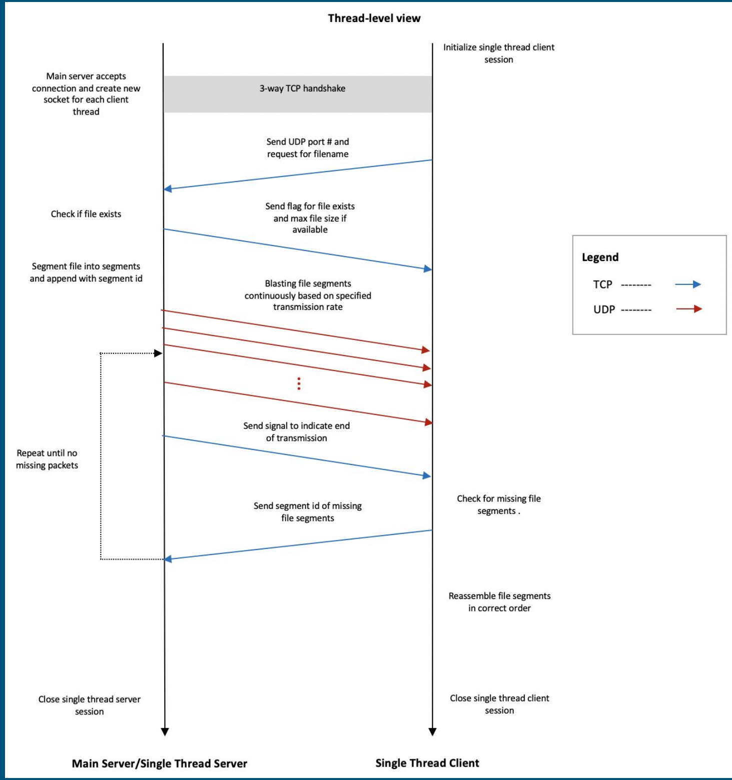
5. The client's thread then creates a UDP socket and sends over the UDP port number that this thread will use for receiving the file packets through its TCP connection. It also sends over the agreed filename of the particular part to download through the TCP connection.



Multi-threaded downloader

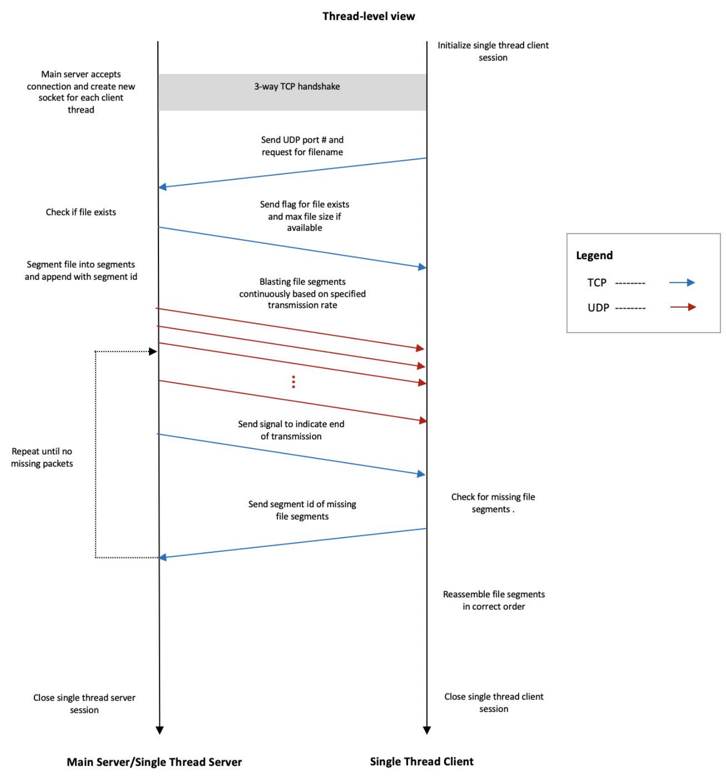
6. The server's thread will send over a flag to indicate whether the file segment exists on the server for error detection and the max file size of the file segment.

7. Each server's thread will segment the file part, as illustrated in Fig. 3.1.3 and append the segment id. It then sends segments to the client's thread through the UDP connection at the user-specified rate. Once the transmission is done, the server's thread signals the client's thread through the TCP connection using a 'DONE' message.



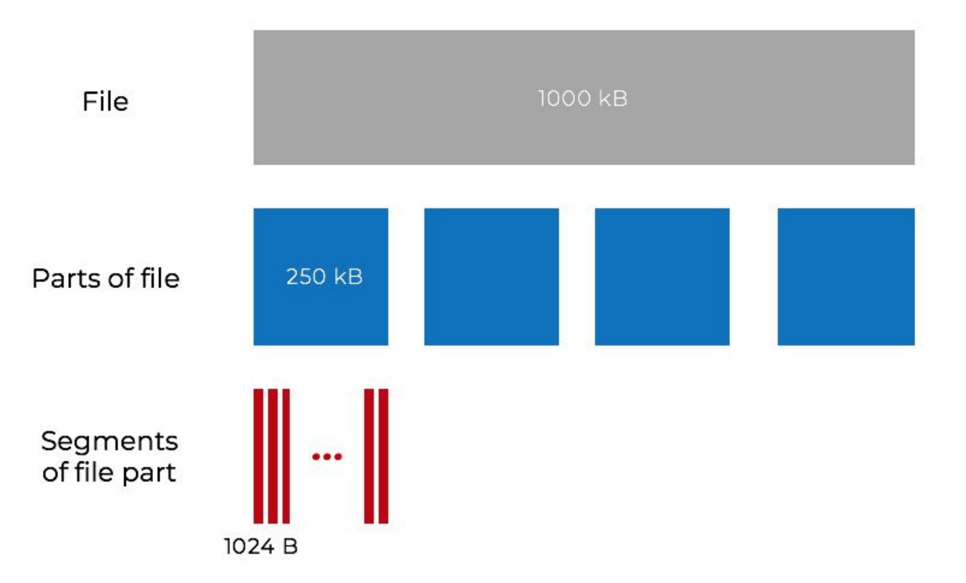
Multi-threaded downloader

8. The client's thread receives segments of the file part, and upon receiving the 'DONE' signal, it checks for missing file segments. If there are missing segments, it will send the segment id of the missing segments and step 7 will be repeated until there is no missing segments. It then assembles the file segments and saves them into a single file part and the client's thread will close.
9. Finally, the client combines parts of the file in sequence and obtains the complete file.



Multi-threaded downloader

File Segmentation details



Demonstration

Ok boss!

Give me
some 50.012
notes

Server

Client

Thank You