TRUSTED TRANSPORT PROTOCOL (TTP)
&
740 FILE TRANSFER PROTOCOL (740FTP)

DESIGN DOCUMENT

By Ramya Balaraman(rbalaram)
& Venkatesh Sriram (vsriram)

CONTENTS

INSTRUCTIONS TO RUN THE PROGRAM

1. Configure Run Configuration arguments of FTPClient & FTPServer programs to accept window size and retransmission timer interval (in milliseconds) as first and second arguments respectively. Currently these parameters have been set to 20 and 7000 respectively.
2. FTPServer has been hardcoded to listen on Port 2221 for testing purposes as port 21 was already bound and could not be used. Modify this port number in FTPServer.java if required.
3. FTPClient has been hardcoded with source port as 2000. Modify this port number if required in FTPClient.java. Also modify the destination IP if it is not localhost.
4. Run FTPServer.java
5. Run **multiple** FTPClient.java on different machines. (Note: Modify the Destination IP if running the client from a different machine) You will be prompted to enter the file name on the console window. Enter any file name from the following:
    a. Sample1.txt
    b. Sample2.rtf
    c. DesignDoc.docx
    d. DesignDoc.pdf
    e. Smallest.gif
    f. Small.gif
    g. Medium.jpg
    h. ClassDiagram.png

    Note: Large files like DesignDoc.docx, DesignDoc.pdf take a considerable time to transfer especially with our version of DatagramService (which includes Test Cases). Please be patient ☺
6. The minimum and maximum delay introduced in the tests is 7.5 and 15 seconds respectively. We have considered the retransmission timer interval to be less than 15seconds during our testing. Please change delay according to timer interval if needed.

7. Note: At times, due to delay and dropped packets introduced by the test cases, the client program repeatedly sends the same acknowledgement (of the last packet correctly received) and it appears like the program is stuck. But please be patient, it will proceed to completion. Test with smaller files like Sample1.txt or Small.gif in this case to test the logic.

EXPECTED OUTPUT (without tests)

Example 1: Without fragmentation/reassembly (File size in this example: 411 bytes)

[Note: Max data bytes in one packet is 1281 + 9 bytes header]
The file requested will be received in the **ClientFiles** folder.

FTP Client:

```
Enter file name
Sample2.rtf
SYN sent to 127.0.0.1:2221 with ISN 10904
Received SYNACK with seq no:46702 and Acknowledgement No 10904
Acknowledgement sent! No:46702

Establised connection to FTP Server at 127.0.0.1:2221
Data sent to 127.0.0.1:2221 with Seq No 10905
Received ACK for packet no:10905
CRC verified!!
Received data with Seq no 46703
Acknowledgement sent! No:46703
CRC verified!!
Received data with Seq no 46704
Acknowledgement sent! No:46704
FTP Client received file!
MD5 Hash verified!!
FIN sent! Seq No:10906
Received FINACK with seq no:46705
Acknowledgement sent for FINACK! No:46705
TTP Client closes connection!
```

FTP Server:

```
FTP Server is listening on Port 2221
Received SYN from:127.0.0.1:2000
SYNACK sent to 127.0.0.1:2000 with Seq no 46702
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:46702
FTP Server continues listening..
Received ACK/REQUEST from existing client
CRC verified!!
Received request from 127.0.0.1:2000
Acknowledgement sent! No:10905
Data written to TTPServer buffer
Client request passed to FTP
FTP Server received file request!
FTP Server continues listening..
Proxy FTP Server servicing FTP Client...
File Requested:Sample2.rtf
TTP Server received data from FTP
Data sent to 127.0.0.1:2000 with Seq No 46703
Received ACK/REQUEST from existing client
Received ACK for packet no:46703
TTP Server received data from FTP
FTP Server has sent file to TTP to send to FTP client!
Data sent to 127.0.0.1:2000 with Seq No 46704
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:46704
FTP Server continues listening..
Received ACK/REQUEST from existing client
FINACK sent to 127.0.0.1:2000
FTP Server continues listening..
Timeout for Packet 46705
Datagram resent!!
Connection 127.0.0.1:2000 closed at server !
Received ACK for packet no:46705
FTP Server continues listening..
```

Example 2: With fragmentation & reassembly (File size in this example: 40,280 bytes)

FTP Client:

```
Enter file name
Medium.jpg
SYN sent to 127.0.0.1:2221 with ISN 33319
Received SYNACK with seq no:8565 and Acknowledgement No 33319
Acknowledgement sent! No:8565

Establised connection to FTP Server at 127.0.0.1:2221
Data sent to 127.0.0.1:2221 with Seq No 33320
Received ACK for packet no:33320
CRC verified!!
Received data with Seq no 8566
Acknowledgement sent! No:8566
CRC verified!!
Received data with Seq no 8567
Acknowledgement sent! No:8567
Acknowledgement sent! No:8568
Acknowledgement sent! No:8569
Acknowledgement sent! No:8570
Acknowledgement sent! No:8571
Acknowledgement sent! No:8572
Acknowledgement sent! No:8573
Acknowledgement sent! No:8574
Acknowledgement sent! No:8575
Acknowledgement sent! No:8576
Acknowledgement sent! No:8577
Acknowledgement sent! No:8578
Acknowledgement sent! No:8579
Acknowledgement sent! No:8580
Acknowledgement sent! No:8581
Acknowledgement sent! No:8582
Acknowledgement sent! No:8583
Acknowledgement sent! No:8584
Acknowledgement sent! No:8585
Acknowledgement sent! No:8586
Acknowledgement sent! No:8587
Acknowledgement sent! No:8588
Acknowledgement sent! No:8589
Acknowledgement sent! No:8590
```

```
Acknowledgement sent! No:8591
Acknowledgement sent! No:8592
Acknowledgement sent! No:8593
Acknowledgement sent! No:8594
Acknowledgement sent! No:8595
Acknowledgement sent! No:8596
Acknowledgement sent! No:8597
Acknowledgement sent! No:8598
FTP Client received file!
MD5 Hash verified!!
FIN sent! Seq No:33352
Received FINACK with seq no:8599
Acknowledgement sent for FINACK! No:8599
TTP Client closes connection!
```

FTP Server:

```
FTP Server is listening on Port 2221
Received SYN from:127.0.0.1:2000
SYNACK sent to 127.0.0.1:2000 with Seq no 8565
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8565
FTP Server continues listening..
Received ACK/REQUEST from existing client
CRC verified!!
Received request from 127.0.0.1:2000
Acknowledgement sent! No:33320
Data written to TTPServer buffer
Client request passed to FTP
FTP Server received file request!
FTP Server continues listening..
Proxy FTP Server servicing FTP Client...
File Requested:Medium.jpg
TTP Server received data from FTP
Data sent to 127.0.0.1:2000 with Seq No 8566
Received ACK/REQUEST from existing client
Received ACK for packet no:8566
TTP Server received data from FTP
FTP Server has sent file to TTP to send to FTP client!
Data sent to 127.0.0.1:2000 with Seq No 8567
Data sent to 127.0.0.1:2000 with Seq No 8568
Data sent to 127.0.0.1:2000 with Seq No 8569
Data sent to 127.0.0.1:2000 with Seq No 8570
Data sent to 127.0.0.1:2000 with Seq No 8571
FTP Server continues listening..
Data sent to 127.0.0.1:2000 with Seq No 8572
Received ACK/REQUEST from existing client
Received ACK for packet no:8567
Data sent to 127.0.0.1:2000 with Seq No 8573
Data sent to 127.0.0.1:2000 with Seq No 8574
Data sent to 127.0.0.1:2000 with Seq No 8575
Data sent to 127.0.0.1:2000 with Seq No 8576
Data sent to 127.0.0.1:2000 with Seq No 8577
Data sent to 127.0.0.1:2000 with Seq No 8578
```

```
Data sent to 127.0.0.1:2000 with Seq No 8579
Data sent to 127.0.0.1:2000 with Seq No 8580
Data sent to 127.0.0.1:2000 with Seq No 8581
Data sent to 127.0.0.1:2000 with Seq No 8582
Data sent to 127.0.0.1:2000 with Seq No 8583
Data sent to 127.0.0.1:2000 with Seq No 8584
Data sent to 127.0.0.1:2000 with Seq No 8585
Data sent to 127.0.0.1:2000 with Seq No 8586
Data sent to 127.0.0.1:2000 with Seq No 8587
Send Window full! Packet 8588 queued!
Send Window full! Packet 8589 queued!
Send Window full! Packet 8590 queued!
Send Window full! Packet 8591 queued!
Send Window full! Packet 8592 queued!
Send Window full! Packet 8593 queued!
Send Window full! Packet 8594 queued!
Send Window full! Packet 8595 queued!
Send Window full! Packet 8596 queued!
Send Window full! Packet 8597 queued!
Send Window full! Packet 8598 queued!
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8588
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8589
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8568
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8590
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8569
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8591
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8592
```

```
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8570
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8593
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8571
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8594
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8595
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8572
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8596
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8573
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8597
Queued packet about to be sent!
Data sent to 127.0.0.1:2000 with Seq No 8598
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8574
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8575
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8576
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8577
FTP Server continues listening..
```

```
Received ACK/REQUEST from existing client
Received ACK for packet no:8578
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8579
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8580
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8581
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8582
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8583
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8584
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8585
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8586
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8587
FTP Server continues listening..
```

```
Received ACK for packet no:8589
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8590
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8591
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8592
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8593
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8594
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8595
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8596
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8597
FTP Server continues listening..
Received ACK/REQUEST from existing client
Received ACK for packet no:8598
FTP Server continues listening..
Received ACK/REQUEST from existing client
FINACK sent to 127.0.0.1:2000
FTP Server continues listening..
Timeout for Packet 8599
Datagram resent!!
Connection 127.0.0.1:2000 closed at server !
Received ACK for packet no:8599
FTP Server continues listening..
```

PROTOCOL SPECIFICATIONS

1. Initial Sequence Number (ISN):
   Sequence number is of data type int and can be a maximum of 4 bytes. The initial sequence number is a randomly generated value from 0 to 65535.

2. Connection Setup: 3-way handshake.
   TTP Client sends SYN, TTP Server responds with SYN ACK, TTP Client sends ACK

3. Payload contains 9 bytes of header

1	2	3	4	5	6	7	8	9

Sequence Number Acknowledgement No Header Flags

4. Sequence Number is assigned packet-wise, and not byte-wise as in TCP.

5. The Acknowledgement number denotes the sequence number of the last received packet, and **not** the byte value of the next byte expected, as in TCP.

6. The header structure of the payload is the following-

N/A	N/A	N/A	FINACKACK	EOF	SYN	ACK	FIN

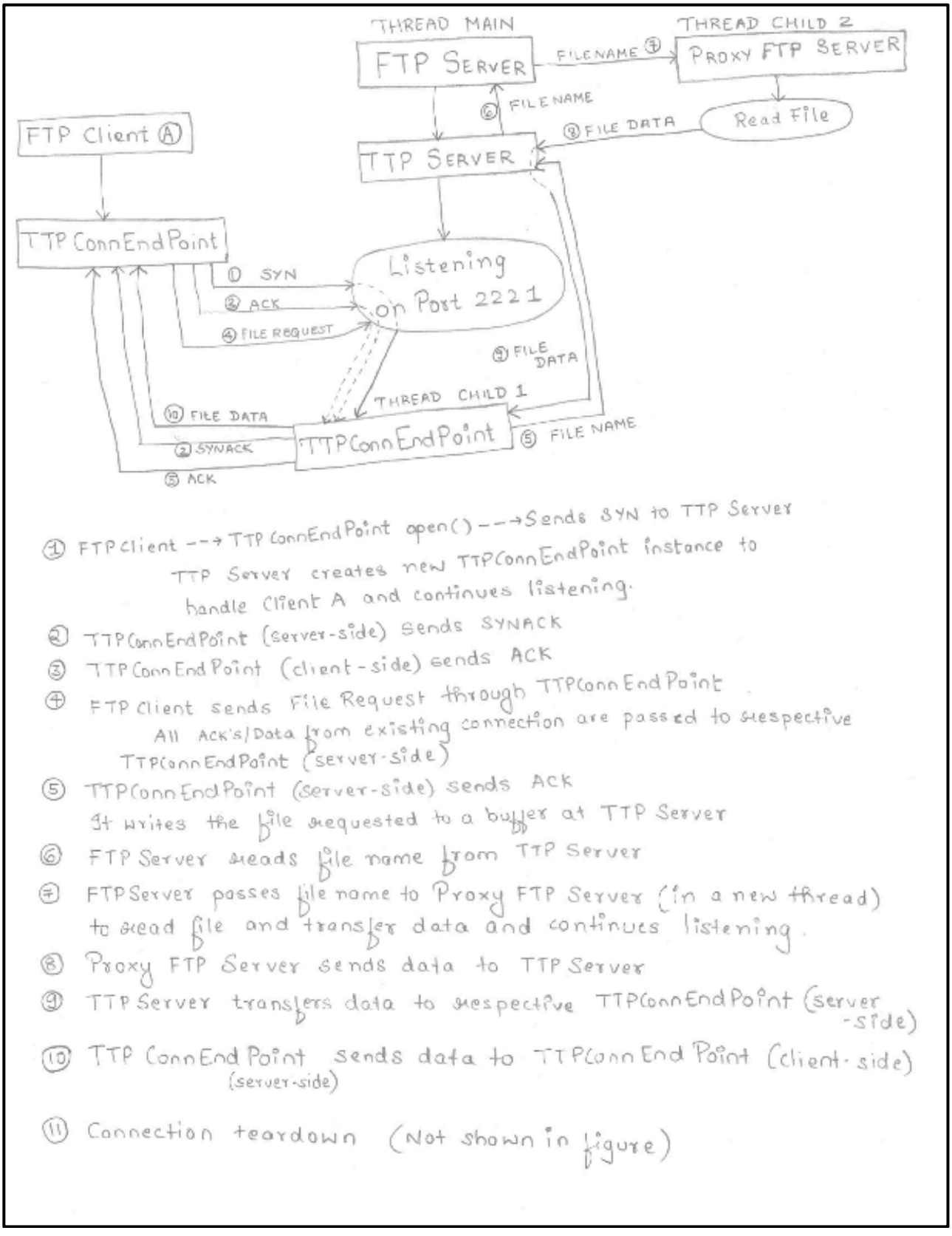
7. Maximum number of bytes in one packet is 1281 + 9 bytes for the header.

8. Fragmentation and reassembly is done using the EOF flag, which signifies the End of File marker.

9. The checksum is calculated over the payload using the same method as the UDP checksum. It is verified at the receiver by recalculating the checksum and comparing it with the value in the checksum field.

10. Connection teardown- A three step process is followed which is analogous to the three-way handshake for connection establishment.
    FIN from client, FINACK from server, FINACKACK from client

ARCHITECTURE



① FTPClient --→ TTP ConnEndPoint open() ---→ Sends SYN to TTP Server
TTP Server creates new TTPConnEndPoint instance to handle Client A and continues listening.

② TTPConnEndPoint (server-side) Sends SYNACK

③ TTPConnEndPoint (client-side) sends ACK

④ FTP Client sends File Request through TTPConnEndPoint
All ACK's/Data from existing connection are passed to respective TTPConnEndPoint (server-side)

⑤ TTPConnEndPoint (server-side) sends ACK
It writes the file requested to a buffer at TTP Server

⑥ FTP Server reads file name from TTP Server

⑦ FTPServer passes file name to Proxy FTP Server (in a new thread) to read file and transfer data and continues listening.

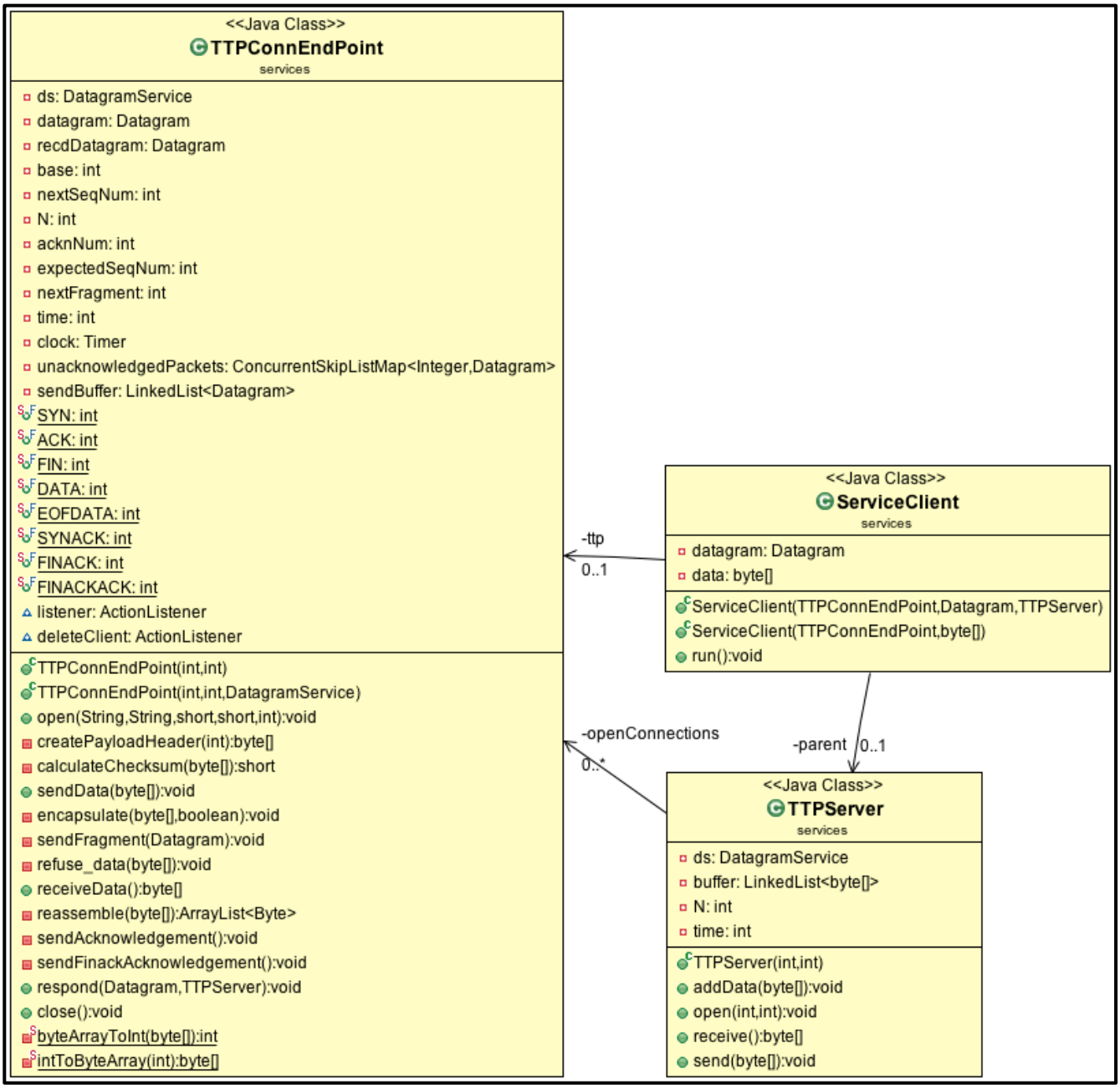
⑧ Proxy FTP Server sends data to TTP Server

⑨ TTP Server transfers data to respective TTPConnEndPoint (server-side)

⑩ TTP ConnEndPoint (server-side) sends data to TTPConnEndPoint (client-side)

⑪ Connection teardown (Not shown in figure)

CLASS DIAGRAM



NOTE:

TTPConnEndPoint class, representing an end point of a reliable connection, can be considered analogous to a Socket. It contains the entire logic of Go-Back-N mechanism.

The TTPServer listens for incoming SYN connection requests from clients who are running TTPConnEndPoint and creates a new instance of TTPConnEndPoint on the server side to handle each connection. The TTPServer thereafter continues listening for new connection requests and the actual work of interfacing with the client endpoint is handled by the server TTPConnEndPoint.

PROGRAM FLOW

1. The FTPServer is configured using Run Configuration to accept 2 command line arguments i.e. TTPServer send window size and TTPServer retransmission timer interval. FTPServer instantiates TTPServer using these parameters.

2. It calls open(int port, int verbose) on TTPServer instance which in turn initializes a DatagramService instance on the specified port.
   Note: FTP Server is hardcoded to listen on port 2221 in this case for testing purposes since FTP port 21 is already bound.

3. FTPServer listens continuously for file requests on Port 2221 by calling receive() on TTPServer instance in a while(true) loop. TTPServer receive() in turn calls receiveDatagram() of the underlying DatagramService instance.

4. FTPClient is configured using Run Configuration in Eclipse to accept 2 command line arguments i.e. send window size and retransmission timer interval.

5. It creates an instance of TTPConnEndPoint, which contains the entire logic for Go-Back-N mechanism. TTPConnEndPoint class, representing an end point of a reliable connection, can be considered analogous to a Socket.

6. FTPClient calls open(int port, int verbose) on this instance which in turn creates a DatagramService instance on the specified port. It also sends a SYN request to the TTPServer.

7. The TTPServer which is continuously listening on the specified port, creates a TTPConnEndPoint instance for every new incoming connection i.e. SYN. The client is then serviced in a separate thread by this instance. TTPServer maintains a ConcurrentSkipListMap of all active connections identified by client IP & port.

8. Note that this TTPConnEndPoint shares the same DatagramService instance as the TTPServer. The difference is that while the TTPServer continuously listens using receiveDatagram() of the underlying DatagramService, the TTPConnEndPoint instance implements responds to the received requests and ACK's using sendDatagram() of the same DatagramService instance albeit in a separate thread. Thus both listening for new connections and interacting with existing clients can happen simultaneously.

9. Thereafter, the TTPServer passes every message from an existing client to the respective TTPConnEndpoint instance for servicing in a new thread.

10. We perform a 3-way handshake to setup connection i.e. SYN from the TTP client, SYNACK from the TTP Server endpoint and ACK from the TTP client.

11. After establishing connection, FTP client sends a request for the file. This request is accepted as an input from the console on running the FTPClient program.

12. The listening TTPServer passes the request to the server endpoint servicing this particular client. If it falls within the receive window of the server endpoint, it acknowledges the packet and writes this request datagram to a buffer (implemented using a Queue) at TTPServer.

13. The FTPServer which is continuously listening through the receive() of TTPServer reads requests from this queue.

14. On receiving a request, the FTPServer launches a ProxyFTPServer instance in a new thread to read the file and transfer the bytes to the FTP client.

15. The ProxyFTPServer instance calls send(byte[]) of the TTPServer to send the file to the FTPClient. It also sends the MD5 hash of the file to the client.

16. The TTPServer in turn passes this file data to the TTPConnEndPoint instance interfacing with the particular client.

17. The FTP Client on receiving this data through its TTPConnEndPoint first verifies the MD5 hash of the file. On verification, the file is created within the ClientFiles folder in the project path.

18. The FTPClient closes the connection by calling close() on the TTPConnEndPoint instance after receiving a file correctly or receiving an error i.e. MD5 hash mismatch.

19. Connection teardown is a 3 step process – FIN from the TTP client endpoint to TTP server endpoint, FINACK from the TTP server endpoint and then ACK from the TTP client endpoint. The TTPServer deletes the entry in its ConcurrentSkipListMap corresponding to the particular connection on teardown. TTPClient endpoint on the other hand, sets its DatagramService instance to null, hence it can no longer send or receive data. The instance will soon be garbage collected.

FEATURES IMPLEMENTED CHECKLIST

1. Go Back N
2. TTP exposes the following functions to the application layer protocols-
   a) open()
   b) receive()
   c) send()
   d) close()
3. Robust against delayed packets, packet reordering, dropped packets and duplicate packets.
4. Fragmentation and Reassembly
5. UDP checksum
6. MD5 hash verification
7. Multi-threaded TTP (handles multiple connections simultaneously).
8. Test cases (Extra-credit)