Internet Technology

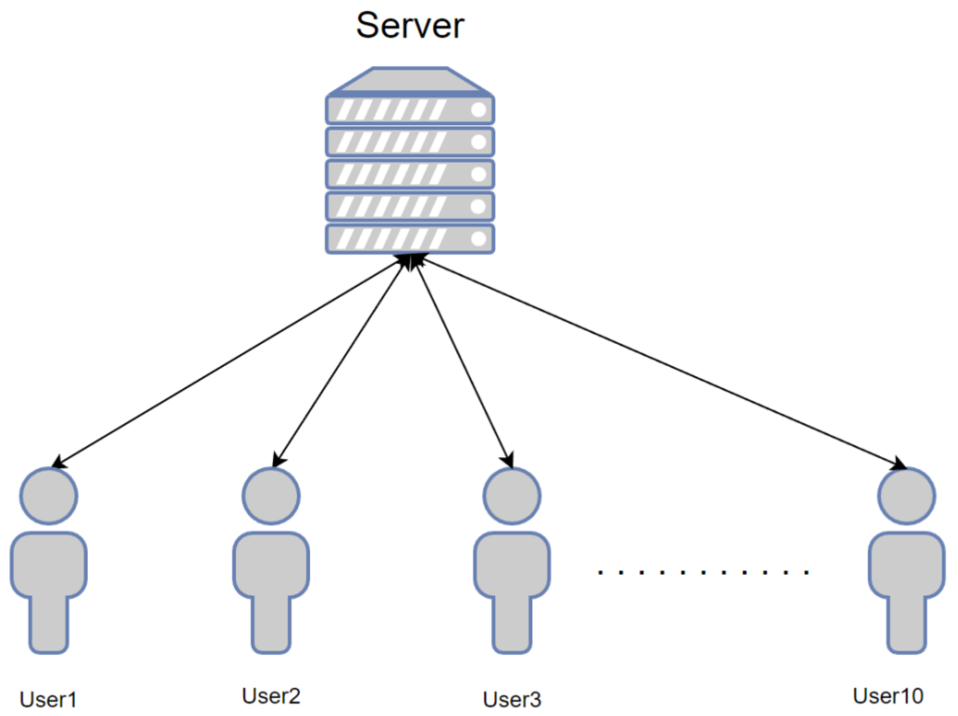
(Report)

Table of Contents

Chat Server Assignment

This application is a chat server, with functionality allowing for multiple users to be connected at once. The application was built using a server/client architecture, where all the clients would connect to a central server that would be responsible for exchanging information between the clients and managing the connections and users.

A protocol specification was also developed for the communication between the server and the clients.



The application is built on sockets, with the server also making use of serverSockets which will listen for requests from client sockets. Communications happens on port 1337 on the localhost IP address (127.0.0.1).

The server uses server sockets to listen to connections from the clients. It starts two threads for every client that connects to the server, one to listen to the client and respond or send messages to them, and another to send ping/pong messages to make sure the client is still connected (health check).

Every client also has two threads, one to listen(reader) to the server's responses/messages and one to send (writer) messages from the user to the server.

Both the server and the client use the java PrintWriter and BufferedReader classes to send and receive messages.

Functionality:

- Allow users to properly establish a connection
- List of connected users
- Broadcast messages to other users
- Privately message other users with encryption
- Share files between users
- Get a list of the connected users from the server
- Allow users to properly disconnect from the server
- Health check function that sends ping messages to make sure clients are connected
- Allows sending surveys to the users
- Encryption of all communication between two clients

Level 1:

Level 1 focuses on the basic functionality of the chat client, including the ability to join the server, broadcast messages to the other users, and quitting/disconnecting properly from the chat server. This functionality will be outlined in this section. Initially a node js server was used to test the clients connection which was developed in this project before moving to creating the server in java too.

The server thread that is responsible for handling client requests waits for requests, once a request is received from the client it processes the request based on the protocol specification, by retrieving the protocol from the request message.

Then a switch statement was used to handle the protocol messages rather than an if statement for efficiency and cleanliness of the code. The functionality for each protocol was also written in separate functions for reusability and readability when necessary.

Level 2:

Level 2 focuses on implementing private messaging functionality and allowing the users to retrieve a list of connected clients.

The users must be able to send a command to the server which returns the full list of connected clients. A user will then be able to use a command that allows them to privately message another connected user through the server, without the messages appearing to other users.

These functionalities were implemented in the server and client switch statements.

Level 3:

This level adds one main feature which is the ability for users to send/transfer files to other users privately. The file would be transferred over a separate socket to allow the client to still be able to chat while it is being uploaded, a checksum (such as MD5) is also implemented to ensure the file is not corrupted, damaged, or modified in any way during the transfer. The file is also only transferred to the recipient after they confirm/acknowledge the transfer, this is important in the case of malicious files being shared to a user without them accepting the file.

Level 4:

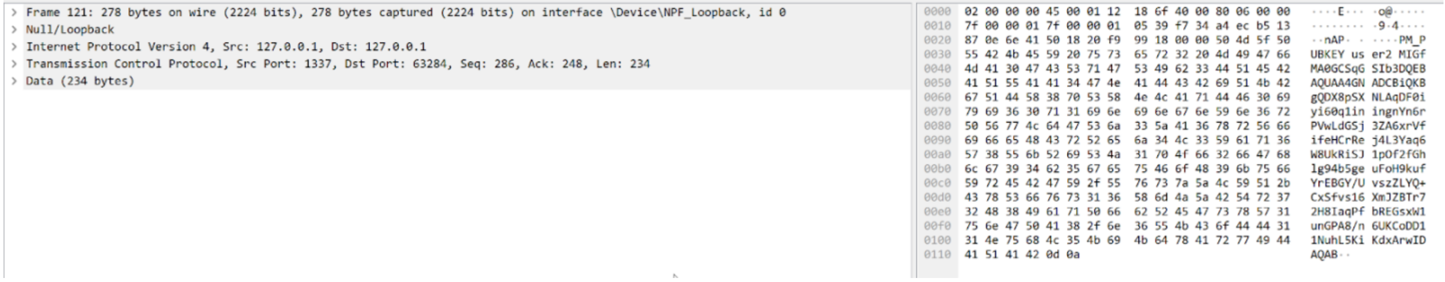
Level 4 adds encryption functionality to the chat server. When a user sends a private message to another user it must be encrypted to maintain the confidentiality of the message, without encryption anyone will be able to read the contents of the message. The implementation uses both symmetric(AES) and asymmetric encryption(RSA).

The client upon joining the server will generate a public and private key (asymmetric) and share the private key with the other clients. Then using a public key a session key can be randomly generated, encrypted, and shared with the recipient. This session key is then decrypted with the private key; now both clients have a session key (symmetric) and can use it to encrypt and decrypt messages instead of relying on the asymmetric encryption. This is advantageous because relying on symmetric encryption alone presents the issue of exchanging the encryption key securely, while relying on asymmetric encryption alone presents the issue of performance since encrypting a message with this method takes a lot of CPU time. Therefore combining both encryption methods is the best method.

After implementation the encryption was tested using wireshark to sniff and inspect the packets being sent over the network.

Wireshark screenshots:

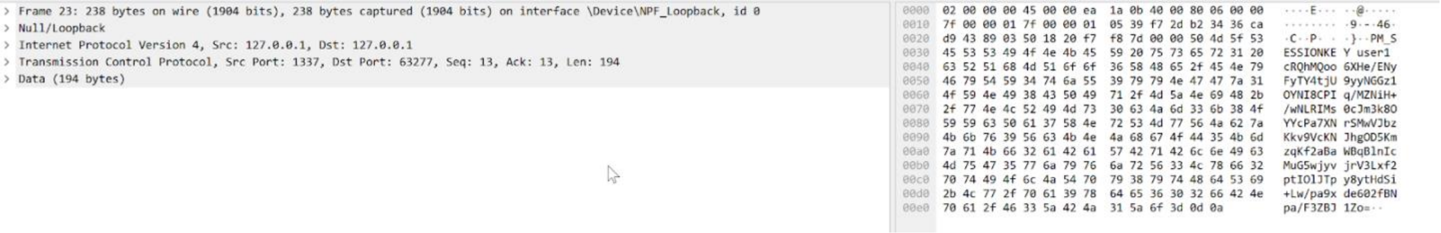
Public key exchange:



```
> Frame 121: 278 bytes on wire (2224 bits), 278 bytes captured (2224 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 1337, Dst Port: 63284, Seq: 286, Ack: 248, Len: 234
> Data (234 bytes)
```

```
0000  02 00 00 00 45 00 01 12  18 6f 40 00 80 06 00 00   ····E··· ·o@·····
0010  7f 00 00 01 7f 00 00 01  05 39 f7 34 a4 ec b5 13   ········ ·9·4····
0020  87 0e 6e 41 50 18 20 f9  99 18 00 00 50 4d 5f 50   ··nAP· · ····PM_P
0030  55 42 4b 45 59 20 75 73  65 72 32 20 4d 49 47 66   UBKEY us er2 MIGf
0040  4d 41 30 47 43 53 71 47  53 49 62 33 44 51 45 42   MA0GCSqG SIb3DQEB
0050  41 51 55 41 41 34 47 4e  41 44 43 42 69 51 4b 42   AQUAA4GN ADCBiQKB
0060  67 51 44 58 38 70 53 58  4e 4c 41 71 44 46 30 69   gQDX8pSX NLAqDF0i
0070  79 69 36 30 71 31 69 6e  69 6e 67 6e 59 6e 36 72   yi60q1in ingnYn6r
0080  50 56 77 4c 64 47 53 6a  33 5a 41 36 78 72 56 66   PVwLdGSj 3ZA6xrVf
0090  69 66 65 48 43 72 52 65  6a 34 4c 33 59 61 71 36   ifeHCrRe j4L3Yaq6
00a0  57 38 55 6b 52 69 53 4a  31 70 4f 66 32 66 47 68   W8UkRiSJ 1pOf2fGh
00b0  6c 67 39 34 62 35 67 65  75 46 6f 48 39 6b 75 66   lg94b5ge uFoH9kuf
00c0  59 72 45 42 47 59 2f 55  76 73 7a 5a 4c 59 51 2b   YrEBGY/U vszZLYQ+
00d0  43 78 53 66 76 73 31 36  58 6d 4a 5a 42 54 72 37   CxSfvs16 XmJZBTr7
00e0  32 48 38 49 61 71 50 66  62 52 45 47 73 78 57 31   2H8IaqPf bREGsxW1
00f0  75 6e 47 50 41 38 2f 6e  36 55 4b 43 6f 44 44 31   unGPA8/n 6UKCoDD1
0100  31 4e 75 68 4c 35 4b 69  4b 64 78 41 72 77 49 44   1NuhL5Ki KdxArwID
0110  41 51 41 42 0d 0a                                  AQAB··
```

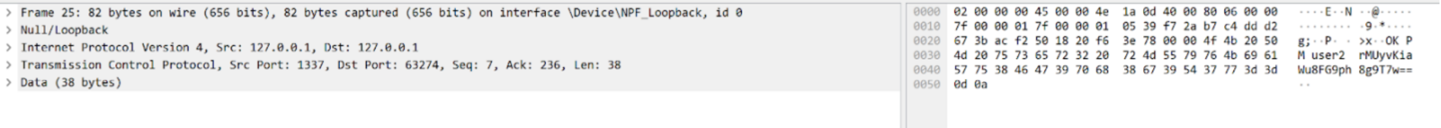
Encrypted session key exchange:



```
> Frame 23: 238 bytes on wire (1904 bits), 238 bytes captured (1904 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 1337, Dst Port: 63277, Seq: 13, Ack: 13, Len: 194
> Data (194 bytes)
```

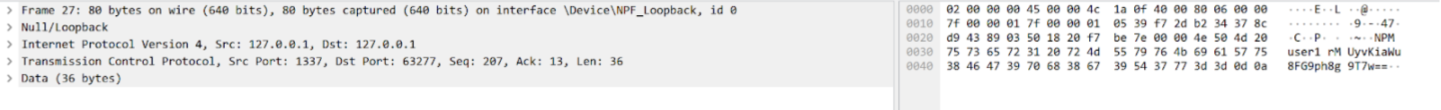
```
0000  02 00 00 00 45 00 00 ea  1a 0b 40 00 80 06 00 00   ····E··· ··@·····
0010  7f 00 00 01 7f 00 00 01  05 39 f7 2d b2 34 36 ca   ········ ·9·-·46·
0020  d9 43 89 03 50 18 20 f7  f8 7d 00 00 50 4d 5f 53   ·C··P· · ·}··PM_S
0030  45 53 53 49 4f 4e 4b 45  59 20 75 73 65 72 31 20   ESSIONKE Y user1
0040  63 52 51 68 4d 51 6f 6f  36 58 48 65 2f 45 4e 79   cRQhMQoo 6XHe/ENy
0050  46 79 54 59 34 74 6a 55  39 79 79 4e 47 47 7a 31   FyTY4tjU 9yyNGGz1
0060  4f 59 4e 49 38 43 50 49  71 2f 4d 5a 4e 69 48 2b   OYNI8CPI q/MZNiH+
0070  2f 77 4e 4c 52 49 4d 73  30 63 4a 6d 33 6b 38 4f   /wNLRIMs 0cJm3k8O
0080  59 59 63 50 61 37 58 4e  72 53 4d 77 56 4a 62 7a   YYcPa7XN rSMwVJbz
0090  4b 6b 76 39 56 63 4b 4e  4a 68 67 4f 44 35 4b 6d   Kkv9VcKN JhgOD5Km
00a0  7a 71 4b 66 32 61 42 61  57 42 71 42 6c 6e 49 63   zqKf2aBa WBqBlnIc
00b0  4d 75 47 35 77 6a 79 76  6a 72 56 33 4c 78 66 32   MuG5wjyv jrV3Lxf2
00c0  70 74 49 4f 6c 4a 54 70  79 38 79 74 48 64 53 69   ptIOlJTp y8ytHdSi
00d0  2b 4c 77 2f 70 61 39 78  64 65 36 30 32 66 42 4e   +Lw/pa9x de602fBN
00e0  70 61 2f 46 33 5a 42 4a  31 5a 6f 3d 0d 0a         pa/F3ZBJ 1Zo=··
```

Encrypted private message exchange: (decrypted message content was “hello”)



```
> Frame 25: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 1337, Dst Port: 63274, Seq: 7, Ack: 236, Len: 38
> Data (38 bytes)
```

```
0000  02 00 00 00 45 00 00 4e  1a 0d 40 00 80 06 00 00   ····E··N ··@·····
0010  7f 00 00 01 7f 00 00 01  05 39 f7 2a b7 c4 dd d2   ········ ·9·*····
0020  67 3b ac f2 50 18 20 f6  3e 78 00 00 4f 4b 20 50   g;··P· · >x··OK P
0030  4d 20 75 73 65 72 32 20  72 4d 55 79 76 4b 69 61   M user2  rMUyvKia
0040  57 75 38 46 47 39 70 68  38 67 39 54 37 77 3d 3d   Wu8FG9ph 8g9T7w==
0050  0d 0a                                              ··
```



```
> Frame 27: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 1337, Dst Port: 63277, Seq: 207, Ack: 13, Len: 36
> Data (36 bytes)
```

```
0000  02 00 00 00 45 00 00 4c  1a 0f 40 00 80 06 00 00   ····E··L ··@·····
0010  7f 00 00 01 7f 00 00 01  05 39 f7 2d b2 34 37 8c   ········ ·9·-·47·
0020  d9 43 89 03 50 18 20 f7  be 7e 00 00 4e 50 4d 20   ·C··P· · ·~··NPM
0030  75 73 65 72 31 20 72 4d  55 79 76 4b 69 61 57 75   user1 rM UyvKiaWu
0040  38 46 47 39 70 68 38 67  39 54 37 77 3d 3d 0d 0a   8FG9ph8g 9T7w==··
```