CS 360 Spring 2012   PUIM Command Line Client and Library   **Assignment Two**
Feb 20, 2012
DUE: Feb 22, 2012, 2:15 pm   Protocol State Diagrams (Client/Server)
DUE: March 5, 2012, 11:59pm   50 points

For this project, you need to produce a Java command line chat client that will work with the instructor's server. You must produce code that will be easy to share with an Android client and Java server in the future. I recommend that shared code be placed in a project **libPUIM_PUNetID** that each application can import and use.

There is no C server for this project, only a Java server.

The PUIM (Pacific University Instant Message) is written on top of TCP streams. The protocol is similar to SMTP or HTTP. There is no explicit packet being transferred between client and server like there is with DNS or the Math Client/Server. All data is transferred as Java Strings. No interoperability with C is provided or expected.

PUIM Version 1
The protocol messages are specified below.

Each line of text is a maximum of 1024 characters including the \n.
Each line must be terminated by a \n.
Lines marked with C are produced by the client. Lines marked with S are produced by the server. Neither C nor S are transmitted.

For ERR messages, any string after the ERR is optional.

Session Startup.
```
C HELLO username
S OK | ERR Username already in use
```

Session Disconnect
```
C DISCONNECT
S OK | MSG FROM:username
```

Send a message to another user.
```
C MSG TO:username
S OK | ERR Username not found | MSG FROM:username
C <TEXT terminated with single . on a line>
C .
S OK
```

Receive a message from another user.
```
S MSG FROM:username
C OK
S <TEXT terminated with single . on a line>
S .
C OK
```

CONFLICTS

If a server and client both initiate the send of a message the CLIENT must back off and accept the server's message by sending an OK. The client then attempts to resend its message after the server completes its transaction.

If the client requests to DISCONNECT but receives MSG FROM:username rather than OK, the client must receive the full message and after the final OK then try again to DISCONNECT. Once a server has received DISCONNECT (even if that DISCONNECT is interrupted by a message from the server) no new messages can be sent to the client.

Example:
```
C MSG TO:doug
S MSG FROM:shereen
C OK
S <TEXT terminated with single . on a line>
S .
C OK
C MSG TO:doug
S OK
C <TEXT terminated with single . on a line>
C .
S OK
```

The Command Line Client

The command line client must take three command line options: server address, server port, and username.   The user can type two commands. EXIT cleanly terminates the connection and exits the client. SEND username message, will send a message to user username. While the user is typing at the console, any incoming messages must also be displayed. Note below that you should be able to send messages to yourself. Bold text is text produced by incoming messages.

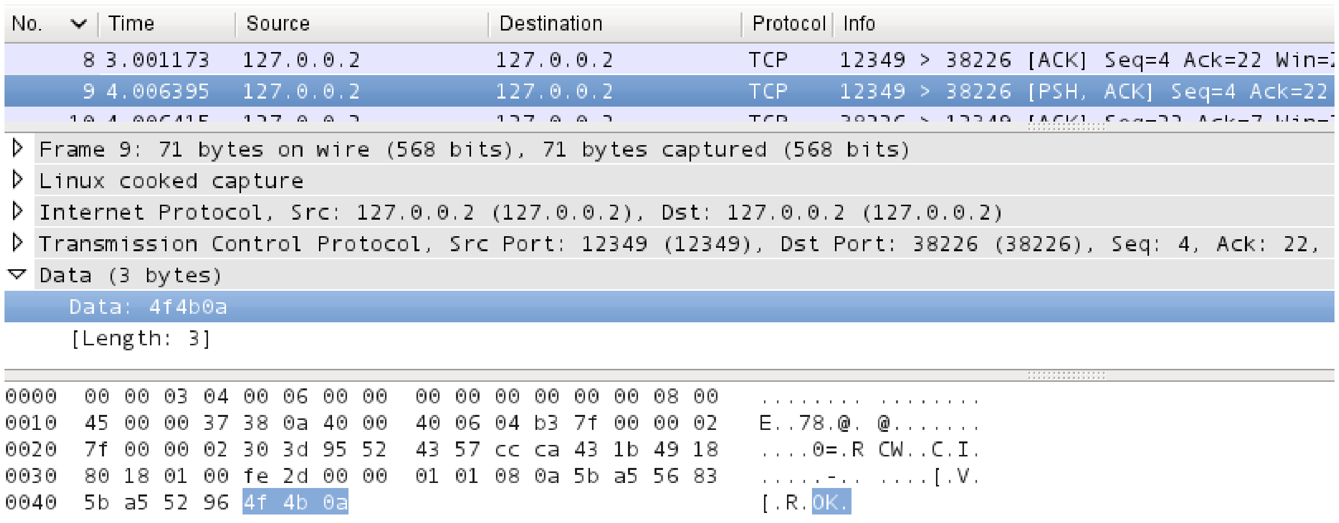
```
bart$ java -jar PUIMCommandLineClient.jar coffee.cs.pacificu.edu 12349 chadd

> SEND chadd This is the message
chadd >> This is the message
> EXIT

bart$
```

State Diagrams
You must produce a state diagram for both the client and server. The diagram must be produced electronically (Visio, PowerPoint, Libre/OpenOffice, Google Drawing, etc). Share a copy of the electronic drawing (PDF) with profchadd@gmail.com and bring a printout to class on Wednesday.

Notes:
The protocol will be changing over the next few assignments. The next assignment is an Android client. It would be useful to write your code in such a way that the Threads or Client could be reusable for Android.

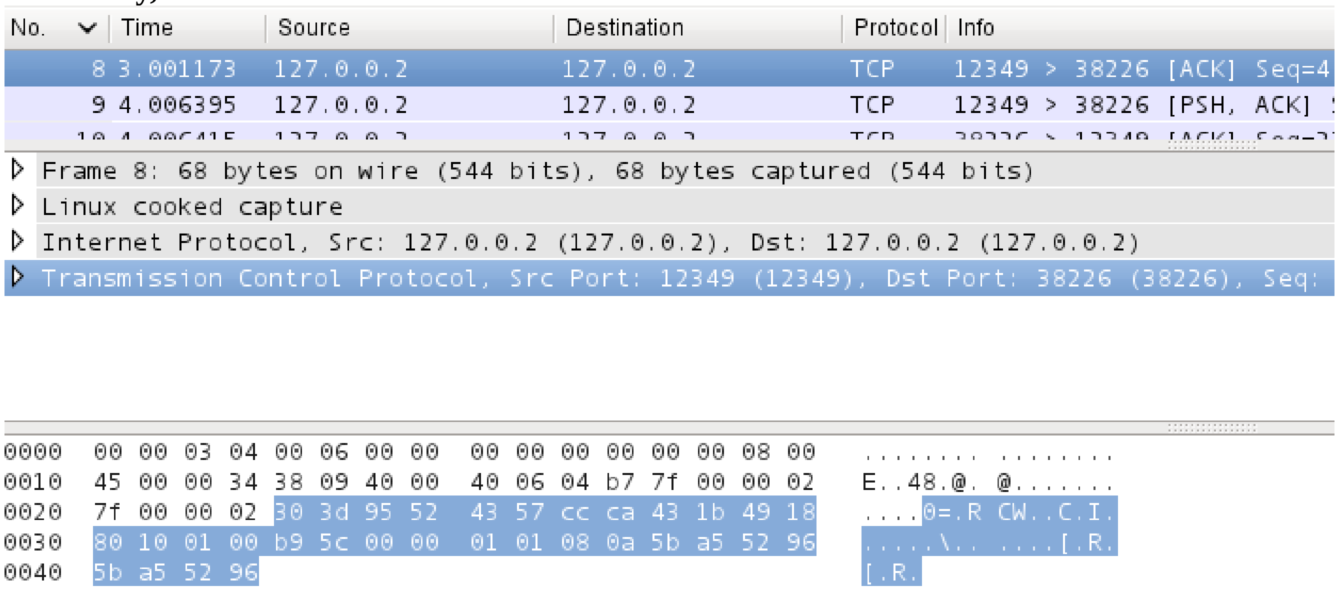
Wireshark: You will be able to see this traffic via Wireshark. You will see TCP protocol packets. Some packets will be marked with **ACK** and appear to be empty.

ACK and Data.



```
No.   Time        Source      Destination   Protocol  Info
  8 3.001173   127.0.0.2   127.0.0.2     TCP       12349 > 38226 [ACK] Seq=4 Ack=22 Win=
  9 4.006395   127.0.0.2   127.0.0.2     TCP       12349 > 38226 [PSH, ACK] Seq=4 Ack=22
 10 4.006415   127.0.0.2   127.0.0.2     TCP       38226 > 12349 [ACK] Seq=22 Ack=7 Win=

▷ Frame 9: 71 bytes on wire (568 bits), 71 bytes captured (568 bits)
▷ Linux cooked capture
▷ Internet Protocol, Src: 127.0.0.2 (127.0.0.2), Dst: 127.0.0.2 (127.0.0.2)
▷ Transmission Control Protocol, Src Port: 12349 (12349), Dst Port: 38226 (38226), Seq: 4, Ack: 22,
▽ Data (3 bytes)
    Data: 4f4b0a
    [Length: 3]

0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 00 00 37 38 0a 40 00  40 06 04 b3 7f 00 00 02   E..78.@. @.......
0020  7f 00 00 02 30 3d 95 52  43 57 cc ca 43 1b 49 18   ....0=.R CW..C.I.
0030  80 18 01 00 fe 2d 00 00  01 01 08 0a 5b a5 56 83   .....-.. ....[.V.
0040  5b a5 52 96 4f 4b 0a                               [.R.OK.
```

ACK Only, no data:



```
No.   Time        Source      Destination   Protocol  Info
  8 3.001173   127.0.0.2   127.0.0.2     TCP       12349 > 38226 [ACK] Seq=4
  9 4.006395   127.0.0.2   127.0.0.2     TCP       12349 > 38226 [PSH, ACK]
 10 4.006415   127.0.0.2   127.0.0.2     TCP       38226 > 12349 [ACK] Seq=2

▷ Frame 8: 68 bytes on wire (544 bits), 68 bytes captured (544 bits)
▷ Linux cooked capture
▷ Internet Protocol, Src: 127.0.0.2 (127.0.0.2), Dst: 127.0.0.2 (127.0.0.2)
▷ Transmission Control Protocol, Src Port: 12349 (12349), Dst Port: 38226 (38226), Seq:

0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 00 00 34 38 09 40 00  40 06 04 b7 7f 00 00 02   E..48.@. @.......
0020  7f 00 00 02 30 3d 95 52  43 57 cc ca 43 1b 49 18   ....0=.R CW..C.I.
0030  80 10 01 00 b9 5c 00 00  01 01 08 0a 5b a5 52 96   .....\.. ....[.R.
0040  5b a5 52 96                                        [.R.
```

Server:

A simple server will be provided on Wednesday. You can run this server as follows:

```
bart$ java -jar PUIMSimpleServer.jar edu.pacificu.cs.cs360.PUIM.PUIMServer 12349
```

Eclipse Project

Name your Eclipse projects **CS360_PUIM_CommandLineClient_PUNetID** and **libPUIM_PUNetID.**