Ewha Tic-Tac-Toe Protocol (ETTTP) : Request Message

- ETTTP request message:
  - Send my current move information to the peer host
    - Host: peer IP address
    - New-Move: my new move at row #1, col #2

[@192.168.0.1]

```
SEND ETTTP/1.0 \r\n
Host: 192.168.0.2 \r\n
New-Move: (1, 2) \r\n
\r\n
```

Ewha Tic-Tac-Toe Protocol (ETTTP) : Response Message

- ETTTP response message:
  - Acknowledge what I received to the peer host
    - Host: peer IP address
    - New-Move: peer's move at row #1, col #2

[@192.168.0.2]

```
ACK ETTTP/1.0 \r\n
Host: 192.168.0.1 \r\n
New-Move: (1, 2) \r\n
\r\n
```

Ewha Tic-Tac-Toe Protocol (ETTTP) : Result Poll Message

- ETTTP Result poll message:
  - Poll the results
  - Each peer sends its own decision for winner based on the Tic-Tac-Toe rule
  - Ex) In case that user A (192.168.0.1) wins

```
RESULT ETTTP/1.0 \r\n
Host: 192.168.0.2 \r\n
Winner: ME \r\n
\r\n
```

```
RESULT ETTTP/1.0 \r\n
Host: 192.168.0.1 \r\n
Winner: YOU \r\n
\r\n
```

Ewha Tic-Tac-Toe Protocol (ETTTP) : Request Message

- ETTTP request message @server:
  - Randomly choose one to decide the first mover
  - Send this decision to the peer host
    - Host: peer IP address
    - New-Move: my new move at row #1, col #2

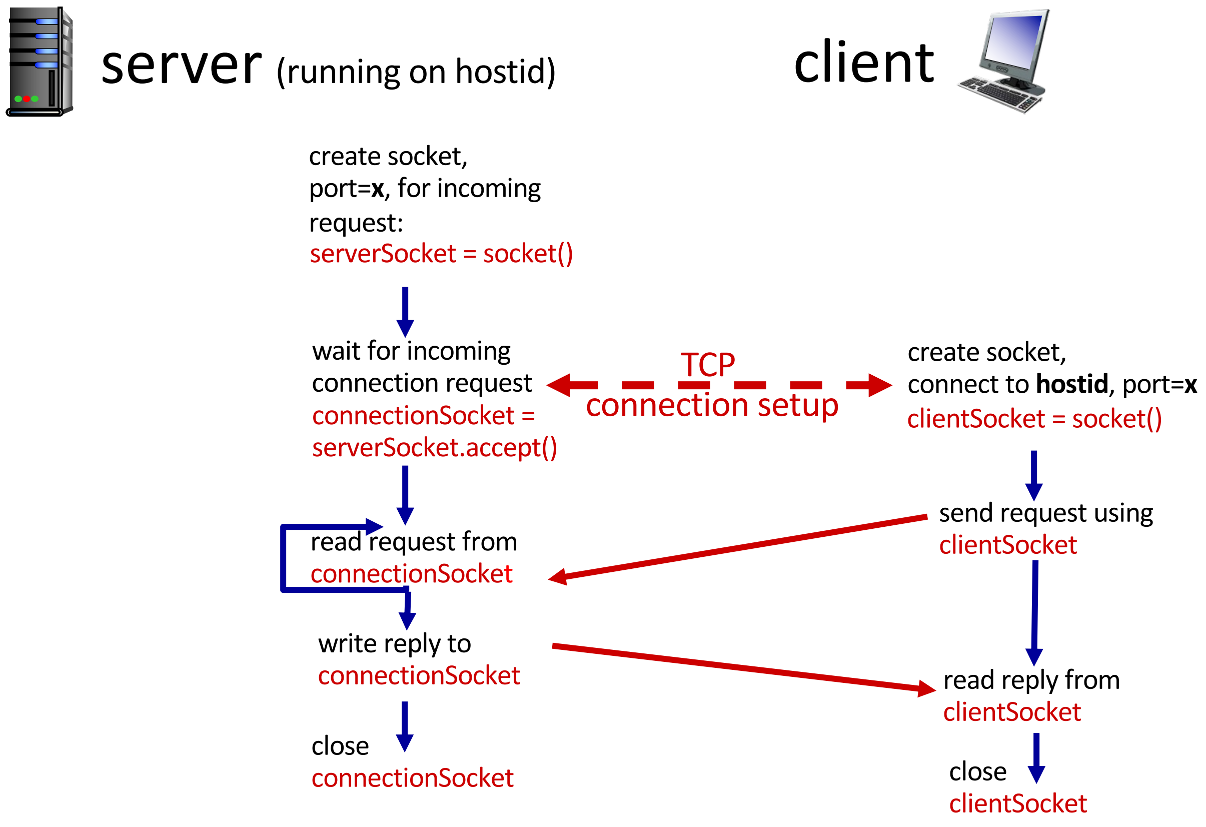
[@192.168.0.2]

```
SEND ETTTP/1.0 \r\n
Host: 192.168.0.1 \r\n
First-Move: YOU \r\n
\r\n
```

Procedure

- 1) Client-Server TCP Connection
  - Port number: 12000
- 2) Once TCP connection is established, open the GUI window at each client and server
- 3) Server randomly selects who is going to be the first mover (Mark: X) and shares with the client
- 4) Only the user with the correct turn can click on a certain area
  - Invalid user's input should neither change the board status nor send any message to the peer
- 5) Each client or server needs to continously check whether the game is over
- 6) Then, the result is shared with the peer
- 7) Both server and client double-check if the results are same, and then, the game is over

Client/server socket interaction: TCP



server (running on hostid)
client
create socket, port=x, for incoming request:
serverSocket = socket()
wait for incoming connection request
connectionSocket = serverSocket.accept()
TCP connection setup
create socket, connect to hostid, port=x
clientSocket = socket()
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

Example app: TCP client

Python TCPClient

```
from socket import *
serverName = ’servername’
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input(‘Input lowercase sentence:’)
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print (‘From Server:’, modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Example app: TCP server

Python TCPServer

create TCP welcoming socket → serverSocket = socket(AF_INET,SOCK_STREAM)

server begins listening for in coming TCP requests → serverSocket.listen(1)

loop forever → while True:

server waits on accept() for incoming requests, new socket created on return → connectionSocket, addr = serverSocket.accept()

read bytes from socket (but not address as in UDP) → sentence = connectionSocket.recv(1024).decode()

close connection to this client (but *not* welcoming socket) → connectionSocket.close()

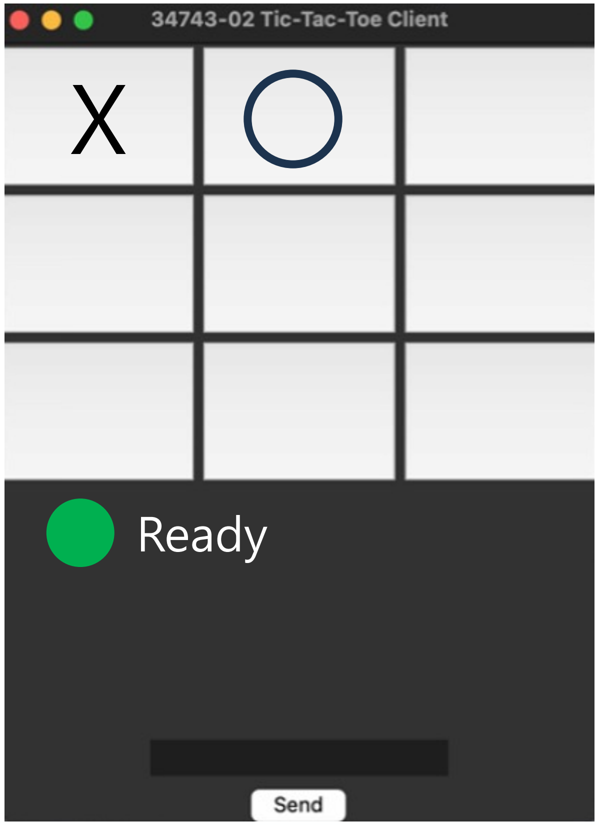
```
 from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
print ‘The server is ready to receive’
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                             encode())
    connectionSocket.close()
```

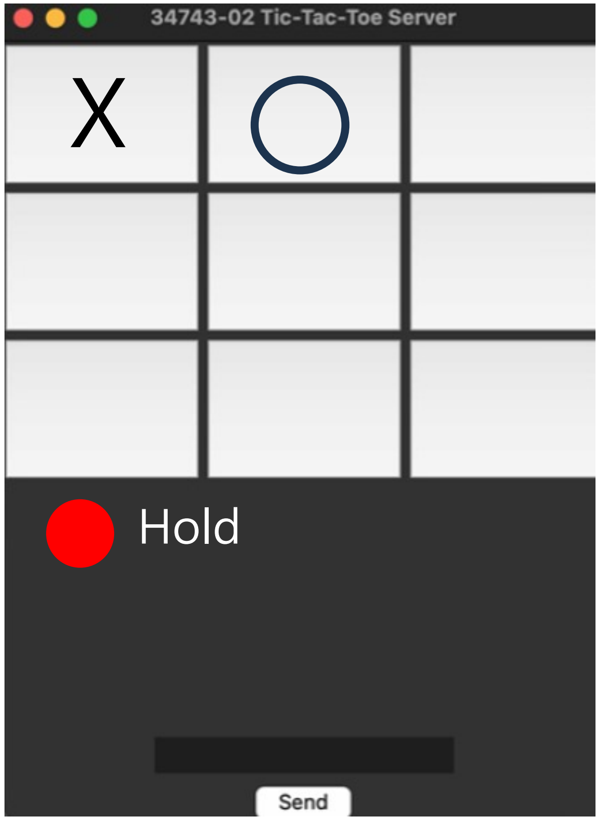
Tkinter (Python's standard GUI)

In case that the client was the first mover

- Client (Mark: X)

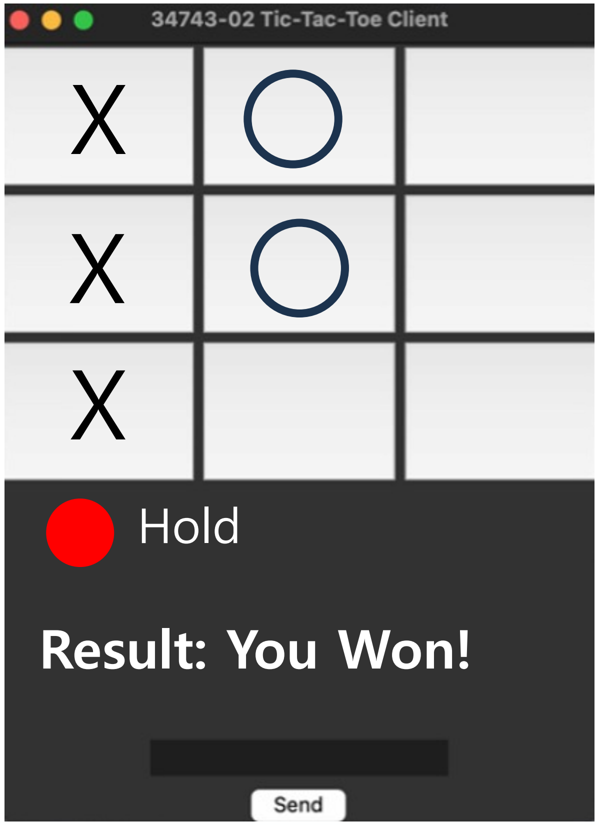


- Server (Mark: O)



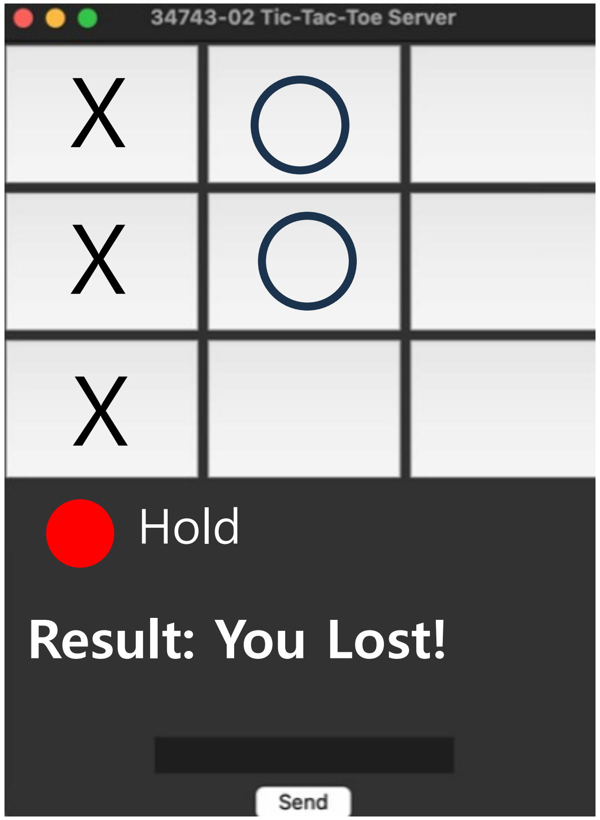
Tkinter (Python's standard GUI)

- Client (Mark: X)



Result: You Won!

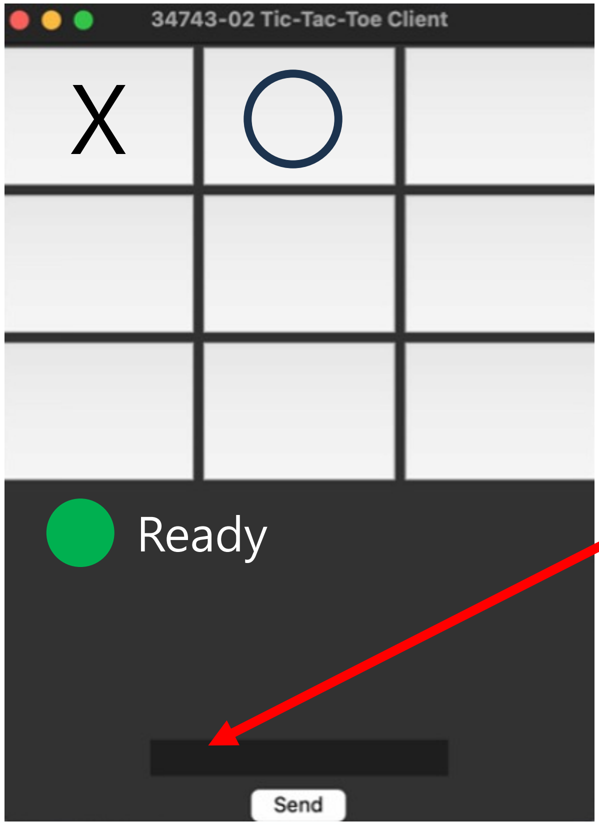
- Server (Mark: O)



Result: You Lost!

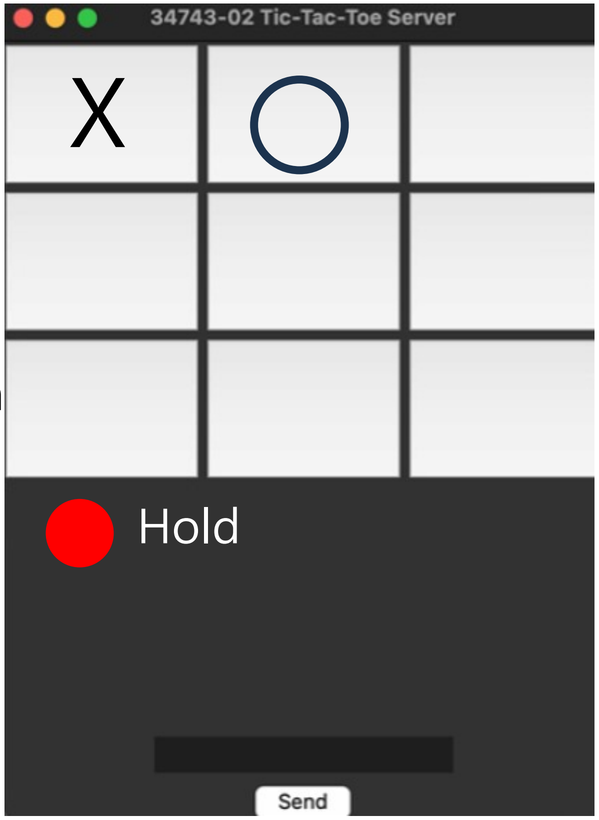
Command Mode (Debugging Mode)

- Client (Mark: X)



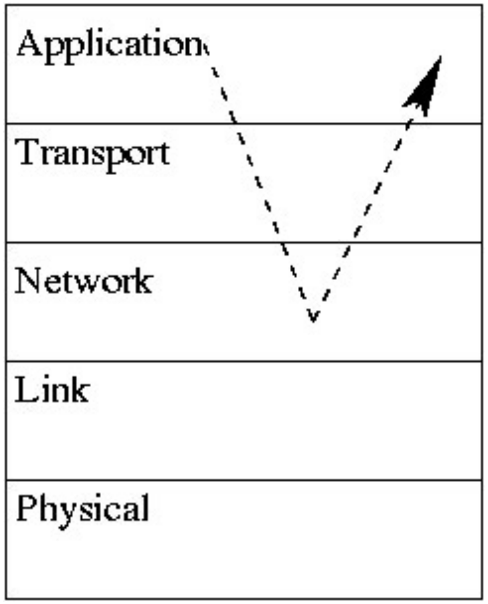
You can directly type or paste an ETTTP message right here! & click on “Send” button!

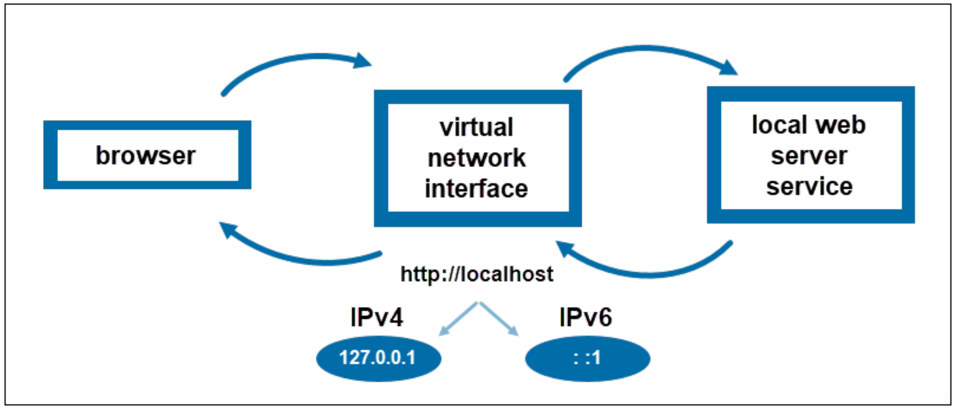
- Server (Mark: O)



Loopback Test in a Single Machine

- Loopback address can be used at both client and server for your test
  - IP Address: 127.0.0.1
  - One machine running two programs
    - 1 python program as client
    - 1 python program as server





Project Grade Guideline

- 1. Accuracy (60%)
  - Try out many different inputs by your own beyond the given input files, and show how the results from your implementation are correct
- 2. Code Analysis and Explanation in report (30%)
- 3. Whether your codes gave informative comments (10%)