CG4002: Computer Engineering Capstone Project

External Communications: Secure wireless Internet communications

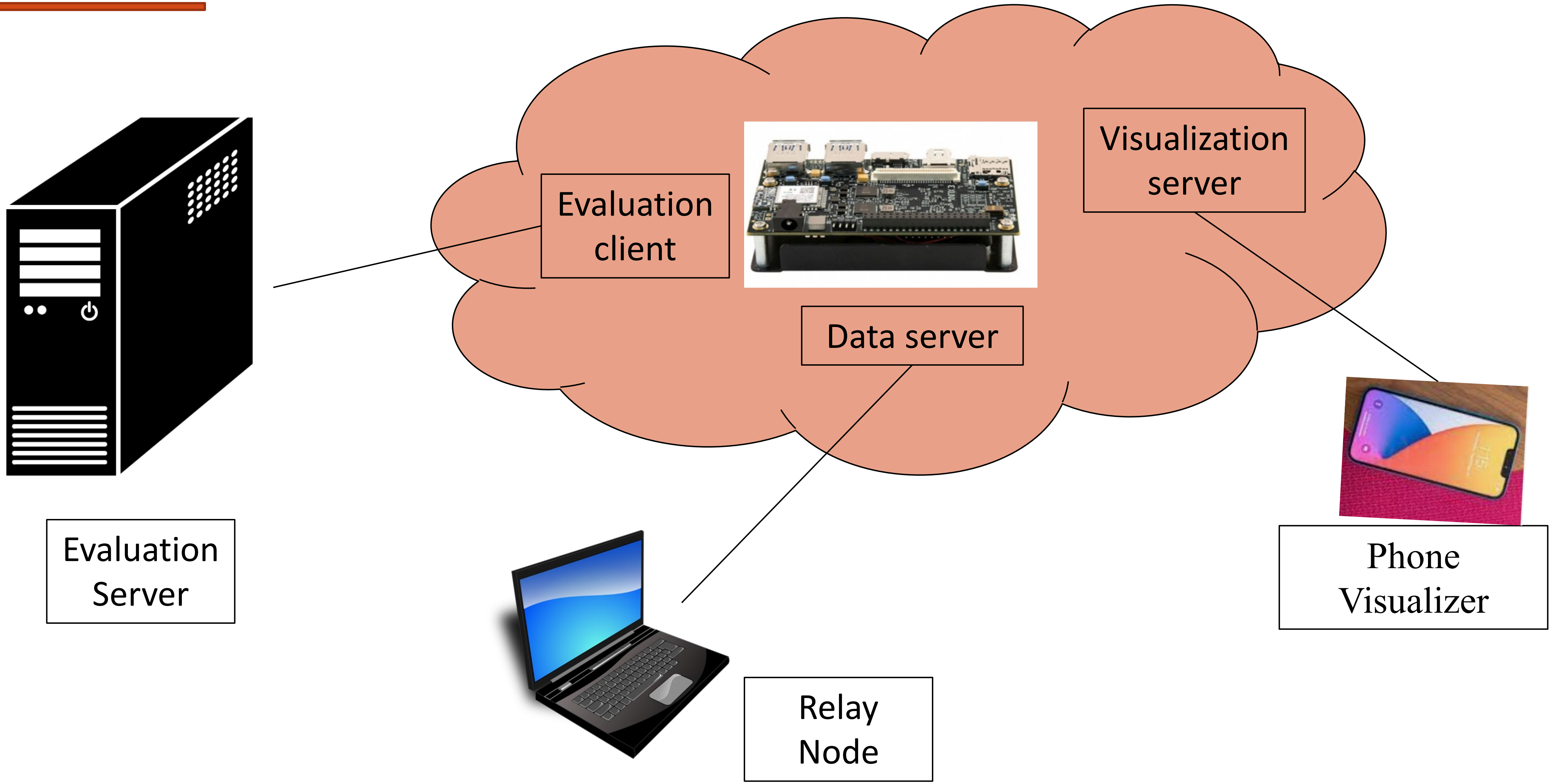
JITHIN VACHERY

jithin@comp.nus.edg.sg

Slides adopted from
Prof. Peh Li Shiuan

Comms External:

Comms2: External:
Internet comms



Evaluation client

Visualization server

Data server

Evaluation Server

Relay Node

Phone Visualizer

Secure wireless communications between system and server

Processes

In C/S model

server process waits to be contacted

client process initiates the communication

Process: program running within a host.

- Within the same host, two processes communicate using inter-process communication (IPC) (defined by OS).
  - You would need to handle
    - Producer-Consumer problem
    - Race conditions, etc
- Processes in different hosts communicate by exchanging messages (according to protocols).

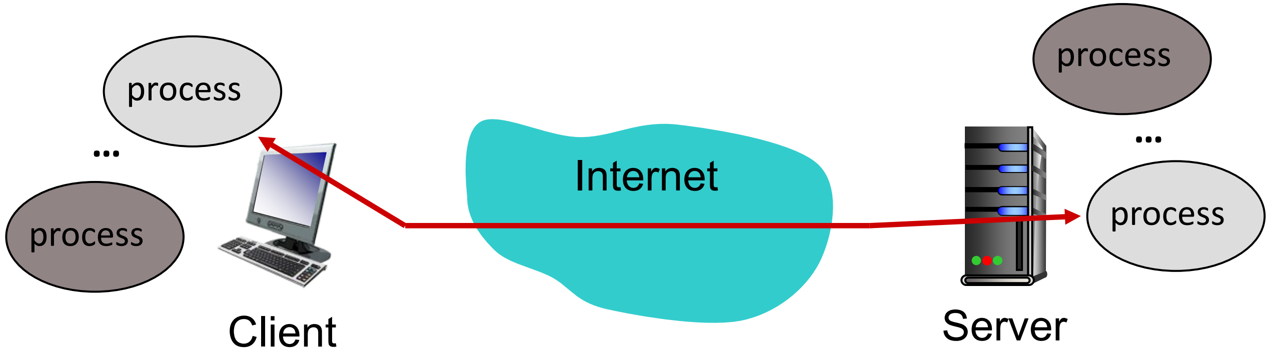
Addressing Processes

IP address is used to identify a host

- A 32-bit integer (e.g. 137.132.21.27)

Question: is IP address of a host suffice to identify a process running inside that host?

A: no, many processes may run concurrently in a host.



Analogy

Postal service:

deliver letter to the doorstep: home address

dispatch letter to the right person in the house: name of the receiver as stated on the letter

Protocol service:

deliver packet to the right host: IP address of the host

dispatch packet to the right process in the host: port number of the process

Addressing Processes

A process is identified by (IP address, port number).

- Port number is 16-bit integer (1-1023 are reserved for standard use).

Example port numbers

- HTTP server: 80
- SMTP server: 25

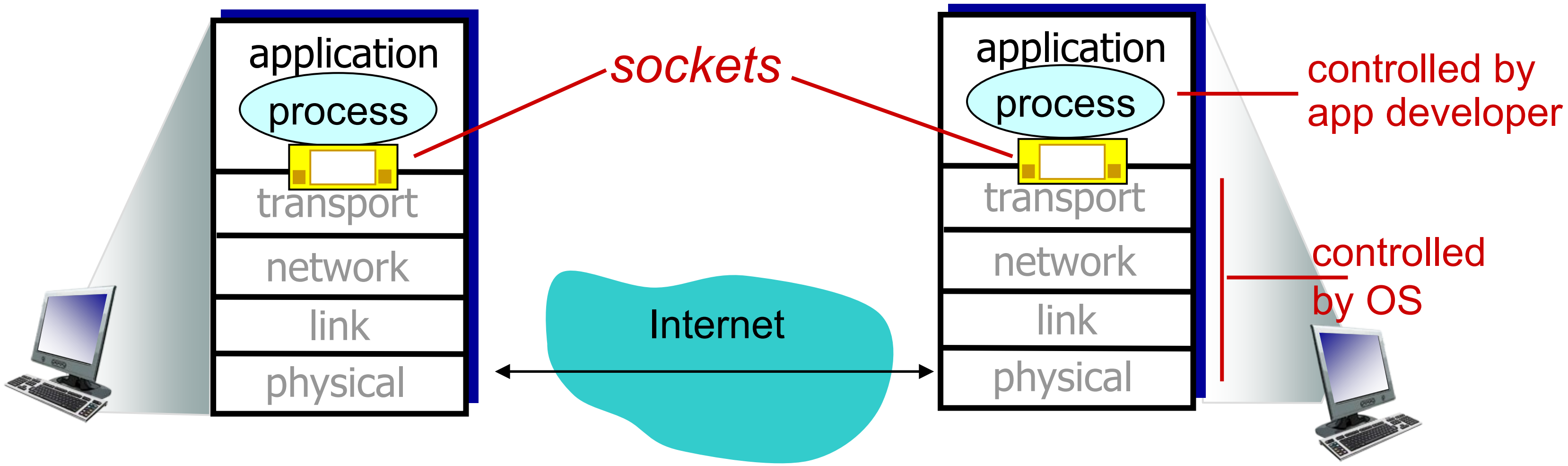
IANA coordinates the assignment of port number:

- http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

Sockets

Socket is the software interface between app processes and transport layer protocols.

- Process sends/receives messages to/from its socket.
- Programming-wise: a set of APIs



Socket Programming

Applications (or processes) treat the Internet as a black box, sending and receiving messages through sockets.

Two types of sockets

- TCP: reliable, byte stream-oriented socket
- UDP: unreliable datagram socket

Now let's write a simple client/server application that client sends a line of text to server, and server echoes it.

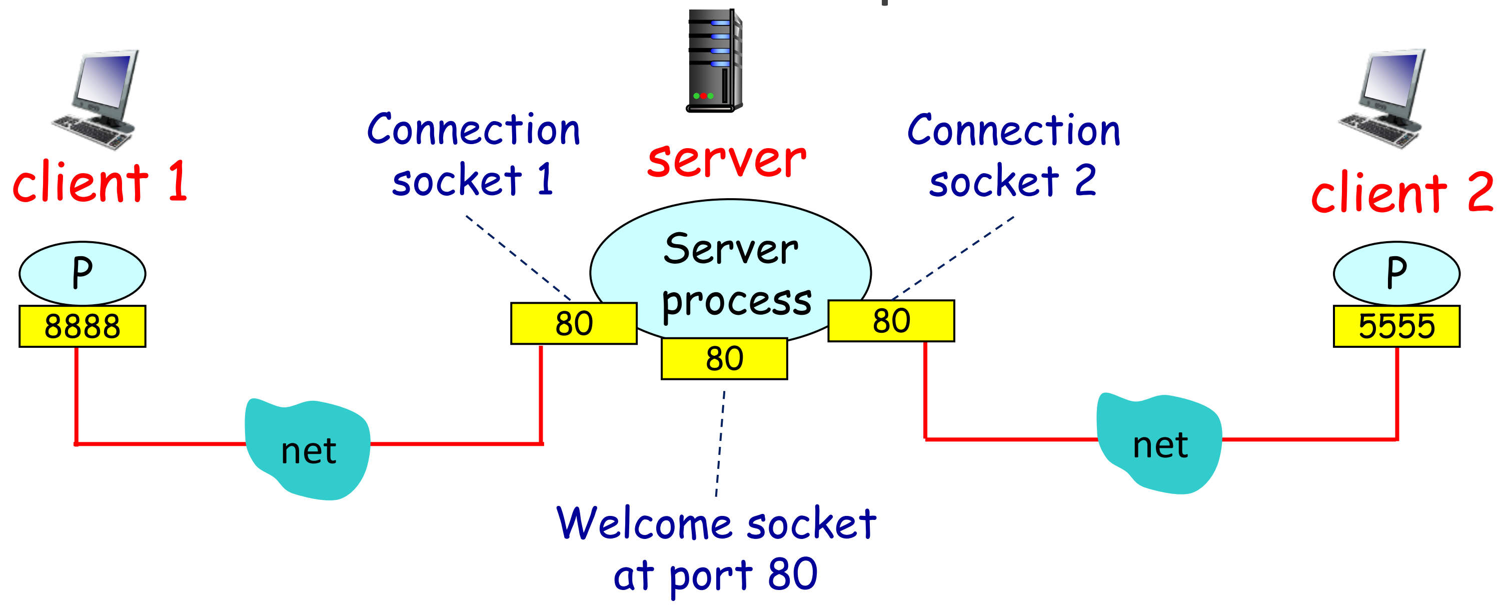
- We will demo both TCP socket version

Socket Programming with TCP

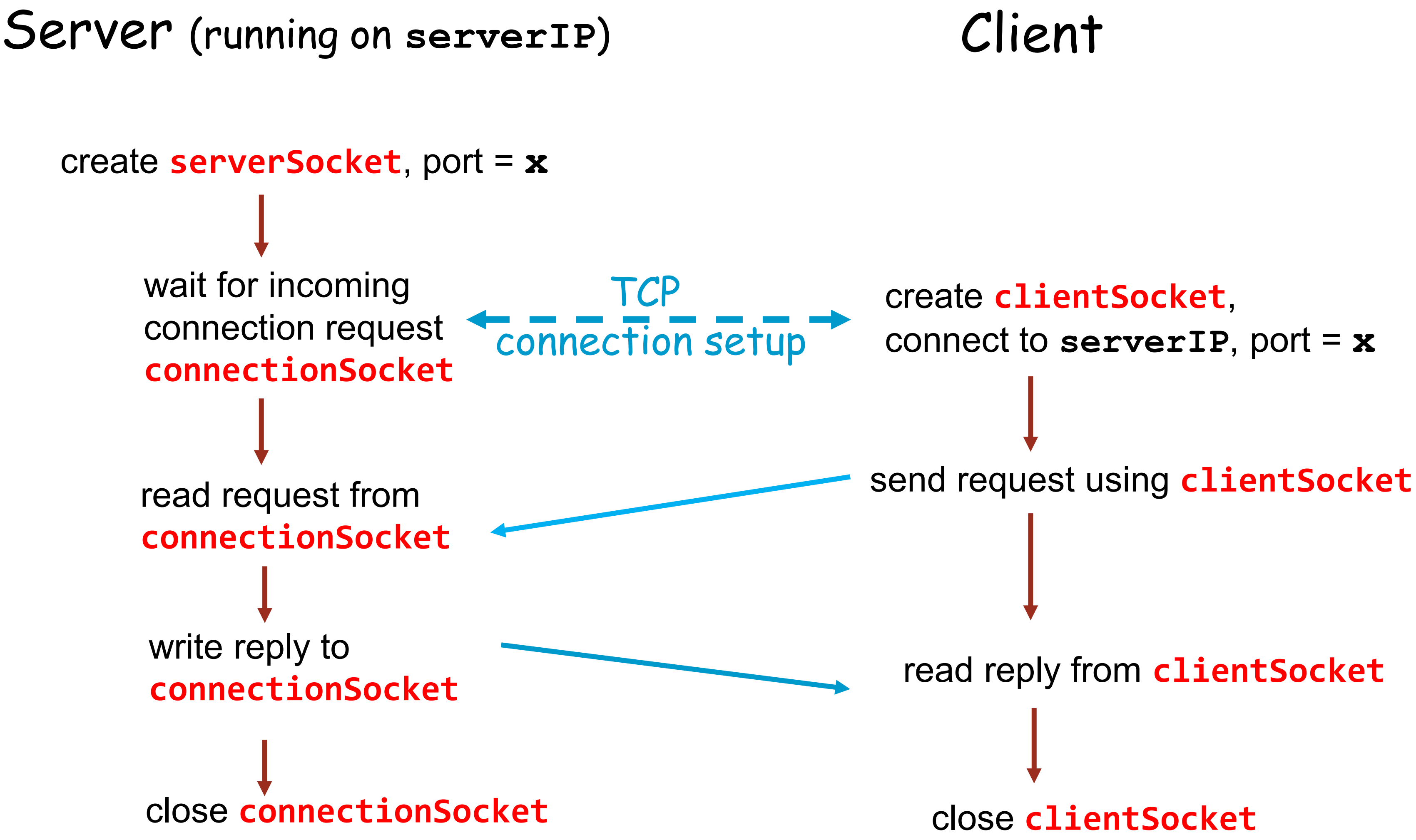
When client creates socket, client TCP establishes a connection to server TCP.

When contacted by client, server TCP creates a new socket for server process to communicate with that client.

- allows server to talk with multiple clients individually.



TCP: Client/server Socket Interaction



Server (running on **serverIP**)

Client

create **serverSocket**, port = **x**

wait for incoming connection request **connectionSocket**

TCP connection setup

create **clientSocket**, connect to **serverIP**, port = **x**

send request using **clientSocket**

read request from **connectionSocket**

write reply to **connectionSocket**

read reply from **clientSocket**

close **connectionSocket**

close **clientSocket**

Example: TCP Echo Server

```
from socket import *

serverPort = 2105

serverSocket = socket(AF_INET, SOCK_STREAM)

serverSocket.bind(('', serverPort))

serverSocket.listen()
print('Server is ready to receive message')

connectionSocket, clientAddr = serverSocket.accept()
message = connectionSocket.recv(2048)

connectionSocket.send(message)

connectionSocket.close()
```

TCP socket

listens for incoming TCP request
(not available in UDP socket)

returns a *new* socket
to communicate with
client socket

Example: TCP Echo Client

```
from socket import *

serverName = 'localhost'
serverPort = 2105

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

message = input('Enter a message: ')

clientSocket.send(message.encode())

receivedMsg = clientSocket.recv(2048)

print('from server:', receivedMsg.decode())

clientSocket.close()
```

establish a connection

no need to attach server name, port

Example: TCP Echo Server

```
from socket import *

serverPort = 2105

serverSocket = socket(AF_INET, SOCK_STREAM)

serverSocket.bind(('', serverPort))

serverSocket.listen()
print('Server is ready to receive message')

connectionSocket, clientAddr = serverSocket.accept()
message = connectionSocket.recv(2048)

connectionSocket.send(message)

connectionSocket.close()
```

TCP socket

listens for incoming TCP request
(not available in UDP socket)

returns a *new* socket
to communicate with
client socket

Processes

In C/S model

server process waits to be contacted

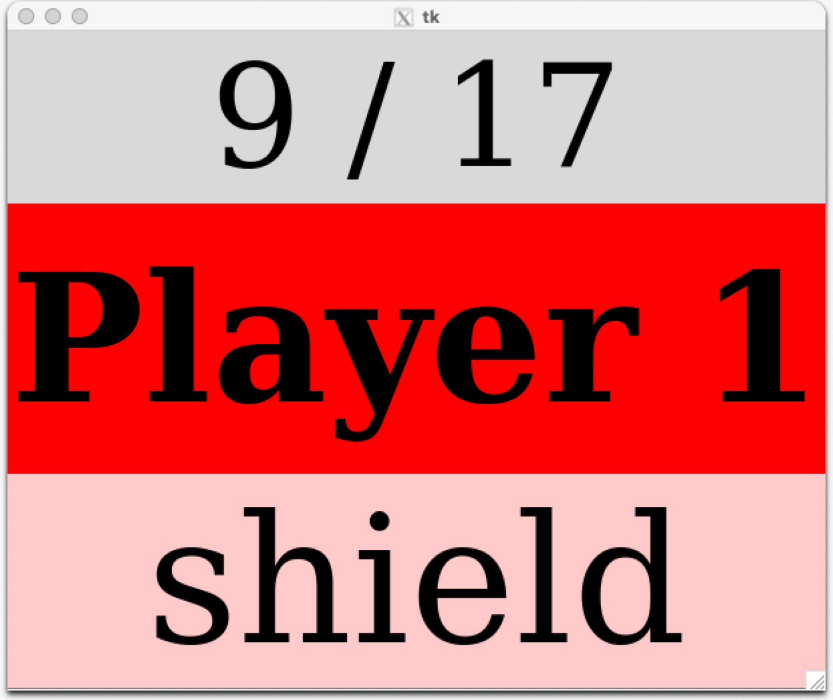
client process initiates the communication

Process: program running within a host.

- Within the same host, two processes communicate using inter-process communication (IPC) (defined by OS).
  - You would need to handle
    - Producer-Consumer problem
    - Race conditions, etc
- Processes in different hosts communicate by exchanging messages (according to protocols).

Eval server

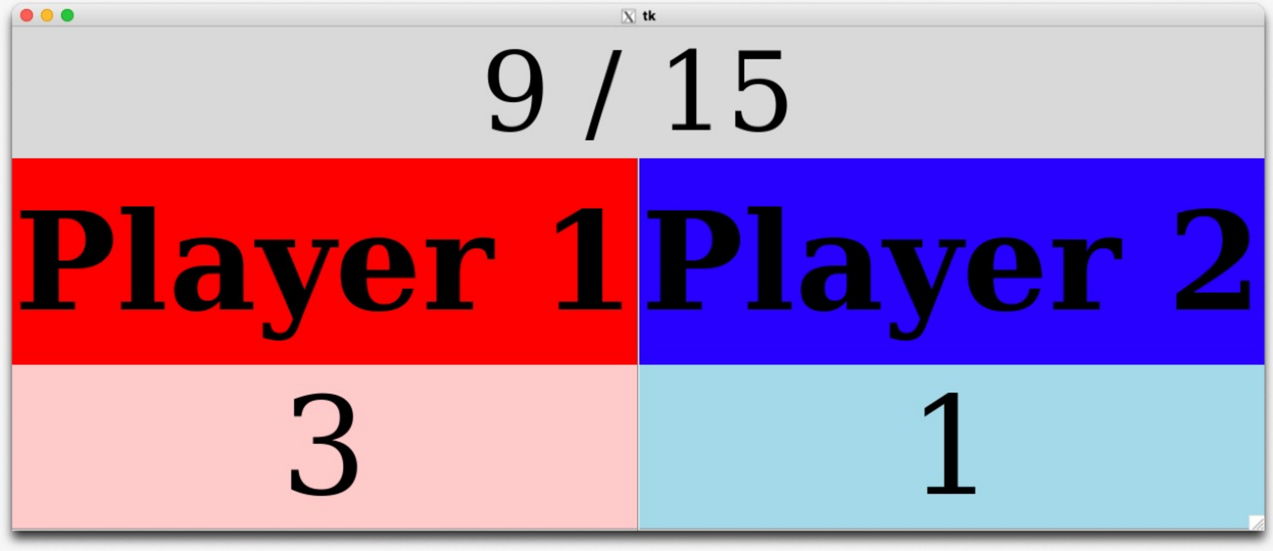
GUI



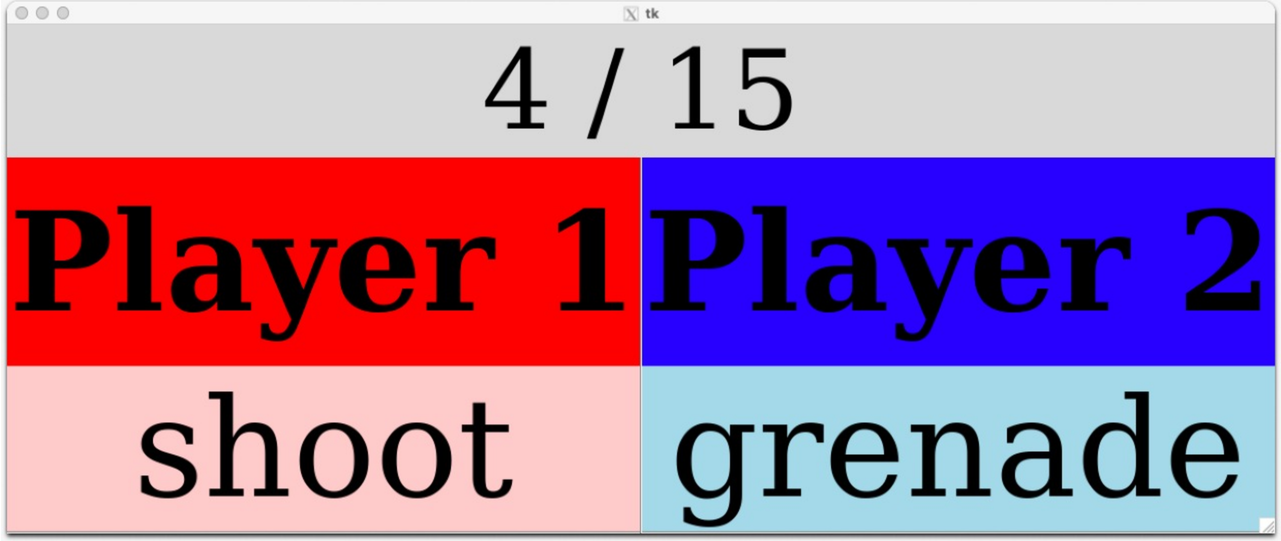
9 / 17

Player 1

shield

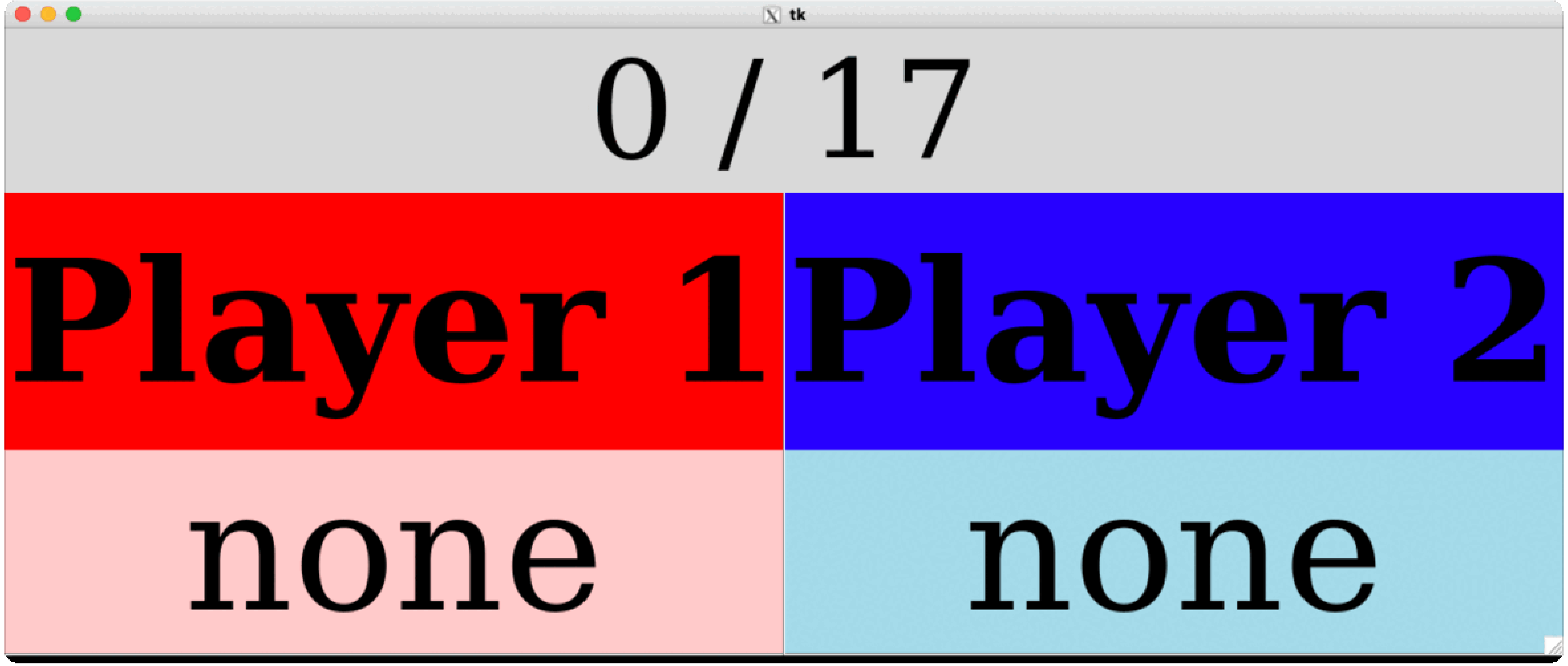


9 / 15	
Player 1	**Player 2**
3	1



4 / 15	
Player 1	**Player 2**
shoot	grenade

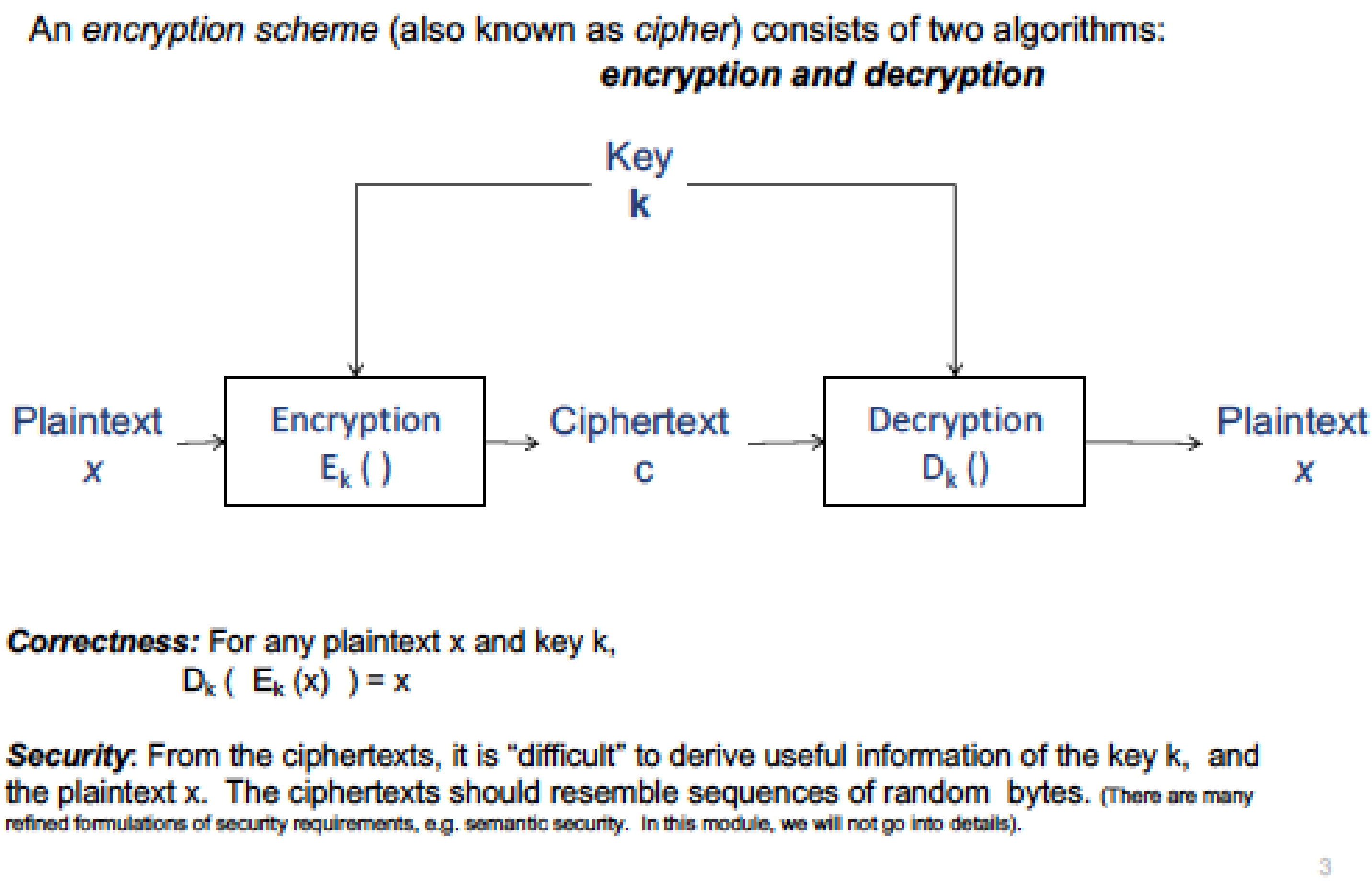
Server python code provided (Canvas)



0 / 17

Player 1	Player 2
none	none

Encryption: Flashback from CS2103



An *encryption scheme* (also known as *cipher*) consists of two algorithms:
encryption and decryption

Key
k
Plaintext
x
Encryption
E_k ( )
Ciphertext
c
Decryption
D_k ()
Plaintext
x

Correctness: For any plaintext x and key k,
$D_k(\ E_k(x)\ ) = x$

Security: From the ciphertexts, it is "difficult" to derive useful information of the key k, and the plaintext x. The ciphertexts should resemble sequences of random bytes. (There are many refined formulations of security requirements, e.g. semantic security. In this module, we will not go into details).

3

[Slide from CS2103, Prof. Chang Ee Chien]

Why do we need encryption in our system?

- Open wireless networks
- Personal data privacy
- Authentication

- Key
  - Your choice – Tell us during evaluation so we can decrypt

Authentication: Server (released on Canvas)

```
decodedMSG = base64.b64decode(encodedMsg)

iv = decodedMSG[:16]

cipher = AES.new(secret_key,AES.MODE_CBC,iv)

decryptedText = cipher.decrypt(decodedMSG[16:]).strip()
```

Authentication: Client (You! ☺)

```
iv = Random.new().read(AES.block_size)

cipher  = AES.new(secret_key,AES.MODE_CBC,iv)

encoded = base64.b64encode(iv + cipher.encrypt(msg))
```

Server python code provided (Canvas)

eval_server.py

- Server expects a secret key
- Server expects message in JSON format: more details in the code
- AES expects base64 encoded message of 128-bits initial value + message
- AES expects padding
- Server returns correct JSON so you can recalibrate

Tips

- Test your wireless comms client on your laptop first, localhost
- Test socket comms and encryption/decryption separately

Evaluation Server JSON

JSON Received P1:

```
{'hp': 4,
'action': 'none’,
 'bullets': 3,
 'grenades': 17,
'shield_time': 3,
'shield_health': 1,
 'num_deaths': 22,
'num_shield': 12}
```

JSON Expected P1:

```
{'hp': 4,
'action': ‘shoot’,
 'bullets': 3,
 'grenades': 1,
'shield_time': 3,
'shield_health’: 10,
 'num_deaths': 2,
'num_shield': 1}
```

The big bad NUS wolf/firewall ☺

Your Ultra96 FPGA boards can be accessed remotely:

1. You need to ssh into sunfire (for students) :

   ssh nusnet_id@sunfire.comp.nus.edu.sg

2. From Sunfire, you can access the boards:

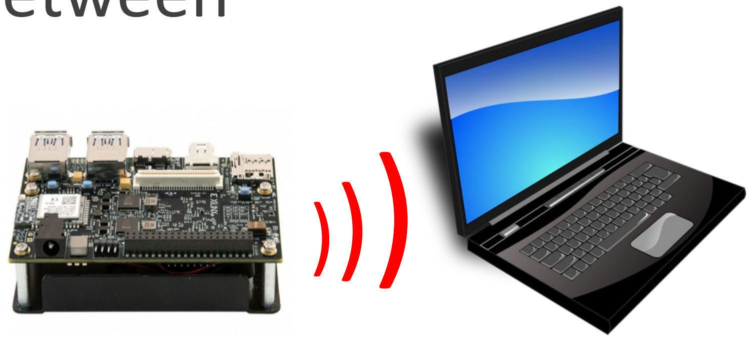
   ssh xilinx@<IP address of your group's board>

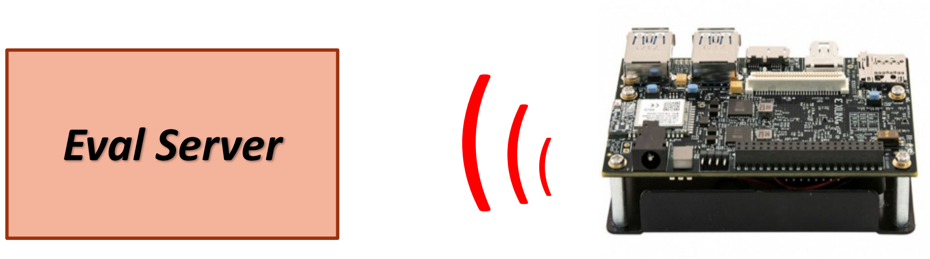
How do you tunnel through the NUS firewall so you can communicate between laptop and Ultra96 FPGA board?

Individual subcomponent test

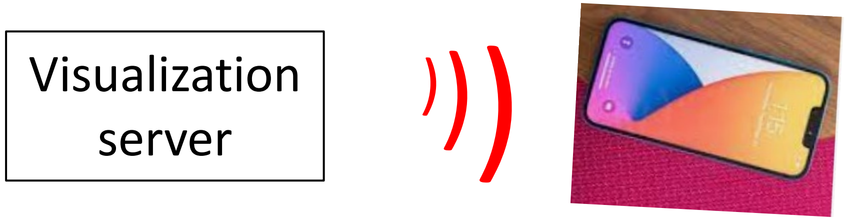
Comms External

- Walkthrough and demo secure socket comms between
  - laptop and Ultra96
  - Ultra96 and evaluation server
  - Visualizer server and Visualizer client





Eval Server



Visualization server