Socket Programming

Peerapon S.

CPE 314: Computer Networks (2/63)

Topics

- Applications and transport layer services
- UDP socket
- TCP socket
- Concurrent communication

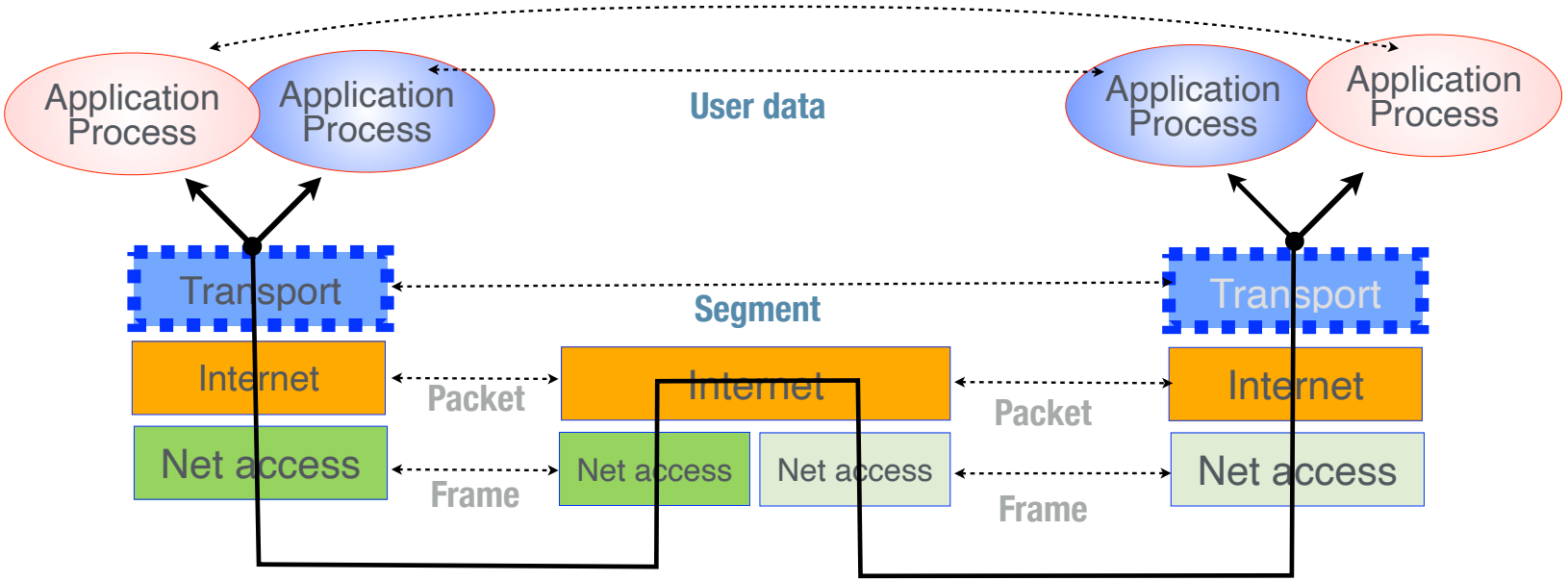
- Readings
  - Forouzan text: Ch. 2.1, 2.2, 2.5
  - Kurose text: Ch.2.7

Learning Objectives

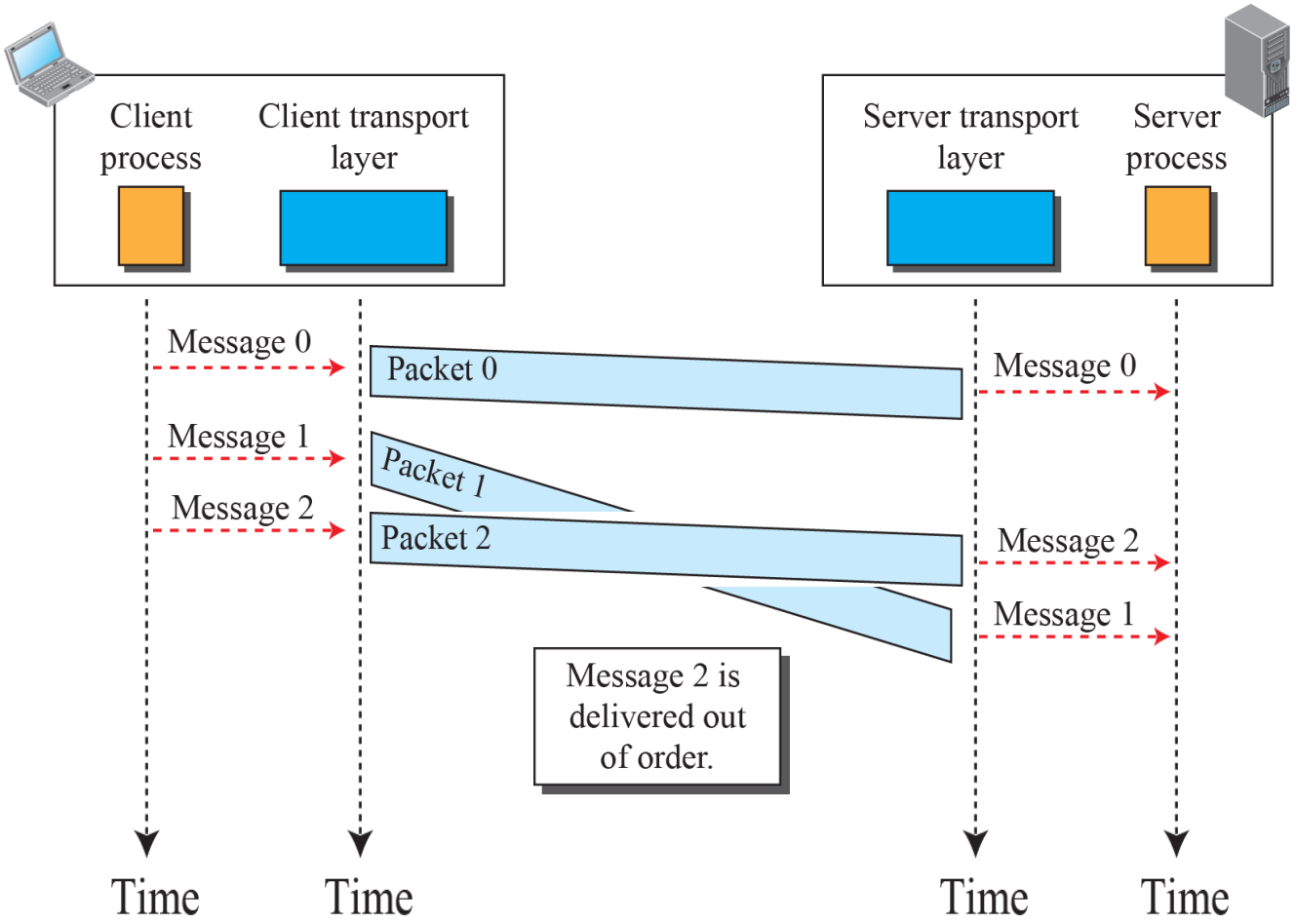
- ☐ Explain the services of transport layer
- ☐ Explain the meanings and roles of socket in network communication
- ☐ Explain the purpose of port number and port range.
- ☐ Explain the differences between UDP and TCP sockets
- ☐ Write a simple network program using UDP/TCP sockets.

Transport Layer (TL) Services

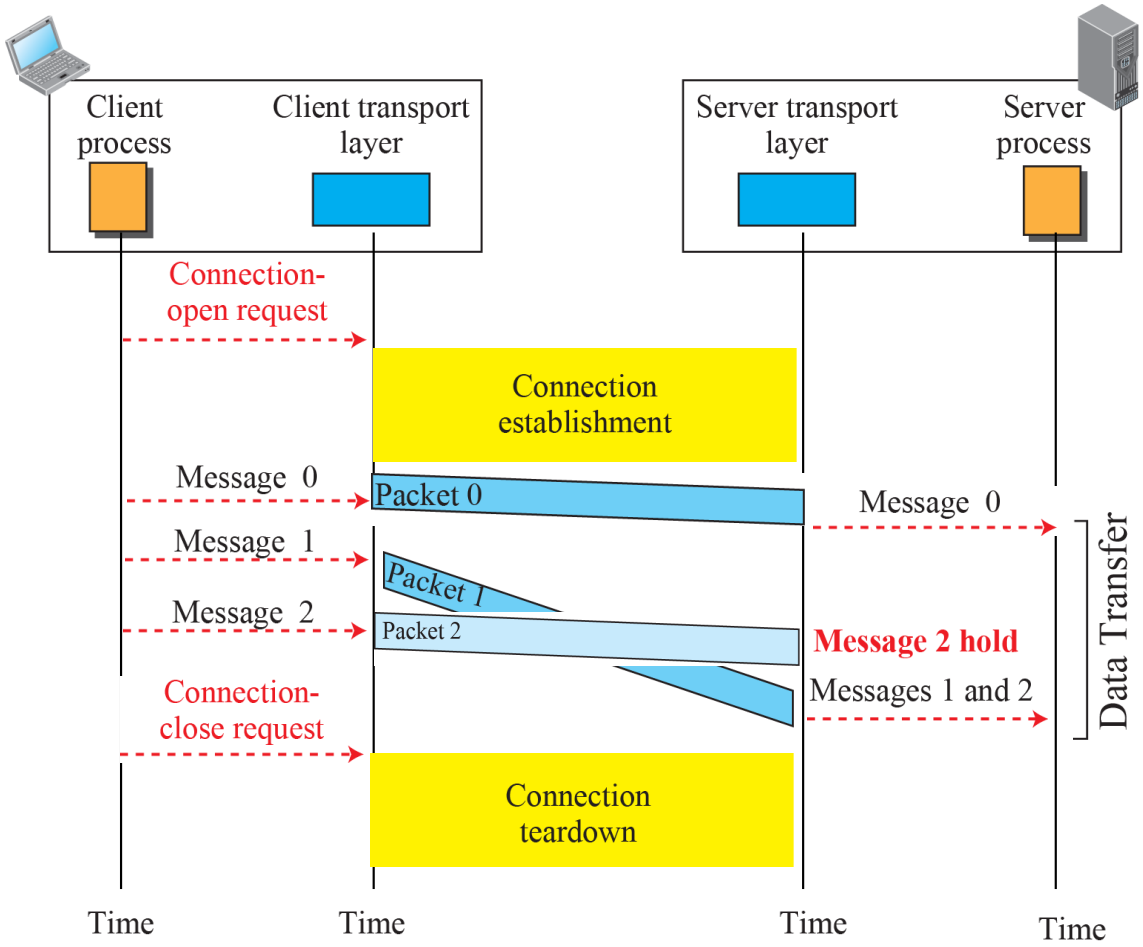
- Process-to-process message delivery service



Connectionless Transport Service



Connection-oriented Transport Service

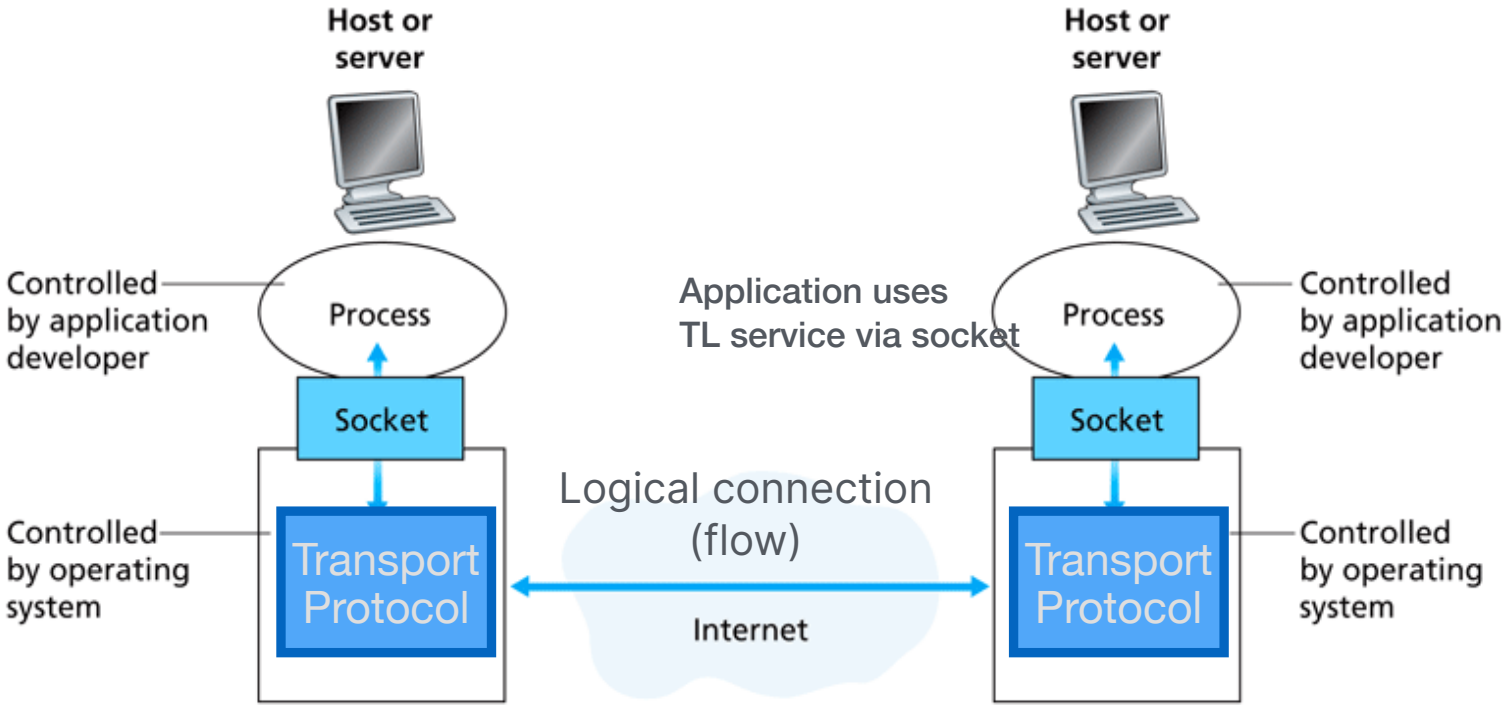


Two Main Internet Transport Layer Protocols

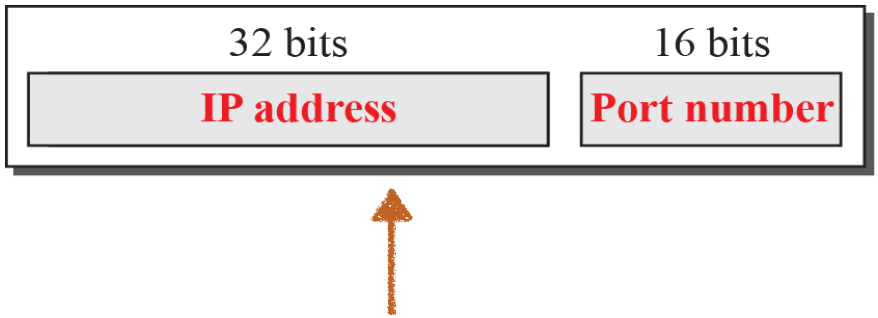
- **UDP: User Datagram Protocol**
  - Connection-less, Unreliable service
  - Independent messages delivery
- **TCP: Transport Control Protocol**
  - Connection-oriented, Reliable service
  - Byte-stream delivery

	Loss-free	Order	No duplicate	Bound delay	Throughput
UDP					
TCP	◉	◉	◉		

Socket Abstraction

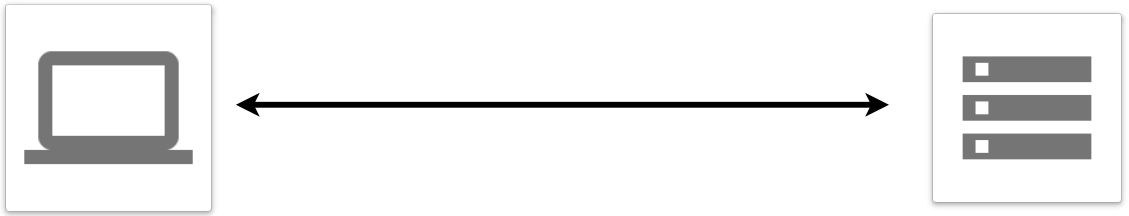


Socket Address



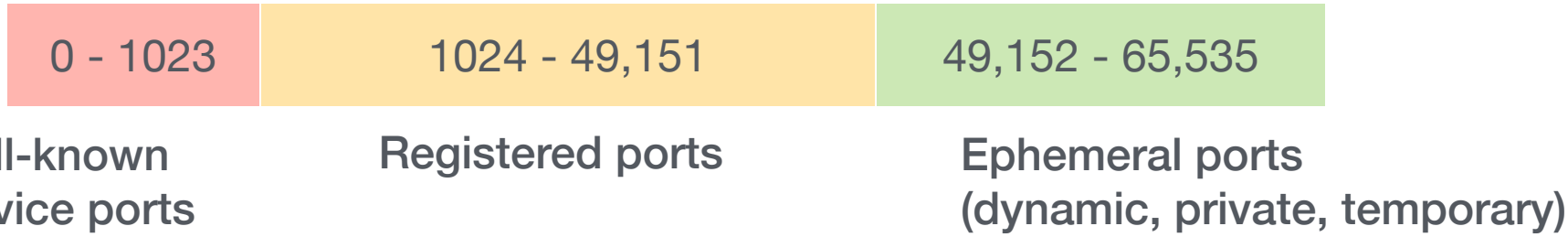
(10.35.21.10, 56780)
Web client application

(www.kmutt.ac.th, 80)
Web server application



Port Range for Server and Client

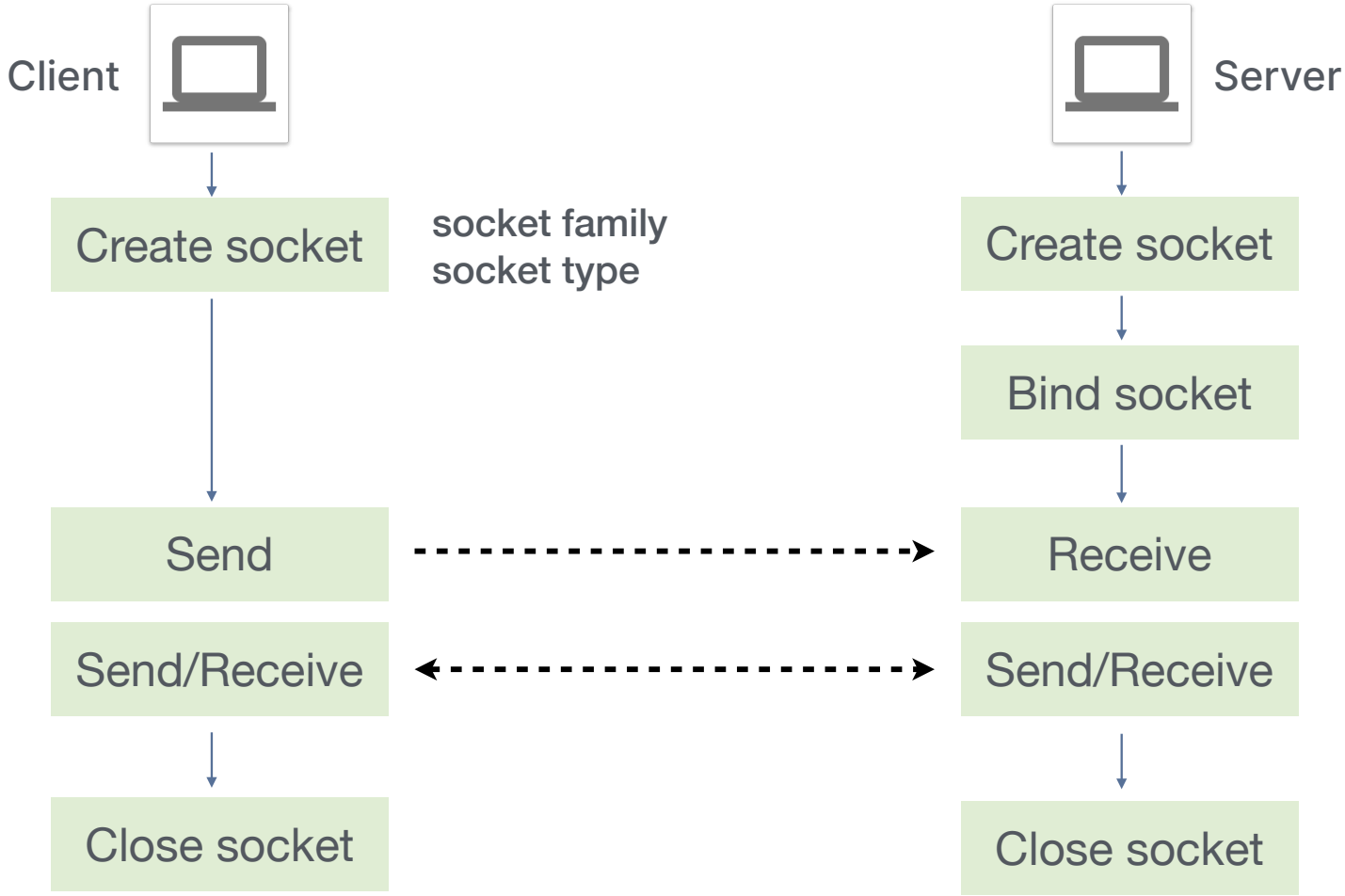
- Only some port numbers can be used by your custom app.



	Local port	Remote port
Client side	Random from Ephemeral range	Service or Registered port
Server side	Service or Registered port	Derived from client segment

UDP Program Demo

Workflow of UDP Socket Communication



UDP Server

```
from socket import *
import sys

MAX_BUF = 2048      # Size of buffer to store received bytes
SERV_PORT = 50000   # Server port number

addr = ('127.0.0.1', SERV_PORT)          # Socket address
s = socket(AF_INET, SOCK_DGRAM)          # Create UDP socket
s.bind(addr)                             # Bind socket to address
print ('UDP server started ...')
```

Change to '' if running on MAC/Linux

```
while(1):
  print ('Client> ', end = '')
  txtin,addr = s.recvfrom(MAX_BUF)  # txtin = received text
                                    # addr = client socket address
  print ('%s' %(txtin).decode('utf-8')) # Convert byte to string
  if txtin == b'quit':          # Break if user types 'quit'.
     print('Terminate server ...')
     break
  else:
     txtout = txtin.upper()     # Change text to upper case
     s.sendto(txtout, addr)     # Send it back to the client
```

Some key Python syntax rules:

- Indentation for code blocks
- ':' at the end of control statement.
- No variable declaration needed.

UDP Client

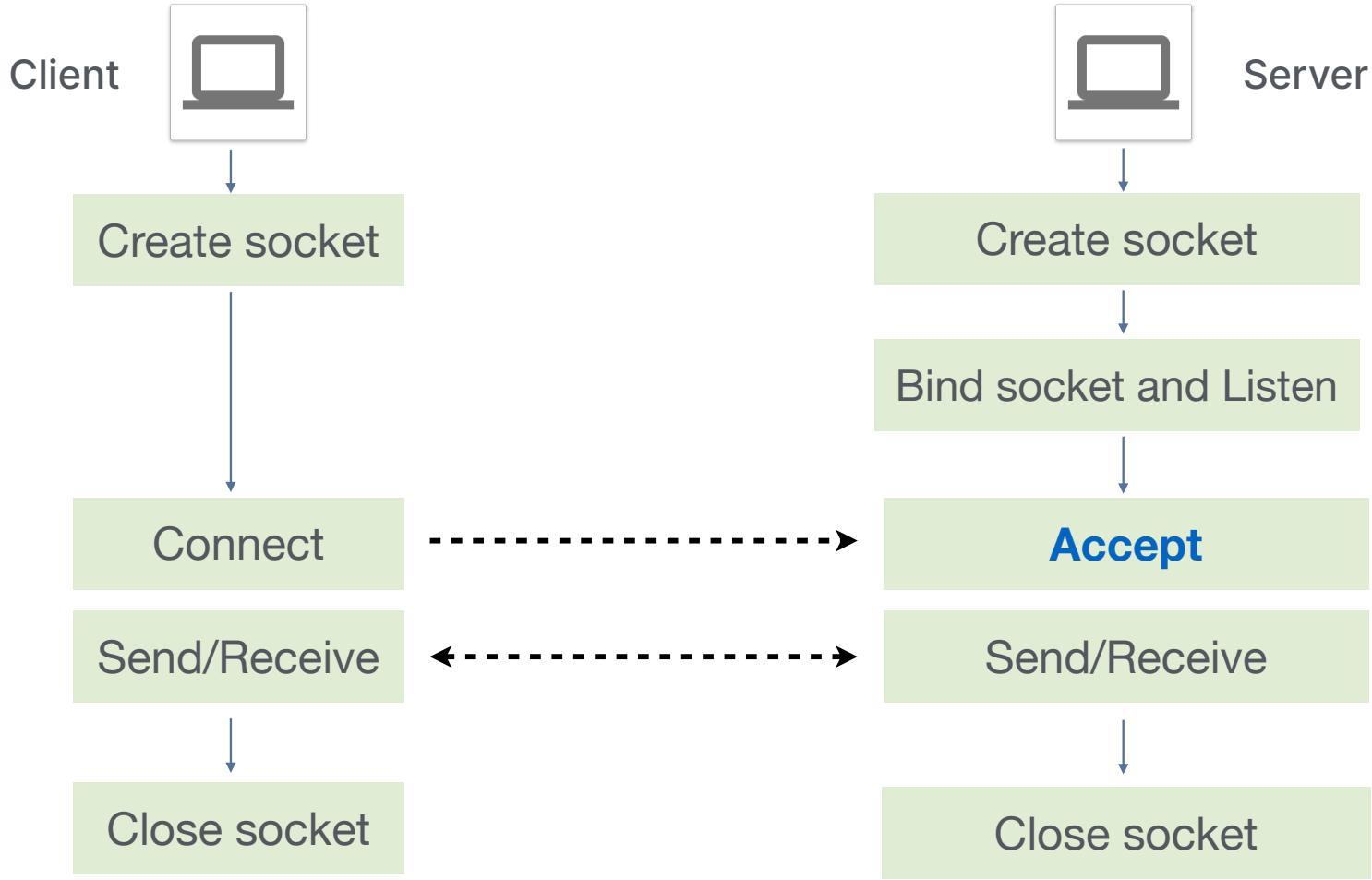
```
from socket import *
import sys

MAX_BUF = 2048
SERV_PORT = 50000

addr = ('127.0.0.1', SERV_PORT)  # Server socket address
s = socket(AF_INET, SOCK_DGRAM)  # Create UDP socket
```

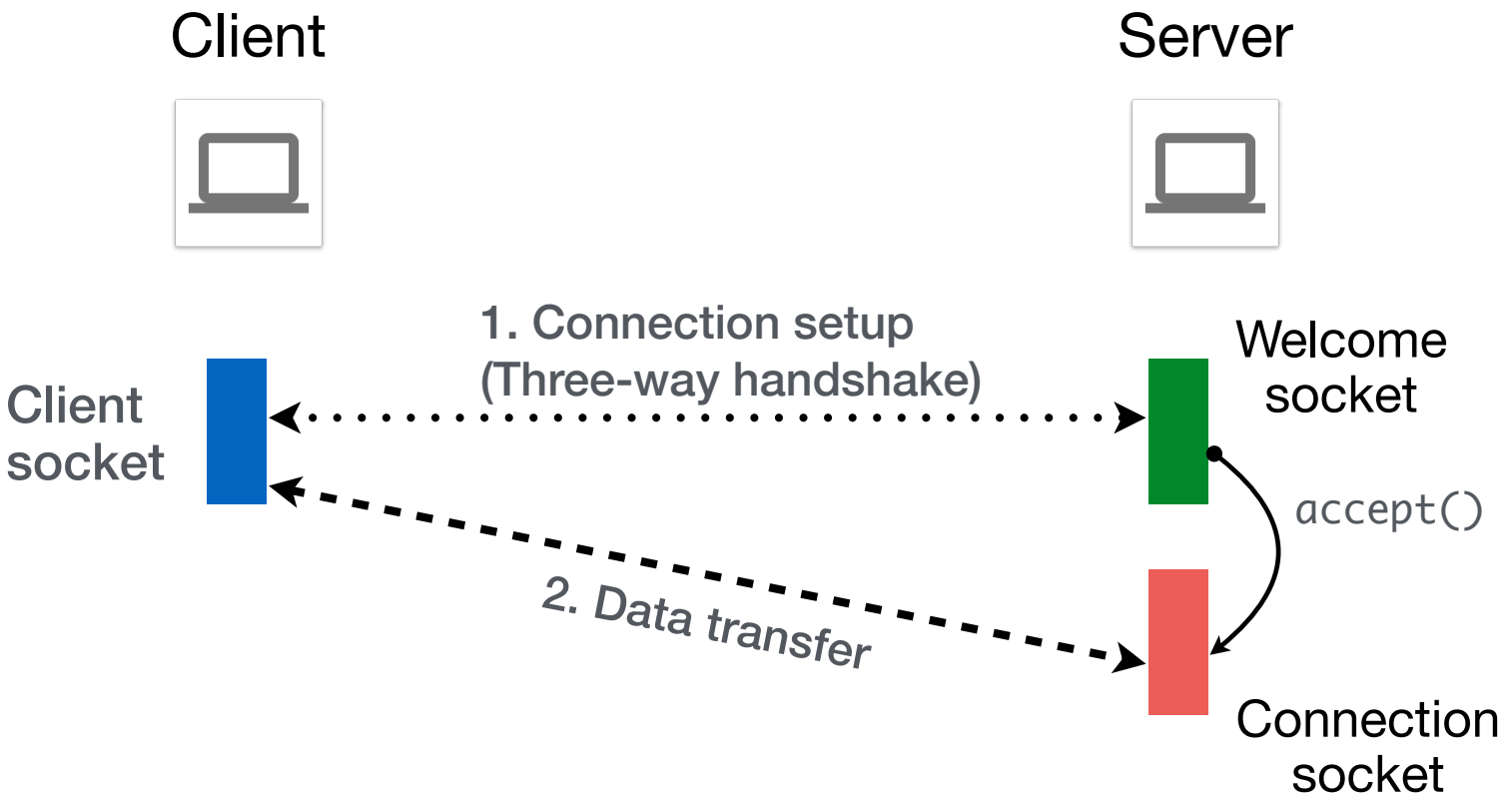
Change to remote server IP adress
if not running in the same computer

```
username = input('Enter your name: ') # text for prompt
while(1):
    print('%s> ' %(username), end='') # Print the prompt
    sys.stdout.flush()
    txtout =  sys.stdin.readline().strip() # Take input from user keyboard
    s.sendto(txtout.encode('utf-8'), addr) # Convert to string to byte and send
    if txtout == 'quit':                   # Exit if user types quit
      break
    modifiedMsg, srvAddr = s.recvfrom(2048) # Wait for modified text from server
    print (modifiedMsg.decode('utf-8'))     # Print the modified text.
```

TCP Program Demo

Workflow of TCP Socket Communication




Client
Server
1. Connection setup
(Three-way handshake)
Client
socket
Welcome
socket
accept()
2. Data transfer
Connection
socket

TCP Server

```
from socket import *
import sys

SERV_PORT = 50000

addr = ('127.0.0.1', SERV_PORT)
s = socket(AF_INET, SOCK_STREAM)
s.bind(addr)
s.listen(1)
print ('TCP server started ...')
```

```
while True:
  sckt, addr = s.accept()
  print ('New client connected ..')

  while True:
     txtin = sckt.recv(1024)
     print ('Client> %s' %(txtin).decode('utf-8'))
     if txtin == b'quit':
       print('Client disconnected ..')
       print('Waiting for a new client ...')
       break
     else:
       txtout = txtin.upper()
       sckt.send(txtout)

  sckt.close()
```

TCP Client

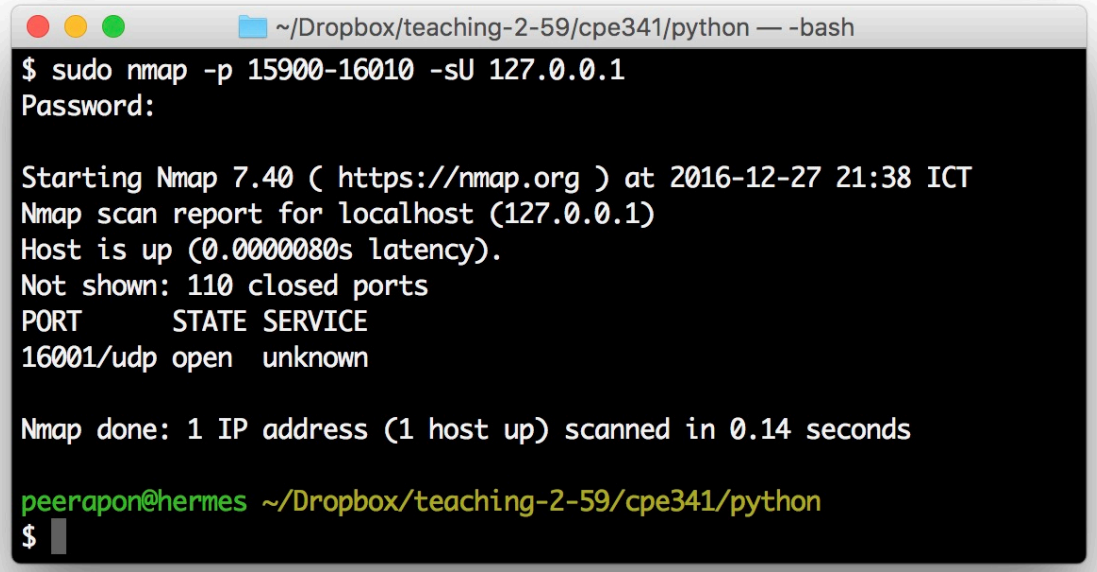
```
from socket import *
import sys

MAX_BUF = 2048
SERV_PORT = 50000

addr = ('127.0.0.1', SERV_PORT)
s = socket(AF_INET, SOCK_STREAM)
s.connect(addr)
```

```
username = input('Enter your name: ')
while True:
    print ('%s> ' %(username), end='')
    sys.stdout.flush()
    txtout = sys.stdin.readline().strip()
    s.send(txtout.encode('utf-8'))
    if txtout == 'quit':
      break
    modifiedMsg = s.recv(2048)
    print (modifiedMsg.decode('utf-8'))
```

Use 'netstat - a' to verify the TCP connection
Note that TCP server does not quit when the client terminates.

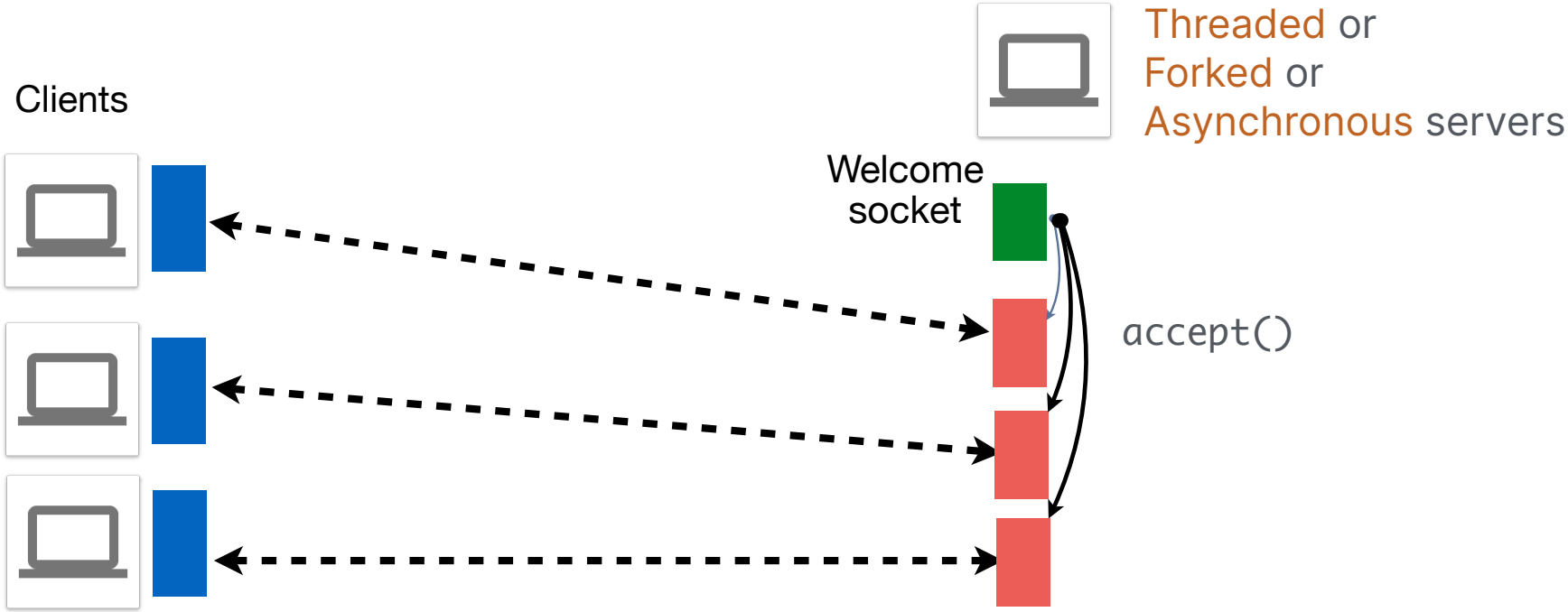
- Scan open ports at a given IP address by using **nmap** utility (https://nmap.org/download.html)

```
# Scan port range 3000 to 4000 in local host for TCP server
nmap -p 3000-4000 127.0.0.1

# Scan port range 3000 to 4000 in local host for UDP server
nmap -p 3000-4000 -sU 127.0.0.1

# Same as above but does not ping before scanning
nmap -P0 –p 3000-4000 -sU 127.0.0.1
```



Concurrent Connections

- Server almost always serves multiple clients simultaneously.
- Use either threaded or forked processes to handle new connections.



See SocketServer module in https://docs.python.org/3/library/socketserver.html

Threaded Server

```
from socket import *
from threading import Thread
import os,sys

SERV_PORT = 50000

def handle_client(s):
  while True:
     txtin = s.recv(1024)
     print ('Client> %s' %(txtin).decode('utf-8'))
     if txtin == b'quit':
        print('Client disconnected ...')
        break
     else:
        txtout = txtin.upper()
        s.send(txtout)
  s.close()
  return
```

```
while True:
  sckt, addr = s.accept()
  ip, port = str(addr[0]), str(addr[1])
  print ('New client connected from ..' + ip + ':' + port)

  try:
    Thread(target=handle_client, args=(sckt,)).start()
  except:
    print("Cannot start thread..")
    import traceback
    trackback.print_exc()
```

Forked Server

```
import os
from socket import *

s = socket(AF_INET,SOCK_STREAM)
s.bind(("",5000); s.listen(5)

while True:
  sckt, addr = s.accept()
if os.fork() == 0: # child process ..
     ...
     sckt.close(); os._exit(0);
  else: # Parent process
     sckt.close()
 s.close()
```

Conclusion

- UDP and TCP sockets to access transport layer services (unreliable / reliable services)
- Port numbers to locally identify a network process
- Socket address to globally identify a network process.
- Different workflows in functions calls for UDP and TCP sockets.
- Concurrent connections achieved by using threaded or forked TCP servers.