

# Network Programming

## 1 Computer Networks

- transmission media and network topologies
- client/server architecture
- layers, protocols, and sockets

## 2 Network Programming

- a simple client/server interaction
- the module socket in Python
- implementing a simple client/server

## 3 Guessing a Secret

- a game: player/dealer communication

MCS 260 Lecture 39  
Introduction to Computer Science  
Jan Verschelde, 18 April 2016

# Network Programming

## 1 Computer Networks

- transmission media and network topologies
- client/server architecture
- layers, protocols, and sockets

## 2 Network Programming

- a simple client/server interaction
- the module socket in Python
- implementing a simple client/server

## 3 Guessing a Secret

- a game: player/dealer communication

# Transmission Media

## types of connections

The unit for transmission speed is  $\text{bps}$ , bits per second.

Four categories of media used for data transmission:

- 1 **twisted pair wire:** pair of copper wires used for telephone communications. Transmission speeds vary from 2,400 to 33,600  $\text{bps}$ .
- 2 **coaxial cable:** for many telephone lines or television signal. Data transmission over short distances guarantees speeds up to  $10^7$   $\text{bps}$ .
- 3 **optical fibre:** signals transmitted via light-emitting diodes encode bits as presence or absence of light. Transmission speeds reach up to  $10^9$   $\text{bps}$ .
- 4 **electromagnetic waves:** wireless over short distances or via satellites across long distances.

# Network Topologies

structure of computer networks

Three most common regular topologies:



a *star network*: one central node is connected to all other nodes



a *ring network*: nodes form a closed circuit, messages circle around the ring of nodes



a *bus network*: all nodes on a single bus, used in the Von Neumann architecture

# Network Programming

## 1 Computer Networks

- transmission media and network topologies
- **client/server architecture**
- layers, protocols, and sockets

## 2 Network Programming

- a simple client/server interaction
- the module socket in Python
- implementing a simple client/server

## 3 Guessing a Secret

- a game: player/dealer communication

# Client/Server Architecture

for software and hardware

Client/server architecture defines the communication between two computers: one is the server and the other acts as the client.

A client places a request or order to a server.

The server processes the request.

The client does not need to know how the server processes the requests.

We distinguish between software and hardware client/server architectures:

- web and database servers offer software services;
- file and print servers offer hardware services.

# Network Programming

## 1 Computer Networks

- transmission media and network topologies
- client/server architecture
- layers, protocols, and sockets

## 2 Network Programming

- a simple client/server interaction
- the module socket in Python
- implementing a simple client/server

## 3 Guessing a Secret

- a game: player/dealer communication

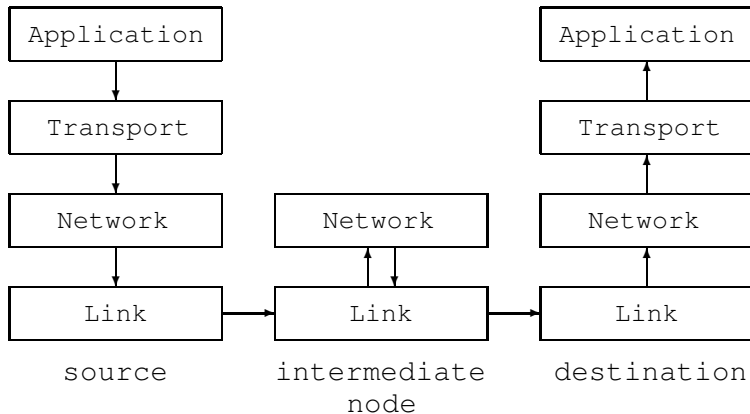
# Four Layers in Internet Communication

application, transport, network, and link

- 1 The **application layer** consists of client and server software. The application prepares the message and defines the destination address.
- 2 The **transport layer** formats the message by chopping it into packets attaching a sequence number and destination address to each packet. When receiving, packets are collected and reassembled into a message.
- 3 The **network layer** determines the (intermediate) address for each packet. This layer detects when a packet has reached its final destination.
- 4 The **link layer** is responsible for the actual transmission of packets.

# Following a Message

through all four layers



# Network Protocols

## TCP and UDP

Network protocols are rules for network communication.

We consider two types of protocols:

### TCP Transmission Control Protocol

First a message is sent that data is coming.

Only after the receiver acknowledges this message will the sender send the data.

All successful transmissions are confirmed, and retransmissions are acknowledged.

### UDP User Datagram Protocol

Unlike TCP, no connection is established prior to sending data. The sender just carries on after sending a message.

TCP is connection oriented, UDP is connectionless.

TCP is more reliable whereas UDP is more streamlined.

# Sockets

Sockets are objects programs use to connect.

Sockets were introduced in 1981 in BSD Unix. Originally used for communication between processes, i.e.: between two programs on same computer.

Sockets support communication across platforms, independent of the operating system.

IP stands for Internet Protocol.

In addition to the IP address of the computer, both server and client must use the same *port*.

A port is a 16-bit integer, some are reserved for particular protocols. Any port between 1,024 and 65,535 is free.

Sockets support both TCP and UDP.

## getting the hostname and IP address

In a Python session, we may obtain the hostname and the IP address of the computer:

```
>>> import socket
>>> hn = socket.gethostname()
>>> hn
'SEO1200-PD00'
>>> ip = socket.gethostbyname(hn)
>>> ip
'131.193.11.176'
```

On some computers, the IP address on return may be '127.0.0.1' also known as localhost.

# Analogy with Telephone Exchange

client/server communication with sockets

Analogy between a telephone exchange and sockets:

- 1 dial company on 1-312-666-9000  
connect to IP address 127.0.0.1
- 2 call answered by reception  
connection established to remote host
- 3 ask for computer center  
route using specified port (8732)
- 4 call answered by computer center  
server handles request from client
- 5 hang up the phone  
close sockets

# Network Programming

## 1 Computer Networks

- transmission media and network topologies
- client/server architecture
- layers, protocols, and sockets

## 2 Network Programming

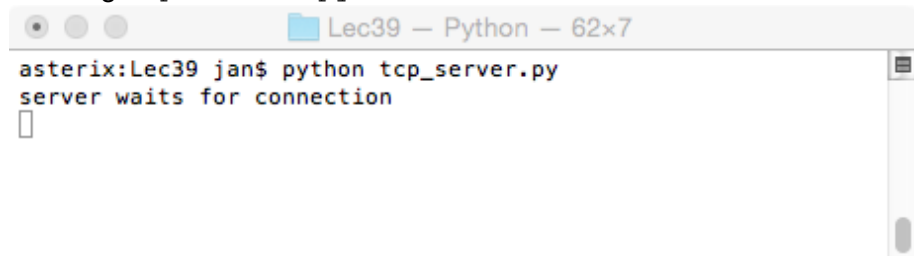
- a simple client/server interaction
- the module socket in Python
- implementing a simple client/server

## 3 Guessing a Secret

- a game: player/dealer communication

## a simple client/server interaction

Running `tcp_server.py`:

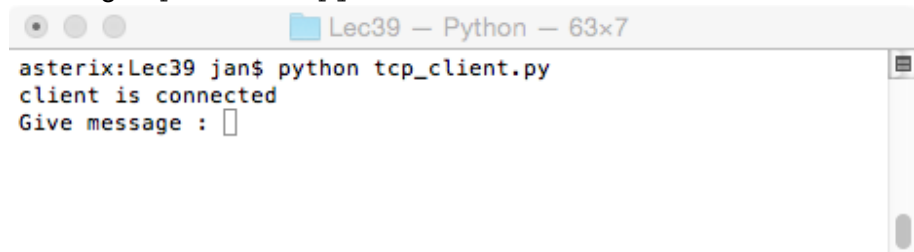
A terminal window titled "Lec39 - Python - 62x7" is shown. The prompt is "asterix:Lec39 jan\$". The user has entered "python tcp\_server.py" and the output is "server waits for connection". There is a cursor on the line below.

```
asterix:Lec39 jan$ python tcp_server.py
server waits for connection
█
```

The client `tcp_client.py` runs in another terminal.

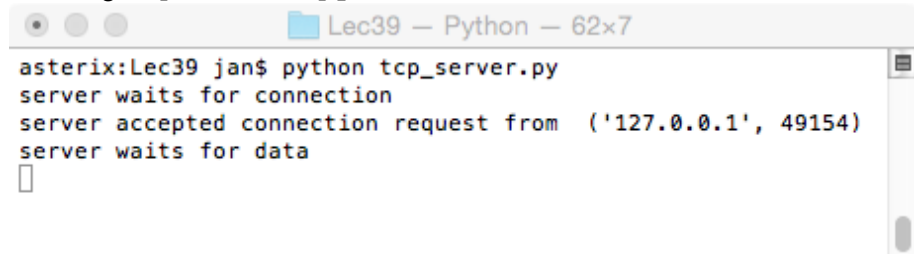
## connecting client and server

Running `tcp_client.py` in another terminal:

A terminal window titled "Lec39 - Python - 63x7" showing the execution of a Python script. The prompt is "asterix:Lec39 jan\$". The user enters "python tcp\_client.py", and the output is "client is connected" followed by a prompt "Give message :".

```
asterix:Lec39 jan$ python tcp_client.py
client is connected
Give message : 
```

Running `tcp_server.py` in one terminal:

A terminal window titled "Lec39 - Python - 62x7" showing the execution of a Python script. The prompt is "asterix:Lec39 jan\$". The user enters "python tcp\_server.py", and the output is "server waits for connection", "server accepted connection request from ('127.0.0.1', 49154)", and "server waits for data" followed by a cursor.

```
asterix:Lec39 jan$ python tcp_server.py
server waits for connection
server accepted connection request from ('127.0.0.1', 49154)
server waits for data

```

## passing data from client to server

Running `tcp_client.py` in another terminal:

```
Lec39 — bash — 63x7
asterix:Lec39 jan$ python tcp_client.py
client is connected
Give message : Hello there!
asterix:Lec39 jan$
```

Running `tcp_server.py` in one terminal:

```
Lec39 — bash — 62x7
asterix:Lec39 jan$ python tcp_server.py
server waits for connection
server accepted connection request from ('127.0.0.1', 49154)
server waits for data
server received "Hello there!"
asterix:Lec39 jan$
```

# Network Programming

## 1 Computer Networks

- transmission media and network topologies
- client/server architecture
- layers, protocols, and sockets

## 2 Network Programming

- a simple client/server interaction
- **the module socket in Python**
- implementing a simple client/server

## 3 Guessing a Secret

- a game: player/dealer communication

# sockets in Python: the socket module

The `socket` module exports the class `socket ()` which returns an object representing a socket.

The first argument of `socket` is the Address Family (AF):

- `AF_UNIX` : for UNIX sockets;
- `AF_INET` : most commonly used for internet

`AF_INET` supports both TCP and UDP, given respectively by `SOCK_STREAM` and `SOCK_DGRAM` as second argument of `socket ()`.

For example:

```
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM
sock = Socket(AF_INET, SOCK_STREAM)
```

# methods on socket objects

Most commonly used methods:

method	description
<code>accept()</code>	accepts connection and returns new socket for passing data
<code>bind()</code>	binds a socket to an address
<code>close()</code>	closes the socket
<code>connect(a)</code>	connects to address <code>a</code>
<code>listen(c)</code>	listen for <code>c</code> connections
<code>shutdown(flag)</code>	shut down for reading, writing, or both
<code>recv(b)</code>	receives data in buffer of size <code>b</code>
<code>send(d)</code>	sends data in <code>d</code>

# Network Programming

## 1 Computer Networks

- transmission media and network topologies
- client/server architecture
- layers, protocols, and sockets

## 2 Network Programming

- a simple client/server interaction
- the module socket in Python
- **implementing a simple client/server**

## 3 Guessing a Secret

- a game: player/dealer communication

## defining connections with `bind` and `connect`

After a socket is created, **both server and client** define

```
server_address = (hostname, number)
```

To bind the socket to the address, **the server** does

```
sock.bind(server_address)
```

and **the client** contacts the server then via

```
sock.connect(server_address)
```

## taking requests with `listen` and `accept`

With `listen()` the server indicates how many incoming connections will be accepted:

```
sock.listen(2) # accept at most 2 connections
```

The server takes requests via the `accept()` method:

```
client, client_address = sock.accept()
```

The `accept()` method returns

- 1 a socket `client` for receiving data;
- 2 the address of the client.

## sending and receiving data with `send` and `recv`

The **client** sends data with `send()`

```
sock.send(data)
```

The **server** receives data applying `recv()`

```
data = client.recv(buffer)
```

to the socket `client` obtained with `accept()`.

When all is over, **both client and server** do

```
sock.close()
```

***What is sent must be of type bytes.***

## the methods `encode()` and `decode()`

To avoid an error such as

```
TypeError: a bytes-like object is required, not 'str'
```

we have to work with `encode()` and `decode()`, for example

```
>>> n = 4
>>> b = str(n).encode()
>>> b
b'4'
>>> type(b)
<class 'bytes'>
>>> c = int(b.decode())
>>> c
4
```

## the script `tcp_server.py`

The script with the definition of the setup:

```
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM

HOSTNAME = ''          # blank so any address can be used
PORTNUMBER = 41267    # number for the port
BUFFER = 80           # size of the buffer

SERVER_ADDRESS = (HOSTNAME, PORTNUMBER)
SERVER = Socket(AF_INET, SOCK_STREAM)
SERVER.bind(SERVER_ADDRESS)
SERVER.listen(2)
```

## tcp\_server.py continued

```
print('server waits for connection')
CLIENT, CLIENT_ADDRESS = SERVER.accept()
print('server accepted connection request from ', \
      CLIENT_ADDRESS)
print('server waits for data')
DATA = CLIENT.recv(BUFFER).decode()
print('server received \"%s\"' % DATA)

SERVER.close()
```

## the program `tcp_client.py`

```
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM

HOSTNAME = 'localhost' # on same host
PORTNUMBER = 41267 # same port number
BUFFER = 80 # size of the buffer

SERVER_ADDRESS = (HOSTNAME, PORTNUMBER)
CLIENT = Socket(AF_INET, SOCK_STREAM)
CLIENT.connect(SERVER_ADDRESS)

print('client is connected')
DATA = input('Give message : ')
CLIENT.send(DATA.encode())
```

# Network Programming

## 1 Computer Networks

- transmission media and network topologies
- client/server architecture
- layers, protocols, and sockets

## 2 Network Programming

- a simple client/server interaction
- the module socket in Python
- implementing a simple client/server

## 3 Guessing a Secret

- a game: player/dealer communication

# A Little Game

player/dealer communication

The dealer generates a secret number.

The player must guess the number.

After each guess, the dealer sends feedback:  
too low, too high, or found the secret

The game terminates when the secret is found.

The player is the client, the dealer is the server

## the setup of the server = the dealer

```
from random import randint
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM

HOSTNAME = ''          # blank so any address can be used
PORTNUMBER = 11267    # number for the port
BUFFER = 80           # size of the buffer

DEALER_ADDRESS = (HOSTNAME, PORTNUMBER)
DEALER = Socket(AF_INET, SOCK_STREAM)
DEALER.bind(DEALER_ADDRESS)
DEALER.listen(1)

print('dealer waits for player to connect')
PLAYER, PLAYER_ADDRESS = DEALER.accept()
print('dealer accepted connection request from ', \
      PLAYER_ADDRESS)
```

## script for the server continued

```
SECRET = randint(0, 9)
print('the secret is %d' % SECRET)

while True:
    print('dealer waits for a guess')
    GUESS = PLAYER.recv(BUFFER).decode()
    print('dealer received ' + GUESS)
    if int(GUESS) < SECRET:
        REPLY = 'too low'
    elif int(GUESS) > SECRET:
        REPLY = 'too high'
    else:
        REPLY = 'found the secret'
    PLAYER.send(REPLY.encode())
    if REPLY == 'found the secret':
        break

DEALER.close()
```

## code for the client = the player

At the start of the script, we define the setup:

```
from socket import socket as Socket
from socket import AF_INET, SOCK_STREAM

HOSTNAME = 'localhost' # on same host
PORTNUMBER = 11267 # same port number
BUFFER = 80 # size of the buffer

DEALER = (HOSTNAME, PORTNUMBER)
PLAYER = Socket(AF_INET, SOCK_STREAM)
PLAYER.connect(DEALER)
```

## script continued

The player is prompted for a guess until the secret number is found.

```
print('player is ready to guess')
while True:
    GUESS = input('Give number : ')
    PLAYER.send(GUESS.encode())
    ANSWER = PLAYER.recv(BUFFER).decode()
    print('>', ANSWER)
    if ANSWER == 'found the secret':
        break

PLAYER.close()
```

# Summary + Assignments

## Assignments:

- 1 Extend the client/server interaction to simulate a password dialogue. After receiving data from a client, the server returns `access granted` or `access denied` depending on whether the received data matches the password.
- 2 Describe how you would implement the game of rock, paper, and scissors in a client/server manner.
- 3 Redefine the quiz of Project Three as a client/server interaction. The server generates, asks the questions, and gives feedback on the answers submitted by the client.
- 4 Write an implementation of the server in the previous exercise.
- 5 Write an implementation of the client in the previous exercise.