

# CSE 3214: Computer Network Protocols and Applications –Socket Programming

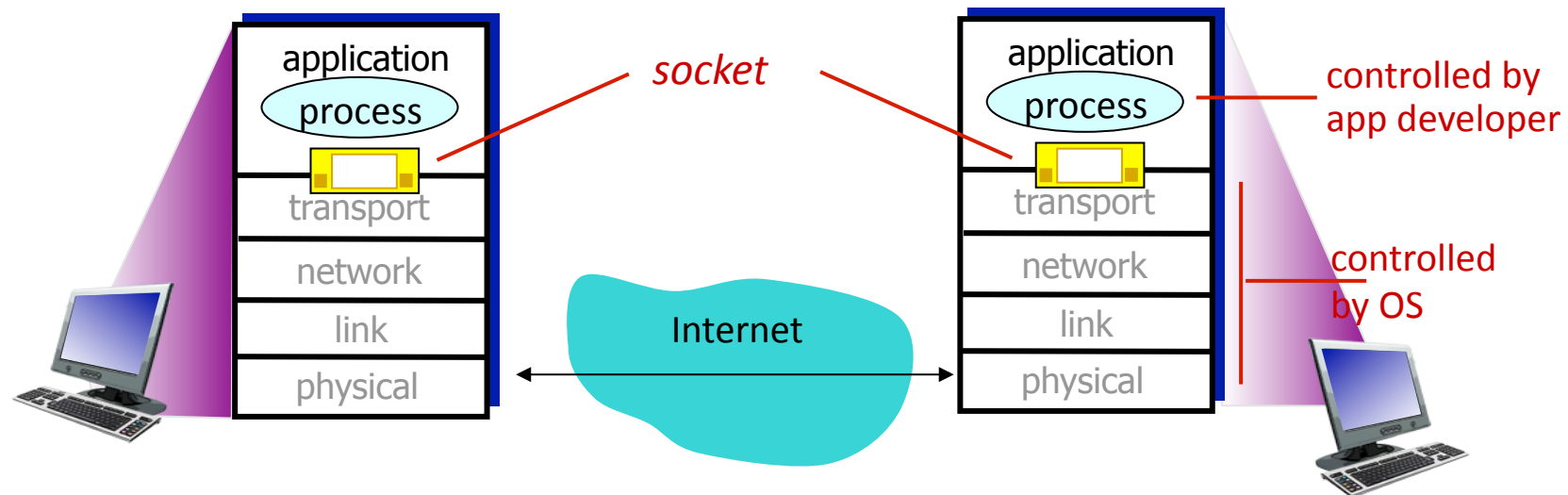
Dr. Peter Lian, Professor  
Department of Computer Science and Engineering  
York University

Email: [peterlian@cse.yorku.ca](mailto:peterlian@cse.yorku.ca)  
Office: 1012C Lassonde Building  
Course website: [http://wiki.cse.yorku.ca/  
course\\_archive/2012-13/W/3214](http://wiki.cse.yorku.ca/course_archive/2012-13/W/3214)

# Socket programming

**goal:** learn how to build client/server applications that communicate using sockets

**socket:** door between application process and end-end-transport protocol

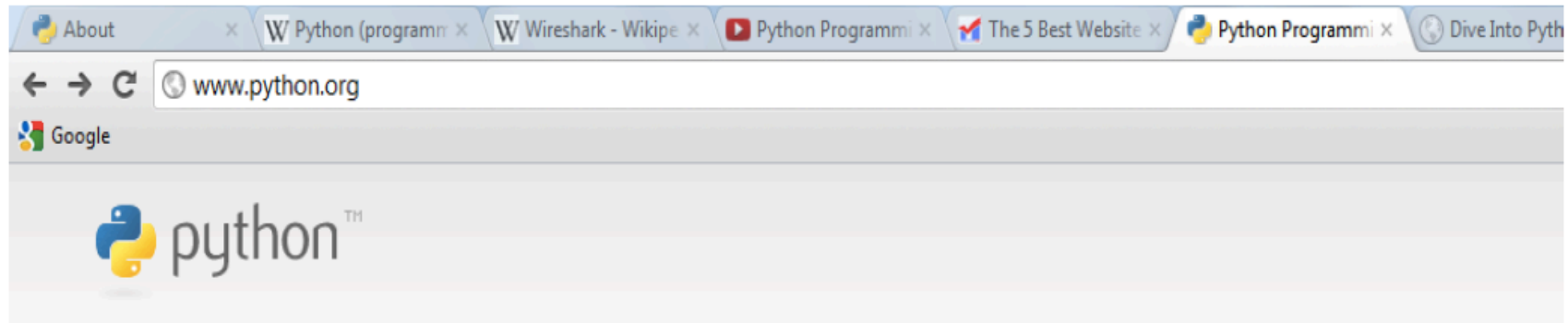


# Socket programming using Python

- Python is a general purpose, high level programming language
- Clear and expressive syntax
- Large and comprehensive library
- Used as scripting language as well as in a wide range of non-scripting contexts
- Available to Windows, Mac, Linux/Unix
- Official Website: <http://www.python.org>

# Socket programming using Python

## Download Python



ABOUT >>

NEWS >>

DOCUMENTATION >>

DOWNLOAD >>

COMUNITY >>

FOUNDATION >>

FOUNDATION >>

CORE DEVELOPMENT >>

Help

Package Index

Quick Links (2.7.3)

» Documentation

» Windows Installer

### Python Programming Language – Official Website

**Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.**

Python runs on Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines.

Python is free to use, even for commercial products, because of its OSI-approved [open source license](#).

New to Python or choosing between Python 2 and Python 3? Read [Python 2 or Python 3](#).

The [Python Software Foundation](#) holds the intellectual property rights behind Python, underwrites the [PyCon conference](#), and funds other projects in the Python community.

[Read more](#), -or- [download Python now](#)

# Socket programming using Python



The image shows a screenshot of a Python Shell window. The window title is "76 \*Python Shell\*". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area contains the following output:

```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> print "hello world"
hello world
>>>
```

The status bar at the bottom right of the window shows "Ln: 5 Col: 4".

Application Layer

# Socket programming using Python

---

*goal:* learn how to build client/server applications that communicate using sockets

*socket:* door between application process and end-end-transport protocol

# Socket programming

---

*Two socket types for two transport services:*

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

## *Application Example:*

1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

# Socket programming *with UDP*

## UDP: no “connection” between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- rcvr extracts sender IP address and port# from received packet

## UDP: transmitted data may be lost or received out-of-order

## Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server



# Client/server socket interaction: UDP

## server (running on serverIP)

create socket, port= x:  
`serverSocket =  
socket(AF_INET,SOCK_DGRAM)`

↓  
read datagram from  
`serverSocket`

↓  
write reply to  
`serverSocket`  
specifying  
client address,  
port number

## client

create socket:  
`clientSocket =  
socket(AF_INET,SOCK_DGRAM)`

↓  
Create datagram with server IP and  
port=x; send datagram via  
`clientSocket`

↓  
read datagram from  
`clientSocket`

↓  
close  
`clientSocket`

# Example app: UDP client

## *Python UDPClient*

include Python's socket library

→ from socket import \*  
serverName = 'hostname'  
serverPort = 12000

create UDP socket for server

→ clientSocket = socket(AF\_INET, SOCK\_DGRAM)

get user keyboard input

→ message = raw\_input('Input lowercase sentence:')

Attach server name, port to message; send into socket

→ clientSocket.sendto(message, (serverName, serverPort))

read reply characters from socket into string

→ modifiedMessage, serverAddress =  
clientSocket.recvfrom(2048)

print out received string and close socket

→ print modifiedMessage  
clientSocket.close()

# Example app: UDP server

## *Python UDPServer*

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    print message
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

create UDP socket →

bind socket to local port number 12000 →

loop forever →

Read from UDP socket into message, getting client's address (client IP and port) →

send upper case string back to this client →

# Socket programming *with TCP*

## client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

## client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

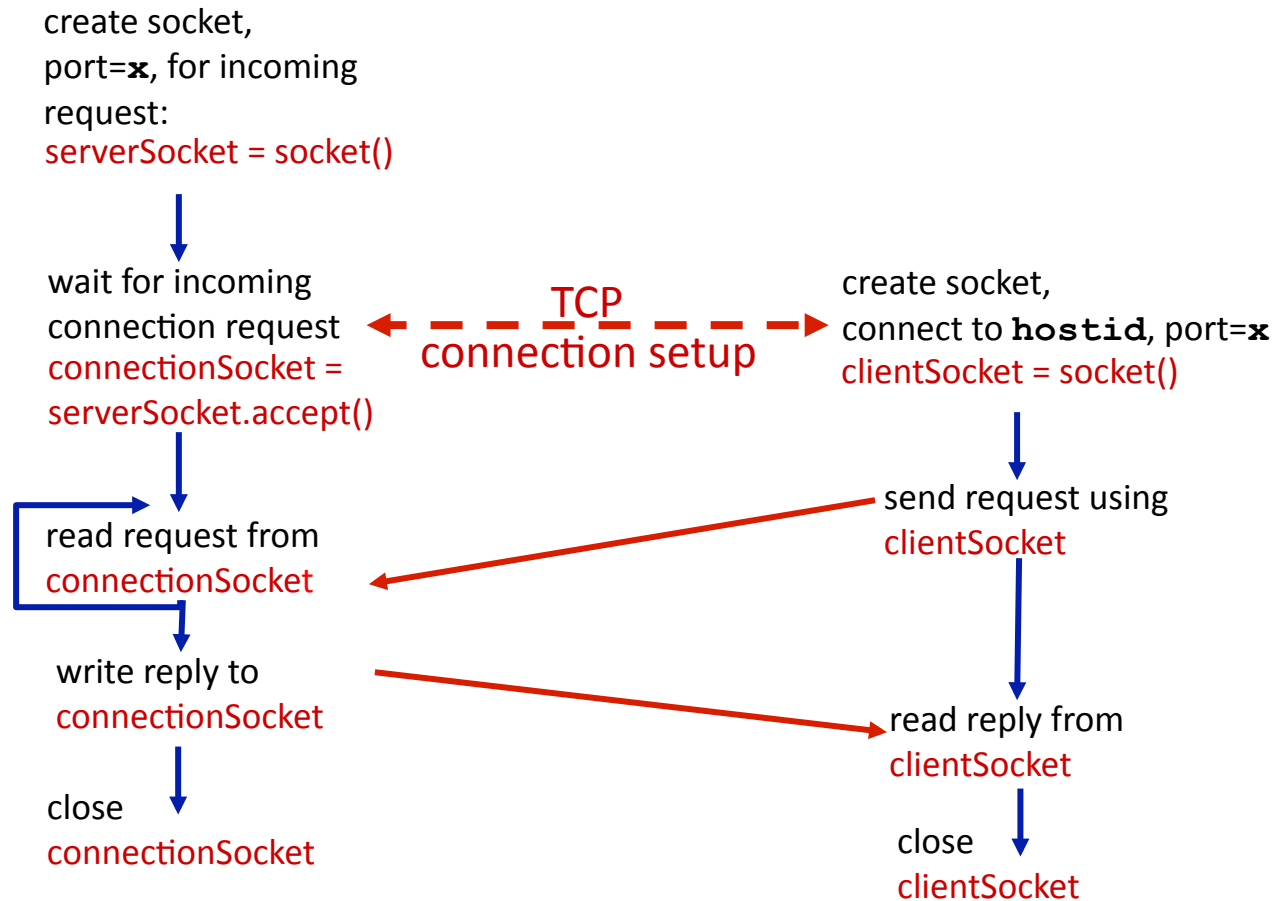
## application viewpoint:

TCP provides reliable, in-order byte-stream transfer (“pipe”) between client and server

# Client/server socket interaction:TCP

server (running on `hostid`)

client



# Example app:TCP client

## *Python TCPClient*

create TCP socket for server,  
remote port 12000

No need to attach server  
name, port

```
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)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

# Example app: TCP server

## *Python TCPServer*

```
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 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

create TCP welcoming socket →

server begins listening for incoming TCP requests →

loop forever →

server waits on accept() for incoming requests, new socket created on return →

read bytes from socket (but not address as in UDP) →

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