Basics of Python Sockets Programming

CS 356 – University of Texas at Austin

Dr. David A. Bryan

Overview

- Python is used extensively on the web…Dropbox, BitTorrent, backend of gmail and Google maps, Eve Online, many others have parts programmed in Python
- We are going to walk through some basic python commands EXTREMELY QUICKLY
  - Look this back over later…
  - Lots online about Python if you have questions
- Many things will be familiar from Java or other languages, some not as much
- Why python?
  - Socket programming is quite easy
  - Starting to be a very common language…

A Few Import Things

- I am showing you Python 3 (actually 3.4.3 on CS machines), the newer version
  - The book's section 2.7 uses the older Python 2
  - Not too many differences, but I will show them where they occur
- Python is case sensitive, like Java. Watch your case on commands, variable names, etc.
- Python **<u>really</u>** cares about indenting, as we will see
- I'm not a python expert by any means
  - I'm too old for that
    - (Heck, I still like C++)
  - If you are a Python expert and I do something stupid (certainly possible), please let me know!

Running Python

- Opening Python…
  - Command on CS machines is **`python3`**
  - **`python`** by itself will run 2.7.6
    - **`python -V`** will tell you the version – 2.7.3
    - (**`python3 -V`** gives back 3.4.3)
- You can run python and type in your code from the command line and interactively type in commands, but ugly, painful…
- Edit files in your favorite editor, then run them with **`python3 <filename>`**

Our First Script

- Minimum possible program
  - Print out Hello world!
- Save this in a file called helloworld.py
- Run it with:
  - **`python3 helloworld.py`**

```
print ('Hello world!')
```

Printing

```
print('Hello world!')
```

- `print` is the ***command*** or ***function*** in python – this is what we are asking the computer to do – in this case print a message
- Everything in parenthesis is being passed to the function. We are passing one thing, the text `'Hello world'`.
- Lastly, single quotes are used to mark off the start and end of the text…
- Python 2 (and the book) uses built-in print (which is not a function)
  - Basically the same, but without ( )s, so:
  - `print 'Hello world’`
  - That won’t work with Python 3

What About Variables?

- Let's put our message into a variable
- Our variable is called hellomsg. We "assign" the value 'Hello world!' to hellomsg
- Now we can pass that to the print as the argument instead of passing it just text…
- We will talk about types in python in just a second…ignore type for now. We don't need to declare it explicitly here in python.
- We could edit helloworld.py to look like this:

```
hellomsg = 'Hello world!'
print(hellomsg)
```

Combining Things for Print

- Plus sign can be used to concatenate (combine) things
  - (remember the space)
- Example: Change helloworld.py to use two variables and concatenate them:

```
hello = 'Hello'
world = 'world!'
print(hello + ' ' + world)
```

Getting input from the user

- We can also get the variable from the user, using the input command (you won't do this with server, but…)
- The argument to input is a prompt to the user
  - We will see the prompt printed to ask the user to enter something (there is a space after that ? in the prompt…)
    - If blank, nothing printed
  - Returns what the user types and assigns to the left hand side (hellomsg in this case)
- Again, in python 2 this was a bit different…called **`raw_input`**, so in book and python 2, you will see **`raw_input`** instead of **`input`**
- We can ask what to print in helloworld.py this way :

```
hellomsg = input('What should I print? ')
print(hellomsg)
```

A Bit on Types and Variables

- Not going to cover all the types and how they work, but we have seen variables are automatically created and we don't need to define a type
- Items still have a type!
- Everything that comes back from input is text (string) in python
- Numeric calculations produce numbers – integers and floats in python
- Need to be able to convert to use them
- We will also see shortly some things come as binary (the "bytes" type) – most notably things from sockets!
- Lists and such exist – you are welcome to use them but not covering them here

So what about numbers?

- Mostly what you expect, but need to be convert back and forth vs. strings:

```
firstgrade = 88
secondgrade = 90
thirdgrade = 81

total = firstgrade + secondgrade + thirdgrade

print('Total is: ' + str(total))
```

What's this str?

Math…

- So what did we do here?
- Created three variables called **`firstgrade`**, **`secondgrade`** and **`thirdgrade`**.
- Since we put numbers into those variables (in this case integers), python made them integers
  - It's automatic
- Added them up and put the answer into another variable called **`total`** (another integer)
- Then we passed total to print…and saw the answer
- The **`str()`** thing is needed to turn the number back into text (a string)…print, like input, expects strings!

More on Math and Types

- Things are pretty much what you expect
  - + adds (it concatenates for strings!)
  - -, /,* all do the usual thing
  - <, >, >=, etc. all do the usual thing
- Assume **`variableb`** and **`variablec`** are integers
  - **`variablea = variableb + variablec`**
    - variablea will be an integer
  - **`variablea = variableb / variablec`**
    - variablea is a float
  - Again, python makes things the right type automatically…
- Can see what something is with type() command:
  - **`type (variablea)`**

Converting Strings

- Strings need to be converted to numbers if we want to use them for math
  – **`int()`** and **`float()`** functions take a string, return an integer or float
  – sort of like the reverse of the **`str()`** we saw earlier

Controlling Flow

- Python has if statements, with else
- Loops, including both for and while
- Python doesn't use { } to mark start of loop parts, it uses indents
- Indents really matter in python, not just to make things pretty
- Did I mention indents matter?

Simple example

- What if we want to know if a temperature (in C) is freezing? Remember that anything above 0C is not freezing, and anything below 0C is freezing.
  - (*I will not apologize for using Celsius. Deal with it.*)
  - Use an if! Don't forget to convert to float.

```
temperature = input('What is the temp (C)? ')
temperature = float(temperature)

if temperature <= 0:
   print('Freezing!')
else:
   print('Not freezing!')
```

The parts here...

Test this...true or false?

```
temperature = input('What is the temp (C)? ')
temperature = float(temperature)

if temperature <= 0:
    print('Freezing!')
else:
    print('Not freezing!')
```

Notice there is a colon

Do this if true...

Do this if false...

Indent really matters!

Again: Indent really matters!

Simple example...continued

- You can do more than one thing
  - again as long as it is indented

```
temperature = input('What is the temp (C)? ')
temperature = float(temperature)

if temperature <= 0:
   print('Freezing!')
   print('Better wear a coat!')
else:
   print('Not freezing!')
   print('I guess you can skip the coat')
```

colons here

"Nesting" ifs

- You can also test inside a test...
- What if we want to check below 0 (parka), 0-15 (jacket) or above 15 (no coat)?

```
temperature = input('What is the temp (C)? ')
temperature = float(temperature)

if temperature <= 0:
    print('Freezing!')
    print('Better wear a parka!')
else:
    if temperature <=15:
        print('Cold, but not freezing')
        print('Wear a jacket')
    else
        print('Not cold!')
        print('I guess you can skip the coat')
```

Python Loops

- Like Java, we have different loops. I'll cover while here, you can look up for loops if you want to (will use later)
- Here we use a variable to store a number, and count to that number in a loop

```
counter = 1


while counter <= 5 :
  print(counter)
  counter = counter + 1
```

Indent matters again!

(`counter += 1` ***Works too)***

What Happened Here?

- We set our counter to 1
- Everything indented after the **`while`** is part of the loop – it is done “while” the loop is true
- The test of the loop is next to the **`while`** – we keep doing it while it is true
  - In this case, kept doing it while counter was less than or equal to 5
- Inside the loop (the indented stuff) we printed the counter, then added one to it
- Loop repeats until the test is false

Comments

- Any line starting with a # is a comment:

```
# this is a comment
# so is this
# need a # for each line
# next line isn't a comment
print ('Hello world!')
```

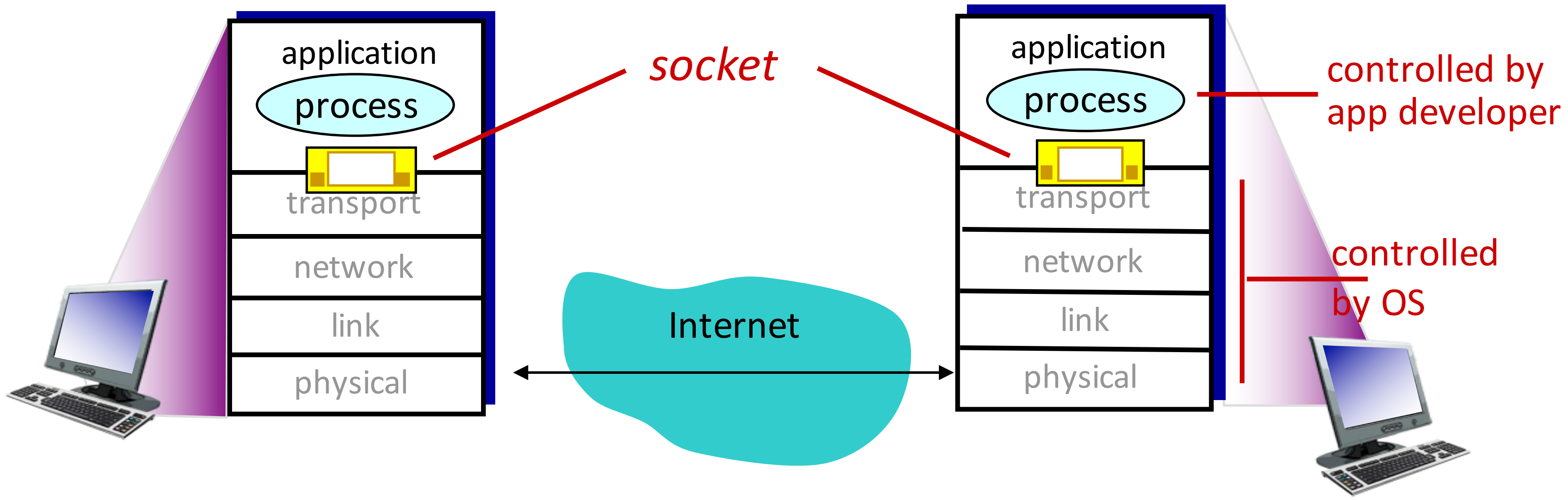
Socket Programming

- So with those basics…let's turn to sockets, which we will use for our project

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

Two socket types for two transport services:

- *UDP:* unreliable datagram
  - Throw it out the door and hope for the best
- *TCP:* reliable, byte stream-oriented
  - Under the hood (transparently) make sure the messages arrive
- We will look at HOW UDP and TCP actually work in the next chapter
- Will look at UDP sockets first (a bit easier), but will use TCP for the project

Socket programming

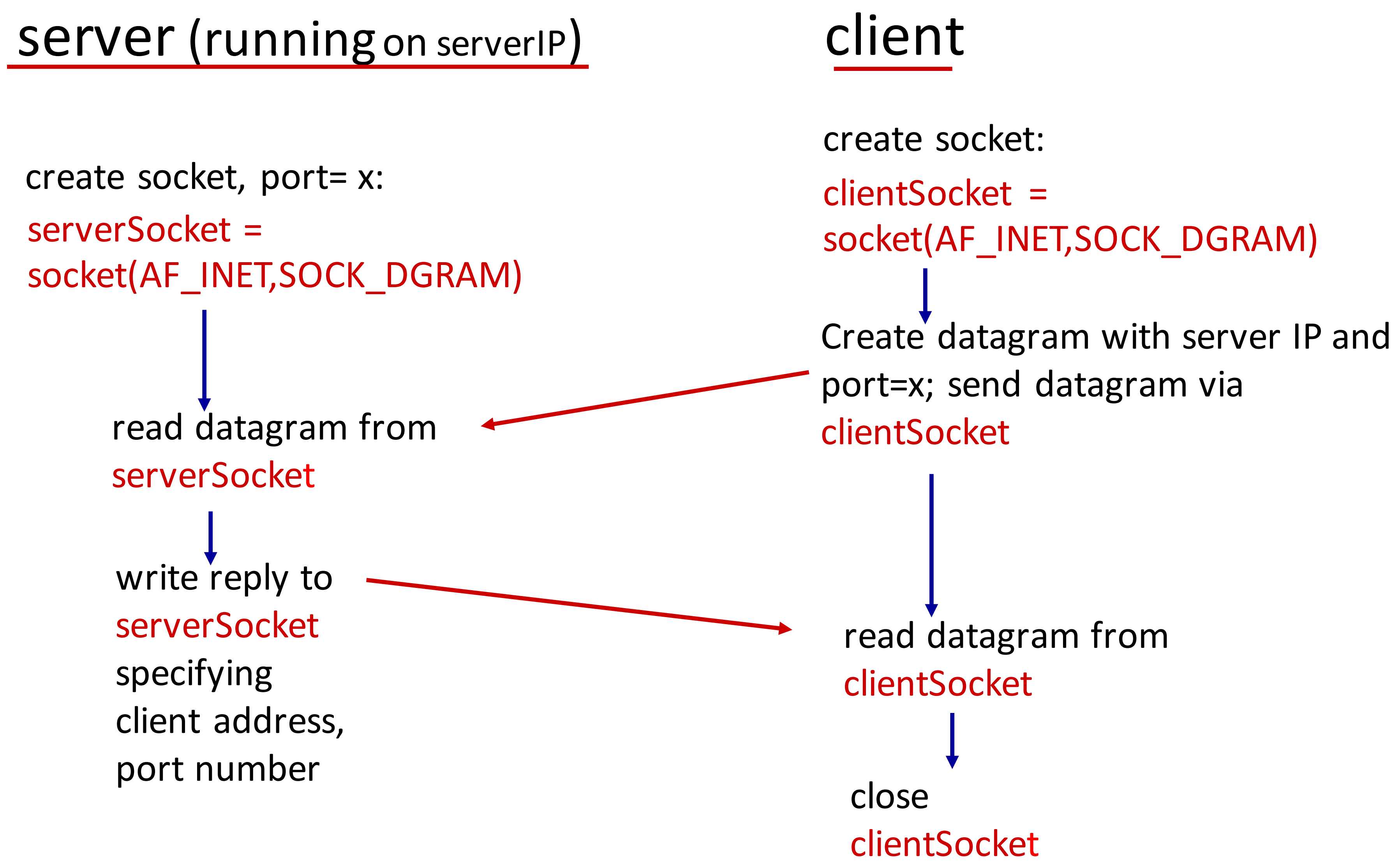
Application Example:

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.
5. In Python 3, things from sockets are in binary – type bytes – some things we want to do can only be done to str, so sometimes will have to convert!

Socket Programming with UDP

- We are doing TCP for our server, but quick look at UDP first
- UDP: no “connection” between client & server
  - no handshaking before sending data
  - sender explicitly attaches IP destination address and port # to each packet
  - receiver 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 = 'localhost'
serverPort = 12000
```

create UDP socket to server

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

get user keyboard input

```
message = input('Input lowercase sentence: ')
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message.encode(),(serverName, serverPort))
```

read reply characters from socket into string

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```

print out received string and close socket

```
print (modifiedMessage.decode())
clientSocket.close()
```

A Few Things

- What is happening here?
- We also need to look at a few things that are different from the book…
- Let's look in some detail here…

Example app: UDP client

Python UDPClient

local machine alias is localhost.
This could also be remote (mentos.cs.utexas.edu)

include Python's socket library

```
from socket import *
serverName = 'localhost'
serverPort = 12000
```

create UDP socket for client

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

get user keyboard input

```
message = input('Input lowercase sentence: ')
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message.encode(),(serverName, serverPort))
```

read reply characters from socket into string

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```

print out received string and close socket

```
print (modifiedMessage.decode())
clientSocket.close()
```

Example app: UDP client

Python UDPClient

We picked a high level port to use...12000 here. You can't open below 1025

include Python's socket library →

```
from socket import *
serverName = 'localhost'
serverPort = 12000
```

create UDP socket for server →

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

get user keyboard input →

```
message = input('Input lowercase sentence: ')
```

Attach server name, port to message; send into socket →

```
clientSocket.sendto(message.encode(),(serverName, serverPort))
```

read reply characters from socket into string →

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```

print out received string and close socket →

```
print (modifiedMessage.decode())
clientSocket.close()
```

Example app: UDP client

Python UDPClient

AF_INET means IPv4 socket
SOCK_DGRAM is datagram, or UDP

include Python's socket library →
```
from socket import *
serverName = 'localhost'
serverPort = 12000
```
create UDP socket for server →
```
clientSocket = socket(AF_INET, SOCK_DGRAM)
```
get user keyboard input →
```
message = input('Input lowercase sentence: ')
```
Attach server name, port to message; send into socket →
```
clientSocket.sendto(message.encode(),(serverName, serverPort))
```
read reply characters from socket into string →
```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```
print out received string and close socket →
```
print (modifiedMessage.decode())
clientSocket.close()
```

Example app: UDP client

Python UDPClient

include Python's socket library →

```
from socket import *
serverName = 'localhost'
serverPort = 12000
```

create UDP socket for server →

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

get user keyboard input →

```
message = input('Input lowercase sentence: ')
```

Attach server name, port to message; send into socket →

```
clientSocket.sendto(message.encode(),(serverName, serverPort))
```

read reply characters from socket into string →

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```

print out received string and close socket →

```
print (modifiedMessage.decode())
clientSocket.close()
```

bytes/str requires encoding/decoding!

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)
    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)

convert to uppercase

send upper case string back to this client

Again, A Few Things

- What is happening here?
- We also need to look at a few things that are different from the book…
- Let's look in some detail here…

Example app: UDP server

two single quotes...
an empty string...
use all our address (localhost will restrict to only requests sent there)

Python UDPServer

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)

convert to uppercase

send upper case string back to this client

```
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)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

Example app: UDP server

message comes back as bytes, bytes supports .upper(), send as bytes (no conversion)

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)
    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)

convert to uppercase

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

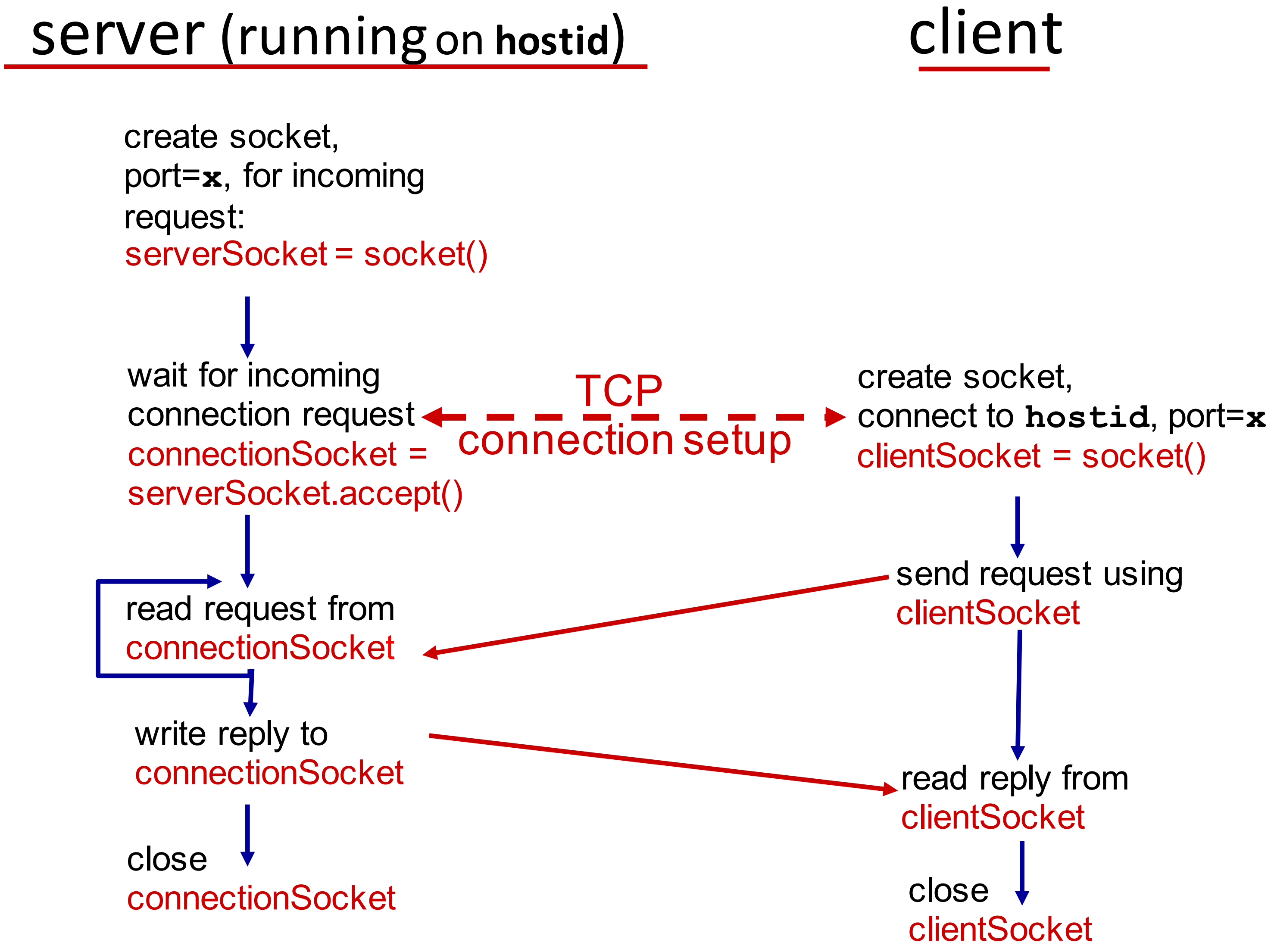
- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

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

THIS IS WHAT WE WILL USE FOR OUR WEB SERVER!!!

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 = 'localhost'
serverPort = 12000
```

create TCP socket for server, remote port 12000 $\rightarrow$

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

connect to the server $\rightarrow$

```
clientSocket.connect((serverName,serverPort))
```

```
sentence = input('Input lowercase sentence: ')
```

No need to include server name, port w/message – we are already connected – just send $\rightarrow$

```
clientSocket.send(sentence.encode())
```

```
modifiedSentence = clientSocket.recv(2048)
```

Receive message from the server, print it, and close socket $\rightarrow$

```
print ('From Server: ' + modifiedSentence.decode())
clientSocket.close()
```

Again, A Few Things

- What is happening here?
- We also need to look at a few things that are different from the book…
- Let's look in some detail here…

Example app: TCP client

Python TCPClient

SOCK_STREAM for a TCP socket

```
from socket import *
serverName = 'localhost'
serverPort = 12000
```

create TCP socket for server, remote port 12000

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

connect to the server

```
clientSocket.connect((serverName,serverPort))
```

```
sentence = input('Input lowercase sentence: ')
```

No need to include server name, port w/message – we are already connected – just send

```
clientSocket.send(sentence.encode())
```

```
modifiedSentence = clientSocket.recv(2048)
```

Receive message from the server, print it, and close socket

```
print ('From Server: ' + modifiedSentence.decode())
clientSocket.close()
```

Example app: TCP client

Python TCPClient

bytes/str requires encoding/decoding!

```
from socket import *
serverName = 'localhost'
serverPort = 12000

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

sentence = input('Input lowercase sentence: ')
clientSocket.send(sentence.encode())

modifiedSentence = clientSocket.recv(2048)
print ('From Server: ' + modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000 → `clientSocket = socket(AF_INET, SOCK_STREAM)`

connect to the server → `clientSocket.connect((serverName,serverPort))`

No need to include server name, port w/message – we are already connected – just send → `clientSocket.send(sentence.encode())`

Receive message from the server, print it, and close socket → `print ('From Server: ' + modifiedSentence.decode())`

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(2048)

    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

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

server begins listening for incoming TCP requests → `serverSocket.listen(1)`

loop forever → `while 1:`

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(2048)`

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

Again, A Few Things

- What is happening here?
- We also need to look at a few things that are different from the book…
- Let's look in some detail here…

Example app: TCP server

Python TCPServer

TWO sockets…one to listen for new connections, new one for each connection!

```
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(2048)

    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)

Let's take a look at the project…

Now…a Few Tools

- Here are a few things you will need for a web server
  - How do we open/read files?
    - This shows us exceptions too…
  - How do we split a big message into lines?
  - How do we split up each line?
  - How can we trim the first character of something off?

Reading a (text) File in Python

- Here is how to open a ***text*** file and read it
  - Can use a loop and readline() to go line by line
- You will want to be sending, but here we print…
- For binary (for pictures), you will need to do some research!

```
filename = 'foo.txt'

try:
     inputfile = open (filename, 'r')

except IOError:
            print ('File Not Found')

contents = inputfile.read()
print(contents)
```

Splitting Lines From a Message

- HTTP messages are terminated with carriage return and linefeed
- When you from socket…one big blob of bytes…
- First need to turn bytes into str with decode, then split it into a list of individual lines…

```
rawMessage = connectionSocket.recv(8192)
message = bytes.decode(rawMessage)
lines = message.split('\r\n')
```

- Now you can use a for loop (see the previous slide) to read each line, or can get at each line individually:

```
firstline = lines[0]
secondline = lines[1]
```

Splitting Each Line

- The split function took an argument of what to split on – we used **`'\r\n'`** above, but can also use it to split on spaces:

```
message = 'hello there'
parts = message.split(' ')

print('First part: ' + parts[0])
print('Second part: ' + parts[1])
```

Splitting Each Line…Count?

- Also good to check how many things you got after you split – will need that to check some things!

```
message = 'hello there'
parts = message.split(' ')

# Check how many parts there are
print(len(parts))

# We were expecting two, so let's check
if (len(parts) != 2):
    print('Something bad happened!')
```

- You can check the length of any list you get back from split

Trimming First Character

- You will need to do this…trust me
- You can use an index into a str. Say we have:

```
mystring = 'hellothere'
```

to skip the first character, use [1:]

```
newstring = mystring[1:]
print (newstring)
```

This prints 'ellothere'

```
newstring = mystring[2:]
print (newstring)
```

This will print 'llothere' – skipping first 2

Some useful libraries

- You can ***ONLY*** use a few libraries for the project
  - There is a built in webserver. No, you can't use it.
- **`socket`**, of course
- **`time`** or **`datetime`** will be needed to get dates
- You can also use (but don't need)
  - **`re`** (regular expressions)
  - **`os.path`** and/or **`os.stat`** (to get file information)
- ***DON'T USE ANY OTHER LIBRARIES OR UTILITIES UNLESS YOU GET APPROVAL FROM ME!***