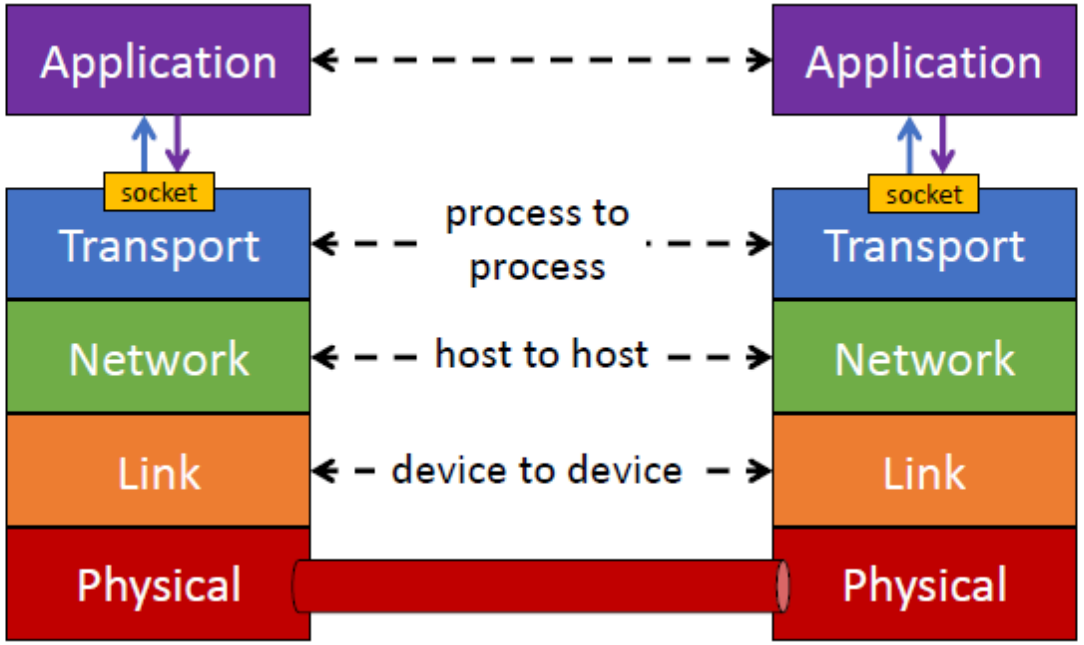
CS2105 Introduction to Computer Networks

Recap

- Sockets
  - Interface through which a process (application) communicates with the transport layer



Recap

- Socket programming
  - TCP
    - Stream socket
    - Connection Oriented
      - Reliable
  - UDP
    - Datagram socket
    - Connectionless
      - Unreliable

Recap

Python UDPClient

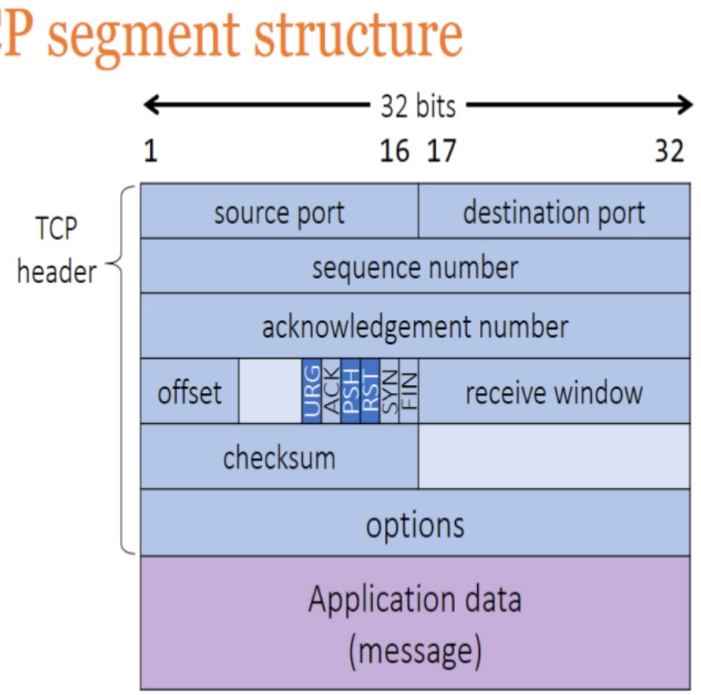
include Python's socket library →
```
from socket import *
```
```
serverName = 'hostname'
serverPort = 12000
```
create UDP socket for server →
```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```
get user keyboard input →
```
message = input('Input lowercase sentence:')
```
Attach 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()
```

Python TCPServer

```
from socket import *
serverPort = 12000
```
create TCP welcoming socket →
```
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
```
server begins listening for incoming TCP requests →
```
serverSocket.listen(1)
print('The server is ready to receive')
```
loop forever →
```
while True:
```
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(1024).decode()
    capitalizedSentence = sentence.upper()
```
close connection to this client (but *not* welcoming socket) →
```
    connectionSocket.send(capitalizedSentence.
                          encode())
    connectionSocket.close()
```

Recap

- TCP
  - Maintains a welcome socket
    - Acts an 'usher'
    - Does not directly participate in message passing
  - When contacted by client, server TCP will create a **new socket** for server process to communicate with client
    - Clients can hence be identified by the TCP connection
  - ➔ *n+1* sockets needed for *n* connections



Recap

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

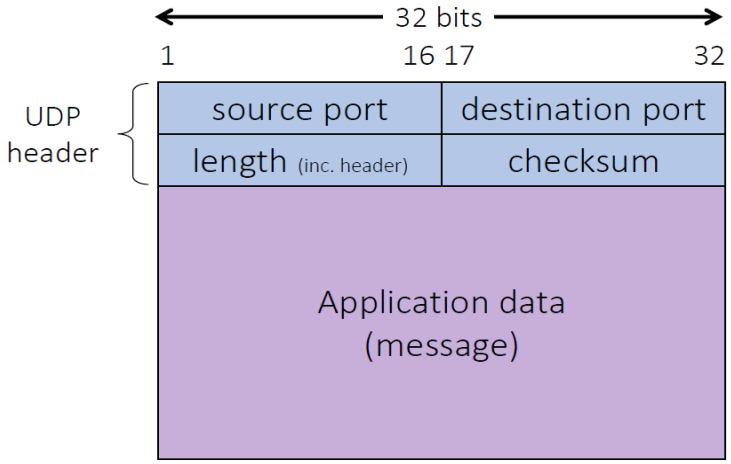
- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

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

Recap

- UDP
  - For each port #, only 1 socket to serve all connections
    - IP datagrams from multiple sources with the **same destination port #** will be directed to the **same UDP socket** at destination (like a mailbox)
  - Connectionless demultiplexing
    - Attach the destination IP address and port number to **each** datagram
  - To send responses to requests, extracts client's IP address from IP header
    - Client's port can be found here>>



Recap

- Checksum
  - Able to detect **single** bit flips
    - Sender
      - Compute checksum value and places the 1s complemented version in checksum field
    - Receiver
      - Computes its own variation of checksum and adds it to the checksum found
      - If there is a 0 anywhere ➔ corrupted!

Tutorial Questions

Question 1

Launch your browser and open its network diagnostic tool (e.g. press F12 if you use Chrome on Windows, or Cmd + Opt + I for Mac). Then click the "Network" tab to observe network communication. Copy-and-paste the following URL in the address bar of your browser: http://tiny.cc/atupaz. Enter your choice and press the "Submit" button.

a) Look at the entry named "formResponse". What is the HTTP request method issued?

POST

b) Briefly explain when HTTP POST and GET methods are used.

(From Wikipedia) The POST request method requests that a web server accepts and stores the data enclosed in the body of the request message. It is often used when uploading a file or submitting a completed web form. In contrast, the HTTP GET request method is designed to retrieve information from the server.

Question 2

[KR, Chapter 2, P21] Suppose that your department has a local DNS server for all computers in the department. You are an ordinary user (i.e. not a network/system administrator). Can you determine if an external Web site was likely accessed from a computer in your department a couple of seconds ago? Explain.

You may use 'dig' (domain information groper) program to query the local DNS server. For example,

dig –t a www.abc.com

If IP address of this Web page has been queried by another computer seconds ago, your local DNS server should keep this knowledge in local DNS cache and is able to answer your query quickly. Otherwise, the query time will be long.

RR Format: <name, value, type, TTL>

Type	Name	Value
A (adress)	Hostname	IP Address
NS (name server)	Domain, e.g nus.edu.sg	Hostname of authoritative name server for domain
CNAME (canonical name)	Alias for real name, e.g. www.comp.nus.edu.sg	The real name, e.g. www0.comp.nus.edu.sg
MX (mail exchange)	Domain of email address	Name of mail server managing the domain

Question 2

1st run:

```
clawyq@clinton:~$ dig -t a www.ntu.edu.sg

; <<>> DiG 9.11.3-1ubuntu1.3-Ubuntu <<>> -t a www.ntu.edu.sg
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49915
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.ntu.edu.sg.                        IN      A

;; ANSWER SECTION:
www.ntu.edu.sg.         15665   IN      A       155.69.7.173

;; Query time: 108 msec
;; SERVER: 172.19.215.140#53(172.19.215.140)
;; WHEN: Sat Feb 16 02:58:30 DST 2019
;; MSG SIZE  rcvd: 59
```

Question 2

2nd run:

```
clawyq@clinton:~$ dig -t a www.ntu.edu.sg

; <<>> DiG 9.11.3-1ubuntu1.3-Ubuntu <<>> -t a www.ntu.edu.sg
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57985
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.ntu.edu.sg.                        IN      A

;; ANSWER SECTION:
www.ntu.edu.sg.         15661   IN      A       155.69.7.173

;; Query time: 10 msec
;; SERVER: 172.19.215.140#53(172.19.215.140)
;; WHEN: Sat Feb 16 02:58:33 DST 2019
;; MSG SIZE  rcvd: 59
```

Question 3

[Modified from KR, Chapter 2, P31] You are given 4 programs: **TCPEchoServer.java**, **TCPEchoClient.java**, **UDPEchoServer.java** and **UDPEchoClient.java**. Compile them on sunfire.

a) Suppose you run **TCPEchoClient** before you run **TCPEchoServer**. What happens? Why?

When creating a local socket, client attempts to make a TCP connection to a non-existent server process. Exception will be thrown.

```
cl4wyq@sunfire [11:11:31] ~/cs2105/tut03files $ python3 TCPClient.py
Traceback (most recent call last):
  File "TCPClient.py", line 6, in <module>
    clientSocket.connect((serverName, serverPort))
ConnectionRefusedError: [Errno 146] Connection refused
```

b) Suppose you run **UDPEchoClient** before you run **UDPEchoServer**. What happens? Why?

UDP client doesn't establish connection to server when creating local socket. Thus it works fine if you start client program first and then server program (but data sent to server are all lost).

```
cl4wyq@sunfire [11:12:56] ~/cs2105/tut03files $ python3 UDPClient.py
Input lowercase sentence:
```

Question 4

[KR, Chapter 3, R7] Suppose a process in Host C has a UDP socket with port number 6789. Suppose both Host A and Host B each sends a UDP segment to Host C with destination port number 6789. Will both of these segments be directed to the same socket at Host C? If so, how will the process at Host C know that these two segments originated from two different hosts?

Yes, both segments will be directed to the same socket. DatagramPacket class provides a method getAddress() for us to learn the IP address of the origin of a packet.

The question is really asking: since UDP does not have a specific connection for each client for ease of identification, how can I tell where did the message come from?

By extracting the IP address from the IP header as discussed earlier.

Question 5

Suppose you have the following 2 bytes: **01011100** and **01100101**.
What is the 1's complement of the sum of these 2 bytes?

Sum: 11000001, checksum: 00111110

Suppose you have the following 2 bytes: **11011010** and **01100101**.
What is the 1's complement of the sum of these 2 bytes?

Sum: 01000000, checksum: 10111111

Question 6

[Modified from KR, Chapter 3, Problem 5] Suppose that UDP receiver computes the checksum for the received UDP segment and finds that it matches the value carried in the checksum field. Can the receiver be absolutely certain that no bit errors have occurred? You may use Q5 as an example to explain.

If sender transmits the following two bytes: 01011100 and 01100101, and the two highlighted bits flip, then checksum remains unchanged and receiver will fail to detect this error. Checksum cannot pick up multi-bit flips!

Question 7

Why is it that UDP takes 1's complement of the sum as checksum, i.e. why not just use sum? With the 1s complement scheme, how does the receiver detect errors?

Computation at receiver side is simplified. To detect errors, receiver adds all the 16-bit words (including the checksum). If the sum contains a zero, then receiver knows there has been an error. All one-bit errors will be detected, but some two-bit errors can be undetected. There is no need to compare received checksum with computed checksum bit by bit.

Question 8

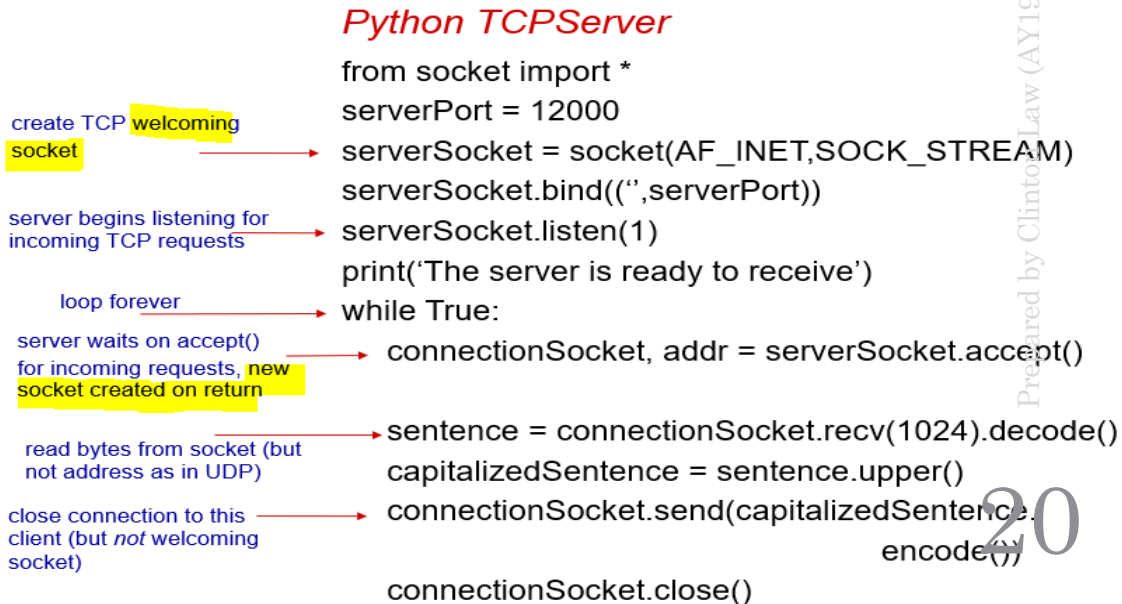
In our rdt protocols, why did we need to introduce sequence numbers?

Sequence numbers are required for a receiver to find out whether an arriving packet contains new data or is a retransmission.

Question 9

Assignment 1 key specifications:

- Implementation and design of a pseudo HTTP protocol
  - What are the valid requests and their associated responses?
- What does it mean to be persistent?
  - What are the things I must implement for persistency?
  - How can I make my code reflect this persistency?
- Think about what actually happens when you call a method.
  - E.g. what does calling .accept() actually do?
  - Reconcile this with what we've learnt in class



Summary

- Sockets
- TCP vs UDP
  - Methods for demultiplexing
- Checksum

Extra Questions

A UDP server needs only one socket to communicate with n different clients simultaneously. How many sockets would a TCP server have ever created for the same situation?

A. 1

B. n

C. $n+1$

D. $2n$

E. None of the above

Extra Questions

9. Consider the following Java code snippet.

```
Socket socket = new ServerSocket(2105).accept();
```

What port number is `socket` bound to when above statement finish execution?

A. It depends on the client's port that's making the connection.

B. TCP port 2105

C. Cannot say; it's operation system dependent and is usually a randomly chosen port.

D. UDP port 2105

E. None of the above

Extra Questions

[1 mark] Consider the following Java code snippet.

```
DatagramSocket s = new DatagramSocket(80);
byte[] buf = new byte[1000];
DatagramPacket pkt = new DatagramPacket(buf, buf.length);
s.receive(pkt);
```

Which of the following statement is TRUE?

A. `s` can be used for both sending and receiving TCP segments.

B. The code snippet will throw an exception at runtime, if port 80 is already in use.

C. A packet received must contain exactly 1000 bytes of data.

D. A packet received must contain at most 1000 bytes of data, inclusive of the size of UDP header.

E. HTTP requests can now be received through `s`.

Thank you!

Answers:
C, B, B