

Chapter 2

Application Layer

A S M Touhidul Hasan, Ph.D.
Assistant Professor,
Department of CSE, UAP

Chapter 2: Outline of Lecture 5

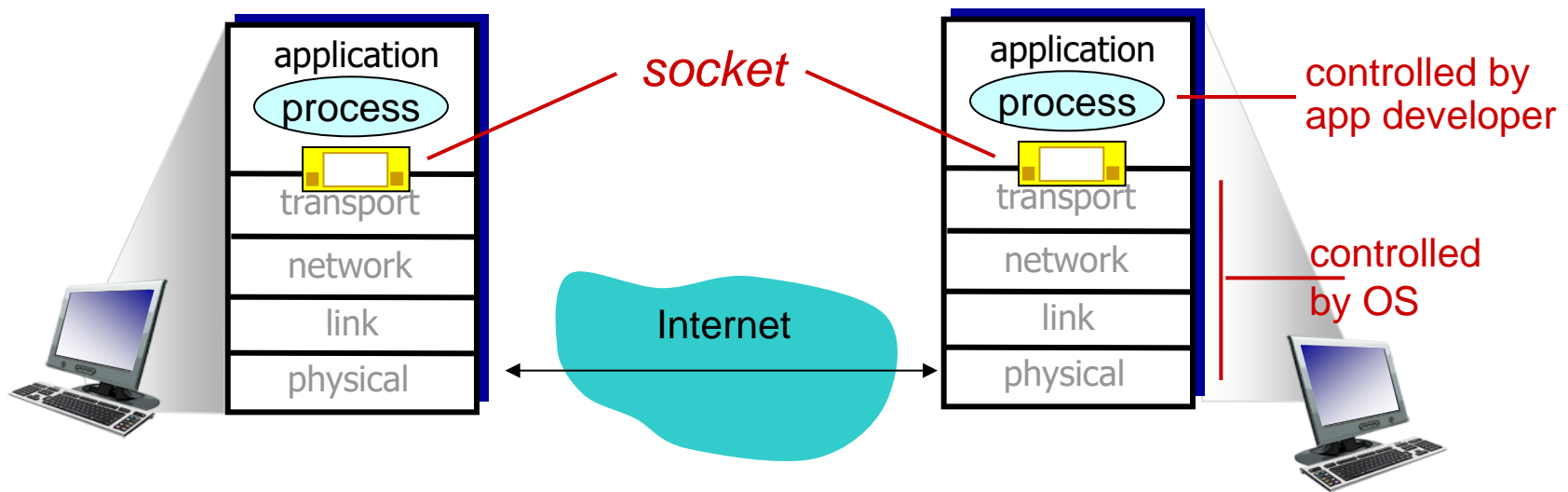
2.7 socket programming with
UDP and TCP

2.8 wireshark

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
- **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(socket.AF_INET,
socket.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
```

create UDP socket →

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

bind socket to local port
number 12000 →

```
serverSocket.bind(("", serverPort))
```

```
print "The server is ready to receive"
```

loop forever →

```
while 1:
```

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

```
message, clientAddress = serverSocket.recvfrom(2048)
```



```
modifiedMessage = message.upper()
```

send upper case string
back to this client →

```
serverSocket.sendto(modifiedMessage, clientAddress)
```


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)

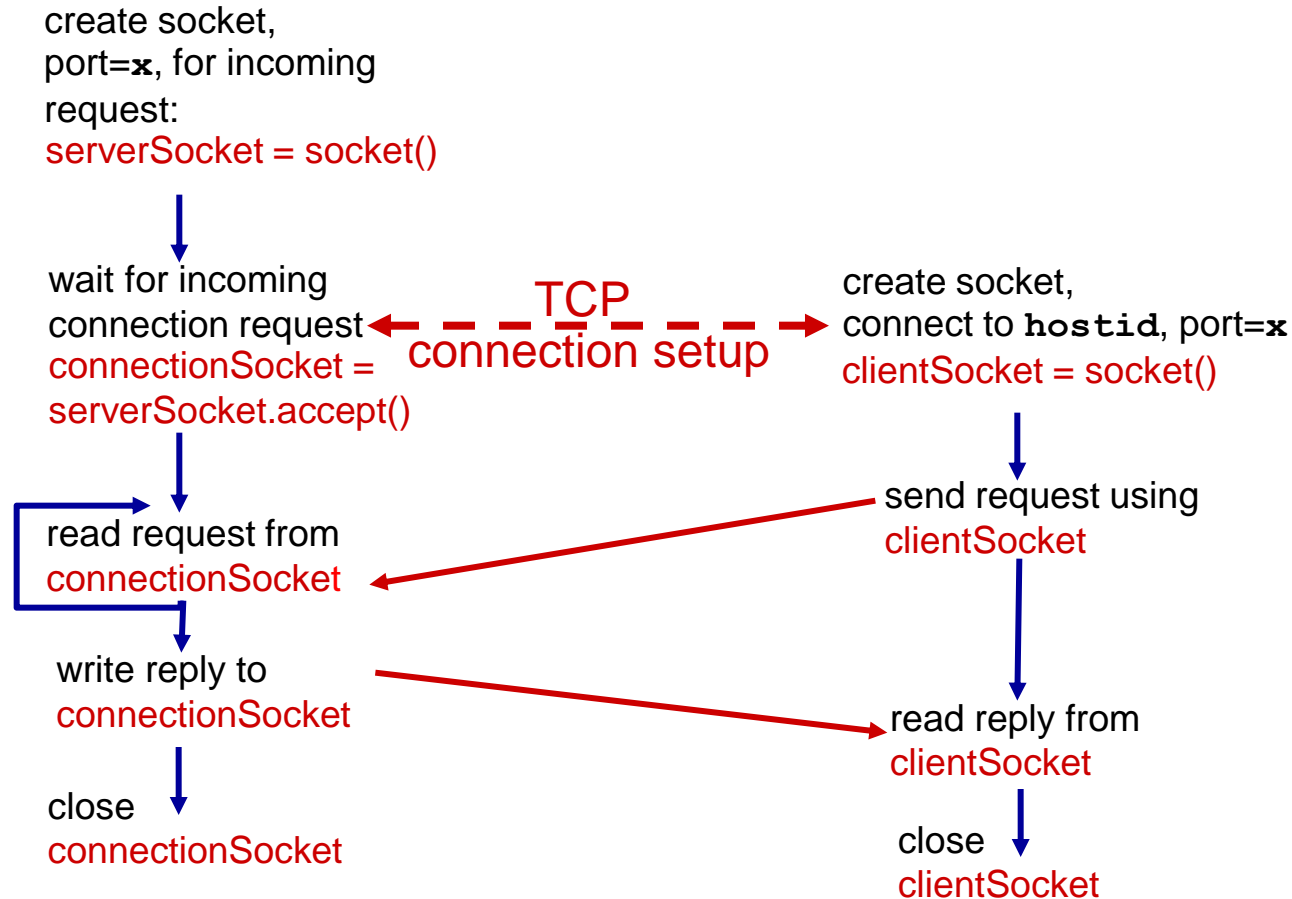
application viewpoint:

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 TCPCClient

```
from socket import *
```

```
serverName = 'servername'
```

```
serverPort = 12000
```

create TCP socket for
server, remote port 12000

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

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

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

No need to attach server
name, port

```
→ clientSocket.send(sentence)
```

```
modifiedSentence = clientSocket.recv(1024)
```

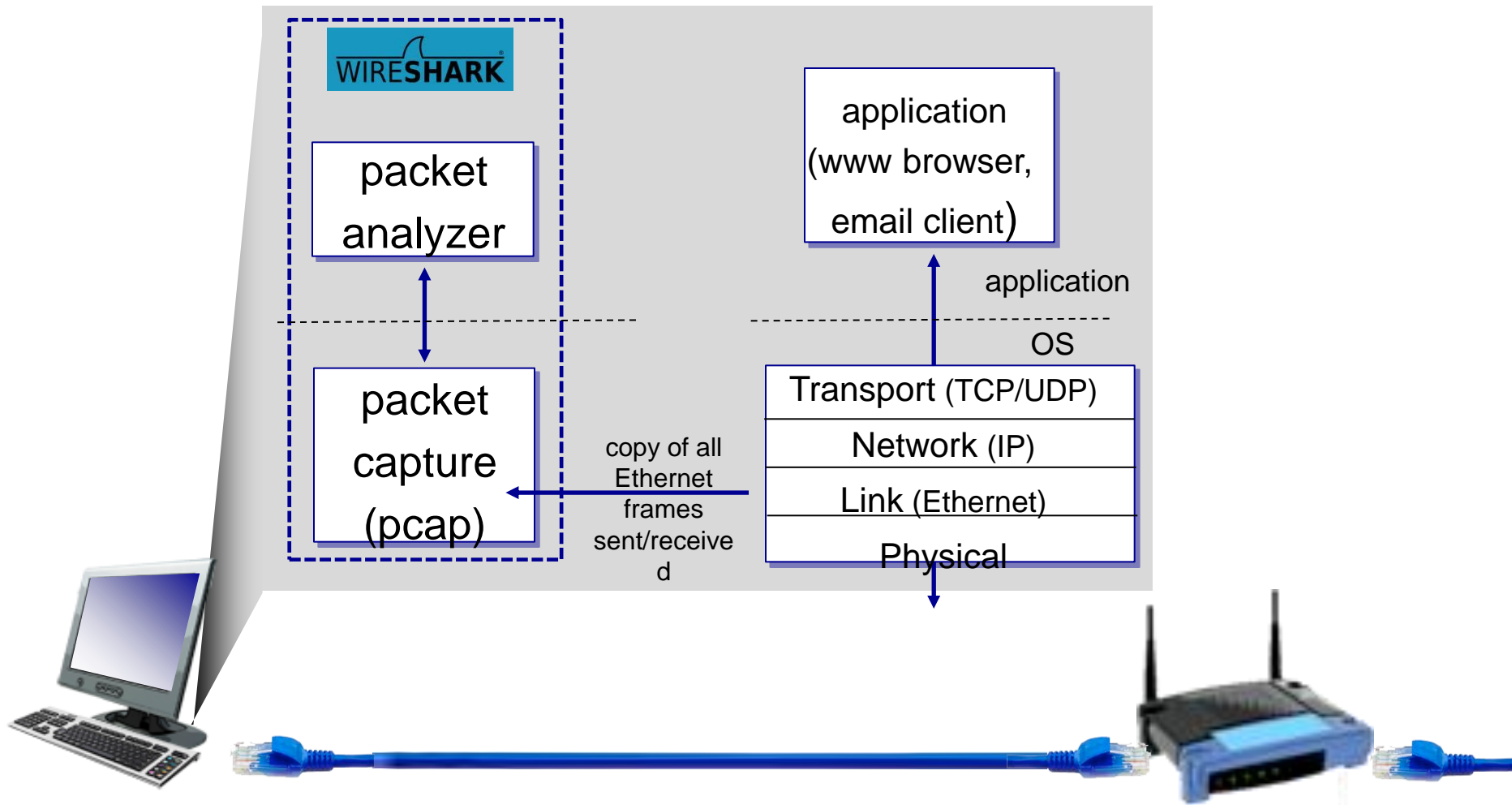
```
print 'From Server:', modifiedSentence
```

```
clientSocket.close()
```

Example app: TCP server

Python TCPServer

create TCP welcoming socket	→	<pre>from socket import * serverPort = 12000 serverSocket = socket(AF_INET, SOCK_STREAM) serverSocket.bind(('', serverPort)) serverSocket.listen(1) print 'The server is ready to receive'</pre>
server begins listening for incoming TCP requests	→	<pre>while 1:</pre>
loop forever	→	<pre> connectionSocket, addr = serverSocket.accept() sentence = connectionSocket.recv(1024) capitalizedSentence = sentence.upper() connectionSocket.send(capitalizedSentence) connectionSocket.close()</pre>
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 <i>not</i> welcoming socket)	→	



What is Wireshark?



- Wireshark is a network packet analyzer.
- A network packet analyzer presents captured packet data in as much detail as possible.
- You could think of a network packet analyzer as a measuring device for examining what's happening inside a network cable, just like an electrician uses a voltmeter for examining what's happening inside an electric cable (but at a higher level, of course).
- In the past, such tools were either very expensive, proprietary, or both. However, with the advent of Wireshark, that has changed. Wireshark is available for free, is open source, and is one of the best packet analyzers available today.

Purposes of Wireshark



- ❑ Network administrators use it to *troubleshoot network problems*
- ❑ Network security engineers use it to *examine security problems*
- ❑ QA engineers use it to *verify network applications*
- ❑ Developers use it to *debug protocol implementations*
- ❑ People use it to *learn network protocol* internals

Demonstration of Captured Packet

The image shows a Wireshark packet capture window titled "tv-netflix-problems-2011-07-06.pcap". The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help) and a toolbar with various icons for packet capture and analysis. A display filter bar at the top shows "Apply a display filter ... <Ctrl-/>".

The packet list pane displays a table of captured packets:

No.	Time	Source	Destination	Protocol	Length	Info
343	65.142415	192.168.0.21	174.129.249.228	TCP	66	40555 → 80 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=491519346 TSecr=551811827
344	65.142715	192.168.0.21	174.129.249.228	HTTP	253	GET /clients/netflix/flash/application.swf?flash_version=flash_lite_2.1&v=1.5&nr
345	65.230738	174.129.249.228	192.168.0.21	TCP	66	80 → 40555 [ACK] Seq=1 Ack=188 Win=6864 Len=0 TSval=551811850 TSecr=491519347
346	65.240742	174.129.249.228	192.168.0.21	HTTP	828	HTTP/1.1 302 Moved Temporarily
347	65.241592	192.168.0.21	174.129.249.228	TCP	66	40555 → 80 [ACK] Seq=188 Ack=763 Win=7424 Len=0 TSval=491519446 TSecr=551811852
348	65.242532	192.168.0.21	192.168.0.1	DNS	77	Standard query 0x2188 A cdn-0.nflximg.com
349	65.276870	192.168.0.1	192.168.0.21	DNS	489	Standard query response 0x2188 A cdn-0.nflximg.com CNAME images.netflix.com.edge
350	65.277992	192.168.0.21	63.80.242.48	TCP	74	37063 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=491519482 TSecr=
351	65.297757	63.80.242.48	192.168.0.21	TCP	74	80 → 37063 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=3295
352	65.298396	192.168.0.21	63.80.242.48	TCP	66	37063 → 80 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=491519502 TSecr=3295534130
353	65.298687	192.168.0.21	63.80.242.48	HTTP	153	GET /us/nrd/clients/flash/814540.bun HTTP/1.1
354	65.318730	63.80.242.48	192.168.0.21	TCP	66	80 → 37063 [ACK] Seq=1 Ack=88 Win=5792 Len=0 TSval=3295534151 TSecr=491519503
355	65.321733	63.80.242.48	192.168.0.21	TCP	1514	[TCP segment of a reassembled PDU]

The packet details pane for packet 349 (Frame 349: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits)) shows the following structure:

- Ethernet II, Src: Globalsc_00:3b:0a (f0:ad:4e:00:3b:0a), Dst: Vizio_14:8a:e1 (00:19:9d:14:8a:e1)
- Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.21
- User Datagram Protocol, Src Port: 53 (53), Dst Port: 34036 (34036)
- Domain Name System (response)
 - [Request In: 348]
 - [Time: 0.034338000 seconds]
 - Transaction ID: 0x2188
 - Flags: 0x8180 Standard query response, No error
 - Questions: 1
 - Answer RRs: 4
 - Authority RRs: 9
 - Additional RRs: 9
 - Queries
 - cdn-0.nflximg.com: type A, class IN
 - Answers
 - Authoritative nameservers

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0020 00 15 00 35 84 f4 01 c7 83 3f 21 88 81 80 00 01 ...5....?!....
0030 00 04 00 09 00 09 05 63 64 6e 2d 30 07 6e 66 6c .....c dn-0.nfl
0040 78 69 6d 67 03 63 6f 6d 00 00 01 00 01 c0 0c 00 ximg.com .....
0050 05 00 01 00 00 05 29 00 22 06 69 6d 61 67 65 73 .....). ".images
0060 07 6e 65 74 66 6c 69 78 03 63 6f 6d 09 65 64 67 .netflix .com.edg
0070 65 73 75 69 74 65 03 6e 65 74 00 c0 2f 00 05 00 esuite.n et../...
```

The status bar at the bottom indicates "Identification of transaction (dns.id), 2 bytes" and "Packets: 10299 · Displayed: 10299 (100.0%) · Load time: 0:0.182 | Profile: Default".

Figure . Wireshark captures packets and lets you examine their contents.

Online Resources of Wireshark

Site: <https://www.wireshark.org/>

User Guide: <https://www.wireshark.org/download/docs/user-guide.pdf>

Chapter 2: summary

our study of network apps now complete!

- ❖ application architectures
 - client-server
 - P2P
- ❖ application service requirements:
 - reliability, bandwidth, delay
- ❖ Internet transport service model
 - connection-oriented, reliable: TCP
 - unreliable, datagrams: UDP
- ❖ specific protocols:
 - HTTP
 - FTP
 - SMTP, POP, IMAP
 - DNS
 - P2P: BitTorrent, DHT
- ❖ socket programming: TCP, UDP sockets

Chapter 2: summary

most importantly: learned about protocols!

- ❖ typical request/reply message exchange:
 - client requests info or service
 - server responds with data, status code
- ❖ message formats:
 - headers: fields giving info about data
 - data: info being communicated

important themes:

- ❖ control vs. data msgs
 - in-band, out-of-band
- ❖ centralized vs. decentralized
- ❖ stateless vs. stateful
- ❖ reliable vs. unreliable msg transfer
- ❖ “complexity at network edge”