Lecture 4: Sockets and system programming

COMP 332, Spring 2018

Victoria Manfredi

WESLEYAN
UNIVERSITY

Acknowledgements: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved as well as from slides by Abraham Matta at Boston University and some material from Computer Networks by Tannenbaum and Wetherall.

Today

1. Announcements
   - homework 1 due today, homework 2 posted
     - tictactoe.py code for homework2 will be posted once homework1 turned in

2. Network applications

3. Network programming
   - TCP sockets
   - UDP sockets

4. Network tools
   - netstat: what connections do you have open
   - netcat: incredibly flexible and useful network tool
   - Wireshark: looking at real traffic

vumanfredi@wesleyan.edu

Network Applications

OVERVIEW

vumanfredi@wesleyan.edu

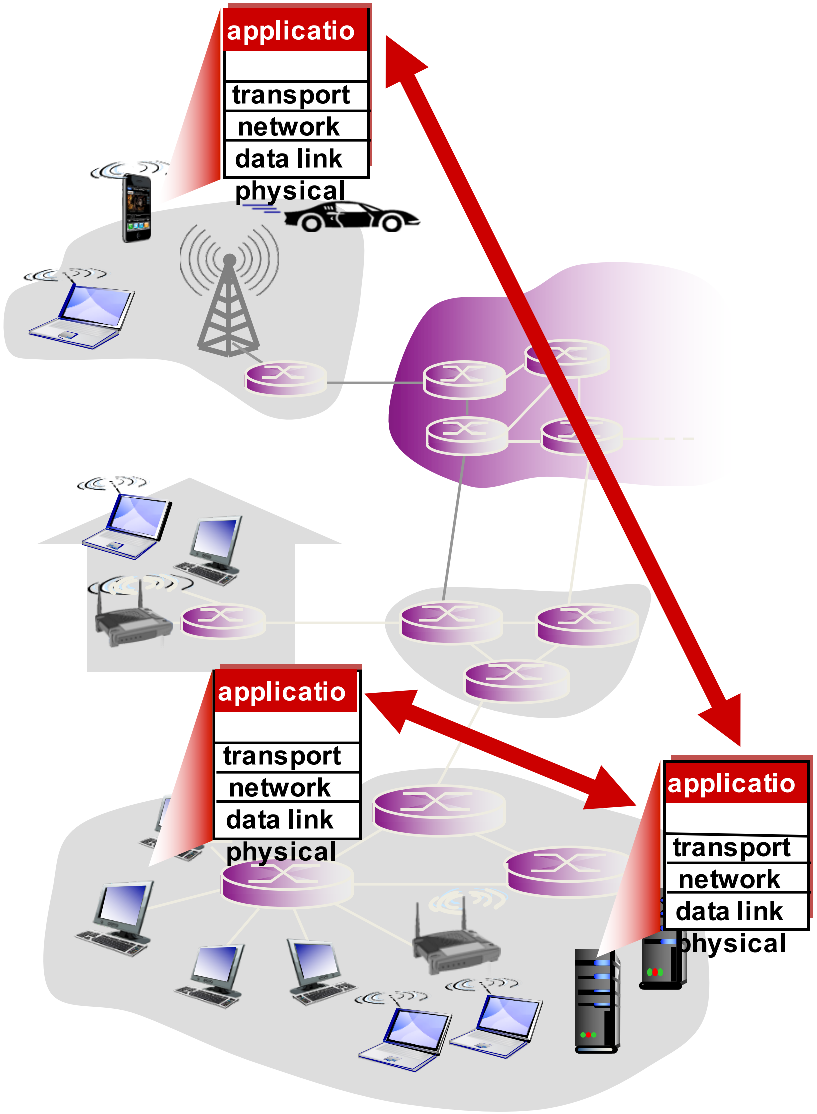
Creating a network app

Write programs that

- run on (different) end systems
- communicate over network
- e.g., web server software communicates with browser software

Q: Do we need to write software for network-core devices?

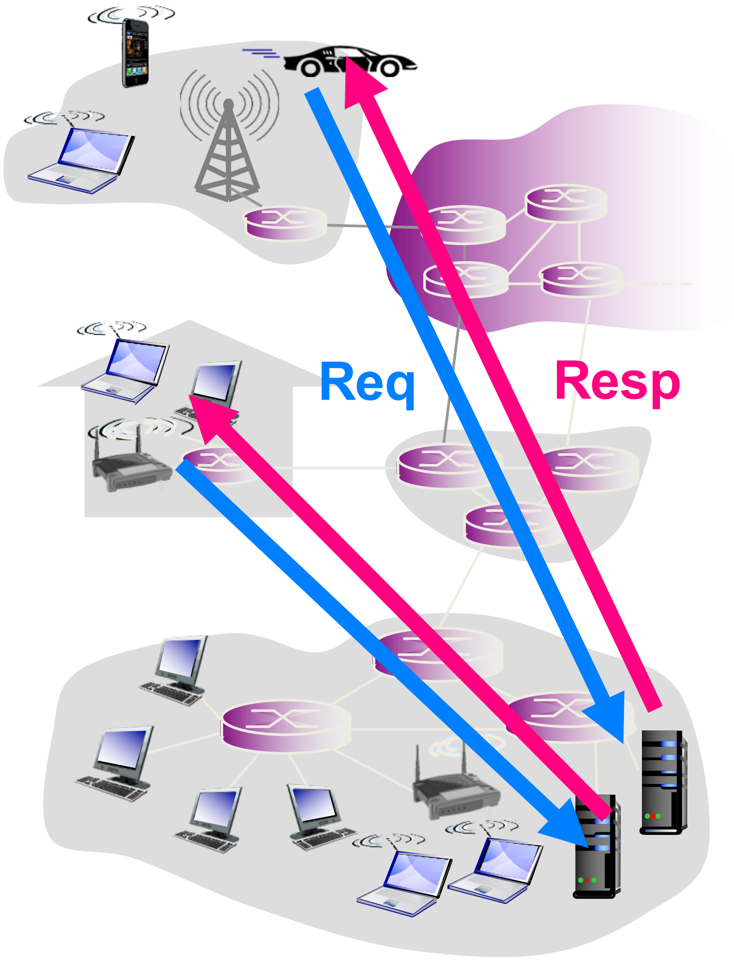
- No, network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



vumanfredi@wesleyan.edu

Client-server architecture

Client host requests and receives service
from always on server host



Server

- always-on, dedicated host
  - e.g., web server
- permanent IP address
- data centers for scaling

Clients

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with other clients

Client and server devices
are not equivalent

vumanfredi@wesleyan.edu

Peer-to-peer (P2P) architecture

Peers request service from other peers, provide service in return to other peers

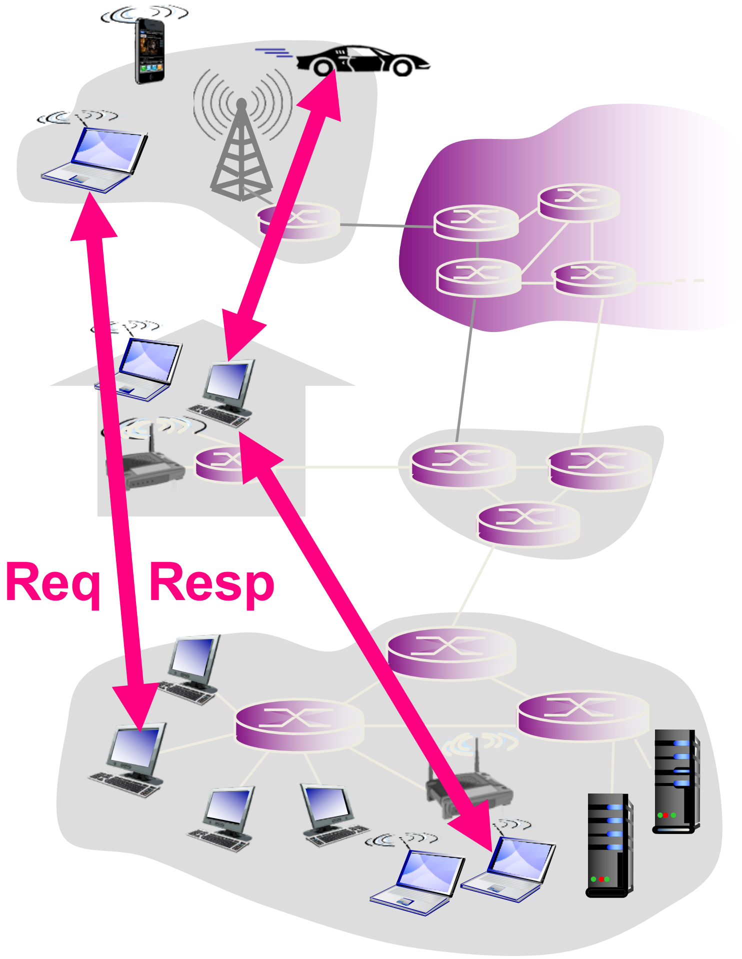
End systems directly communicate

- self scalability – new peers bring new service capacity, as well as new service demands
- minimal/no use of always-on server
- E.g., Skype, BitTorrent

Complex management

- peers are intermittently connected and change IP addresses
- Q: why is this complex?

All devices are equivalent: a client can also be a server



vumanfredi@wesleyan.edu

Processes communicating

Process

- program in execution, running within a host

Processes within same host

- communicate by using inter-process communication (defined by OS)

Processes on different hosts

- communicate by exchanging messages

Clients, servers

- client process
  - process that initiates communication
- server process
  - process that waits to be contacted

Aside

- applications with P2P architectures also have client & server processes

Our goal learn how to build client/server applications that use sockets to communicate

vumanfredi@wesleyan.edu

Network Programming

OVERVIEW

vumanfredi@wesleyan.edu

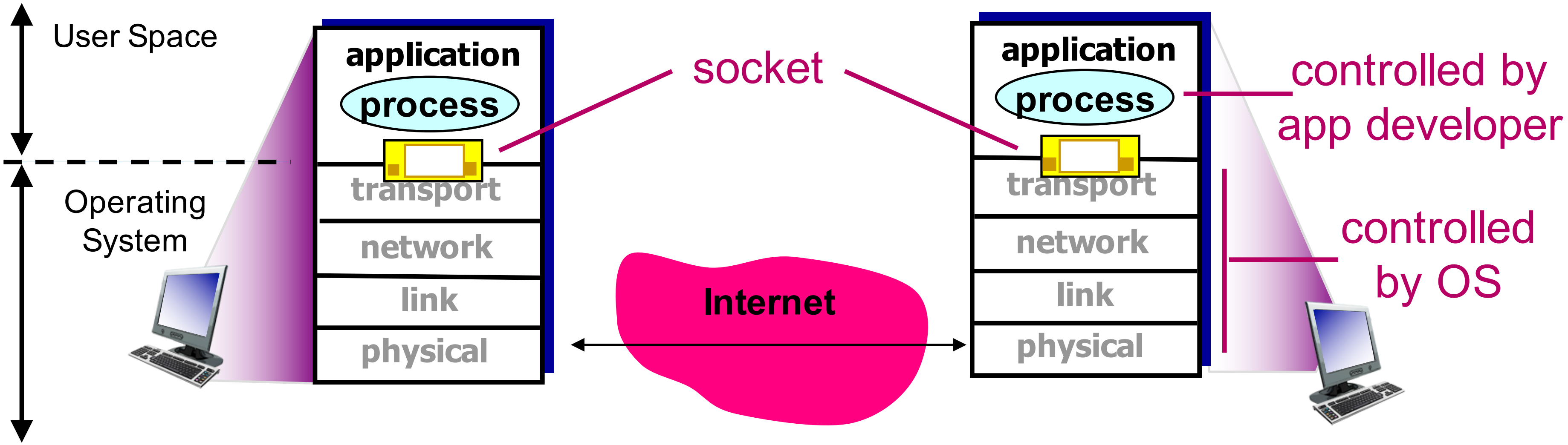
Application Programming Interface

Socket

- interface that transport layer provides to apps to access network
- analogous to door
  - sending process shoves msg out door, relies on transport infrastructure on other side of door to deliver msg to socket at receiving process

Client and server processes

- send/receive messages to/from their respective sockets



vumanfredi@wesleyan.edu

Python socket module

import socket

- gives access to BSD (Berkeley Socket Distribution) socket interface
  - POSIX sockets <-> Berkeley sockets <-> BSD sockets
  - available on pretty much every modern operating system

Resources

- https://docs.python.org/3/howto/sockets.html
- https://docs.python.org/3/library/socket.html

Socket exceptions

- https://docs.python.org/3/library/socket.html#exceptions

You must read/write bytes from/to a socket

- encode string to bytes: string.encode('utf-8')
- decode string from bytes: string.decode('utf-8')

Sockets

Address families

- AF_UNIX
    - local, inter-process communication
- AF_INET4
    - Internet protocol v4
- AF_INET6
    - Internet v6

Part of process identifier:
e.g., <ip address, port>

To send HTTP message to wesleyan.edu web server

- IP address: 129.133.7.68
- port number: 80

Socket types

- SOCK_DGRAM
    - UDP packets
- SOCK_STREAM
    - TCP packets
- SOCK_RAW
    - don’t let OS process transport header on packet, have OS send/receive raw packet

Different types of service
offered by different
socket types

2 main socket types for 2 transport services

TCP (Transmission Control Protocol)

- **connection-oriented**
  - before data exchange takes place, a logical connection is first established
- **reliable, byte stream-oriente**d
  - delivery is in-order, error- and loss-free, no duplication

App reads in-order, error-free bytes from socket

UDP (User Datagram Protocol)

- **connection-less**
  - data is sent directly in a best-effort way
- **unreliable**
  - data can arrive out-of-order, be lost, corrupted, duplicated

App reads whatever is currently at socket, whether out-of-order, missing etc.

Any reliability must be implemented by app

vumanfredi@wesleyan.edu

Send data (from python reference)

socket.send(bytes) - TCP

- Send data to the socket. The socket must be connected to a remote socket. Returns the number of bytes sent. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data

socket.sendall(bytes) - TCP

- Send data to the socket. The socket must be connected to a remote socket. Unlike send(), this method continues to send data from bytes until either all data has been sent or an error occurs. None is returned on success. On error, an exception is raised, and there is no way to determine how much data, if any, was successfully sent.

socket.sendto(bytes, address) - UDP

- Send data to the socket. The socket should not be connected to a remote socket, since the destination socket is specified by address.

vumanfredi@wesleyan.edu

Receive data (from python reference)

Socket.recv(num_bytes)

- Receive data from the socket. The return value is a bytes object representing the data received. The maximum amount of data to be received at once is specified by *bufsize*.

Partial Send/Recv

socket.sendall()

- generally preferable to use to eliminate partial send

socket.recv()

- app needs way to know whether it has read everything from socket
  - “end” flag
  - a priori knowledge of number of bytes to read
  - …
- typically put recv() in while loop
  - keep reading until nothing left to read from socket

vumanfredi@wesleyan.edu

Endianness

Big endian

- big end first: largest byte (containing most significant bit) first

Little endian

- little end first: smallest byte (containing least significant bit) first

Network byte order

- big endian

UTF-8 byte order

- stays the same regardless of endian-ness of machine
- i.e., you shouldn't need to worry about byte order

vumanfredi@wesleyan.edu

Network Programming

TCP SOCKETS

vumanfredi@wesleyan.edu

Socket programming with TCP

Client must first contact server before sending data

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

How?

- create TCP socket
  - specify server IP addr, port #
  - "handshake" occurs
    - TCP Syn/Synack/Ack exchanged
    - if succeeds, connection established, can send data

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

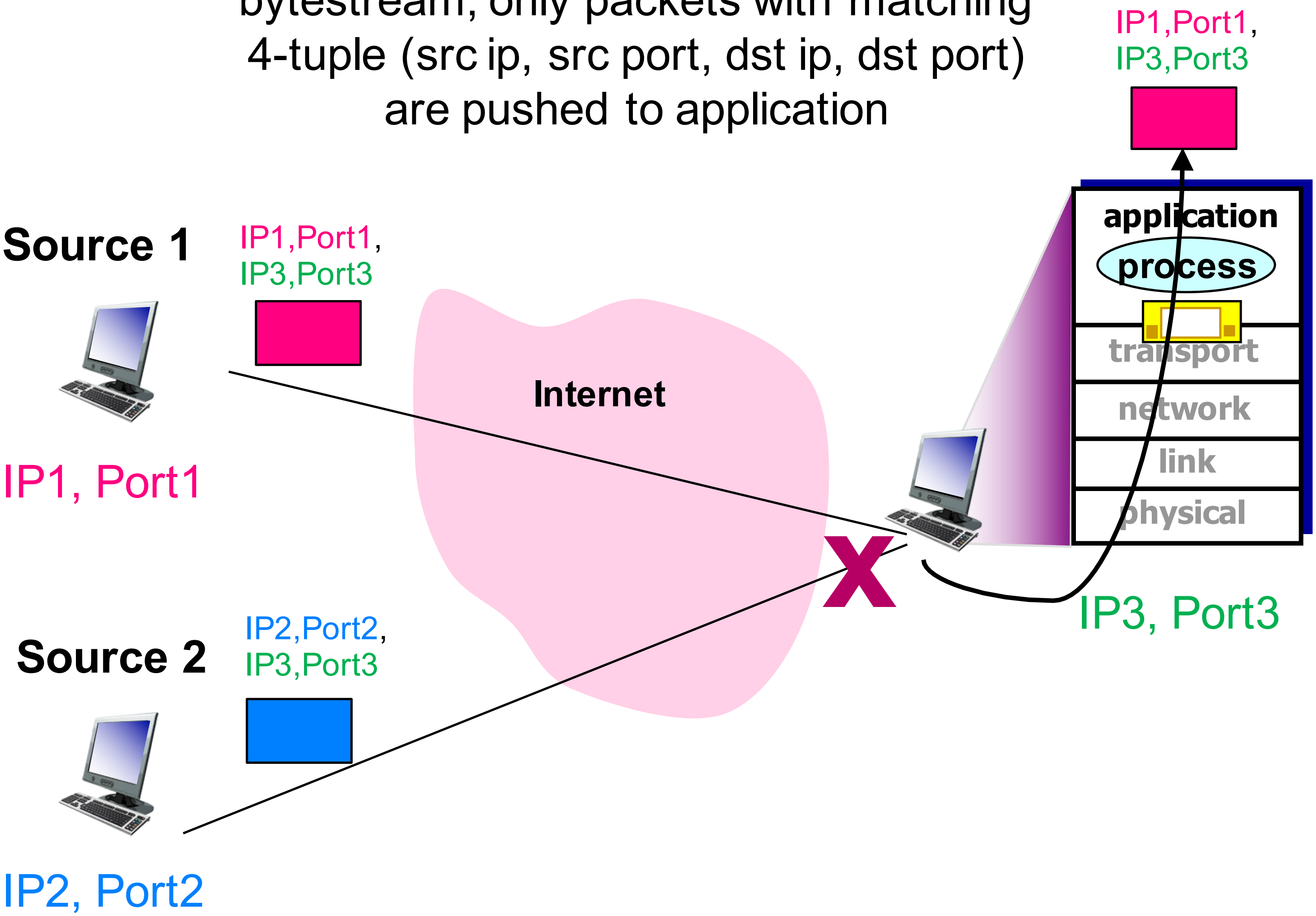
Application viewpoint

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

vumanfredi@wesleyan.edu

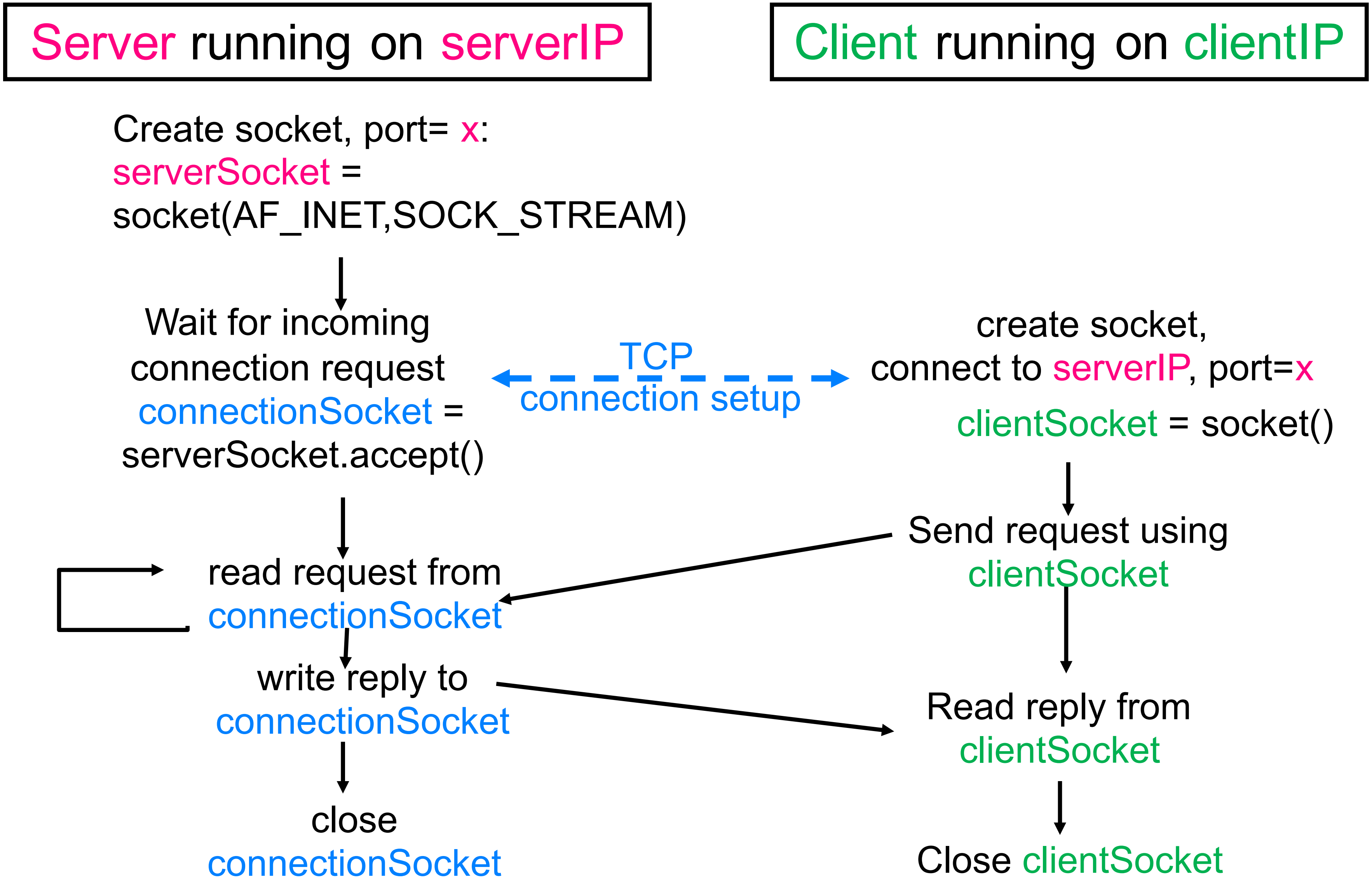
TCP Socket

Establish connection, read/write bytestream, only packets with matching 4-tuple (src ip, src port, dst ip, dst port) are pushed to application



vumanfredi@wesleyan.edu

Client/server socket interaction: TCP



Server running on serverIP

Client running on clientIP

Create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_STREAM)

Wait for incoming
connection request
connectionSocket =
serverSocket.accept()

TCP
connection setup

create socket,
connect to serverIP, port=x
clientSocket = socket()

Send request using
clientSocket

read request from
connectionSocket

write reply to
connectionSocket

Read reply from
clientSocket

close
connectionSocket

Close clientSocket

vumanfredi@wesleyan.edu

Application example

1. Client
   - reads a line of characters (data) from its keyboard and sends data to server via socket

2. Server
   - receives data from socket and converts characters to uppercase

3. Server
   - sends modified data to client

4. Client
   - receives modified data and displays line on its screen

vumanfredi@wesleyan.edu

Application example: 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 True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                        encode())
    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)

vumanfredi@wesleyan.edu

Application example: TCP client

Python TCPClient

```
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.encode())
modifiedSentence = clientSocket.recv(1024)
print (‘From Server:’, modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

vumanfredi@wesleyan.edu

echo_client.py and echo_server.py

Look at code and run:

available on class schedule

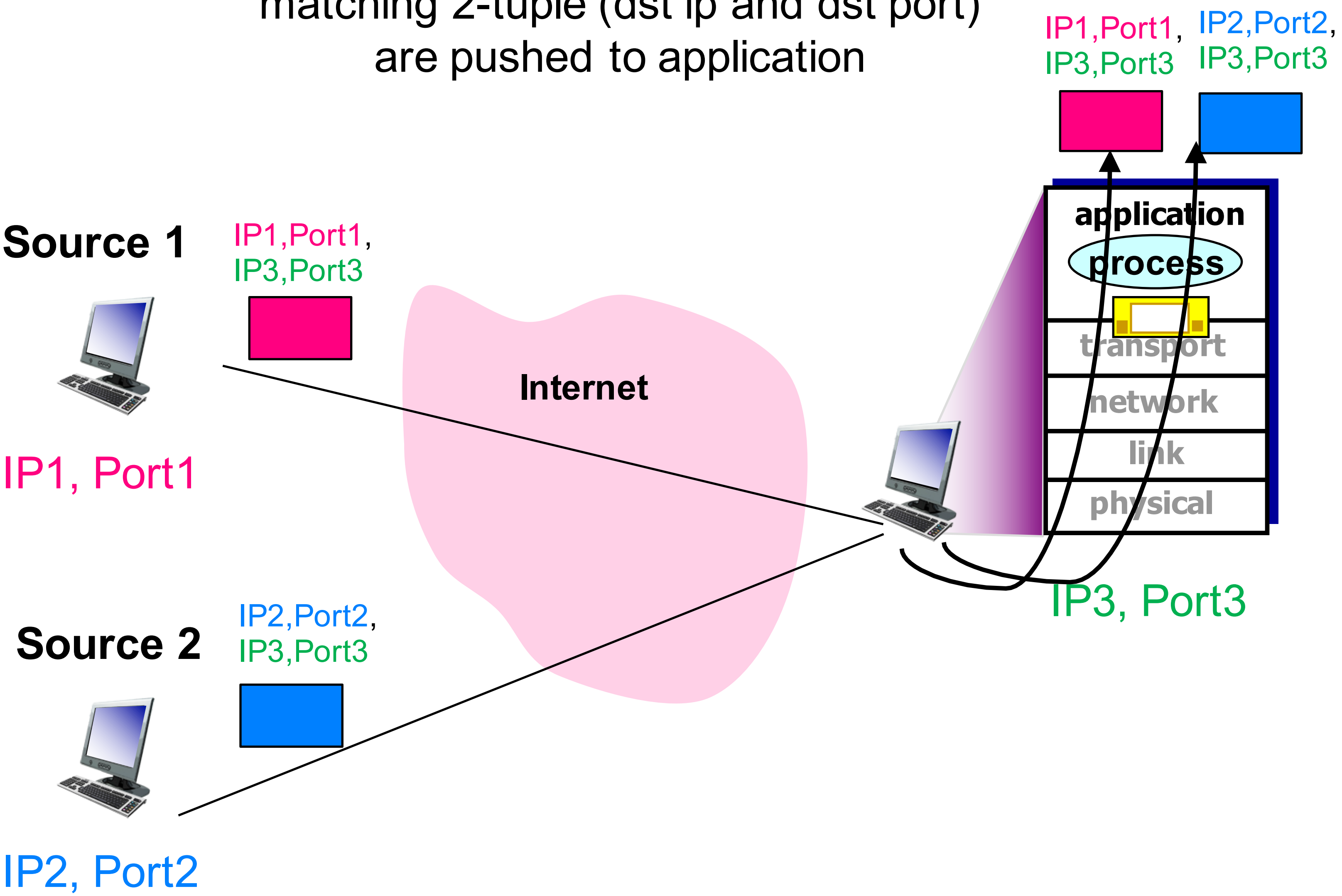
Network Programming

UDP SOCKETS

vumanfredi@wesleyan.edu

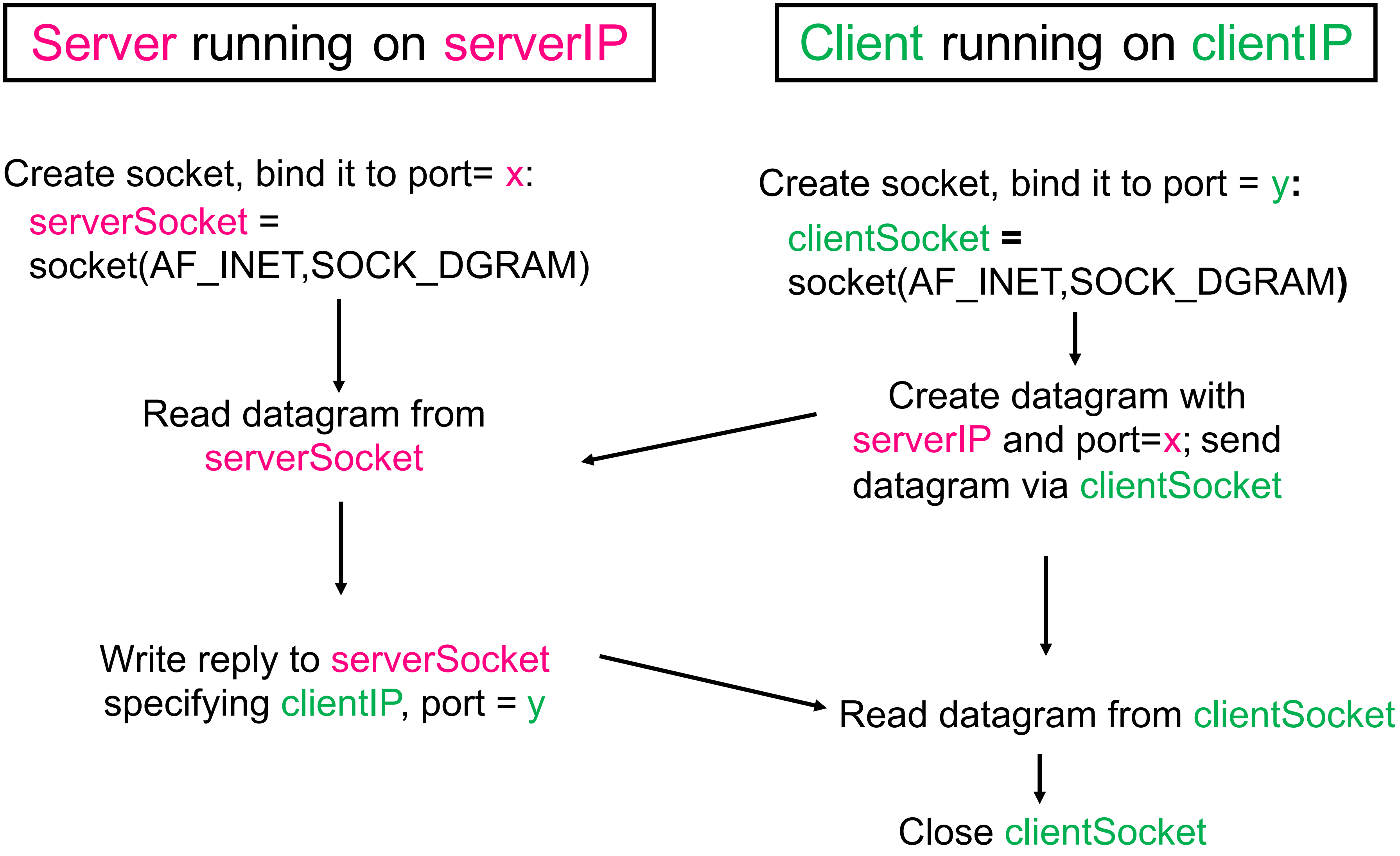
UDP Socket

Read/write packets, only packets with matching 2-tuple (dst ip and dst port) are pushed to application



vumanfredi@wesleyan.edu

Client/server socket interaction: UDP



Server running on serverIP

Create socket, bind it to port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

Read datagram from
serverSocket

Write reply to serverSocket
specifying clientIP, port = y

Client running on clientIP

Create socket, bind it to port = y:
clientSocket **=**
socket(AF_INET,SOCK_DGRAM**)**

Create datagram with
serverIP and port=x; send
datagram via clientSocket

Read datagram from clientSocket

Close clientSocket

vumanfredi@wesleyan.edu

Application example: UDP server

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

Python UDPServer

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

vumanfredi@wesleyan.edu

Application example: 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.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()
```

Network Programming

USEFUL TOOLS

vumanfredi@wesleyan.edu

Netstat: what network connections do you have?

What ports are open?

- netstat | less

Display routing table info

- netstat -r

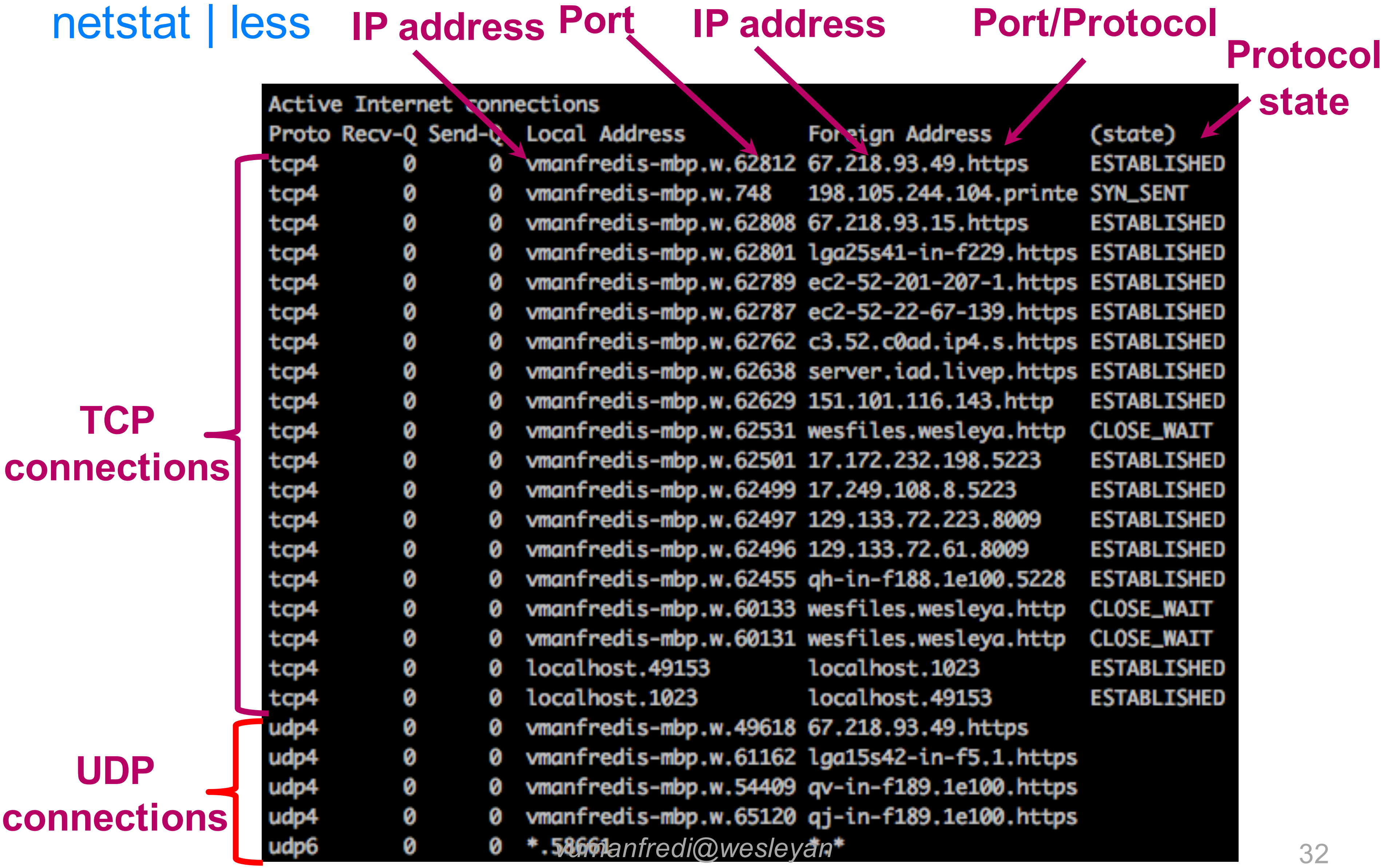
On Linux only

- TCP connections
  - ss –ta
- UDP connections
  - ss -ua
- Unix connections
  - ss -xa

vumanfredi@wesleyan.edu

What network connections do I have?

netstat | less



IP address

Port

IP address

Port/Protocol

Protocol state

TCP connections

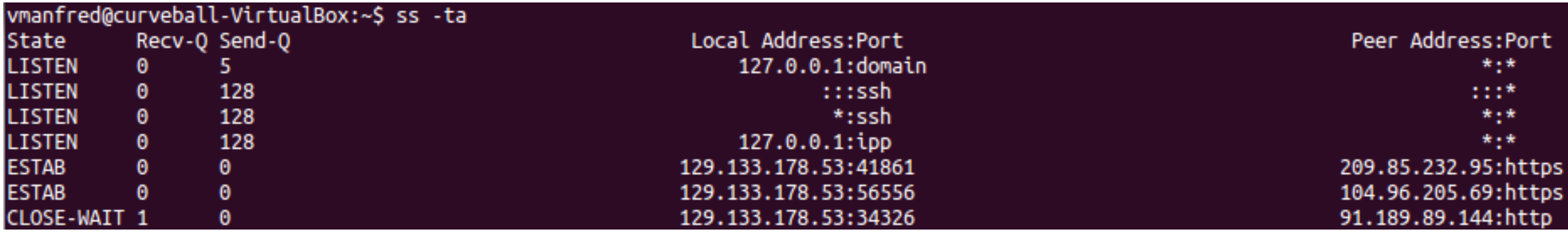
UDP connections

```
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  vmanfredis-mbp.w.62812 67.218.93.49.https     ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.748   198.105.244.104.printe SYN_SENT
tcp4       0      0  vmanfredis-mbp.w.62808 67.218.93.15.https     ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62801 lga25s41-in-f229.https ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62789 ec2-52-201-207-1.https ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62787 ec2-52-22-67-139.https ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62762 c3.52.c0ad.ip4.s.https ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62638 server.iad.livep.https ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62629 151.101.116.143.http   ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62531 wesfiles.wesleya.http  CLOSE_WAIT
tcp4       0      0  vmanfredis-mbp.w.62501 17.172.232.198.5223    ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62499 17.249.108.8.5223      ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62497 129.133.72.223.8009    ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62496 129.133.72.61.8009     ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.62455 qh-in-f188.1e100.5228  ESTABLISHED
tcp4       0      0  vmanfredis-mbp.w.60133 wesfiles.wesleya.http  CLOSE_WAIT
tcp4       0      0  vmanfredis-mbp.w.60131 wesfiles.wesleya.http  CLOSE_WAIT
tcp4       0      0  localhost.49153        localhost.1023         ESTABLISHED
tcp4       0      0  localhost.1023         localhost.49153        ESTABLISHED
udp4       0      0  vmanfredis-mbp.w.49618 67.218.93.49.https
udp4       0      0  vmanfredis-mbp.w.61162 lga15s42-in-f5.1.https
udp4       0      0  vmanfredis-mbp.w.54409 qv-in-f189.1e100.https
udp4       0      0  vmanfredis-mbp.w.65120 qj-in-f189.1e100.https
udp6       0      0  *.58661                *.*
```

What network connections do I have?

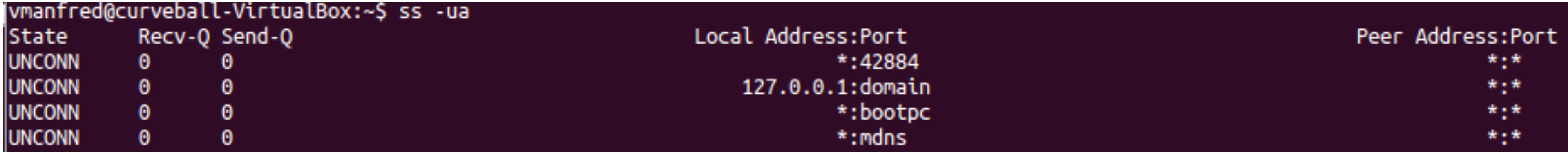
ss (socket statistics, works in linux only)

TCP connections



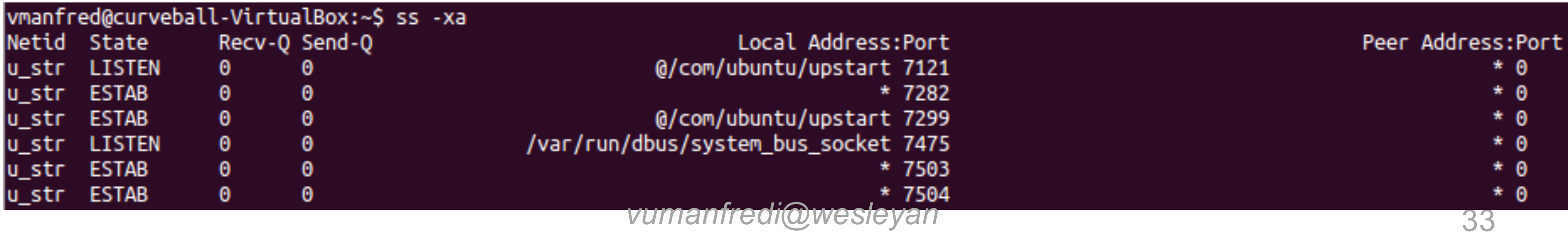
```
vmanfred@curveball-VirtualBox:~$ ss -ta
State      Recv-Q Send-Q                        Local Address:Port                            Peer Address:Port
LISTEN     0      5                              127.0.0.1:domain                                         *:*
LISTEN     0      128                                  :::ssh                                            :::*
LISTEN     0      128                                   *:ssh                                             *:*
LISTEN     0      128                              127.0.0.1:ipp                                          *:*
ESTAB      0      0                           129.133.178.53:41861                          209.85.232.95:https
ESTAB      0      0                           129.133.178.53:56556                          104.96.205.69:https
CLOSE-WAIT 1      0                           129.133.178.53:34326                          91.189.89.144:http
```

UDP connections



```
vmanfred@curveball-VirtualBox:~$ ss -ua
State      Recv-Q Send-Q                        Local Address:Port                            Peer Address:Port
UNCONN     0      0                                     *:42884                                         *:*
UNCONN     0      0                              127.0.0.1:domain                                       *:*
UNCONN     0      0                                     *:bootpc                                        *:*
UNCONN     0      0                                     *:mdns                                          *:*
```

Unix connections



```
vmanfred@curveball-VirtualBox:~$ ss -xa
Netid  State      Recv-Q Send-Q                      Local Address:Port                        Peer Address:Port
u_str  LISTEN     0      0                      @/com/ubuntu/upstart 7121                                   * 0
u_str  ESTAB      0      0                                         * 7282                                   * 0
u_str  ESTAB      0      0                      @/com/ubuntu/upstart 7299                                   * 0
u_str  LISTEN     0      0                 /var/run/dbus/system_bus_socket 7475                             * 0
u_str  ESTAB      0      0                                         * 7503                                   * 0
u_str  ESTAB      0      0                                         * 7504                                   * 0
```

Netcat: useful for testing

Be a TCP server: listen for connections on port 51234

- `nc -l 51234`

Be a TCP client: connect to port 51234 on locahost

- `nc localhost 51234`
- type a string and press enter: you should see it show up at server
- type a string at server and press enter: you should see it at client

Look at connections you created

- `netstat | grep 51234`

Connect to www.wesleyan.edu

- `nc -u www.wesleyan.edu 80`
- once connected, enter

```
        GET / HTTP/1.1
        Host: www.wesleyan.edu          # followed by two enters
```

vumanfredi@wesleyan.edu

netcat

Create a chat app with nc:

nc -l 5000 on one machine with ip addr x

nc x 5000 on another machine

vumanfredi@wesleyan.edu

Packet sniffing

WIRESHARK

How can I look at network traffic?

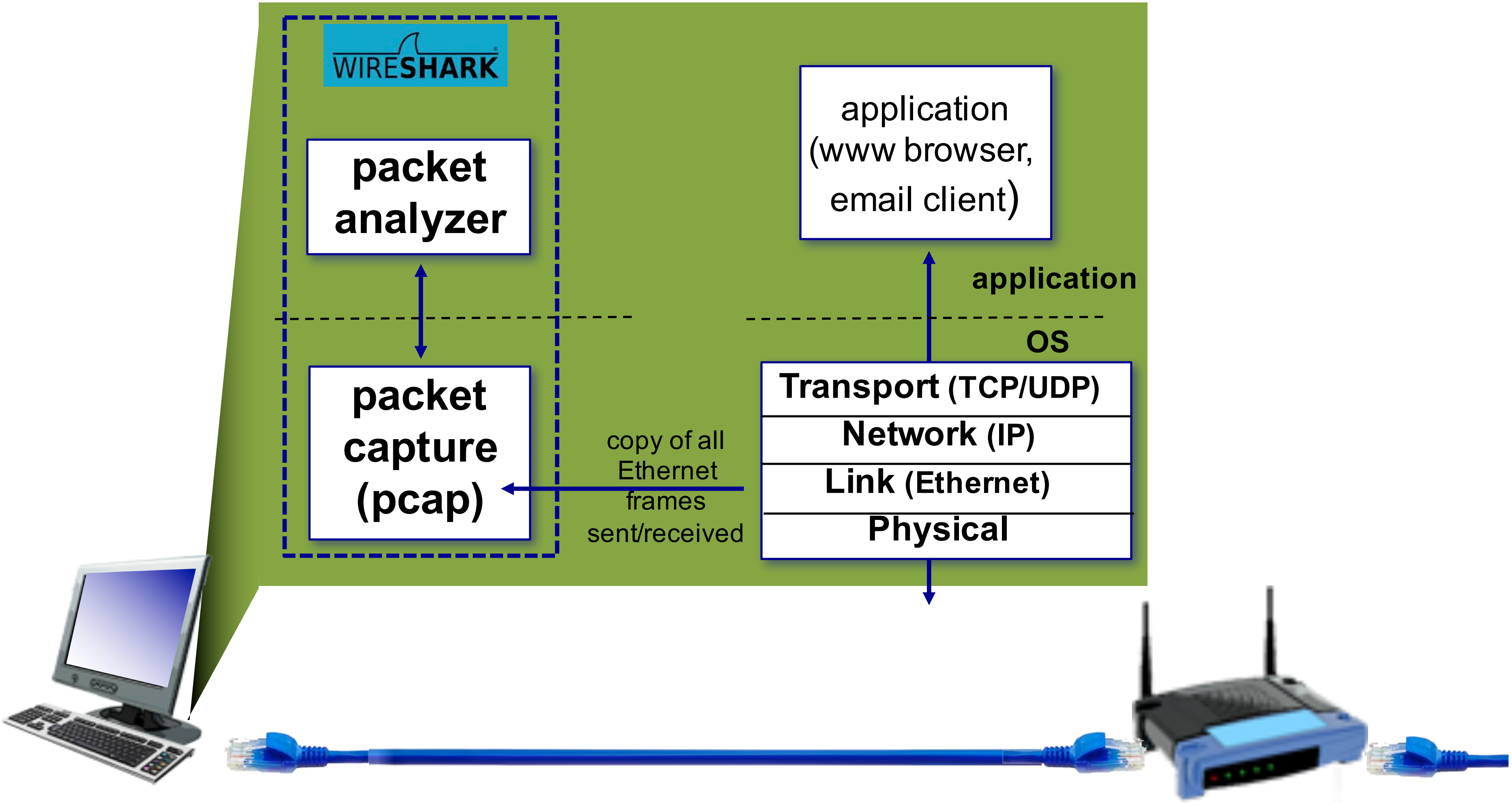
Packet sniffer

- passively observes messages transmitted and received on a particular network interface by processes running on your computer
- often requires root privileges to run

Popular packet sniffers

- Wireshark (also command-line version, tshark)
- tcpdump (Unix) and WinDump (Windows)
- use command line sniffers to analyze packet traces with bash script

Packet sniffer operation



packet analyzer

packet capture (pcap)

copy of all Ethernet frames sent/received

application (www browser, email client)

application

OS

Transport (TCP/UDP)

Network (IP)

Link (Ethernet)

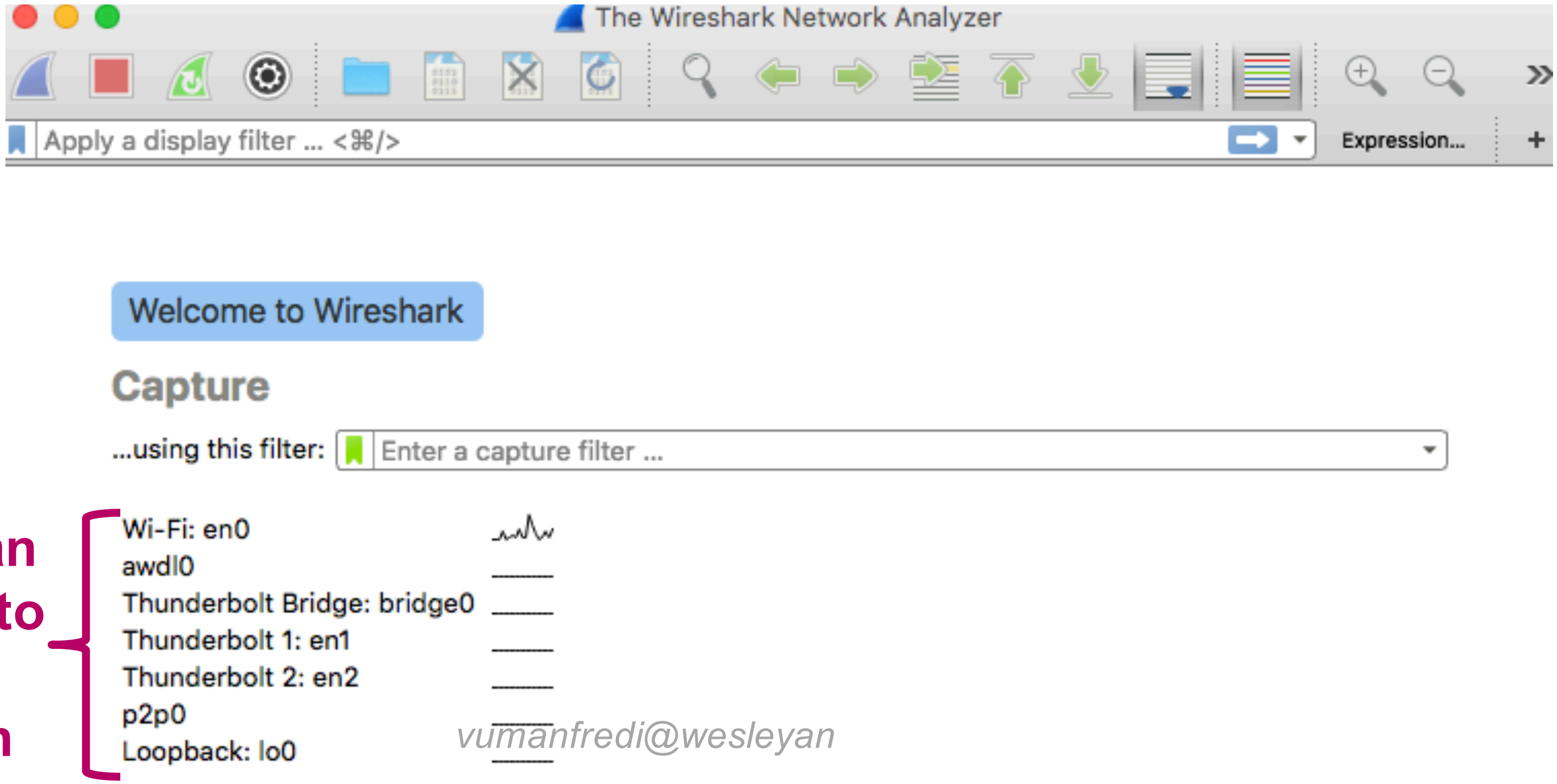
Physical

vumanfredi@wesleyan

Wireshark

Install
- https://www.wireshark.org/download.html

Run
- type Wireshark in terminal, or double-click icon
- Wireshark display may look different for Linux vs. Mac vs. Windows



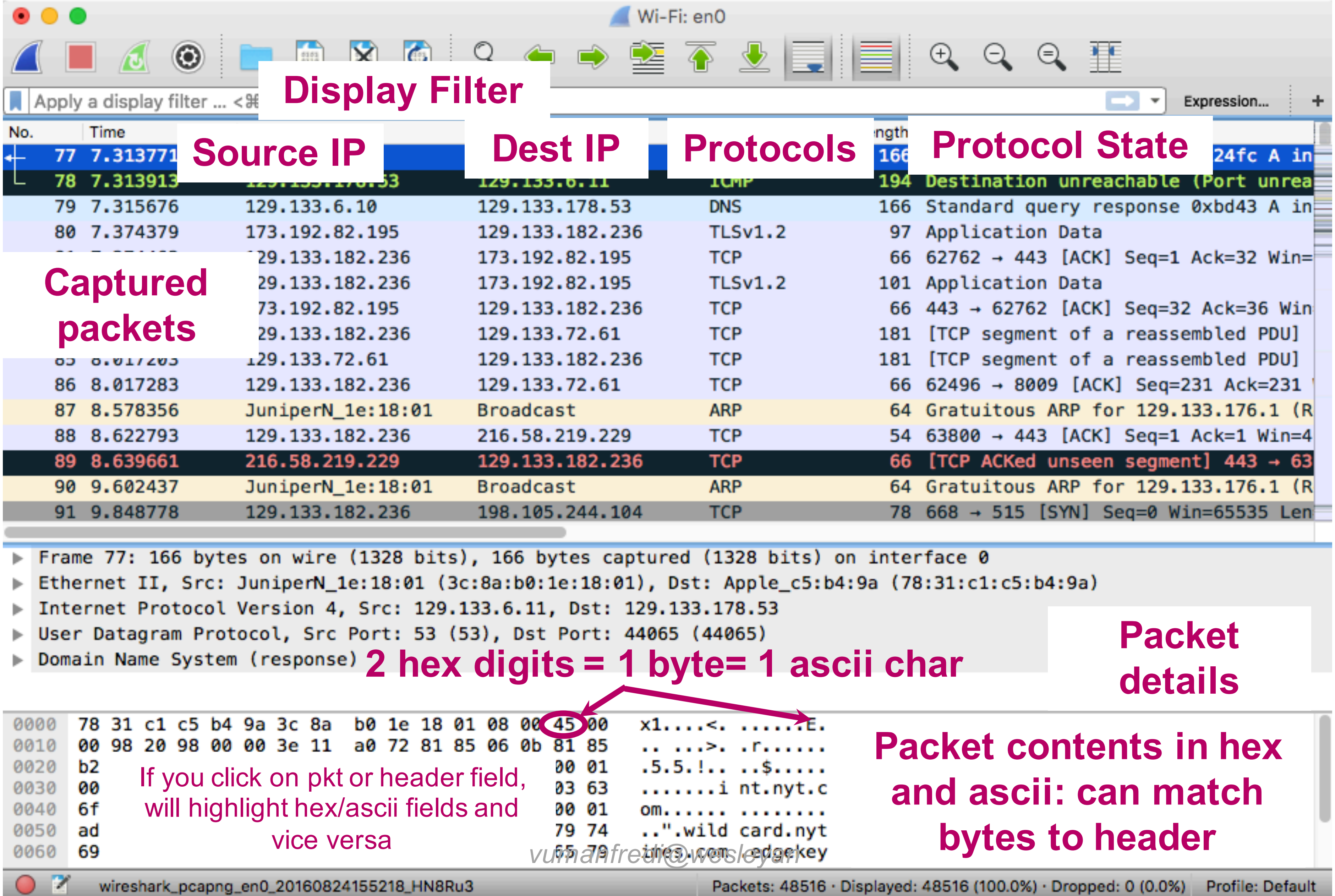
…using this filter: Enter a capture filter …

Wi-Fi: en0
awdl0
Thunderbolt Bridge: bridge0
Thunderbolt 1: en1
Thunderbolt 2: en2
p2p0
Loopback: lo0

Choose an interface to capture traffic on

vumanfredi@wesleyan

What do we see?



Display Filter

Source IP

Dest IP

Protocols

Protocol State

Captured packets

Packet details

2 hex digits = 1 byte= 1 ascii char

If you click on pkt or header field, will highlight hex/ascii fields and vice versa

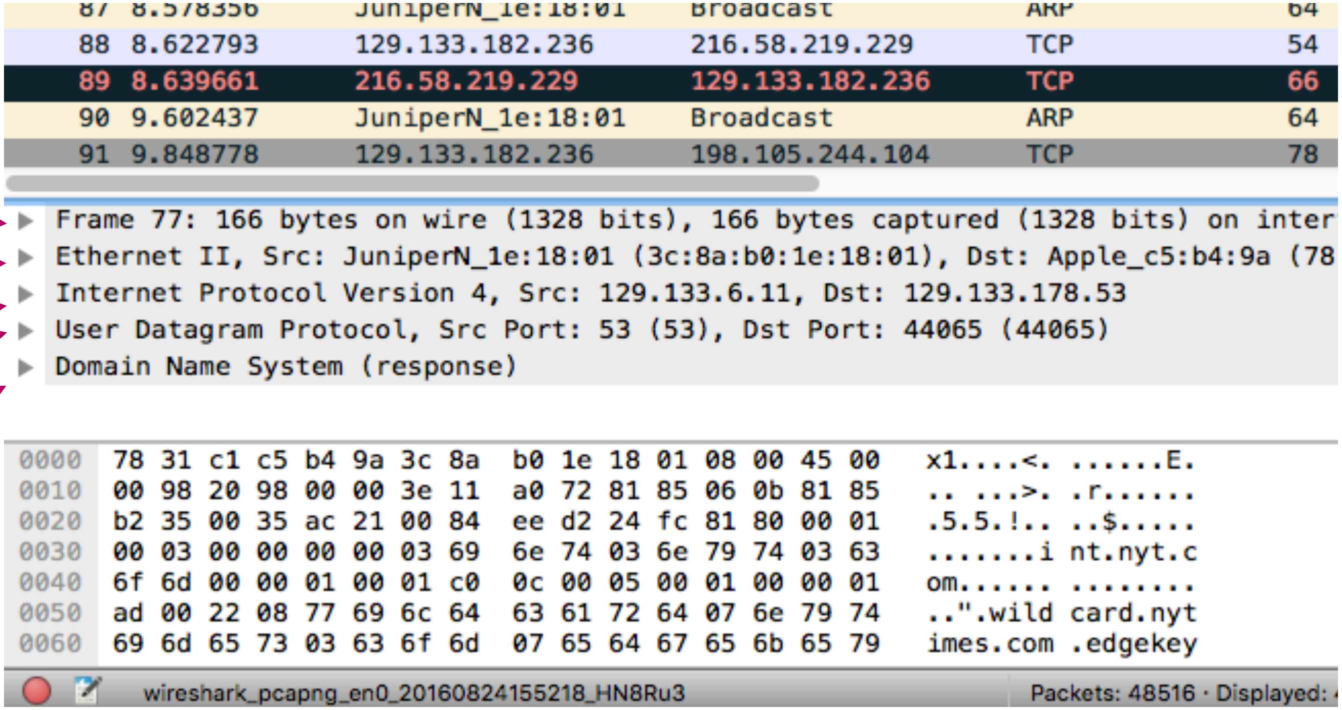
Packet contents in hex and ascii: can match bytes to header

```
Wi-Fi: en0
Apply a display filter ... <⌘                                  Expression...  +
No.  Time                                                       ngth
  77 7.313771                                                  166          24fc A in
  78 7.313913                                                  194 Destination unreachable (Port unrea
  79 7.315676  129.133.6.10          129.133.178.53      DNS        166 Standard query response 0xbd43 A in
  80 7.374379  173.192.82.195        129.133.182.236     TLSv1.2     97 Application Data
               29.133.182.236        173.192.82.195      TCP         66 62762 → 443 [ACK] Seq=1 Ack=32 Win=
               29.133.182.236        173.192.82.195      TLSv1.2    101 Application Data
               73.192.82.195         129.133.182.236     TCP         66 443 → 62762 [ACK] Seq=32 Ack=36 Win
               29.133.182.236        129.133.72.61       TCP        181 [TCP segment of a reassembled PDU]
               129.133.72.61         129.133.182.236     TCP        181 [TCP segment of a reassembled PDU]
  86 8.017283  129.133.182.236       129.133.72.61       TCP         66 62496 → 8009 [ACK] Seq=231 Ack=231
  87 8.578356  JuniperN_1e:18:01     Broadcast           ARP         64 Gratuitous ARP for 129.133.176.1 (R
  88 8.622793  129.133.182.236       216.58.219.229      TCP         54 63800 → 443 [ACK] Seq=1 Ack=1 Win=4
  89 8.639661  216.58.219.229        129.133.182.236     TCP         66 [TCP ACKed unseen segment] 443 → 63
  90 9.602437  JuniperN_1e:18:01     Broadcast           ARP         64 Gratuitous ARP for 129.133.176.1 (R
  91 9.848778  129.133.182.236       198.105.244.104     TCP         78 668 → 515 [SYN] Seq=0 Win=65535 Len

▶ Frame 77: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface 0
▶ Ethernet II, Src: JuniperN_1e:18:01 (3c:8a:b0:1e:18:01), Dst: Apple_c5:b4:9a (78:31:c1:c5:b4:9a)
▶ Internet Protocol Version 4, Src: 129.133.6.11, Dst: 129.133.178.53
▶ User Datagram Protocol, Src Port: 53 (53), Dst Port: 44065 (44065)
▶ Domain Name System (response)

0000  78 31 c1 c5 b4 9a 3c 8a  b0 1e 18 01 08 00 45 00   x1....<. ......E.
0010  00 98 20 98 00 00 3e 11  a0 72 81 85 06 0b 81 85   .. ...>. .r......
0020  b2                                         00 01   .5.5.!.. ..$.....
0030  00                                         03 63   ........i nt.nyt.c
0040  6f                                         00 01   om...... ........
0050  ad                                         79 74   ..".wild card.nyt
0060  69                                         65 79   ides.com edgekey

wireshark_pcapng_en0_20160824155218_HN8Ru3     Packets: 48516 · Displayed: 48516 (100.0%) · Dropped: 0 (0.0%)   Profile: Default
```

What do we see?

Layers

- **Physical**
- **Link**
- **Network**
- **Transport**
- **Application**



```
87  8.578356    JuniperN_1e:18:01   Broadcast         ARP   64
88  8.622793    129.133.182.236     216.58.219.229    TCP   54
89  8.639661    216.58.219.229      129.133.182.236   TCP   66
90  9.602437    JuniperN_1e:18:01   Broadcast         ARP   64
91  9.848778    129.133.182.236     198.105.244.104   TCP   78
```

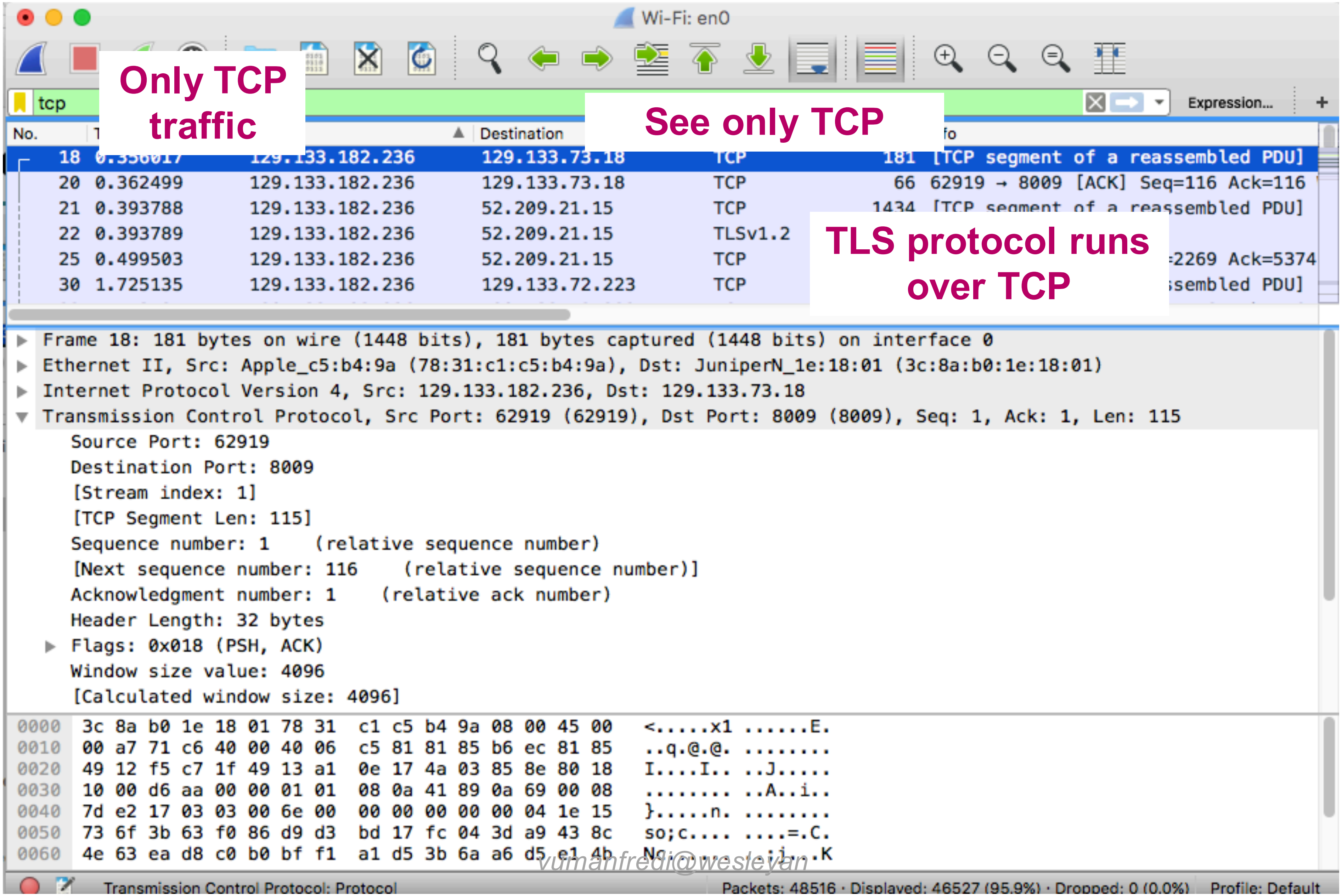
```
▶ Frame 77: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on inter
▶ Ethernet II, Src: JuniperN_1e:18:01 (3c:8a:b0:1e:18:01), Dst: Apple_c5:b4:9a (78
▶ Internet Protocol Version 4, Src: 129.133.6.11, Dst: 129.133.178.53
▶ User Datagram Protocol, Src Port: 53 (53), Dst Port: 44065 (44065)
▶ Domain Name System (response)
```

```
0000  78 31 c1 c5 b4 9a 3c 8a  b0 1e 18 01 08 00 45 00   x1....<. ......E.
0010  00 98 20 98 00 00 3e 11  a0 72 81 85 06 0b 81 85   .. ...>. .r......
0020  b2 35 00 35 ac 21 00 84  ee d2 24 fc 81 80 00 01   .5.5.!.. ..$.....
0030  00 03 00 00 00 00 03 69  6e 74 03 6e 79 74 03 63   .......i nt.nyt.c
0040  6f 6d 00 00 01 00 01 c0  0c 00 05 00 01 00 00 01   om...... ........
0050  ad 00 22 08 77 69 6c 64  63 61 72 64 07 6e 79 74   ..".wild card.nyt
0060  69 6d 65 73 03 63 6f 6d  07 65 64 67 65 6b 65 79   imes.com .edgekey
```

wireshark_pcapng_en0_20160824155218_HN8Ru3   Packets: 48516 · Displayed:

vumanfredi@wesleyan

Add a filter



Only TCP traffic

See only TCP

TLS protocol runs over TCP

```
No.  T                                ▲ Destination
  18 0.356017  129.133.182.236   129.133.73.18    TCP        181 [TCP segment of a reassembled PDU]
  20 0.362499  129.133.182.236   129.133.73.18    TCP         66 62919 → 8009 [ACK] Seq=116 Ack=116
  21 0.393788  129.133.182.236   52.209.21.15     TCP       1434 [TCP segment of a reassembled PDU]
  22 0.393789  129.133.182.236   52.209.21.15     TLSv1.2
  25 0.499503  129.133.182.236   52.209.21.15     TCP                         =2269 Ack=5374
  30 1.725135  129.133.182.236   129.133.72.223   TCP                         ssembled PDU]
```

```
▶ Frame 18: 181 bytes on wire (1448 bits), 181 bytes captured (1448 bits) on interface 0
▶ Ethernet II, Src: Apple_c5:b4:9a (78:31:c1:c5:b4:9a), Dst: JuniperN_1e:18:01 (3c:8a:b0:1e:18:01)
▶ Internet Protocol Version 4, Src: 129.133.182.236, Dst: 129.133.73.18
▼ Transmission Control Protocol, Src Port: 62919 (62919), Dst Port: 8009 (8009), Seq: 1, Ack: 1, Len: 115
     Source Port: 62919
     Destination Port: 8009
     [Stream index: 1]
     [TCP Segment Len: 115]
     Sequence number: 1    (relative sequence number)
     [Next sequence number: 116    (relative sequence number)]
     Acknowledgment number: 1    (relative ack number)
     Header Length: 32 bytes
   ▶ Flags: 0x018 (PSH, ACK)
     Window size value: 4096
     [Calculated window size: 4096]
```

```
0000  3c 8a b0 1e 18 01 78 31  c1 c5 b4 9a 08 00 45 00   <.....x1 ......E.
0010  00 a7 71 c6 40 00 40 06  c5 81 81 85 b6 ec 81 85   ..q.@.@. ........
0020  49 12 f5 c7 1f 49 13 a1  0e 17 4a 03 85 8e 80 18   I....I.. ..J.....
0030  10 00 d6 aa 00 00 01 01  08 0a 41 89 0a 69 00 08   ........ ..A..i..
0040  7d e2 17 03 03 00 6e 00  00 00 00 00 00 04 1e 15   }.....n. ........
0050  73 6f 3b 63 f0 86 d9 d3  bd 17 fc 04 3d a9 43 8c   so;c.... ....=.C.
0060  4e 63 ea d8 c0 b0 bf f1  a1 d5 3b 6a a6 d5 e1 4b   Nc...... ..;j...K
```

```
Transmission Control Protocol: Protocol     Packets: 48516 · Displayed: 46527 (95.9%) · Dropped: 0 (0.0%)     Profile: Default
```

Using wireshark

Run traceroute and see what traffic is generated

vumanfredi@wesleyan.edu

Using wireshark

Run ping and see what traffic is generated

vumanfredi@wesleyan.edu