Lecture 4: IP Addresses, Sockets, and System Programming

COMP 332, Fall 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 solution code will be posted once homework1 submitted

2. Network applications

3. Network programming
   - TCP sockets

4. Network tools
   - Wireshark: looking at real traffic

Internet Organization

IP ADDRESSES

Every device on Internet has an IP address

IPv4 addresses

- 4 bytes
  - space of addresses: 0-255 . 0-255 . 0-255 . 0-255
  - hostnames are human-readable, IP addresses are machine-readable
- Loopback address: send traffic to yourself
  - traffic sent here is "looped back" through network stack on machine on which sending process is running
  - 127 . * .* .*
  - typically 127.0.0.1, also called localhost
- Private subnet addresses
  - 10 .* .* .*
  - 172.16-31 .* .*
  - 192.168 .* .*

Subnet: shared prefix portion of addr

IPv6 addresses

- 16 bytes: we're running out of 4 byte addresses …

Who owns what address ranges?

Amazon

- 50.19.*.* → 256 x 256 = 65536 addresses
- 54.239.98.* → 256 addresses
- …

Facebook

- 57.240.0.0/17
- 157.240.10.0/24
- 157.240.1.0/24
- …

Google

- 64.233.160.0 to 64.233.191.255
- 66.102.0.0 to 66.102.15.255
- …

Wesleyan

- 129.133.21.*
- …

How are IP addresses assigned?

Your ISP or institution has block of IP addresses

- you are assigned one of those IP addresses
- (possible you will get NAT'd address …)

Static IP address

- manual configuration: set in network settings

Dynamic IP address

- using Dynamic Host Configuration Protocol (DHCP) in network-layer
- client (you) broadcasts request for IP address
- DHCP server on network assigns you address from address pool
  - typically get IP address for fixed period of time
  - router can be configured to act as DHCP server

Actually …

Many hosts have multiple IP addresses

How?

- IP address associated with network interface not host
- network interface card (NIC): connects computer to network

A host may have 1 or more network interfaces

- my laptop has (at least) 2 NICs: 1 wireless and 1 wired (via USB)
- router needs at least two interfaces
  - otherwise can’t connect multiple networks together
- Cisco core router: can have up to 10,000 interfaces!
  - one interface per link: router has many IP addresses

VirtualBox Virtual Machine (VM)

- you can set the number and type of network interfaces for VM

What’s my IP address?

ifconfig

- what network interfaces does my machine have?
- what are my IP and MAC # addresses?
- configure/enable/disable an interface

Linux

Ethernet 0
IPv4 address
IPv6 address

Loopback address

```
vmanfred@curveball-VirtualBox:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:e2:65:b0
          inet addr:129.133.178.53  Bcast:129.133.191.255  Mask:255.255.240.0
          inet6 addr: fe80::a00:27ff:fee2:65b0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:102302 errors:0 dropped:0 overruns:0 frame:0
          TX packets:29698 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:141037591 (141.0 MB)  TX bytes:2394226 (2.3 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1912 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1912 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:146886 (146.8 KB)  TX bytes:146886 (146.8 KB)
```

What's host's IP address?

Host

```
> host www.google.com
www.google.com has address 74.125.141.99
www.google.com has address 74.125.141.103
www.google.com has address 74.125.141.105
www.google.com has address 74.125.141.147
www.google.com has address 74.125.141.104
www.google.com has address 74.125.141.106
www.google.com has IPv6 address 2607:f8b0:400c:c06::93
```

What's host name for IP address?

```
> host 8.8.8.8
8.8.8.8.in-addr.arpa domain name pointer google-public-dns-a.google.com.
```

What’s host’s IP address?

dig

```
> dig www.google.com

; <<>> DiG 9.8.3-P1 <<>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4619
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.google.com.                IN      A

;; ANSWER SECTION:
www.google.com.         56      IN      A       74.125.141.104
www.google.com.         56      IN      A       74.125.141.103
www.google.com.         56      IN      A       74.125.141.105
www.google.com.         56      IN      A       74.125.141.147
www.google.com.         56      IN      A       74.125.141.99
www.google.com.         56      IN      A       74.125.141.106

;; Query time: 7 msec
;; SERVER: 129.133.52.12#53(129.133.52.12)
;; WHEN: Mon Jan 22 14:06:38 2018
;; MSG SIZE  rcvd: 128
```

DNS resolver used

Is host up?

Ping

- sends ICMP echo request to host
- host sends ICMP echo reply back
- If no reply within timeout period, packet deemed lost

```
> ping stanford.edu
PING stanford.edu (171.67.215.200): 56 data bytes
64 bytes from 171.67.215.200: icmp_seq=0 ttl=237 time=94.951 ms
64 bytes from 171.67.215.200: icmp_seq=1 ttl=237 time=94.738 ms
64 bytes from 171.67.215.200: icmp_seq=2 ttl=237 time=95.525 ms
64 bytes from 171.67.215.200: icmp_seq=3 ttl=237 time=194.993 ms
64 bytes from 171.67.215.200: icmp_seq=4 ttl=237 time=97.139 ms
64 bytes from 171.67.215.200: icmp_seq=5 ttl=237 time=95.878 ms
64 bytes from 171.67.215.200: icmp_seq=6 ttl=237 time=95.667 ms
^C
--- stanford.edu ping statistics ---
7 packets transmitted, 7 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 94.738/109.842/194.993/34.770 ms
```

Is one IP address per machine enough?

What happens if you run multiple network applications?

- many processes running on computer
  - process is program in execution

How do messages received by computer get to right process?

- messages are addressed to (IP address, port #) pair
- different processes on computer will connect to network using same IP address but different port numbers

2 key functions of Internet core

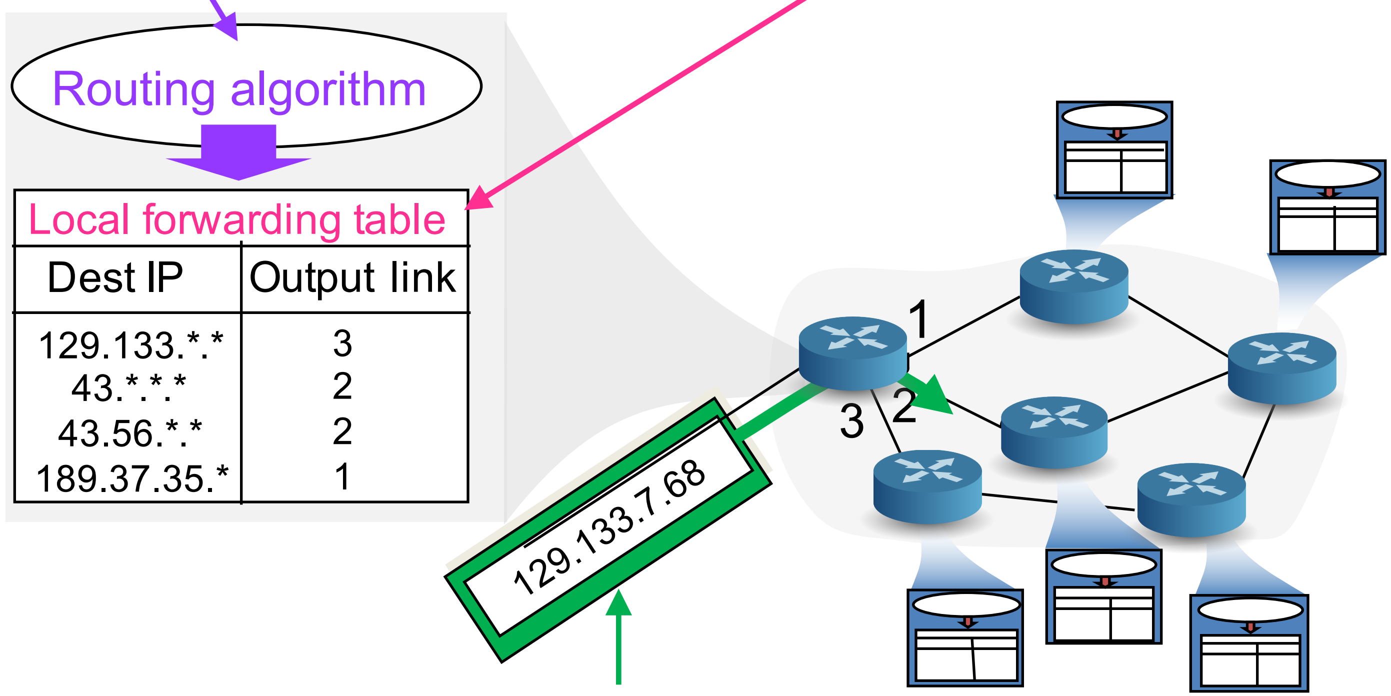
How does Internet router determine outgoing link for packet?

1. Routing

- view Internet as giant graph
- run shortest path algorithms

2. Forwarding

- use paths to choose best output link for packet destination IP address
- if one link fails, chooses another

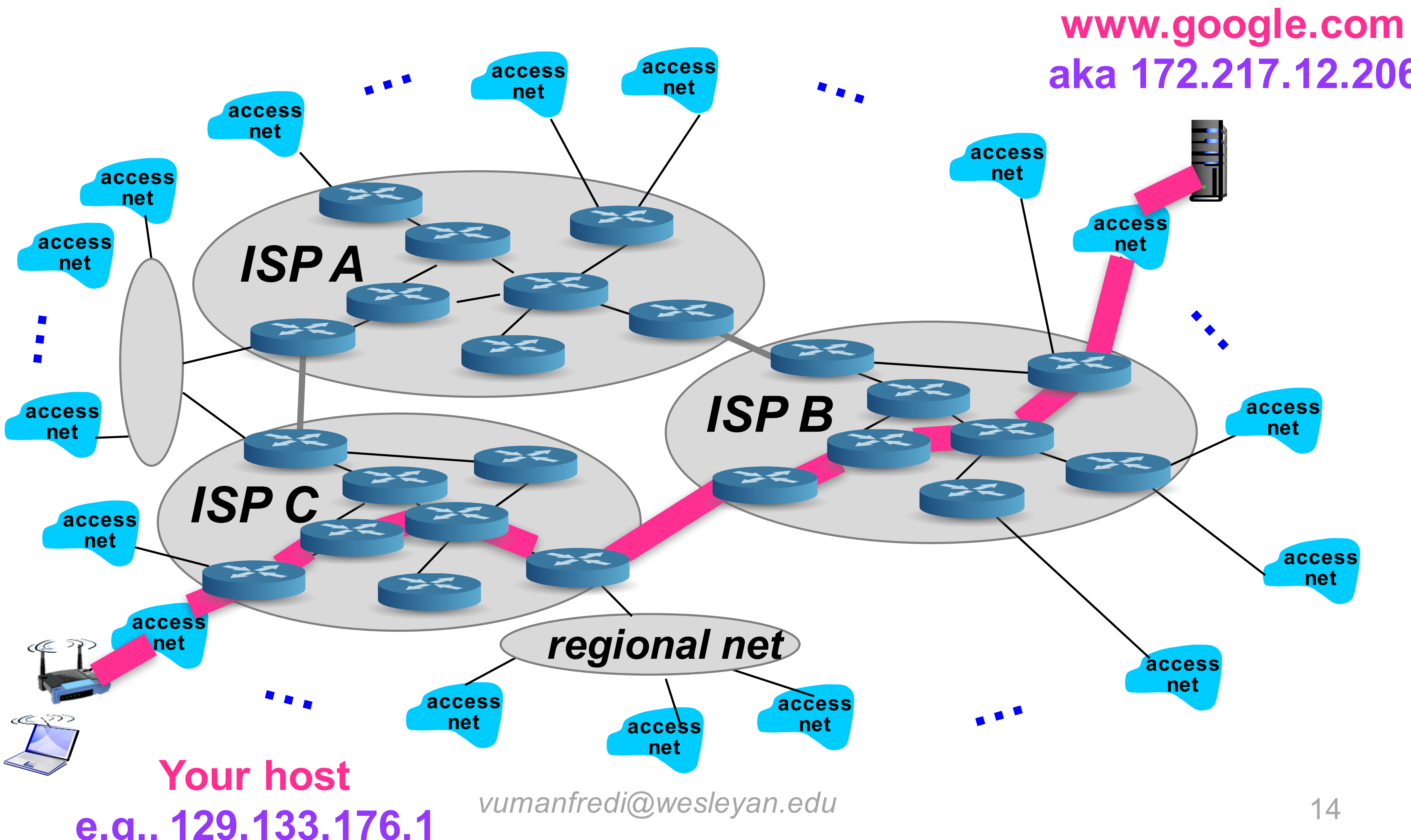


Dest IP	Output link
129.133.*.*	3
43.*.*.*	2
43.56.*.*	2
189.37.35.*	1

Dest IP addr in header of arriving packet (sequence of bits)

Routing of packets across Internet

Each router uses its forwarding table to choose outbound link based on packet's destination



Network Applications

OVERVIEW

vumanfredi@wesleyan.edu

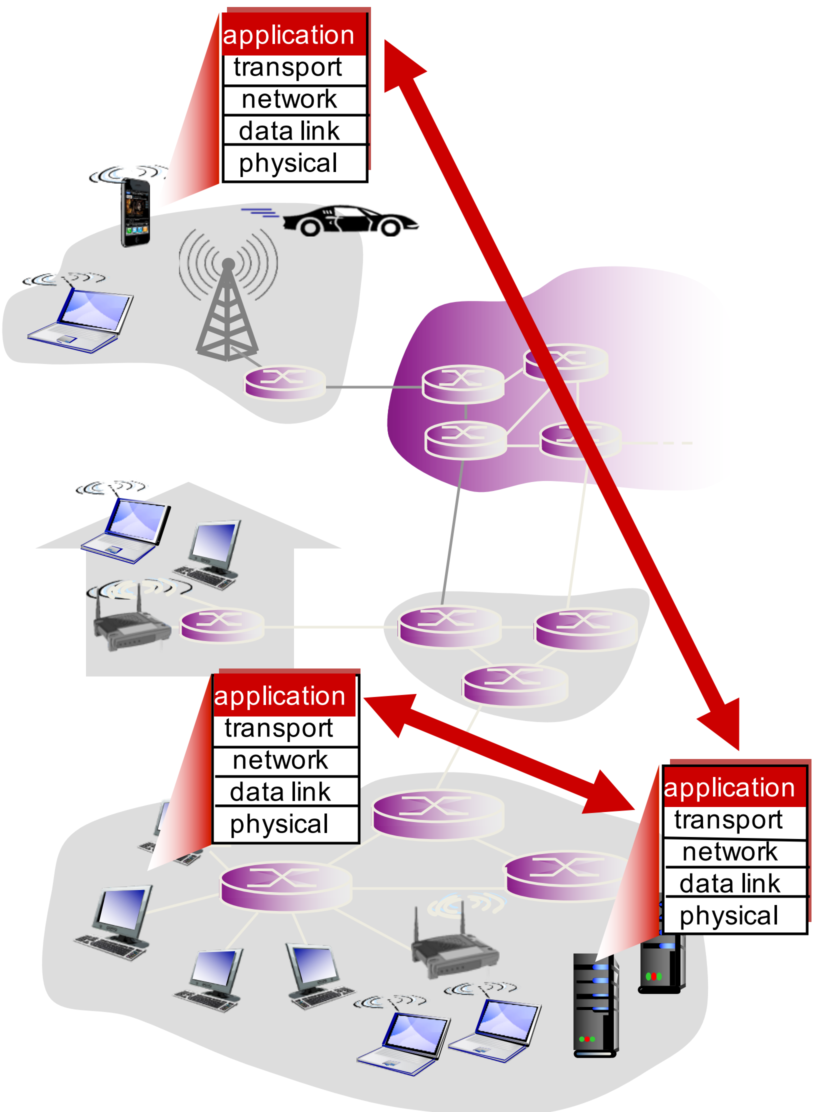
Creating a network app

Write programs that

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

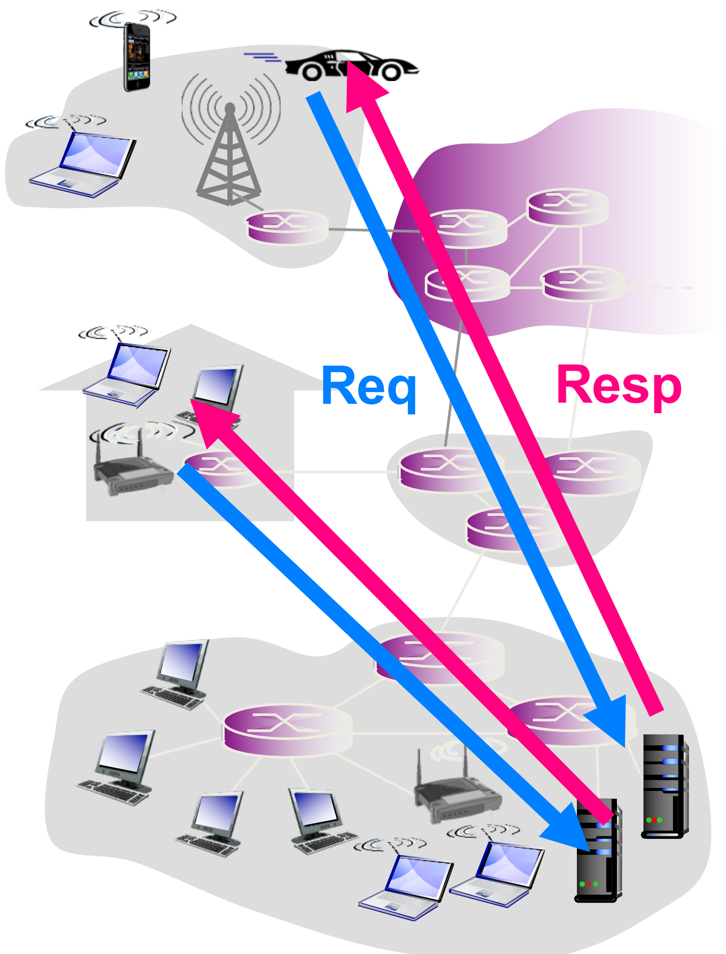
Q: Do we need to write software for network-core devices?

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



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

Peer-to-peer (P2P) architecture

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

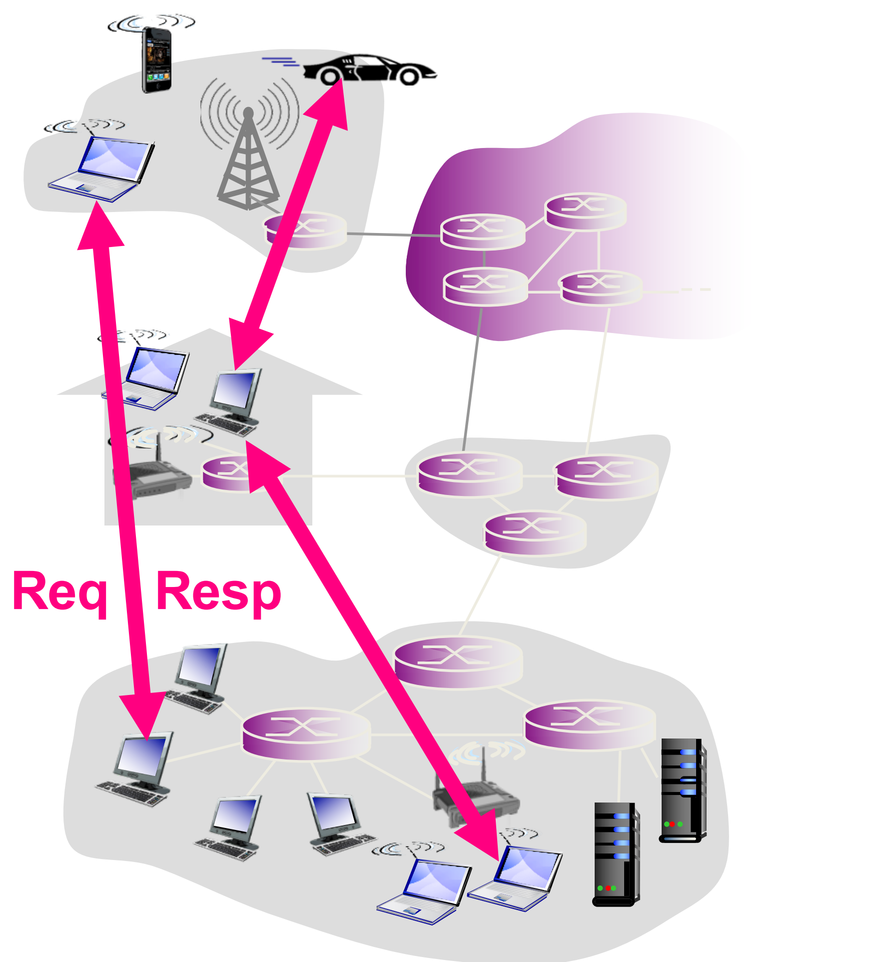
End systems directly communicate

- 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



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

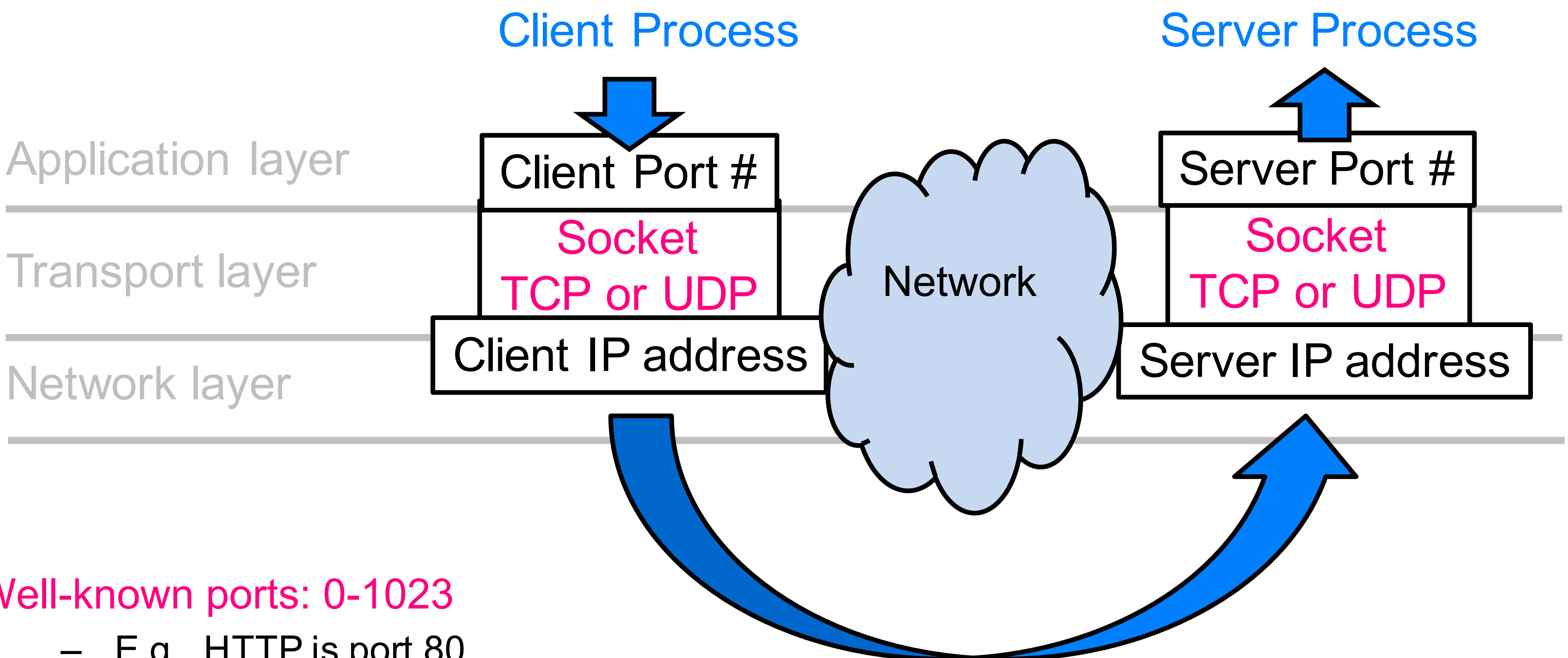
OVERVIEW

vumanfredi@wesleyan.edu

How do two processes talk over a network?

Via sockets

- interface transport layer provides to apps to access network
- connection endpoint with associated IP addr, port #



Well-known ports: 0-1023

- E.g., HTTP is port 80

Registered ports: 1023-49151

Available ports: 49152-65535

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-orient**ed
  - 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

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.

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

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

Network Programming

TCP SOCKETS

vumanfredi@wesleyan.edu

Socket programming with TCP

Client must first contact server before sending data

- server process must 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

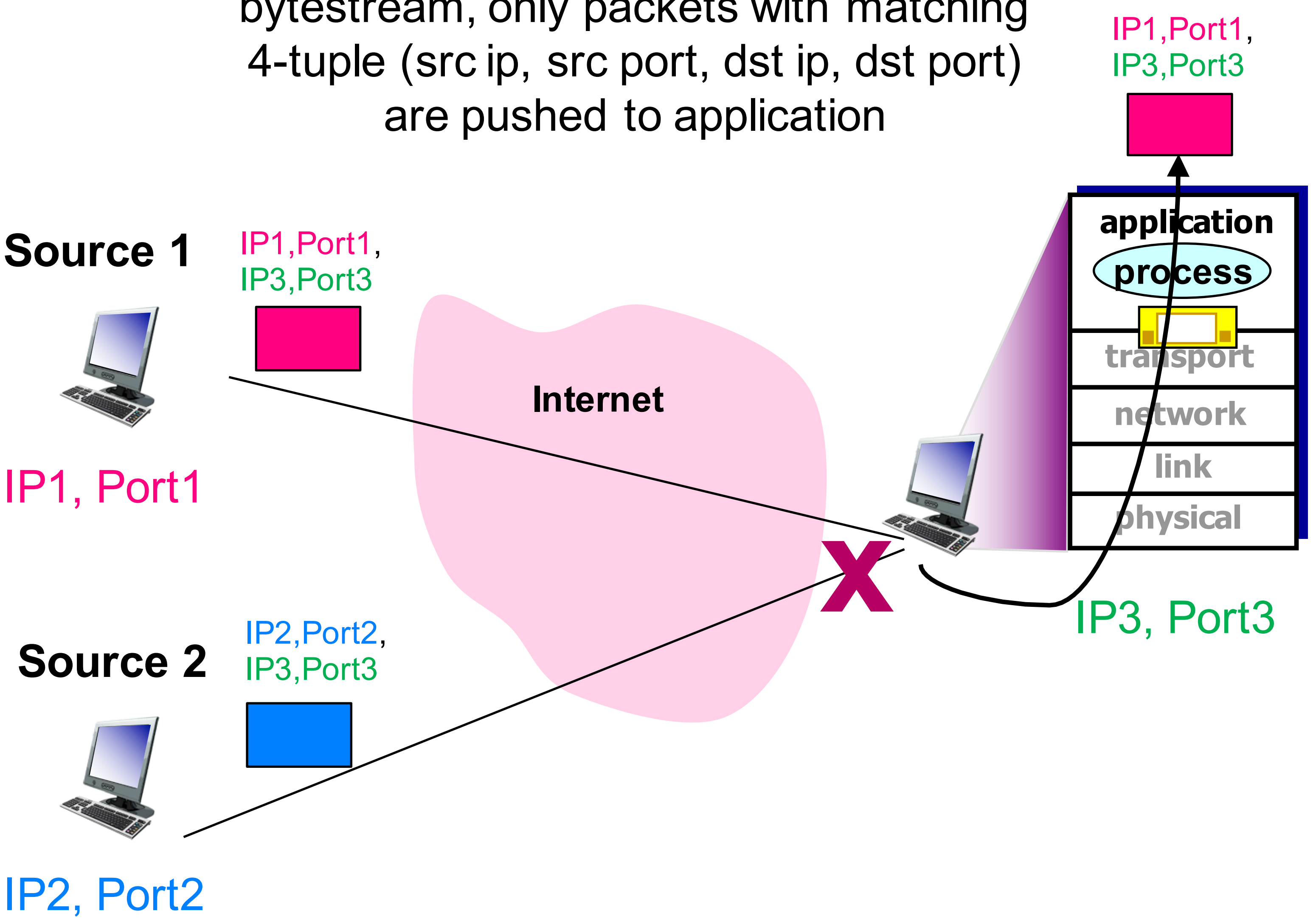
- server TCP creates new socket for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

Application viewpoint

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

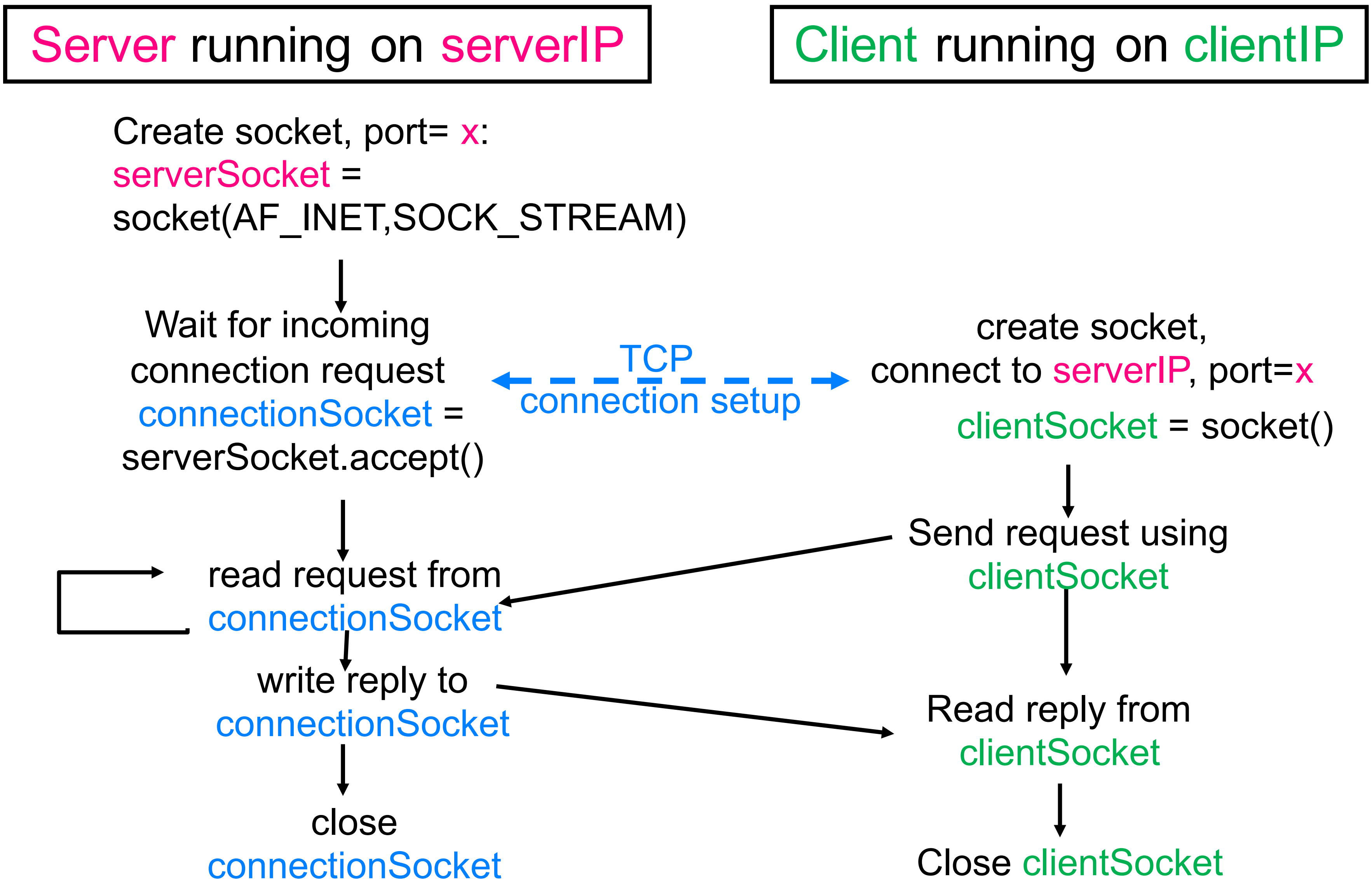
TCP Socket

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



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

Application example: TCP server

Python TCPServer

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

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

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()`

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

```
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()
```

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

echo_client.py and echo_server.py

Look at code and run:

available on class schedule

Packet sniffing

WIRESHARK

How can I look at network traffic?

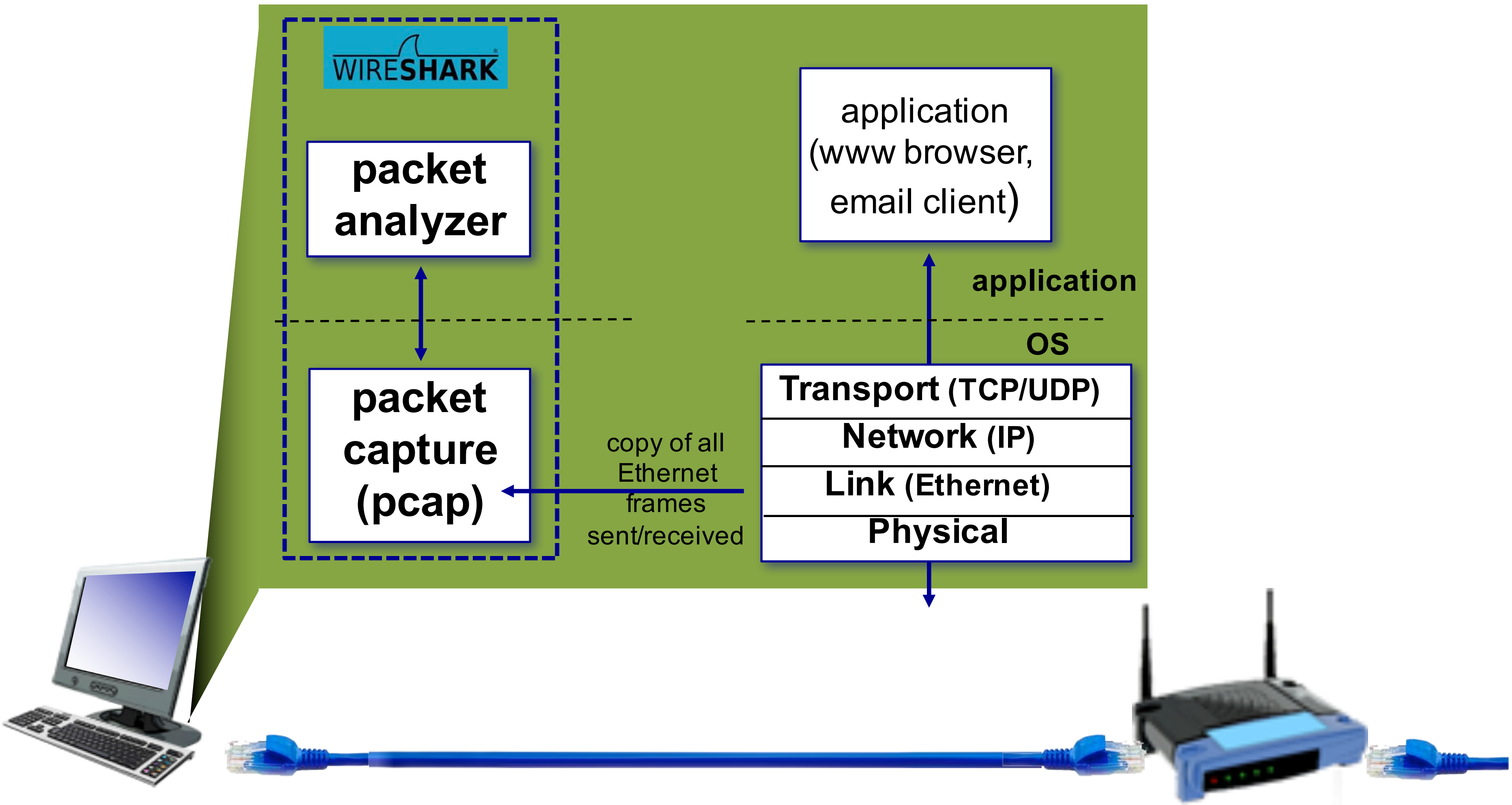
Packet sniffer

- 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



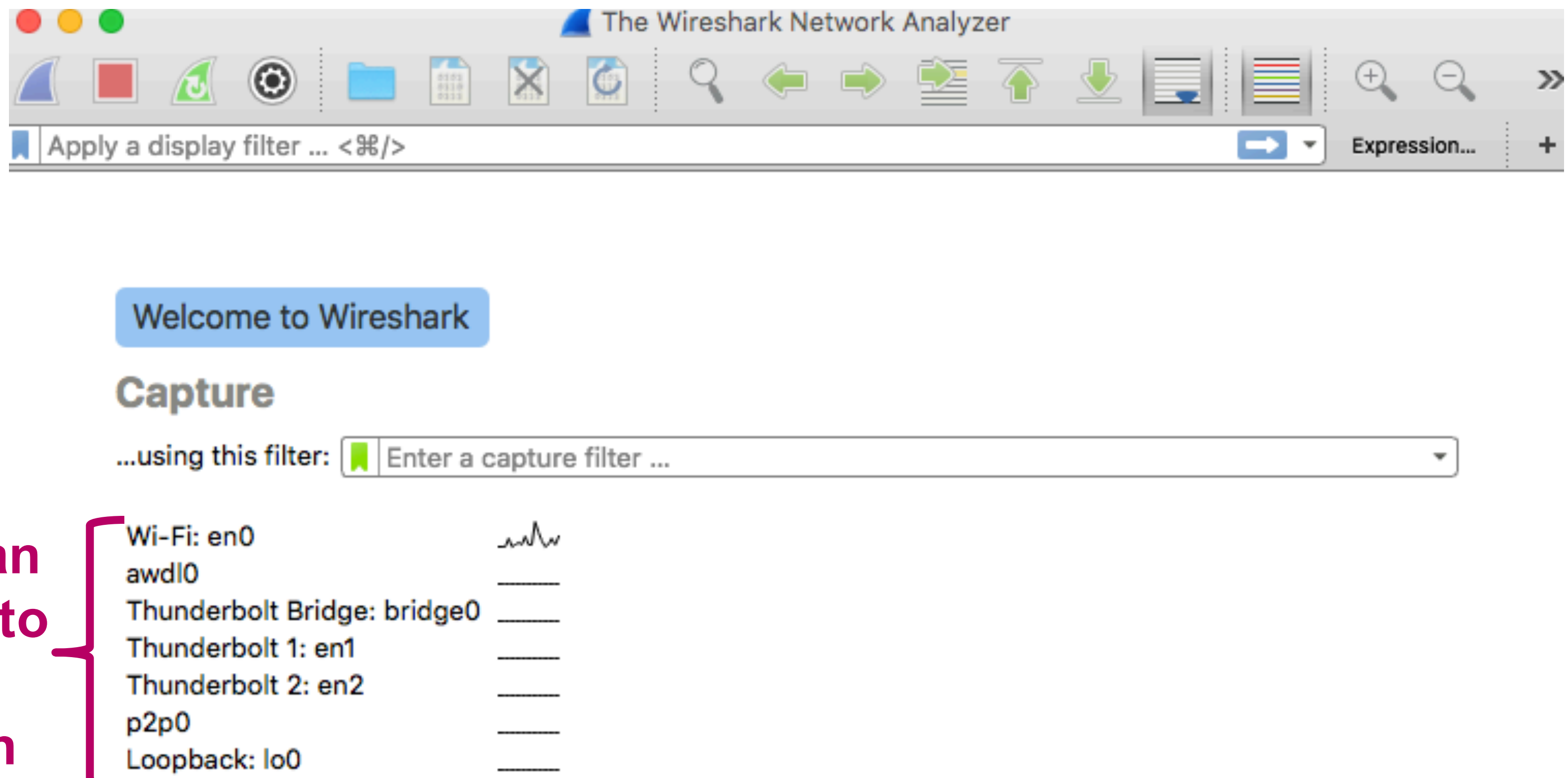
Wireshark

Install

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

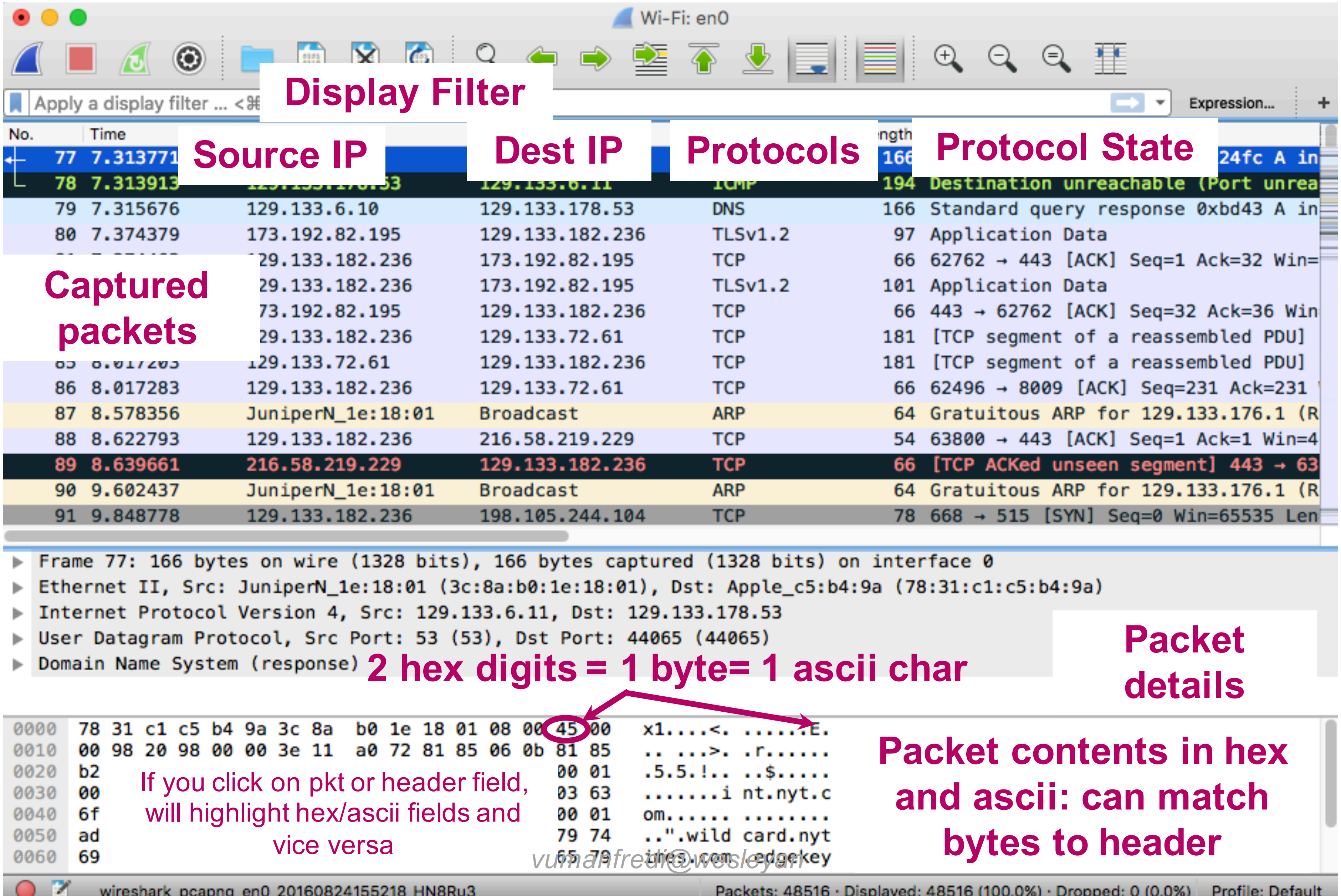
Run

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

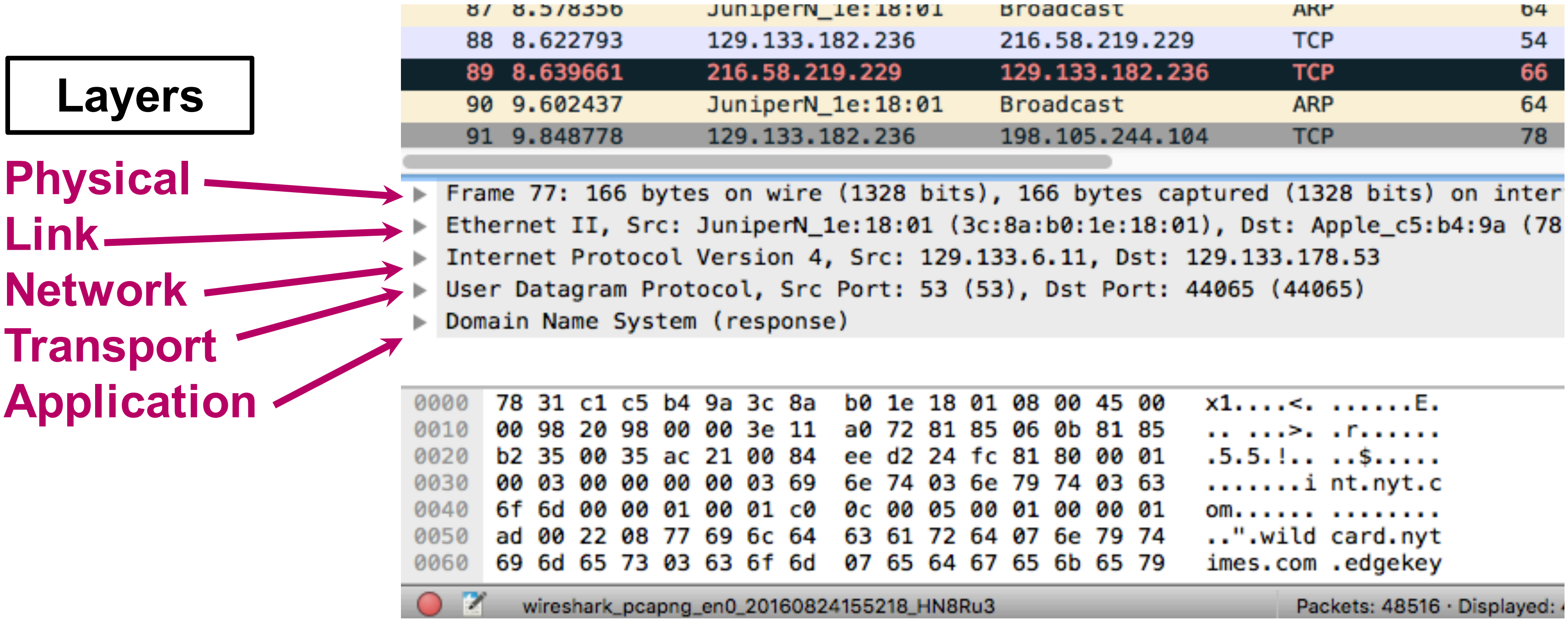


Choose an interface to capture traffic on

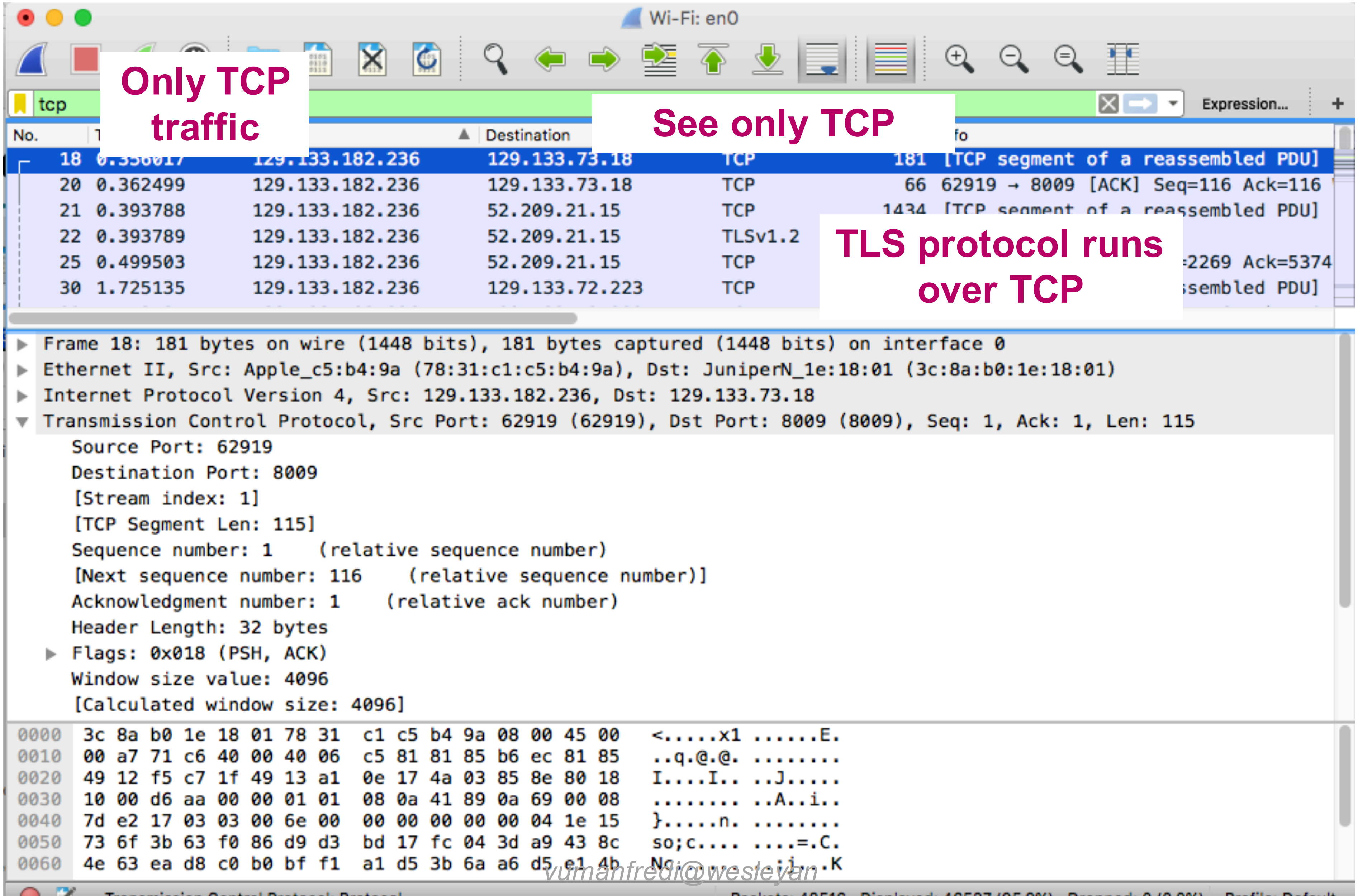
What do we see?



What do we see?



Add a filter


Wi-Fi: en0
tcp
Expression...
Only TCP traffic
See only TCP
TLS protocol runs over TCP
No.
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
Transmission Control Protocol: Protocol
Packets: 48516 · Displayed: 46527 (95.9%) · Dropped: 0 (0.0%)
Profile: Default