COMP 206 – Intro to Software Systems

Lecture 18 – Internet 2

November 9th, 2018

Outline

- More details about how the internet works:
  - More details on how we manage addresses
  - Ports and port ranges
  - Network standardization conventions
- Looking at an internet socket in detail:
  - The 3-way handshake
- Ideas about writing internet software:
  - Sending and receiving on the same socket

Brief history of the internet

- Early network ideas in the 60's: break data into packets, how to organize connections, clients and servers, simple transmission
- 1970's: Early versions of ARPANET (The Advanced Research Projects Agency Network) completed and demonstrated. Ethernet @ Xerox.
- 1980's: First implemented versions of modern TCP/IP.
- 1989: World Wide Web proposed, public in 1991

McGill and the Internet

- Canada on the net: around 1985 with NetNorth
  - McGill is an early member, with researchers at what is now called "Centre for Intelligent Machines" (CIM) having many of the first connected devices
  - Historical information: the NetNorth Polcies circa 1990: http://retirees.uwaterloo.ca/~rwwatt/nnpp.html

- The first search engine Archie, written by Alan Emtage, Bill Heelan, and Mike Parker at McGill University in Montreal Canada is released on September 10, 1990

How to keep track of all of these addresses?

- IPv6 allows close to 2^128 devices to be online and uniquely addressable:
  - Log in to every device from your laptop to your toaster!
  - Each address is 16 bytes... hard to even type correctly, let alone remember!

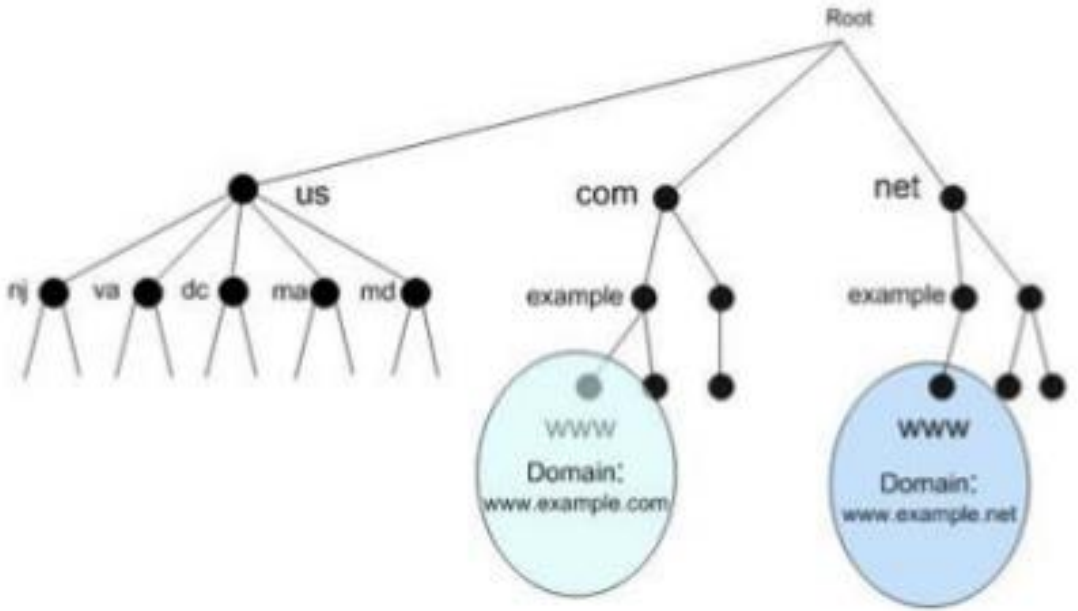
- Routers and servers are organized into a hierarchy and direct traffic

- Domain Name Service (DNS) is the book-keeper:
  - Each human readable name is mapped to a unique address (it can change over time)
  - Each computer and server can offer ***name resolution***, the ability to look-up addresses in the part of the internet that it's responsible for

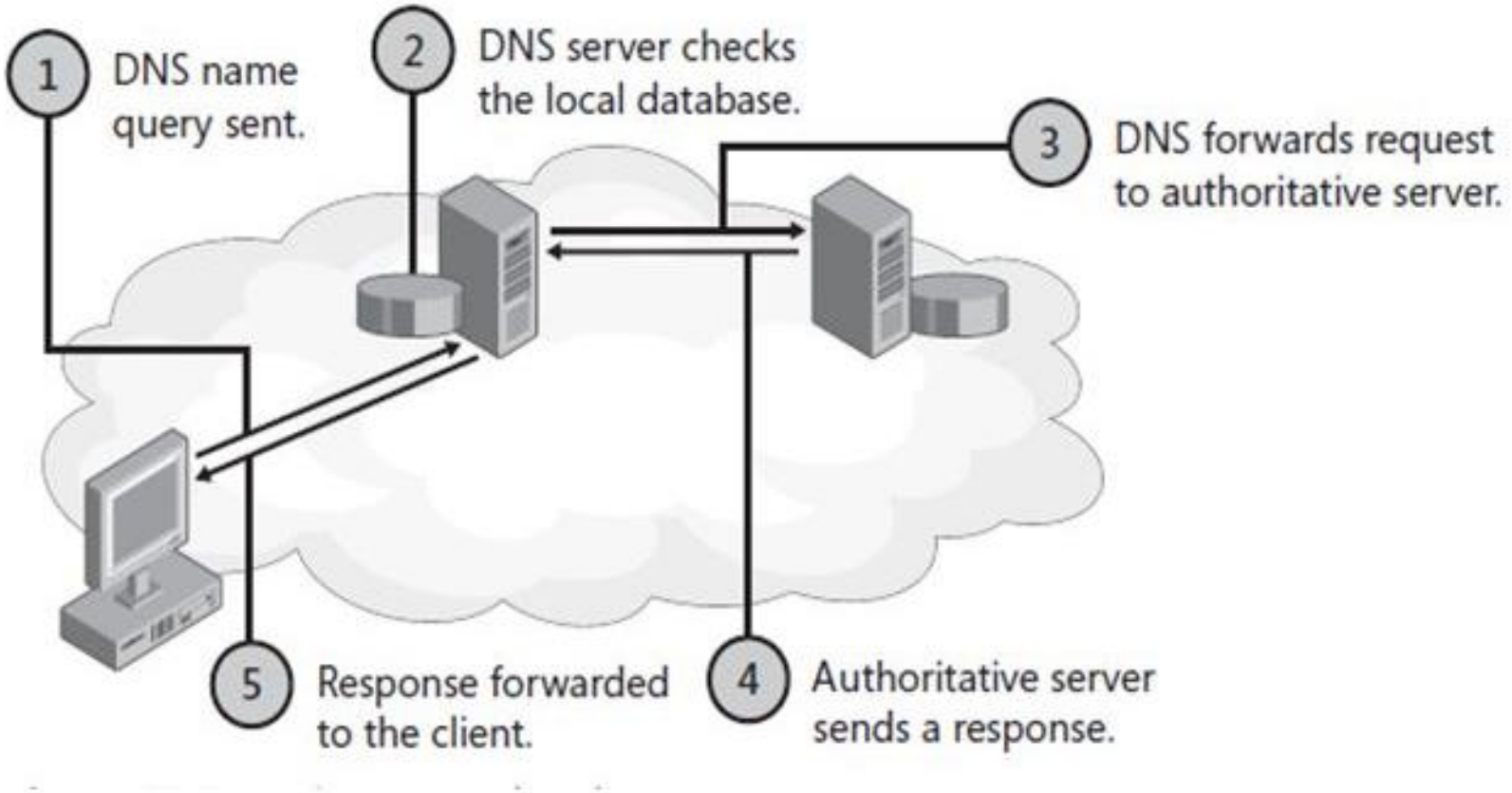
The DNS Hierarchy

- Each part of the name-space is recorded by a master server (along with redundancy)
- That either holds the entry, or knows where to look

- DNS queries go "up-then-down"
  - Originates at a leaf, checks if parent knows, if not, sends upwards
  - Once reach a level with the knowledge (maybe root), go downwards to ask the authority



DNS Dataflow



Networking Tools in Linux

- Commands:
  - ssh, ftp, scp, etc : Command-line networking
  - ifconfig : Report on network interfaces
  - ping : Send trivial packets to <source>, useful to look-up DNS addresses
  - iwconfig : Manage wireless connections
  - iptables, route, etc : Turn your PC into a network traffic switch
- Security:
  - Enforced through protected port ranges, optional firewalls (e.g., ufw), user authentication (e.g. LDAP)

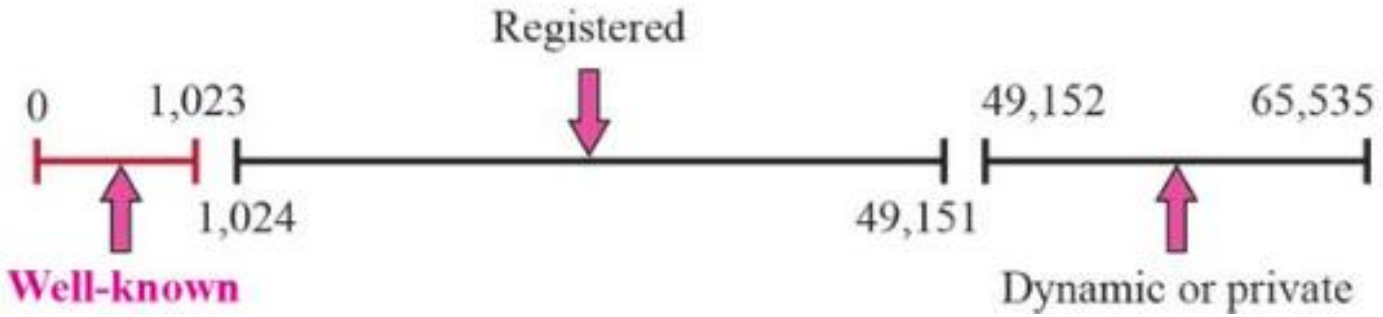
Beyond Addresses: Ports

- A computer must often maintain multiple connections
  - E.g. A web server streaming video to multiple users
  - E.g., Any computer at all since so many apps and resources live on the "cloud" currently

- But, we can only have one single address per device, so isn't this a collision?

ICANN Ranges

- **Well-known ports : 0 ~ 1,023**
  - Assigned and controlled by ICANN
- **Registered ports : 1,024 ~ 49,151**
  - Not assigned and controlled by ICANN
  - Can be registered with ICANN to prevent duplication
- **Dynamic ports : 49,152 ~ 65,535**
  - Can be used as temporary or private port number



A 2-byte integer: max value (2^16)

Within the standard range, known applications

- On the right is a standard list, but it can differ over time
- Constant ones to know for 206:
  - 20 & 21: FTP
  - 22 SSH/SCP
  - 25 SMTP (mail)
  - 53 DNS
  - 80 HTTP
  - 443 HTTPS

Which ports can I access?

- Access to ports enables the good internet functionality that we want, but it's also an opening for unwanted access

- A "firewall" is a software system designed to monitor internet communication and restrict dangerous activities.

- Firewalls can be implemented in the backbone, such as in a wifi router or McGill's central internet connection, as well as by each end-point device (client and server)

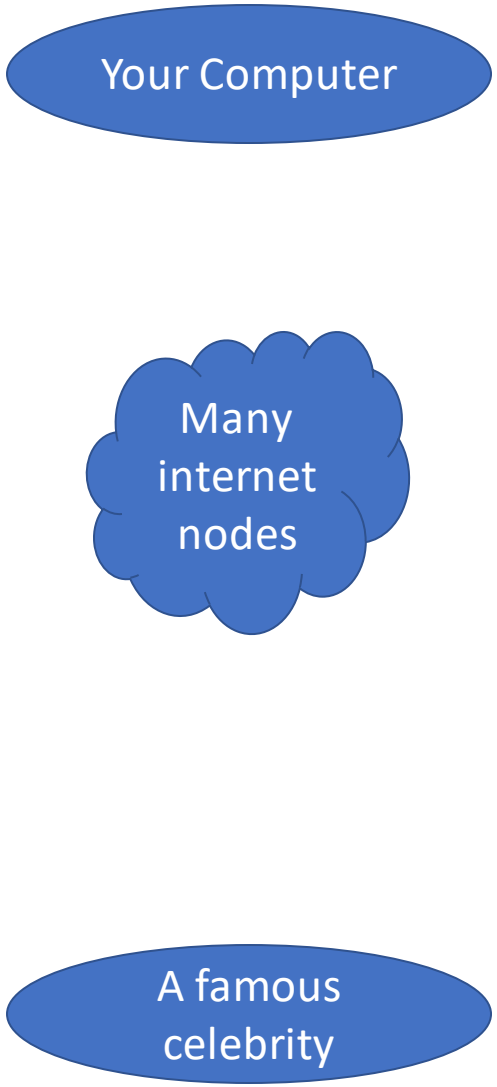
Typical firewall restrictions

- It is common that all ports outside those used for named applications are closed to traffic outside the "trusted" network
- For McGill, this can mean some functions do not work connecting to mimi from home.
  - VPN'ing is one way to fix this, it means your computer is now "virtually" part of the trusted network, but it still may not allow everything
- Because we cannot entirely change the networking settings everywhere, we oftentimes use "localhost", or 127.0.0.1 to demonstrate low-level functions. Do not be fooled, this still uses the "internet", but only the local implementation on a single computer.
- We will look at a few creative "work-arounds" in a while

Endianness on the internet

- Each device that hops our traffic through the internet needs to read the destination address to decide where to send the packet next

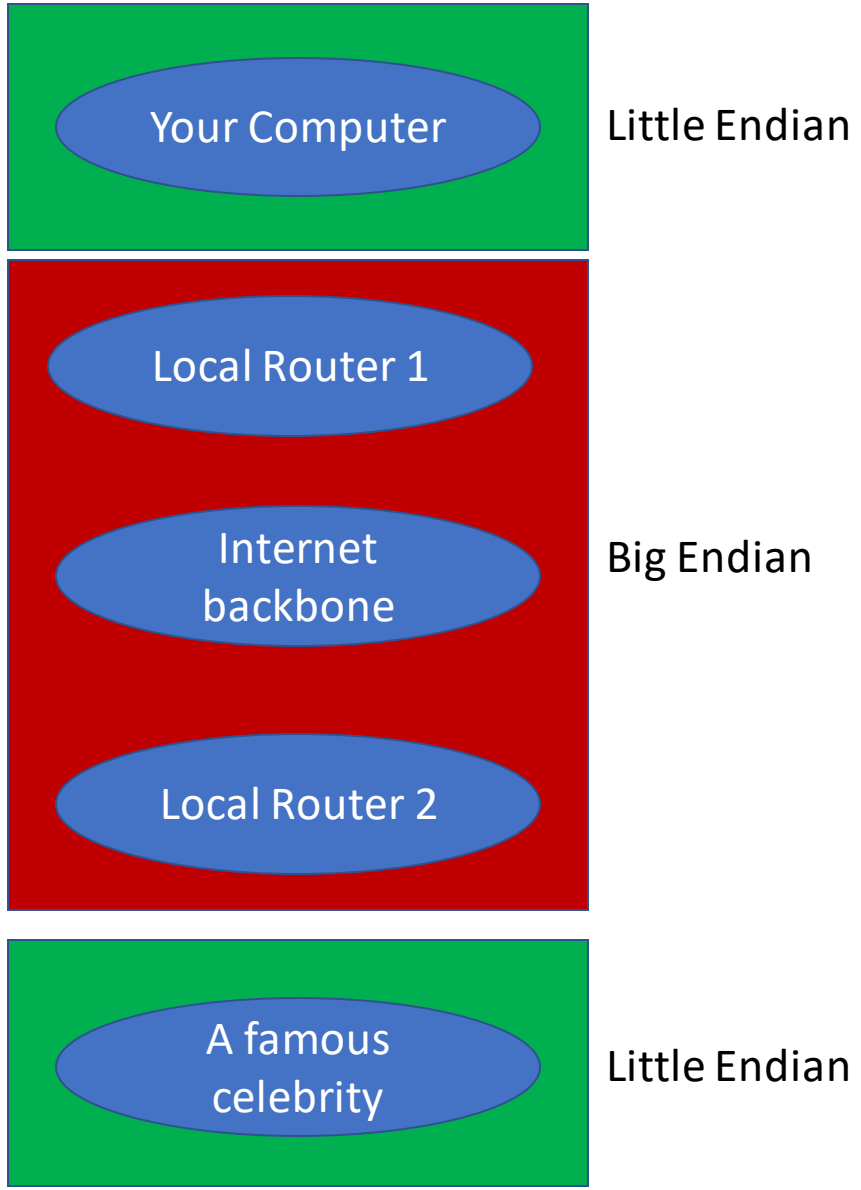
- What if those computers do not have the same endianness?



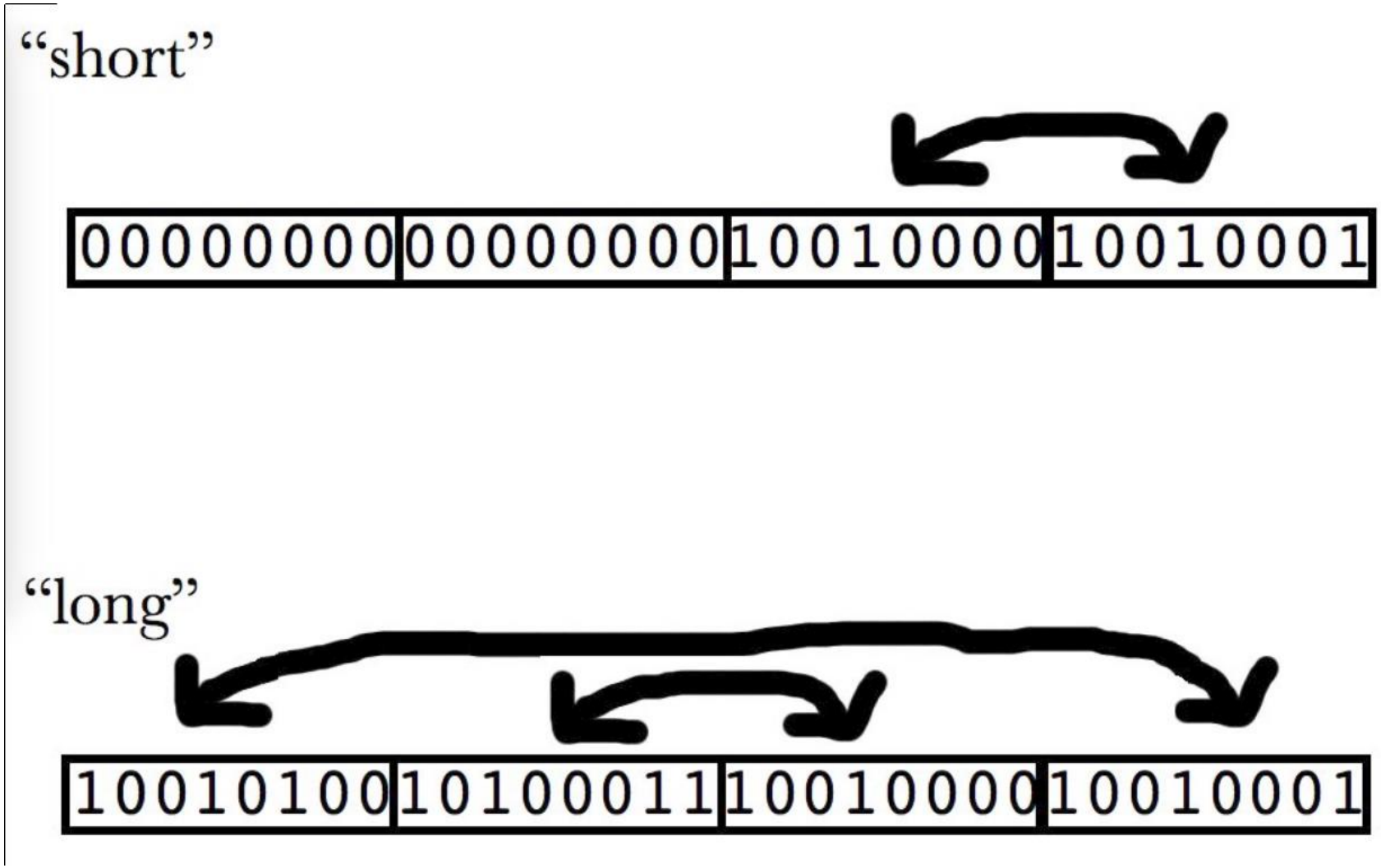
Endianness on the internet

- Each device that hops our traffic through the internet needs to read the destination address to decide where to send the packet next

- What if those computers do not have the same endianness?
  - In fact they very rarely will all agree.
  - Therefore, we must choose a standard: Big Endian is "network byte order". Everyone converts to that when the message is sent



Converting means swapping the right bytes



How to convert in a cross-platform way?

```
#include <arpa/inet.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

Description

The **htonl**() function converts the unsigned integer *hostlong* from host byte order to network byte order.

The **htons**() function converts the unsigned short integer *hostshort* from host byte order to network byte order.

The **ntohl**() function converts the unsigned integer *netlong* from network byte order to host byte order.

The **ntohs**() function converts the unsigned short integer *netshort* from network byte order to host byte order.

On the i386 the host byte order is Least Significant Byte first, whereas the network byte order, as used on the Internet, is Most Significant Byte first.

What else must be standarized?

- Anything important that we could inerpret differently between any 2 computers on the net. What have we seen so far that's important?
  - \0 ends C strings
  - \t means tab

  - \n means newline (or does it?)
  - \r means carriage-return (which carriage?)
  - \r\n means carriage-return then newline (...wait...)

What else must be standarized?

- Anything important that we could inerpret differently between any 2 computers on the net. What have we seen so far that's important?
  - \0 ends C strings
  - \t means tab

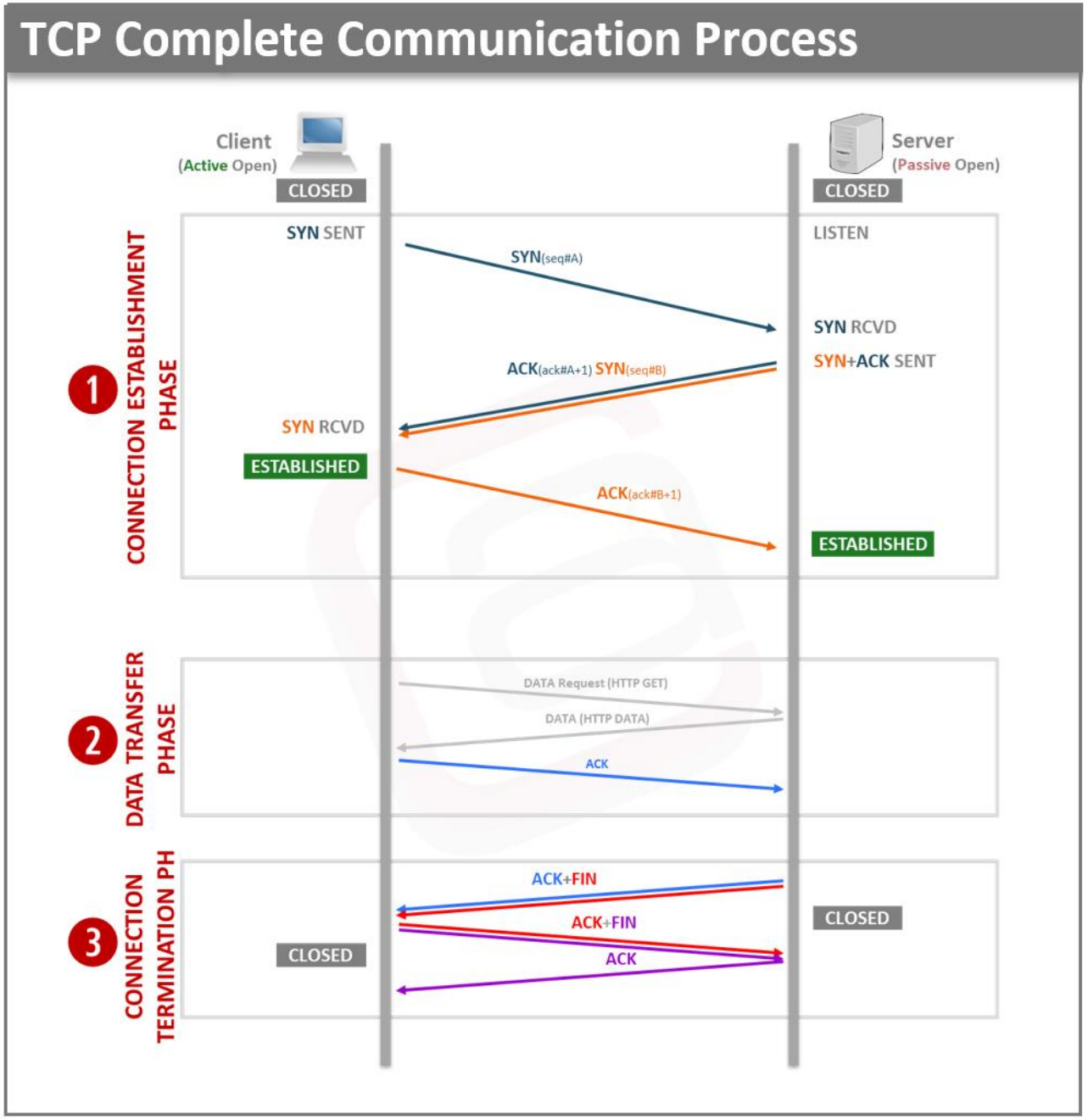
  - \n means newline (or does it?)
  - \r means carriage-return (which carriage?)
  - \r\n is the network "newline"

Thinking about a connection

- We said that 2 communicating generals could never coordinate, so how can we get data across a socket?
- #1: We don’t need to attack anyone, so the constraints on success are easier to meet
- #2: The Internet's idea is "best effort" in that we will always acknowledge the possibility of failure and move on eventually
- But within these bounds, do everything we get to get the data across

TCP Phases to know

- 1) 3-way handshake establishes the communication link, sets up parameters, agrees on plan to re-send, control congestion, etc
- 2) Each side is now able to send and receive
- 3) 3-way tear-down, ensures that both sides know the session is ending.



Next example: both send and receive

- We will look at the echo_server and echo_client in ExampleCode/Lecture18

- We'll use this code as a boot-strap towards our full chat functionality:

- Explore changing some of the parameters
  - Send a lot of data. Does it all arrive?
  - Try to run it on a different server. Which ports are OK and not?

Further Reading

- A sockets tutorial from RPI:
  - http://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/socket.html