

# 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

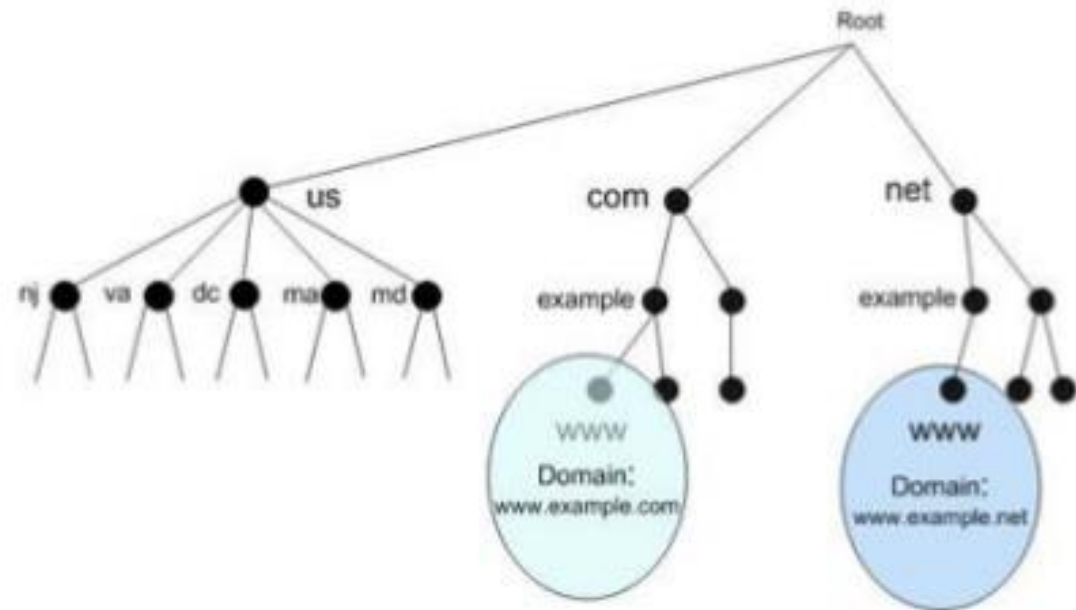
- Canada on the net: around 1985 with NetNorth
  - McGill is an early member, with researchers at what is now called "Centre for Intelligent Machines" (CIM) having many of the first connected devices
  - Historical information: the NetNorth Policies circa 1990: <http://retirees.uwaterloo.ca/~rwwatt/nnpp.html>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

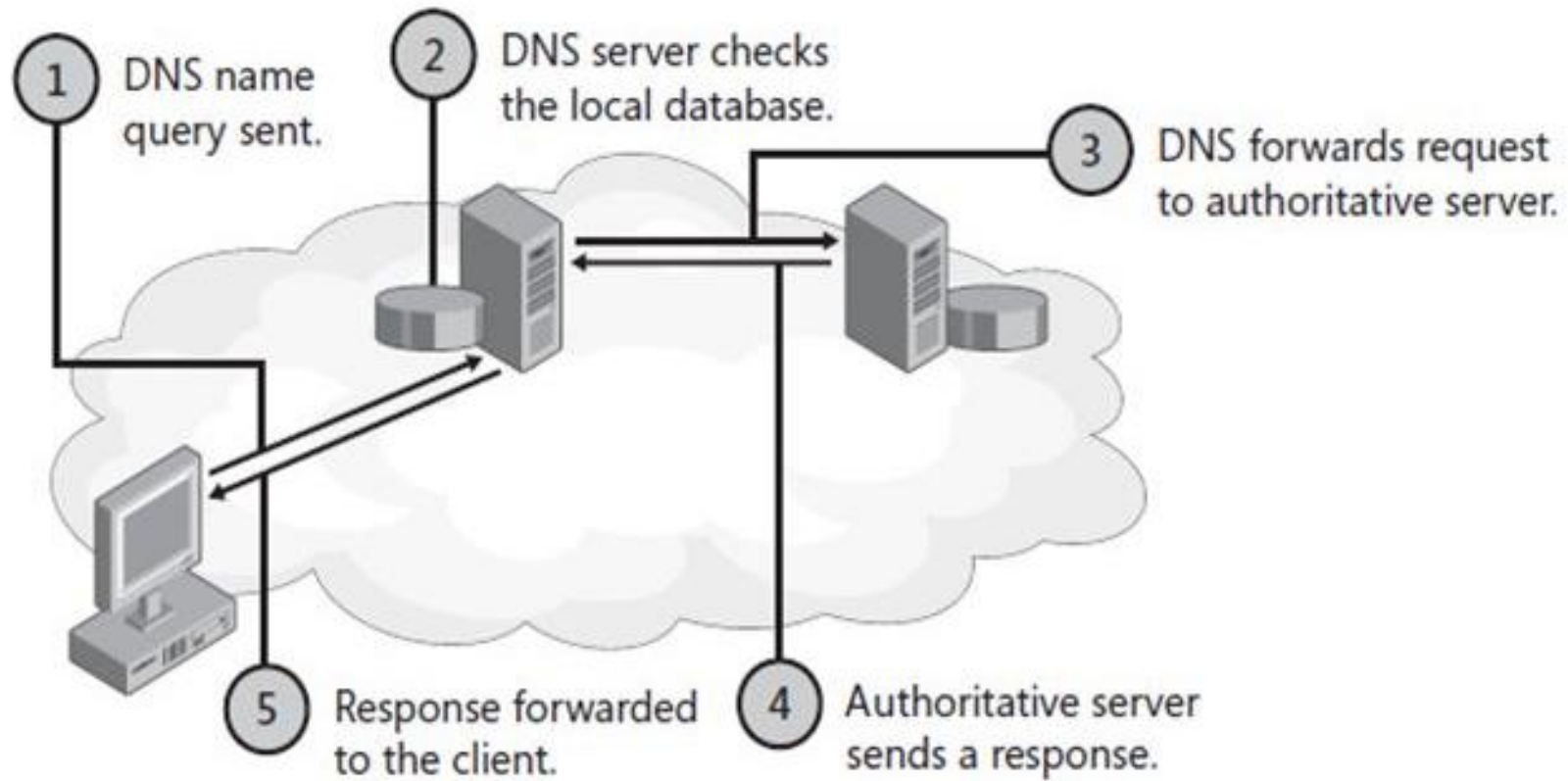
- IPv6 allows close to  $2^{128}$  devices to be online and uniquely addressable:
  - Log in to every device from your laptop to your toaster!
  - Each address is 16 bytes... hard to even type correctly, let alone remember!
- Routers and servers are organized into a hierarchy and direct traffic
- Domain 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

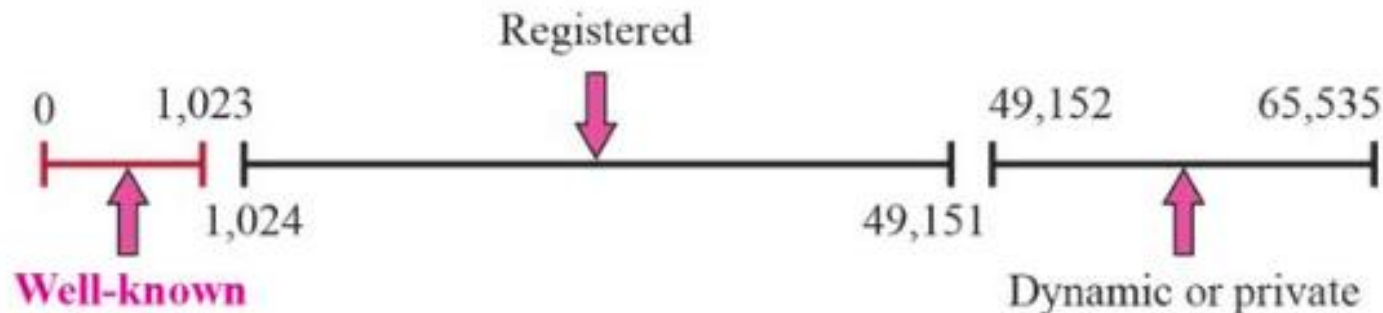
- ◆ Assigned and controlled by ICANN

## ❑ 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

20-21 FTP	560 mmonit	3050 Interbase DB	7212 GhostSurf
22 SSH/SCP	563 JSTP over SSL	3074 XBOX Live	7648-7649 CU SeeMe
23 Telnet	587 SMTP	3124 HTTP Proxy	8000 Internet Radio
25 SMTP	591 FileMaker	3127 MySquid	8080 HTTP Proxy
42 WINS Replication	593 Microsoft DCOM	3128 HTTP Proxy	8086-8087 Kaspersky AV
43 WHOS	631 Internet Printing	3222 GLBP	8118 Proxy
49 TACACS	636 LDAP over SSL	3260 iSCSI Target	8200 VMware Server
53 DNS	639 MSDP (PIM)	3306 MySQL	8500 Adobe ColdFusion
67-68 DHCP/BOOTP	646 LDP (MPLS)	3389 Terminal Server	8767 TeamSpeak
69 TFTP	691 MS Exchange	3689 iTunes	8866 Radmin
70 Gopher	860 iSCSI	3690 Subversion	9100 HP JetDirect
79 Finger	873 rsync	3724 World of Warcraft	9101-9103 Bacula
80 HTTP	902 VMware Server	3784-3785 Ventrilo	9119 Mail
88 Kerberos	989-990 FTP over SSL	4333 mSQL	9800 WebDAV
102 MS Exchange	993 IMAP over SSL	4444 Radmin	9898 Exim
110 POP3	995 POP3 over SSL	4664 Google Desktop	9988 Bandwidth
113 Ident	1025 Microsoft RPC	4672 Samba	9999 Urdin
119 NNTP (Usenet)	1026-1029 Windows Messenger	4899 Radmin	10000 Webmin
123 NTP	1080 SOCKS Proxy	5000 UUPnP	10000 BackupExec
135 Microsoft RPC	1080 MySquid	5001 Slingbox	10113-10116 NetQ
137-139 NetBIOS	1194 OpenVPN	5001 iperf	11371 OpenPGP
143 IMAP	1214 Kazaa	5004-5005 RTP	12035-12036 Setpoint LFE
161-162 SNMP	1241 Nessus	5050 Yahoo! Messenger	12345 NetBus
177 XDMCP	1311 Dell OpenManage	5060 SIP	13720-13721 NetBackup
179 BGP	1337 WASTE	5190 AMIRCQ	14567 WinMTR
201 AppleTalk	1433-1434 Microsoft SQL	5222-5223 XFP/Power	15118 Dnssec/Oddcast
264 BGMF	1512 WINS	5432 PostgreSQL	19226 AdminSecure
318 TSP	1589 Cisco VQP	5500 VNC Server	19638 Evim
381-383 HP Openview	1701 L2TP	5554 Radmin	20000 Usermin
389 LDAP	1723 MS PPTP	5631-5632 pcAnywhere	24800 Synergy
411-412 Direct Connect	1725 Stream	5800 VNC over HTTP	25999 X5e
443 JSTP over SSL	1741 CiscoWorks 2000	5900+ VNC Server	27015 Net-LFE
445 Microsoft DS	1755 MS Media Server	6000-6001 X11	27374 Sun?
464 Kerberos	1812-1813 RADIUS	6112 Battle.net	28960 Call of Duty
465 SMTP over SSL	1863 SSH	6129 GameWare	31337 Back Office
497 Reberspect	1985 Cisco HSRP	6257 WinMTR	33434+ Traceroute
500 Radmin	2000 Cisco SCCP	6346-6347 Gnutella	
512 rmxk	2002 Cisco ACS	6500 Kaspersky Arcade	
513 Radmin	2049 NFS	6566 SANE	
514 syslog	2082-2083 cPanel	6588 AnalogX	
515 LPD/LPR	2100 Oracle XDB	6665-6669 IRC	

Legend

- Chat
- Encrypted
- Gaming

# 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?



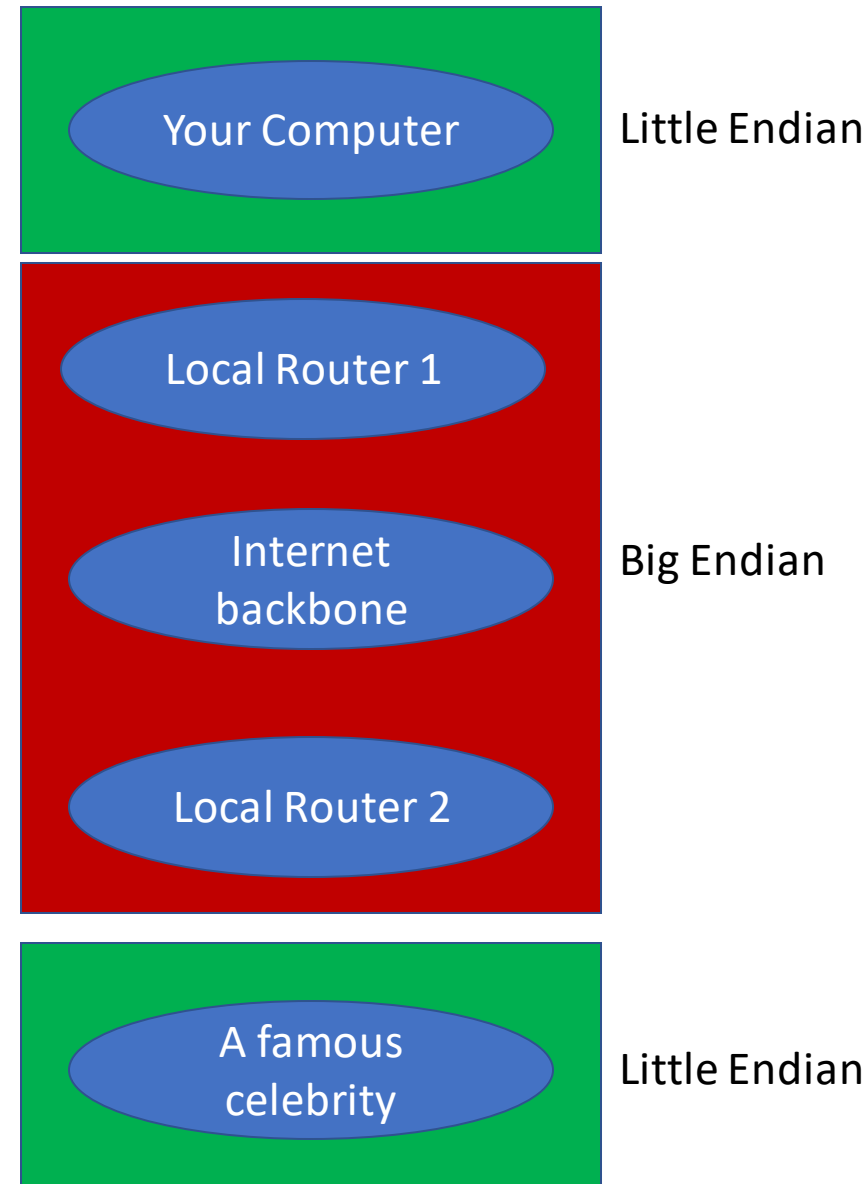
Your Computer

Many  
internet  
nodes

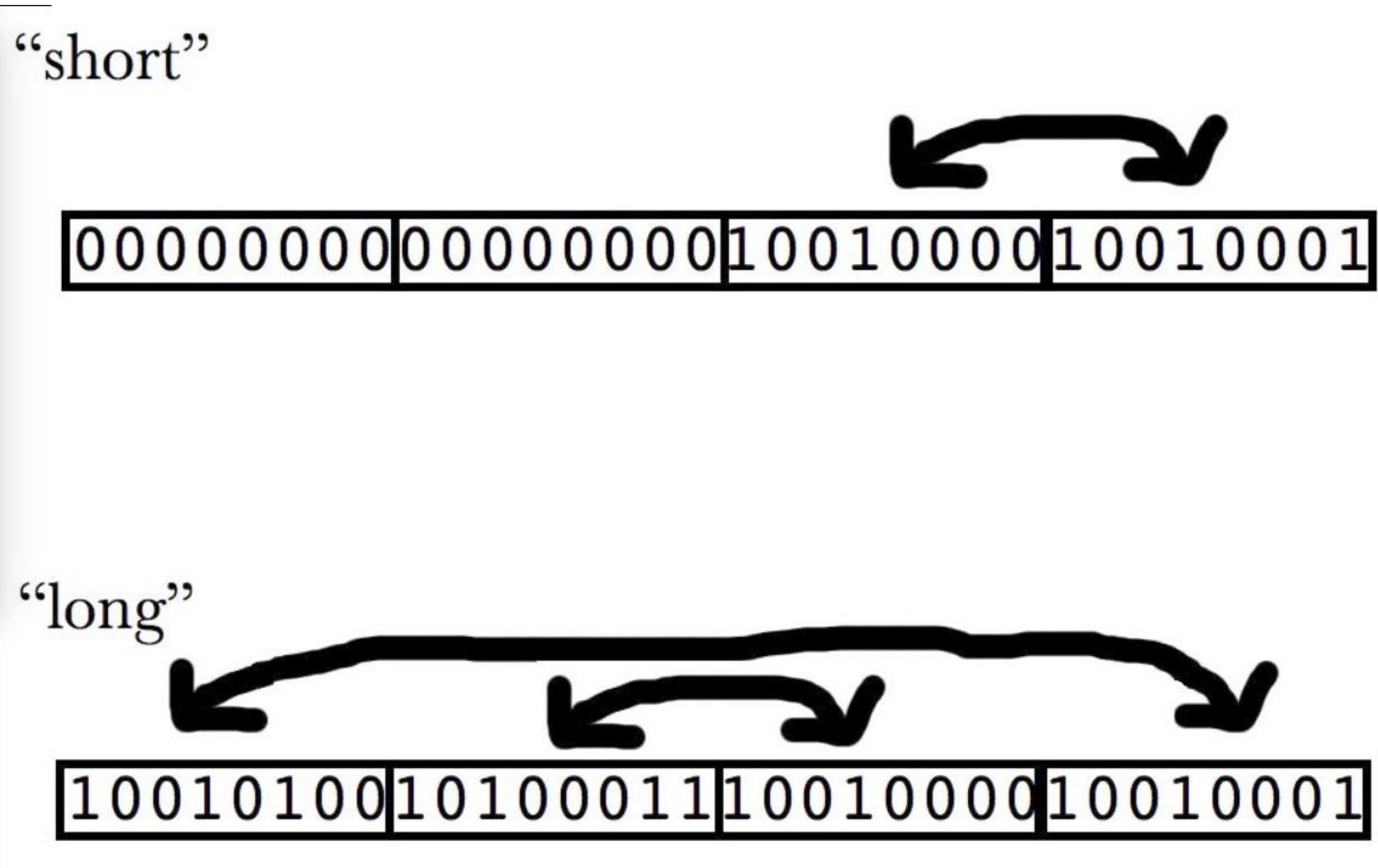
A famous  
celebrity

# 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 standardized?

- Anything important that we could interpret 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 standardized?

- Anything important that we could interpret 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"

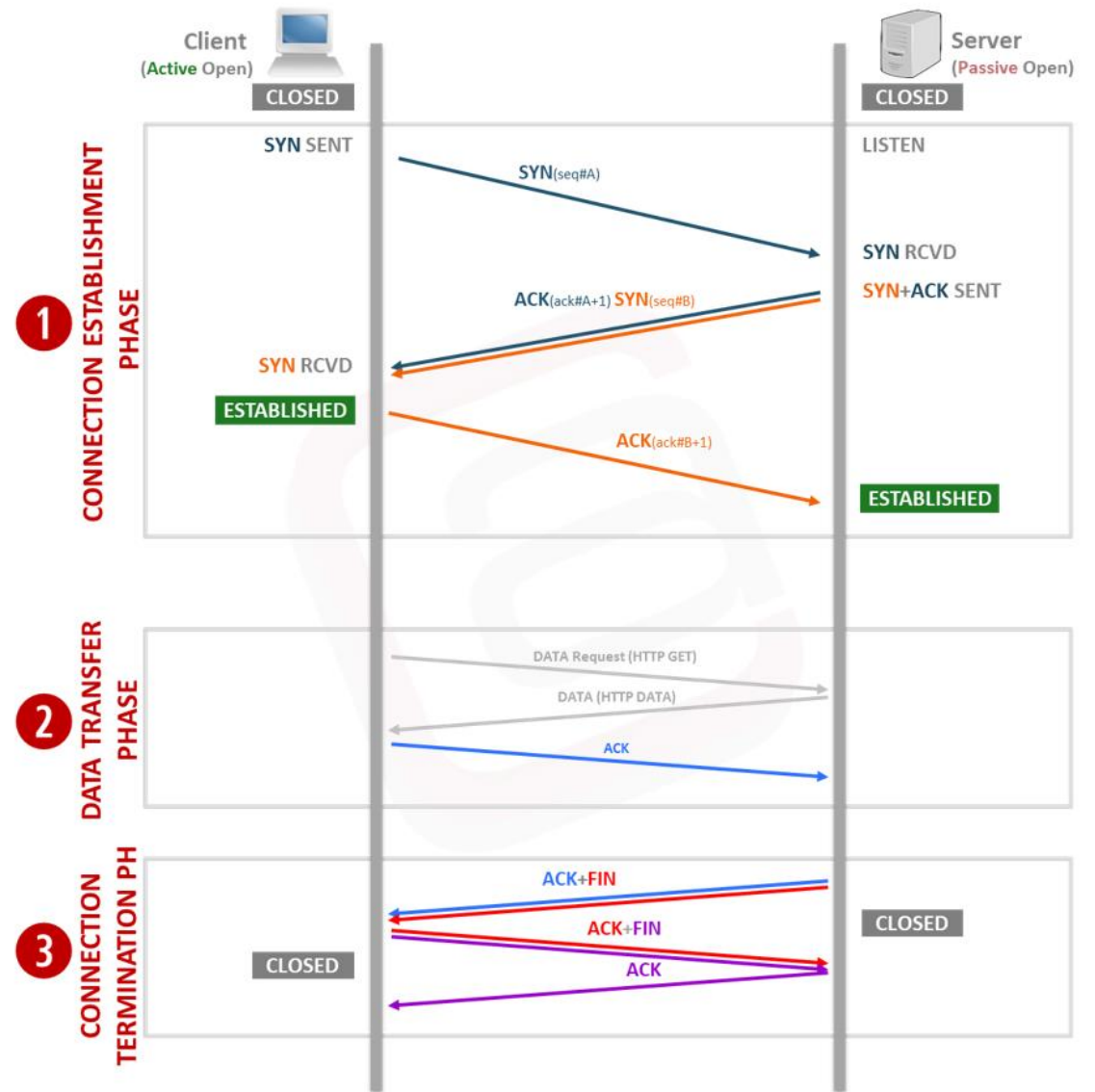
# Thinking about a connection

- 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.

## TCP Complete Communication Process



# 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>