Distributed Systems
Lecture 3

Networking Basics

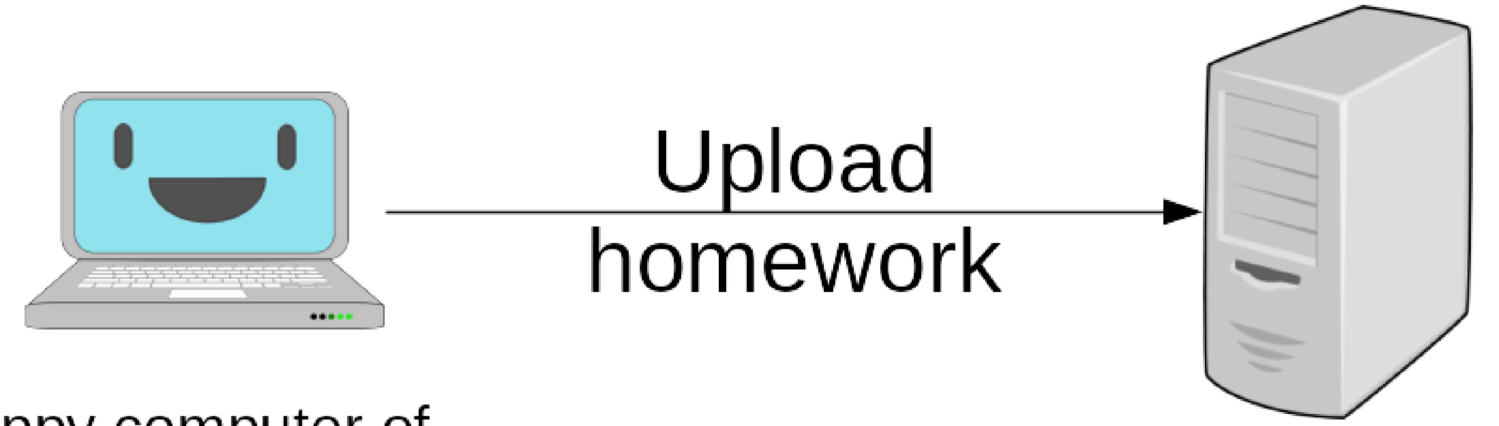
Joseph Phillips

Last modified 2019 September 12

Topics

- Networking
  - Ethernet
  - IP
  - TCP and UDP
- Byte ordering
- Character sets
  - Unicode
  - UTF 16
  - UTF 8
- Error handling
  - Java
  - C
  - C++
- Networking in Alternative Universes
  - Transport Layer Interface
  - Plan 9/Inferno

Motivation



Upload homework

Happy computer of happy student who just finished assignment

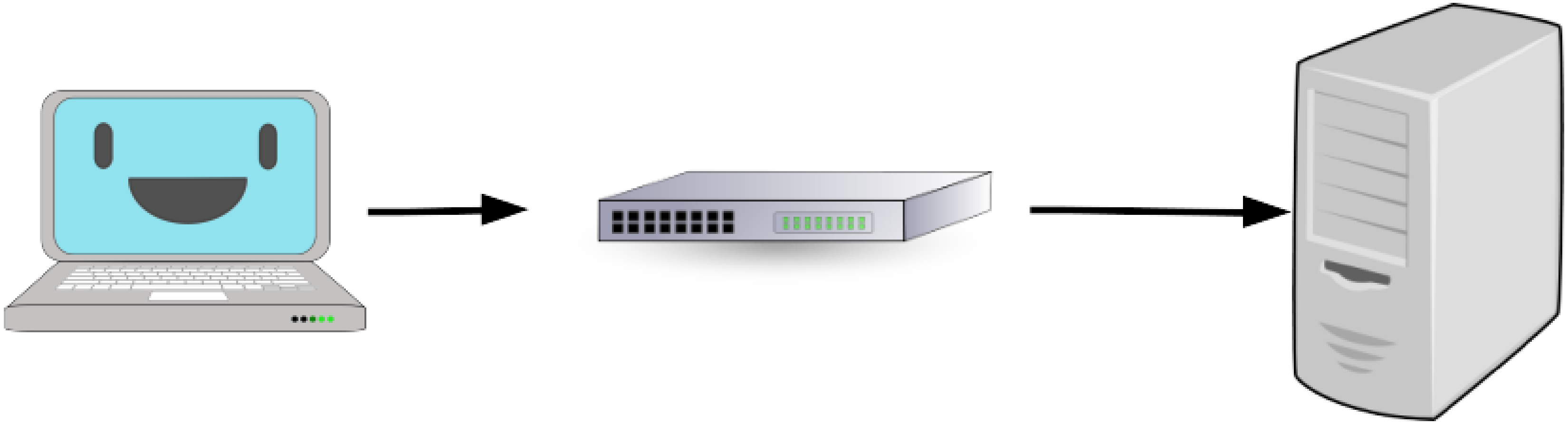
DePaul D2L server

A happy student!

They just finished their assignment.

Now, if there only was a way to upload it?

Attempt #1



The student attaches computer to same network as D2L server

Attempt #1 (cont’d)

- **Question**: How can we get all computers to talk to each other on the same network?
- **Answer**: There are several approaches. Among the most common are:
  - Ethernet
  - IEEE 802.2/802.3
  - Wi-Fi
  - ZigBee

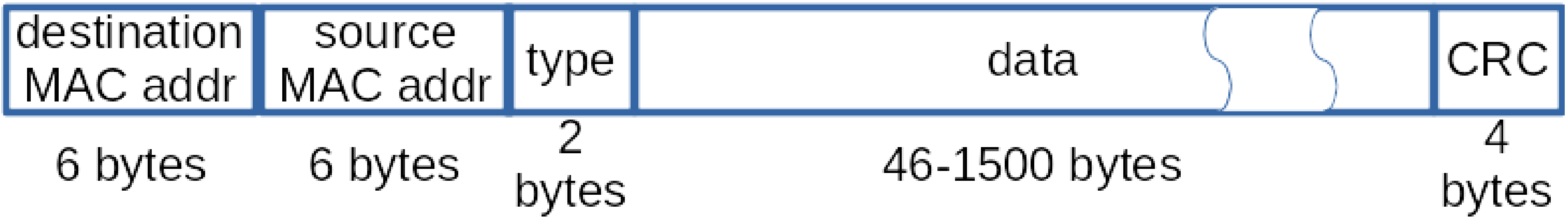
But first, make sure each ***network card*** is unique

- MAC: To distinguish one ***network interface*** from another
  - Media Access Control
  - 48-bits
  - Assigned by IEEE
    - network card manufacturers buy them in blocks
  - Unique for each ***network card***
    - Replace computer’s network card? ***It has a new MAC address!***

Your turn!

What type of “computer”
would have more than one MAC address?

And now, an Ethernet Frame



destination MAC addr	source MAC addr	type	data	CRC
6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes

- MAC addresses
  - ***to*** and ***from*** whom?
- Type: *What the heck am I carrying?*
  - `0x0800`: IPv4
  - `0x86DD`: IPv6
  - `0x0806`: Address Resolution Protocol (ARP)
  - `0x8035`: Reverse Address Resolution Protocol (RARP)
  - (There are many more)

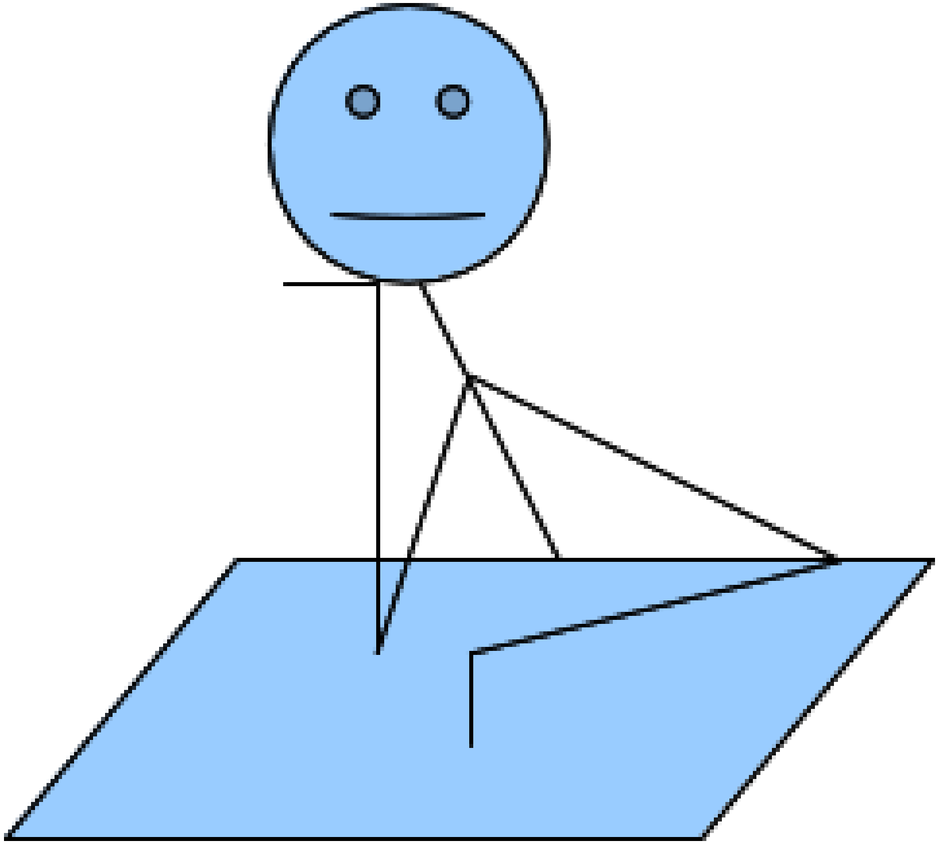
- Data
  - An IP frame (stay tuned!)
- CRC
  - “Cyclic Redundancy Check”
  - Fixed-sized hash value
  - Good at detecting “burst errors” (sequence of wrong data)
  - Easy to compute in hardware

Variations on basic Ethernet

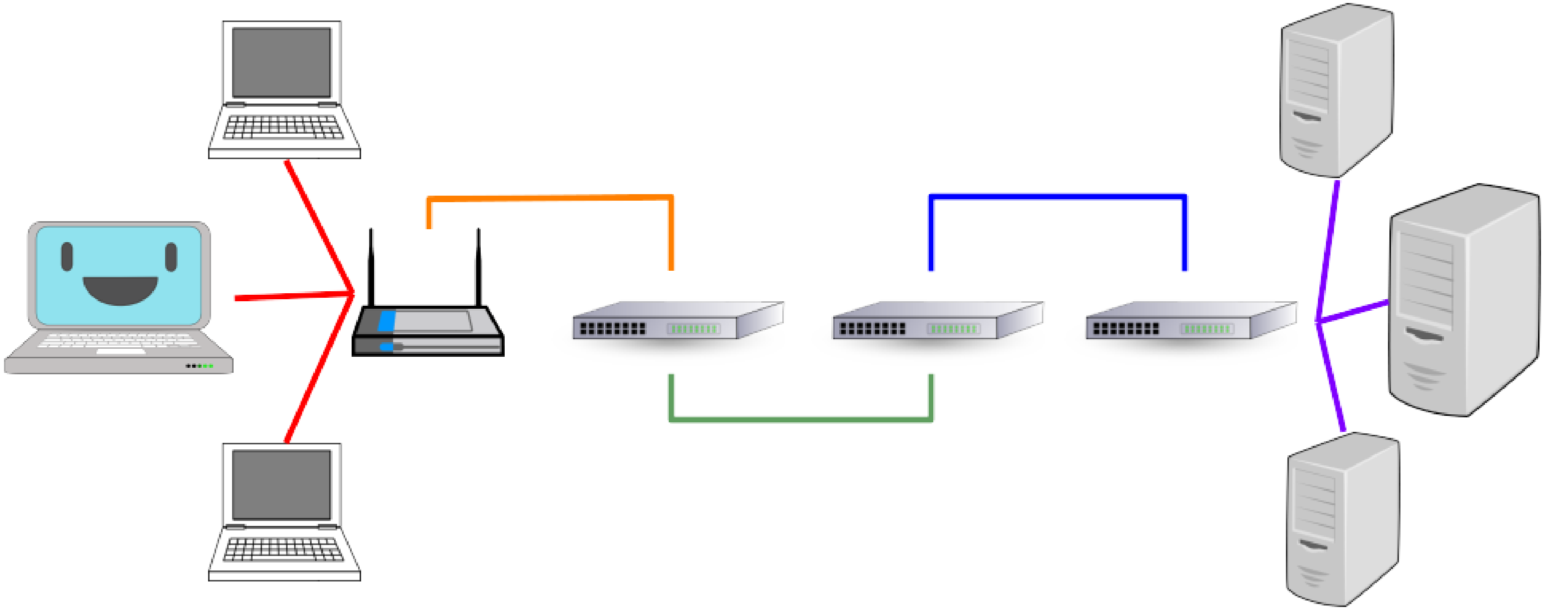
- IEEE 802.2/802.3
  - Similar to Ethernet frame
  - Steal some 8 bytes from data for more meta-info
- IEEE 802.11 (Wi-Fi), Has many more fields
  - Power management,
  - “this is a re-try frame”, etc.
- IEEE 802.15.4 (ZigBee): For Internet-of-Things
  - Multi-hop relay
  - Low power (e.g. battery)
  - Discovery of the number of devices in the network
  - Security

Ruh-roh!

Astute student:
“*DePaul is **too cheap** to lay cable to every student’s house. Students are **too lazy** to walk to DePaul!*”



Attempt #2

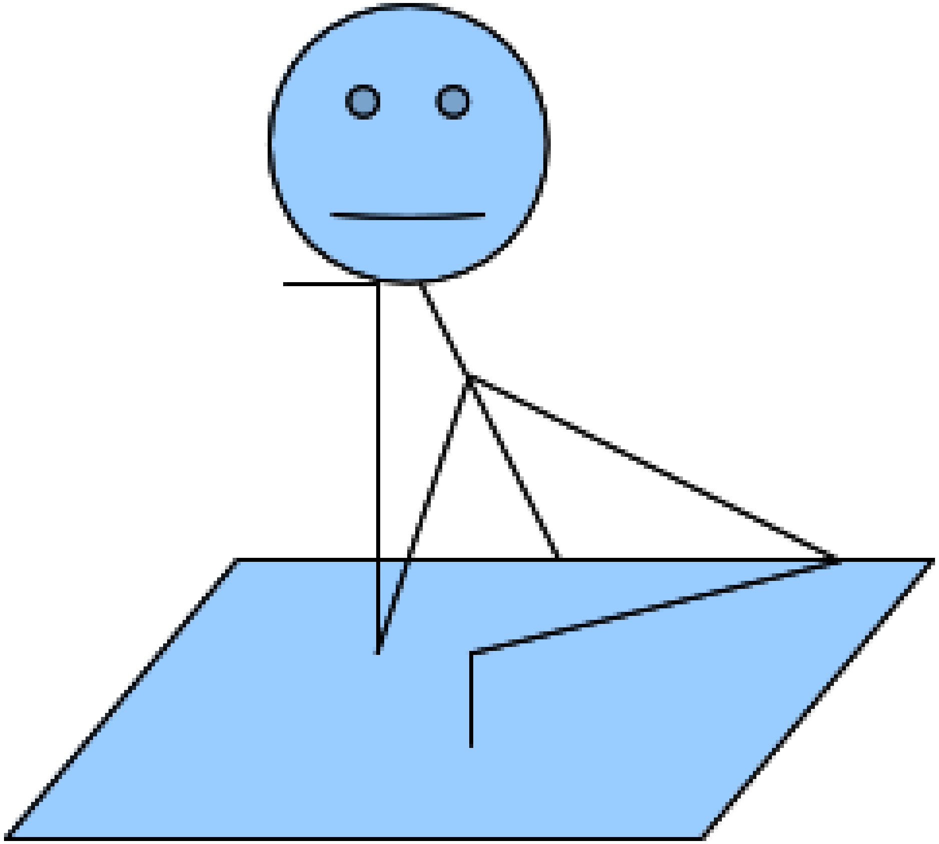


- Let’s ***not*** use just ***one*** network.
- Let’s use a ***network-of-networks***!
  - a.k.a. an ***internet***
  - “***i***nternet”: (small “i”) a network-of-networks
  - “***I***nternet”: (big “I”) ***The*** worldwide network-of-networks-of-networks

Attempt #2

Astute student: “*Hey! All those networks have **different protocols!***

*How can we get **any machine** on one network to talk to **any machine** on another network?*”



But first: Addressing computers on different networks

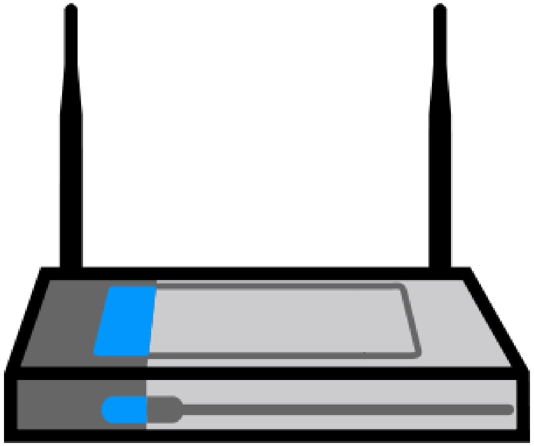
- With ***one network*** we could use the MAC address
- With ***an internet*** we need another scheme
- The ***IP address!***
  - Abstracts away from MAC address
  - A distributed registry of IP addresses to networks that host them (and know their MAC addresses)
- ***DNS*** (Domain Name Service)
  - Computers love integers, but humans love strings
  - IP address => name
  - name => IP address

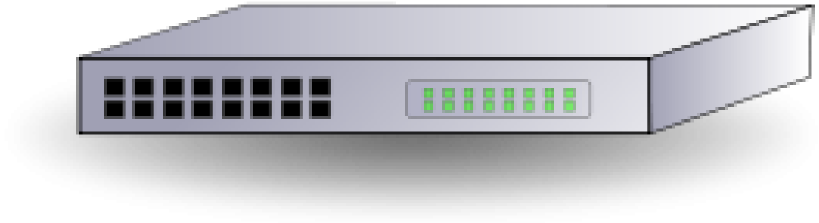
And now . . . Attempt #2: Talking

- We can address another computer on another network
- Now we have to ***talk***
- There really are two issues here . . .
  - A multi-network issue (“hardware”)
  - A multi-protocol issue (“software”)

Attempt #2: “Hardware”

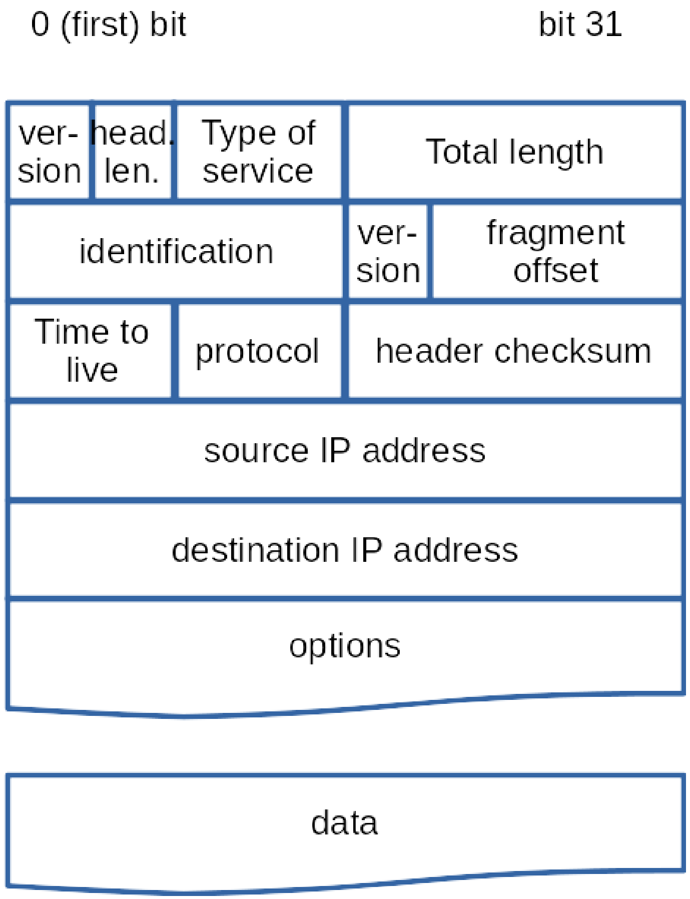
- Introducing the ***bridge***
  - a device to bridge different networks
- A ***router***
  - bridge between different ***types*** of networks
  - e.g. Ethernet and Asymmetric Digital Subscriber Line (ADSL)
  - e.g. Your home wireless





Attempt #2: “Software”

- IP
  - Says where it is going with an ***IP address***
    - Not just a ***this-network-only*** MAC address
  - Version
    - E.g. IPv4
  - Type of service:
    - minimize delay, or,
    - maximize throughput, or,
    - maximize reliability, or,
    - minimize monetary cost
  - Identification
    - Integer incremented for subsequent datagrams
  - Time to live:
    - How many more “hops” it can do before it dies
    - Prevents infinite looping



0 (first) bit

bit 31

ver-sion | head. len. | Type of service | Total length

identification | ver-sion | fragment offset

Time to live | protocol | header checksum

source IP address

destination IP address

options

data

IPv4 vs. IPv6

	IPv4	IPv6
Address	4 octets (32 bits)	16 octets (128 bits)
Notation	Decimal-dot	Hexadecimal-colon
Security	Achievable by higher levels	Encryption and authentication at this level
Maximum packet	65535 bytes	4,294,867,295 bytes (“jumbogram”)

But wait! There’s more . . .

- Besides IP, there are other low-level protocols
  - IGMP: Internet Group Management Protocol
    - Manages multi-cast groups
  - ICMP: Internet Control Message Protocol
    - Router-to-router communication about errors, etc.
    - Used by ping, traceroute

ping

- “Are you there? / Can I reach you?”

```
$ ping www.google.com
PING www.google.com (172.217.4.36) 56(84) bytes of data.
64 bytes from lga15s46-in-f4.1e100.net (172.217.4.36): icmp_seq=1 ttl=54
time=11.4 ms
64 bytes from lga15s46-in-f4.1e100.net (172.217.4.36): icmp_seq=2 ttl=54
time=11.6 ms
64 bytes from lga15s46-in-f4.1e100.net (172.217.4.36): icmp_seq=3 ttl=54
time=9.11 ms
64 bytes from lga15s46-in-f4.1e100.net (172.217.4.36): icmp_seq=4 ttl=54
time=9.15 ms
64 bytes from lga15s46-in-f4.1e100.net (172.217.4.36): icmp_seq=5 ttl=54
time=8.42 ms
^C
--- www.google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 8.422/9.960/11.630/1.335 ms
```

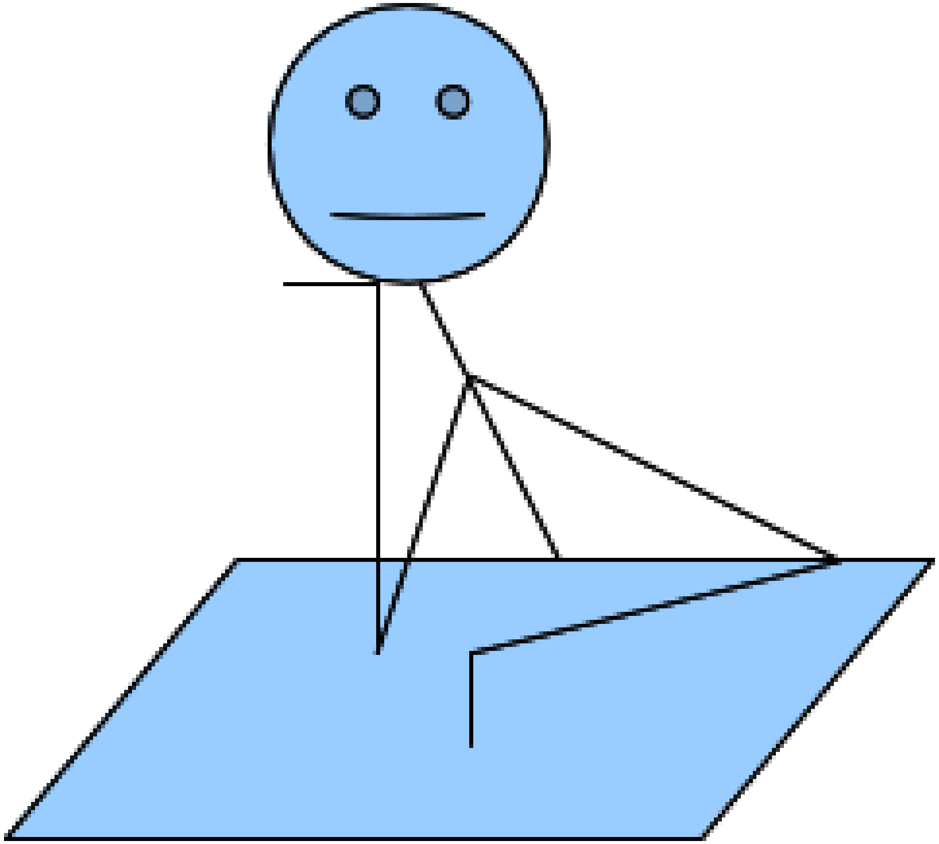
traceroute

- “How did my packet get to you?”

```
$ traceroute 140.192.36.187
traceroute to 140.192.36.187 (140.192.36.187), 30 hops max, 60 byte packets
 1  gateway (192.168.0.1)  1.807 ms  2.609 ms  3.661 ms
 2  96.120.24.253 (96.120.24.253)  21.038 ms  21.320 ms  21.968 ms
 3  be141-sur07.area4.il.chicago.comcast.net (68.85.179.161)  22.698 ms  22.336 ms  24.710 ms
 4  be-110-ar01.area4.il.chicago.comcast.net (68.86.184.253)  27.049 ms  27.665 ms  27.357 ms
 5  be-33491-cr02.350ecermak.il.ibone.comcast.net (68.86.91.165)  28.351 ms  31.459 ms
31.797 ms
 6  be-10563-pe01.350ecermak.il.ibone.comcast.net (68.86.82.158)  29.445 ms  12.555 ms
21.235 ms
 7  50.242.150.34 (50.242.150.34)  22.936 ms  22.140 ms  22.447 ms
 8  0.ae4.cr2.ord6.scnet.net (204.93.204.87)  23.143 ms  23.439 ms 0.ae12.cr2.ord6.scnet.net
(204.93.204.83)  25.377 ms
 9  0.ae2.ar10.ord6.scnet.net (204.93.204.81)  24.552 ms  24.166 ms  25.500 ms
10  unknown.servercentral.net (50.31.167.142)  31.764 ms  31.978 ms  31.264 ms
11  140.192.10.5 (140.192.10.5)  33.778 ms  34.763 ms  34.340 ms
12  mfc-cst-5e-te13-6.netequip.depaul.edu (140.192.9.229)  16.235 ms  20.583 ms  19.894 ms
```

Ruh-roh again!

- Astute student “*Hey, what you propose is an **unreliable stream of bytes**, it has **no meaning**.*”

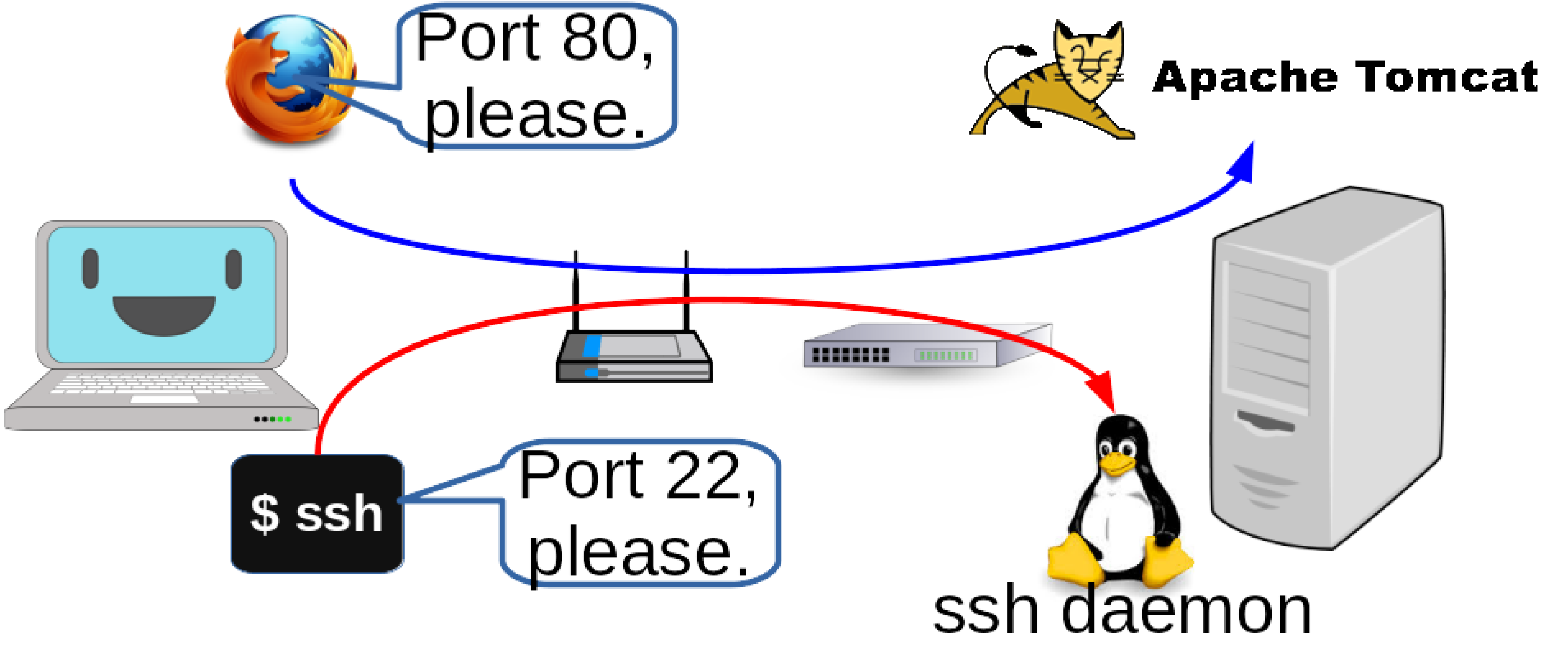


Attempt #3

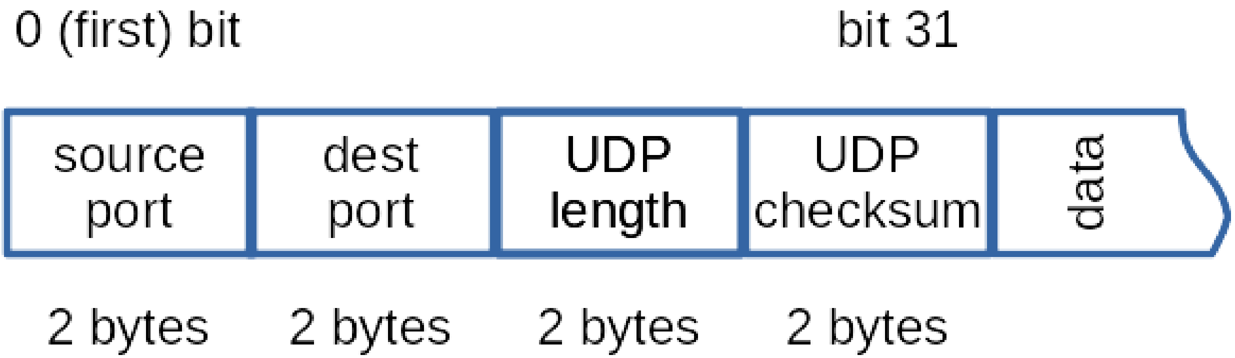
- Let us build a protocol on top of IP
- Nay, let us build 2 protocols
  - UDP
  - TCP

But first, make sure we are talking to the correct service

- Introducing the ***port***
  - Implies which service
  - An integer 1 to 65535 ***and*** a protocol (TCP or UDP)
  - Example:
    - ssh => TCP port 22;   http => TCP port 80
    - Both ssh and http packets can travel same path, but not get confused



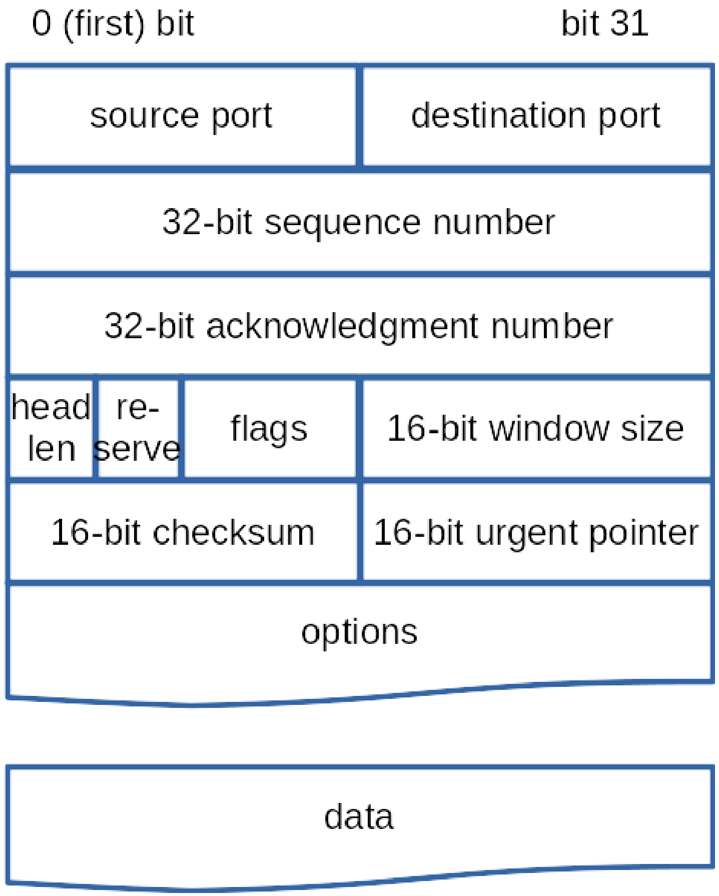
And now, UDP



- Port
  - Implies which service on a given machine
  - TCP port 22: Secure SHell (ssh)
  - TCP port 23: Telnet
  - TCP port 80: http
  - TCP port 443: https
- Checksum
  - Helps detect errors

TCP

- Ports, checksum, and . . .
- Sequence number
  - For receiver to reassemble stream
- Acknowledgment number
  - Next sequence number expected



0 (first) bit

bit 31

source port

destination port

32-bit sequence number

32-bit acknowledgment number

head len

re-serve

flags

16-bit window size

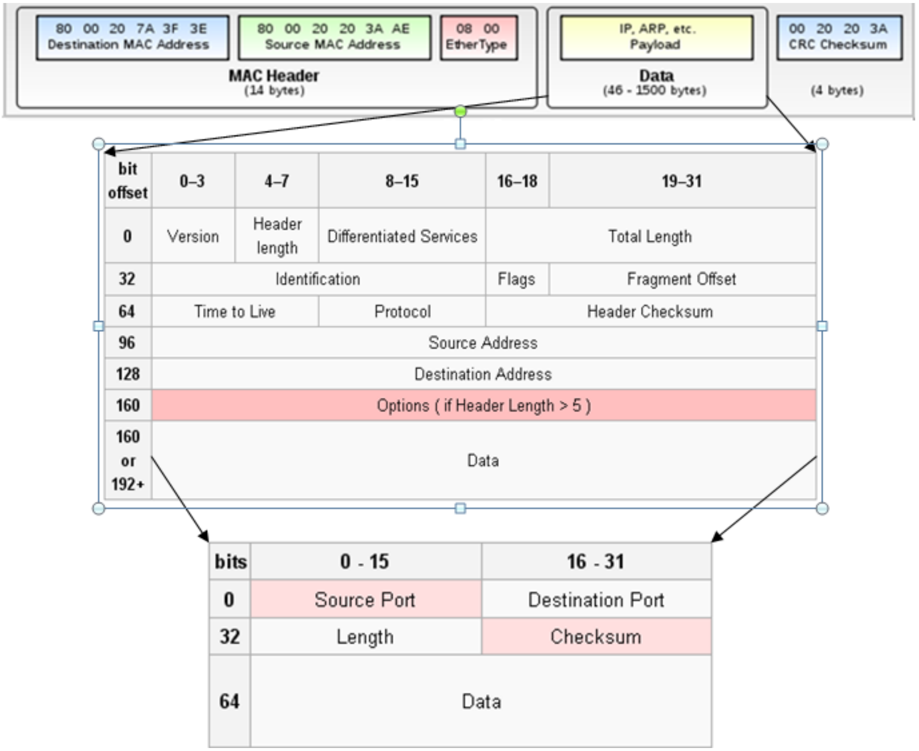
16-bit checksum

16-bit urgent pointer

options

data

Yay! TCP/IP Saves the Day!



bit offset	0–3	4–7	8–15	16–18	19–31
0	Version	Header length	Differentiated Services	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options (if Header Length > 5)				
160 or 192+	Data				

bits	0 - 15	16 - 31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

Application (e.g. FTP)

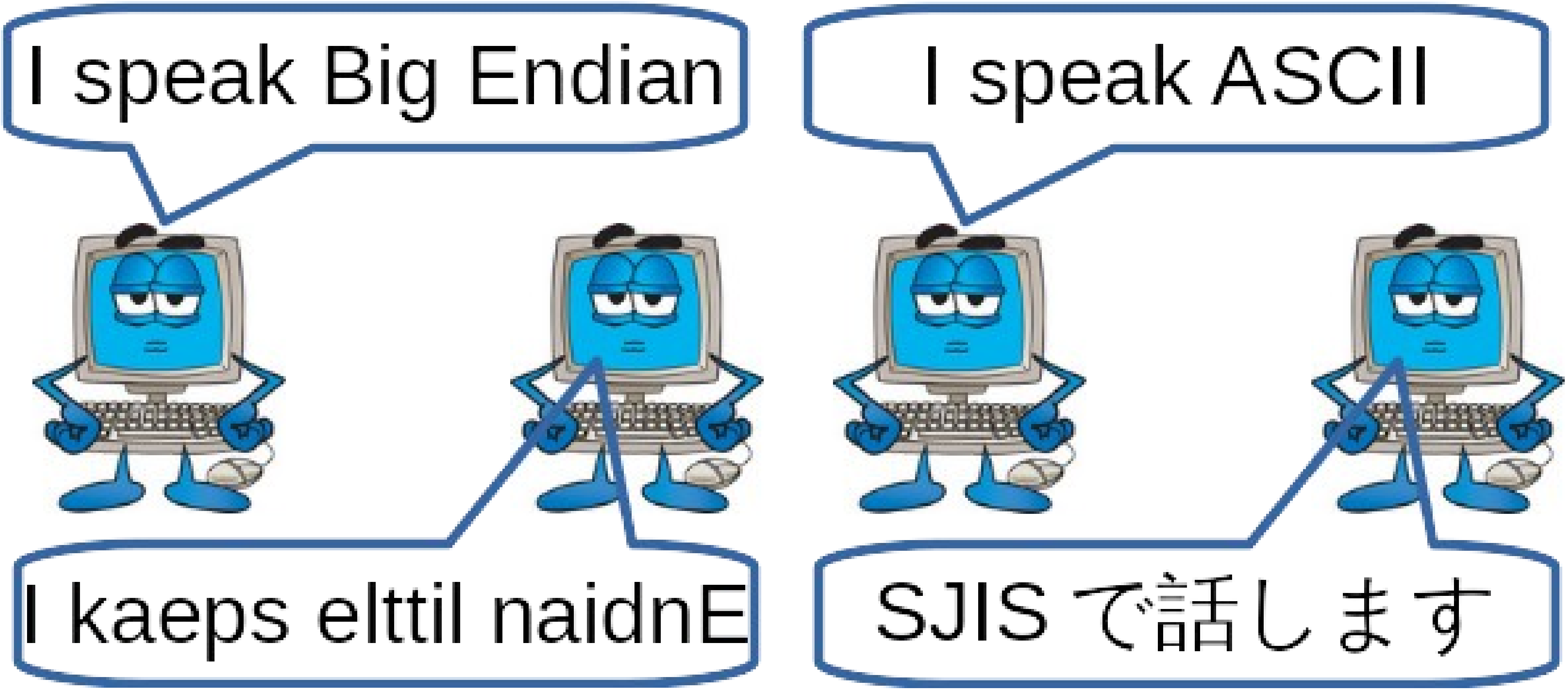
Transport (e.g. UDP)

Internet (e.g. IP)

Network (e.g. Ethernet)

Hardware (e.g. voltages)

Well, not quite: must consider application layer interoperability



Byte-ordering: Java

- We define the network ordering to be Big Endian
- Java coder? You’re covered!
  - Java is defined to represent data in Big Endian

Byte-ordering: C/C++

- We define the network ordering to be Big Endian
- C/C++ coder? Just a *little* more work . . .

```
#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);
```

And if you *know* the endian you want

```
#include <endian.h>

uint16_t htobe16(uint16_t host_16bits);  // Host To Big Endian
uint16_t htole16(uint16_t host_16bits);  // Host To Little Endian
uint16_t be16toh(uint16_t big_endian_16bits);
uint16_t le16toh(uint16_t little_endian_16bits);

uint32_t htobe32(uint32_t host_32bits);
uint32_t htole32(uint32_t host_32bits);
uint32_t be32toh(uint32_t big_endian_32bits);
uint32_t le32toh(uint32_t little_endian_32bits);

uint64_t htobe64(uint64_t host_64bits);
uint64_t htole64(uint64_t host_64bits);
uint64_t be64toh(uint64_t big_endian_64bits);
uint64_t le64toh(uint64_t little_endian_64bits);
```

Your Turn!

Write your own
uint64_t htonll(uint64_t hostLongLong)
uint64_t ntohll(uint64_t netLongLong)

Character Sets

- Unicode
  - Extends ASCII
  - Defines 1,112,064 “code points”
    - Chars of living and dead langs
    - control chars
- Represent in hexadecimal
  - U + *hhhh*
- Does ***not*** say how to represent bits of *hhhh*.

Unicode, cont’d

- UTF-32:
  - 4 bytes used to represent each char
  - Advantage:
    - Straight-forward
  - Disadvantage:
    - Wasteful of bytes
  - Used by:
    - C/C++ internal wchar_t

- UTF-16:
  - Either 2 bytes or 4 bytes
  - Advantage:
    - Less bytes then UTF-32 (for most chars but Chinese)
  - Disadvantage:
    - More complicated to handle 2 byte vs 4 byte chars
  - Used by:
    - Java internal Char

Unicode, cont’d

- UTF-8
  - Uses 1, 2, 3 or 4 “octets” (8-bit bytes)
  - Advantage:
    - More efficient than even UTF-16 (esp. for Latin chars)
  - Disadvantage:
    - More complicated to handle variable-lengthed chars
  - Used by:
    - Unix
    - Web pages

Unicode in C/C++

- Includes:
  - wchar.h, locale.h
- locale:
  - LC_COLLATE: string collation
  - LC_CTYPE: character type
  - LC_MESSAGES: natural language messages
  - LC_MONETARY: money formatting
  - LC_NUMERIC: number formatting
  - LC_TIME: date and time formatting
  - LC_ALL: Everything
  - setlocale(LC_ALL,""); // Set default for current system
  - setlocale(LC_ALL,"en_US.UTF-8"); // Set for particular lang and charset
- Types:
  - wchar_t, wchar_t*
- Constants:
  - wchar_t hiraganaA = L'\x3042';
  - wchar_t* L"日本語で";

Unicode in C: Output
wchar_t both internal & external

```
#include <stdio.h>
#include <locale.h>
#include <wchar.h>

int wprintf(const wchar_t *format, ...);
int fwprintf(FILE *stream, const wchar_t *format, ...);
int swprintf(wchar_t *wcs, size_t maxlen,
                    const wchar_t *format, ...);

setlocale(LC_ALL,"en_US.UTF-8");
wprintf(L"char: '%c' wchar_t: '%lc'\n",'A',L'\x3042');
wprintf(L"char*: %s, wchar_t* %ls\n","in English",L"日本語で");
```

Unicode in C: Output
wchar_t internal, char external

```
void   printUtf8Char(wchar_t c
                    )
{

  if  (c <= 0x7F)
    putchar(c);
  else
  if  (c <= 0x7FF)
  {
    putchar( 0xC0 | (c >> 6) );
    putchar( 0x80 | (c & 0x3F) );
  }
  else
```

Unicode in C: Output wchar_t internal, char external

```
  if  (c <= 0xFFFF)
  {
    putchar( 0xE0 | (c >> 12) );
    putchar( 0x80 | (0x3F&(c>>6)));
    putchar( 0x80 | (0x3F & c) );
  }
  else  if  (c <= 0x1FFFFF)
  {
    putchar( 0xF0 | (c >> 18) );
    putchar( 0x80 | (0x3F & (c >> 12)) );
    putchar( 0x80 | (0x3F & (c >>  6)) );
    putchar( 0x80 | (0x3F & c) );
  }

}
```

Your turn!

- Japanese has two alphabets:
  - Hiragana
  - Katakana
- For Hiragana:
  - The first char is あ (say “*ah*”) (U+3042)
  - The last char is ん (do not say, but pronounce “n”) (U+3093)
- Write a program to print the chars of Hiragana

Unicode in C: Input

- Includes:
  - #include <wchar.h>, #include <stdlib.h>
- wchar_t *fgetws(wchar_t *ws, int n, FILE *stream);
- Conversion from UTF-8/UTF-16 to wchar_t:
  - mbstowcs(), mbsrtowcs()
    - Multi-byte to wide char
  - wcstombs(), wcrtomb()
    - Wide char to multi-byte
  - Uses LC_TYPE of locale()

mbrtowc() example

```
//  PURPOSE:  To translate the unicode sequence pointed to by 'charPtr'
//    into a wchar_t C-string.  Resulting wchar_t C-string placed in 'dest'
//    of length 'destLength', if the resulting string would fit.  Otherwise,
//    space allocated on heap.  In either case 'numWideChars' is set to the
//    number of wchar_t instances that result (not including the ending
//     '\0').  Returns resulting wchar_t C-string (either 'dest' or heap
//     address).
wchar_t*      utf8CPtrToWCPtr (const char*    charPtr,
                               size_t&        numWideChars,
                               wchar_t*       dest,
                               size_t         destLength
                              )
{
  //  I.  Application validity check:
  if  ( (charPtr == NULL)  ||  (dest == NULL) )
    throw L"NULL ptr to charPtrToWcharPtr()";

  //  II.  Translate:
  wchar_t*    space          = dest;
  size_t      spaceLen       = destLength - 1;
  wchar_t*    spaceEnd       = space + spaceLen;

  //  II.A.  Initialize 'mbstate':
  mbstate_t   mbstate;
  memset(&mbstate,'\0',sizeof(mbstate_t));

  //  II.B.  Each iteration translates to one 'wchar_t':
  wchar_t*    spaceRun;
  size_t      charPtrStep;
  size_t      length         = strlen(charPtr);
```

```
for  ( spaceRun = space;
       ;
       spaceRun++, charPtr += charPtrStep, length -= charPtrStep
     )
{
  //  II.B.1.  Translate to wchar_t and see how many more chars to
  //           advance in 'charPtr':
  wchar_t   resultWC;

  charPtrStep = mbrtowc(&resultWC,charPtr,length,&mbstate);

  //  II.B.2.  Stop loop if at end:
  if  (charPtrStep == 0)
    break;

  //  II.B.3.  Handle errors:
  if  (charPtrStep == (size_t)-1)
    throw L"Illegal unicode byte sequence";

  //  II.B.4.  Allocate more space if needed:
  if  (spaceRun == spaceEnd)
  {
    wchar_t*       tempWCPtr;
    size_t         numWCharsToCopy = spaceRun - space;

    spaceLen       += spaceLen;
    tempWCPtr       = (wchar_t*)malloc((spaceLen+1)*sizeof(wchar_t));

    if  (tempWCPtr == NULL)
      throw L"calloc() failure in charPtrToWcharPtr()";
```

mbrtowc() example, cont’d

```
memcpy(tempWCPtr,space,numWCharsToCopy*sizeof(
wchar_t));

    if  (space != dest)
     free(space);

    space          = tempWCPtr;
    spaceRun       = space + numWCharsToCopy;
    spaceEnd       = space + spaceLen;
   }

   //  II.B.5.  Store char:
   *spaceRun = resultWC;
  }

  //  II.C.  End wide char string:
  *spaceRun = L'\0';

  //  IV.  Finished:
  numWideChars       = spaceRun - space;
  return(space);
 }
```

Error Handling: Java

- Exceptions (From Vaibhav Aggarwal http://vaibhavblogs.org/2012/12/common-java-networking-exceptions/):
  - BindException
  - ClosedChannelException
  - ConnectException
  - InterruptedIOException
  - NoRouteToHostException
  - PortUnreachableException
  - ProtocolException
  - SocketException
  - SocketTimeoutException
  - SSLException
  - UnknownHostException
  - UnknownServiceException

Error handling: C

- include
  - errno.h, string.h
- Defines global var ‘errno’ that holds error code for last system call
  - Match with constants:
    - E2BIG Argument list too long
    - EACCES Permission denied
    - EADDRINUSE Address already in use
    - EADDRNOTAVAIL Address not available
    - EAFNOSUPPORT Address family not supported
    - EAGAIN Resource temporarily unavailable
    - There are many more!
  - char* strerror (int errnum)
    - Returns ptr to error message for ‘errnum’
  - void perror(const char* s)
    - Prints ‘s’, then the error message, to stderr

Your turn!

- What is wrong with the following:

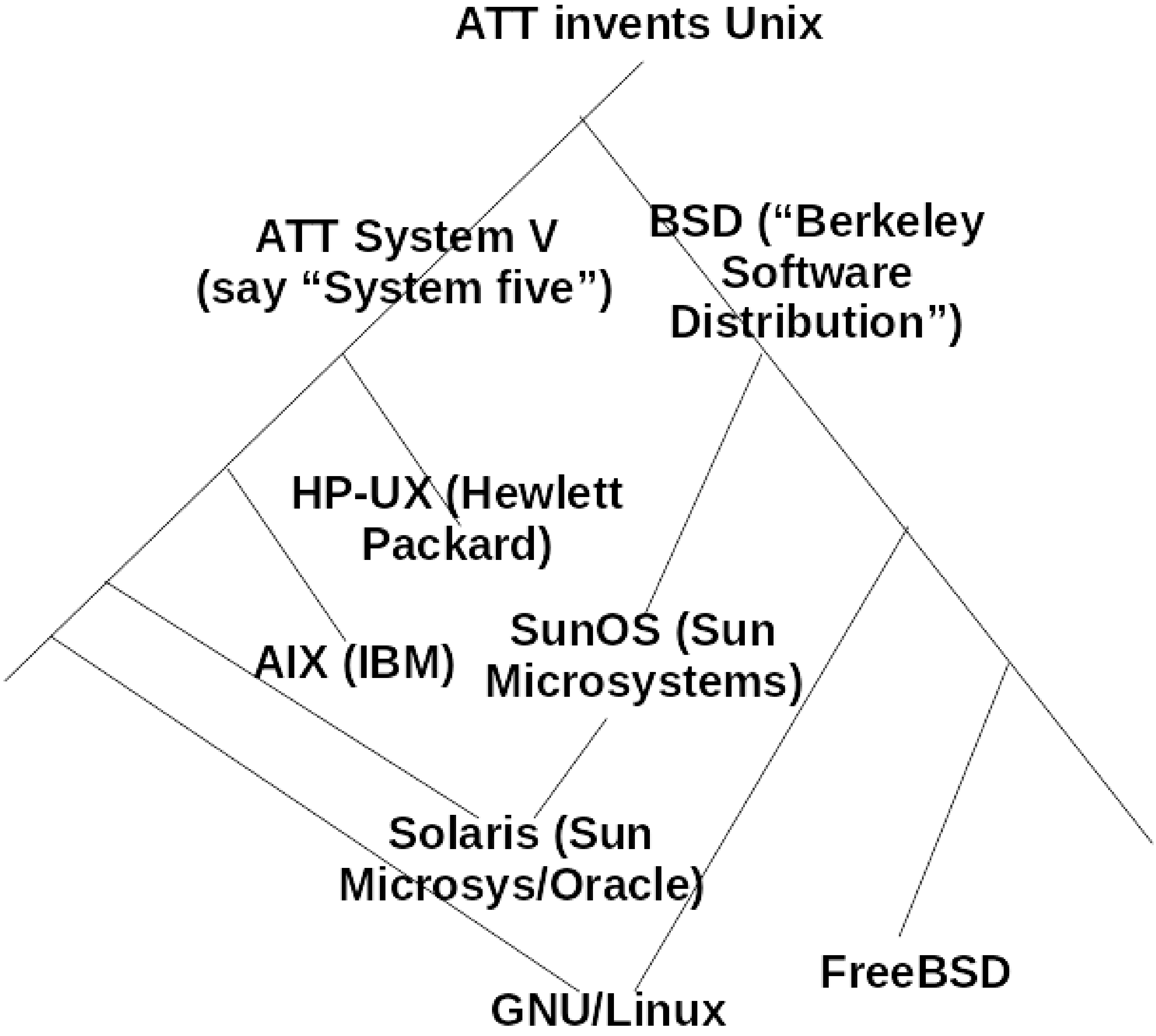
```
if (somecall() == -1) {
        printf("somecall() failed\n");
        if (errno == ...) { ... }
      }
```

- How would you fix it?

Did GOD ordain that networking look like this?

Not quite . . .
In a galaxy far, far away

A History of Derivations of Unix



ATT invents Unix

ATT System V
(say “System five”)

BSD (“Berkeley
Software
Distribution”)

HP-UX (Hewlett
Packard)

AIX (IBM)

SunOS (Sun
Microsystems)

Solaris (Sun
Microsys/Oracle)

GNU/Linux

FreeBSD

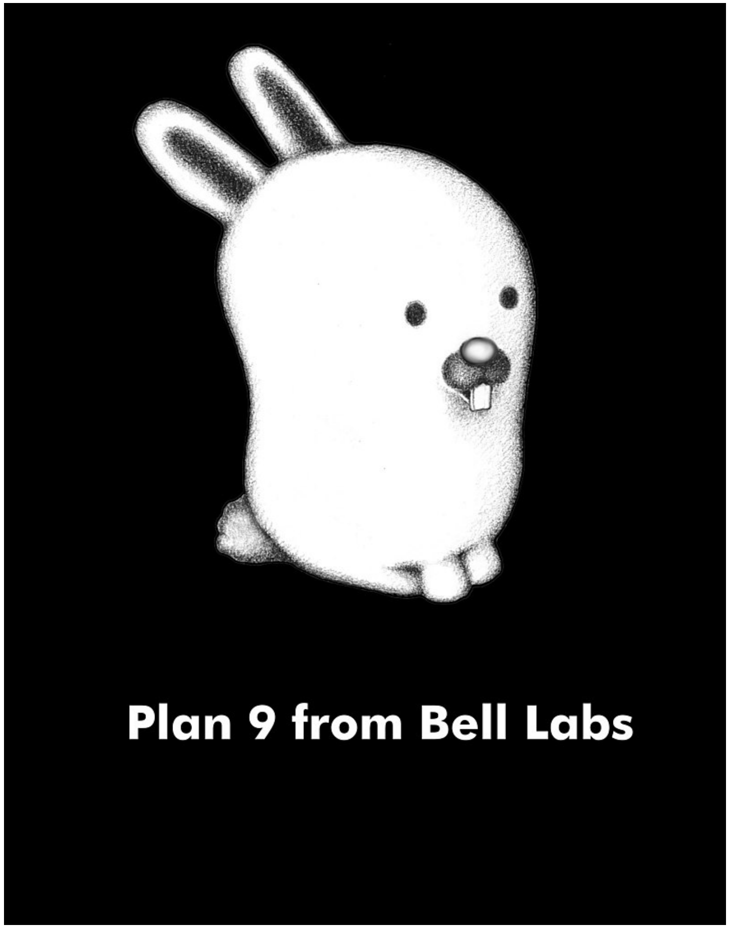
Alternatives to sockets

- Berkeley Software Distribution (BSD)
  - Sockets
- ATT System V (Pronounced “System five”)
  - “Transport Layer Interface”

BSD sockets	System V TLI
read, write, close	read, write, t_close
socket, socketpair	t_open
bind, listen	t_bind, t_unbind
getsockopt, setsockopt, select	t_optmgmt, t_look
strerror	t_strerror
connect	t_connect
accept	t_listen, t_accept, t_snddis

Plan 9/Inferno

- Distributed OS
- From Bell Labs
- Derived from Unix
- Big ideas of Plan 9/Inferno
  1. “Everything is a file” (even more so than Unix)
  2. Computable namespaces
     - Generalization of filepaths
  3. File service protocol
- Name from 1958 movie called “the worst ever made”, “*Plan 9 from Outer Space*”



“Everything is a file”

- Background, consider Linux’s /proc directory
  - presents information on processes
  - appears ***as if*** it were in files in directories
- But Plan 9/Inferno takes it further:
  - networking and other services used as if they were files

Echo server in Plan 9

```
// The Organization of Networks in Plan 9
// Dave Presotto, Phil Winterbottom
// presotto,philw@plan9.bell−labs.com
int echo_server(void)
{
  int dfd, lcfd;
  char adir[40], ldir[40];
  int n;
  char buf[256];

  int afd = announce("tcp!*!echo", adir);

  if(afd < 0)
    return −1;

  for(;;)
  {
    /* listen for a call */
    lcfd = listen(adir, ldir);
    if  (lcfd < 0)
      return −1;
    /* fork a process to echo */
    switch (fork())
    {
    case 0:
      /* accept the call and open the data file */
      dfd = accept(lcfd, ldir);
      if (dfd < 0)
        return −1;
      /* echo until EOF */
      while ((n = read(dfd, buf, sizeof(buf))) > 0)
        write(dfd, buf, n);
      exits(0);
    case −1:
      perror("forking");
    default:
      close(lcfd);
      break;
    }
  }
}
```

Plan 9/Inferno System calls

- Designed as a distributed form of Unix
  - exits()
    - Like C/Unix exit(), but returns a string
    - So error codes do not have to be standardized across systems
  - rfork(int flags)
    - Like C/Unix fork(), but more general
    - Flags include:
    - `RFMEM` If set, the kernel will force sharing of the entire address space, typically by sharing the hardware page table directly. (***QUESTION: What does this remind you of?***)
    - `RFNOWAIT` If set, the child process will be dissociated from the parent. Upon exit the child will not leave a status for the parent to collect. (***QUESTION: What does this remind you of?***)
    - `RFFDG` If set, the invoker's file descriptor table is copied; otherwise the two processes share a single table.

Plan 9/Inferno System calls, cont’d

- void* rendezvous (void* tag, void* value)
  - allows two processes to synchronize and exchange a value.
  - In conjunction with the shared memory system calls, it enables parallel programs to control their scheduling.

References:

- W. Richard Stevens "*TCP/IP Illustrated, Volume 1*" Addison-Wesley Professional Computing Series. 1994
- Douglas E. Comer "*Internetworking with TCP/IP: Principles, Protocols, and Architecture. 6th Ed*." Pearson Education, 2014.
- Vaibhav Aggarwal http://vaibhavblogs.org/2012/12/common-java-networking-exceptions/
- Dave Presotto, Phil Winterbottom "*The Organization of Networks in Plan 9*"
- Charles Forsyth "*The Name Game: Featuring Plan 9 and Inferno*" YouTube talk
- http://man.cat-v.org/plan_9/2/rendezvous (man page for rendezvous() system call)
- Joseph Phillips, Applied Philosophy of Science