*Advanced Programming Techniques*

# Networks

Christopher Moretti

# Network Precursors

"A telegraph. So now I have told my secret."

"A telegraph?" repeated Madame de Villefort.

"Yes, a telegraph. I had often seen one placed at the end of a road on a hillock, and in the light of the sun its black arms, bending in every direction, always reminded me of the claws of an immense beetle, and I assure you it was never without emotion that I gazed on it, for I could not help thinking how wonderful it was that these various signs should be made to cleave the air with such precision as to convey to the distance of three hundred leagues the ideas and wishes of a man sitting at a table at one end of the line to another man similarly placed at the opposite extremity, and all this effected by a simple act of volition on the part of the sender of the message."

*The Count of Monte Cristo*,
Chapter 60
(Alexandre Dumas, père, 1844)

F E D C B A

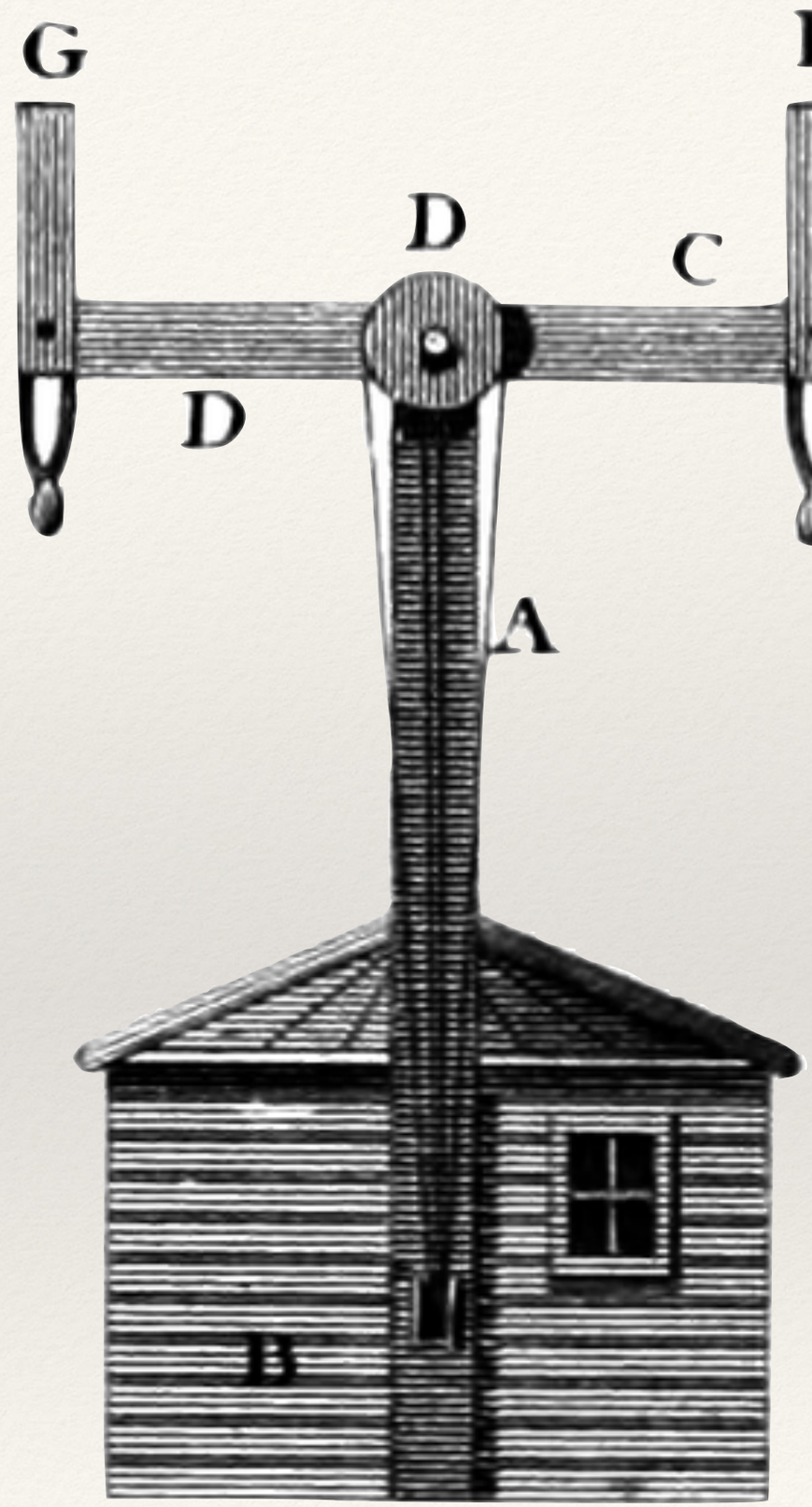
M L K I H G

S R Q P O N

Y X W V U T

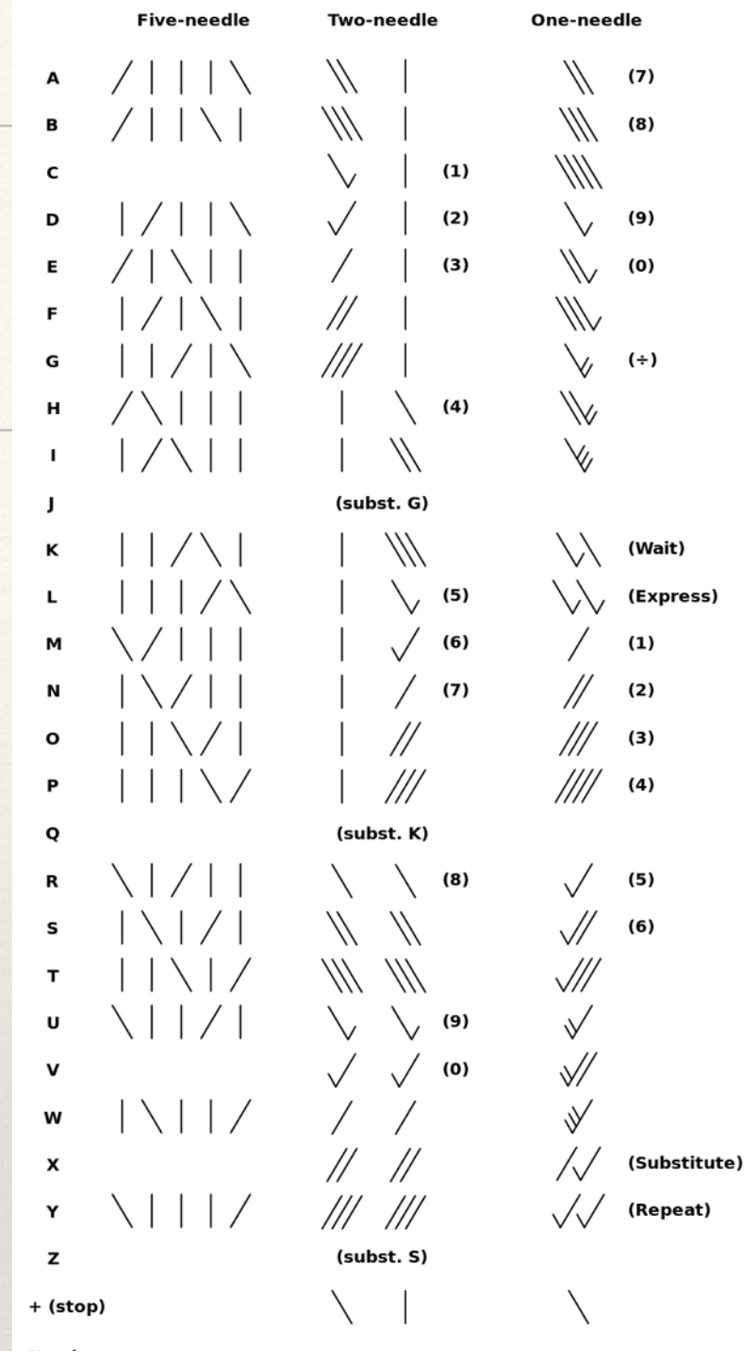
4 3 2 1 & Z

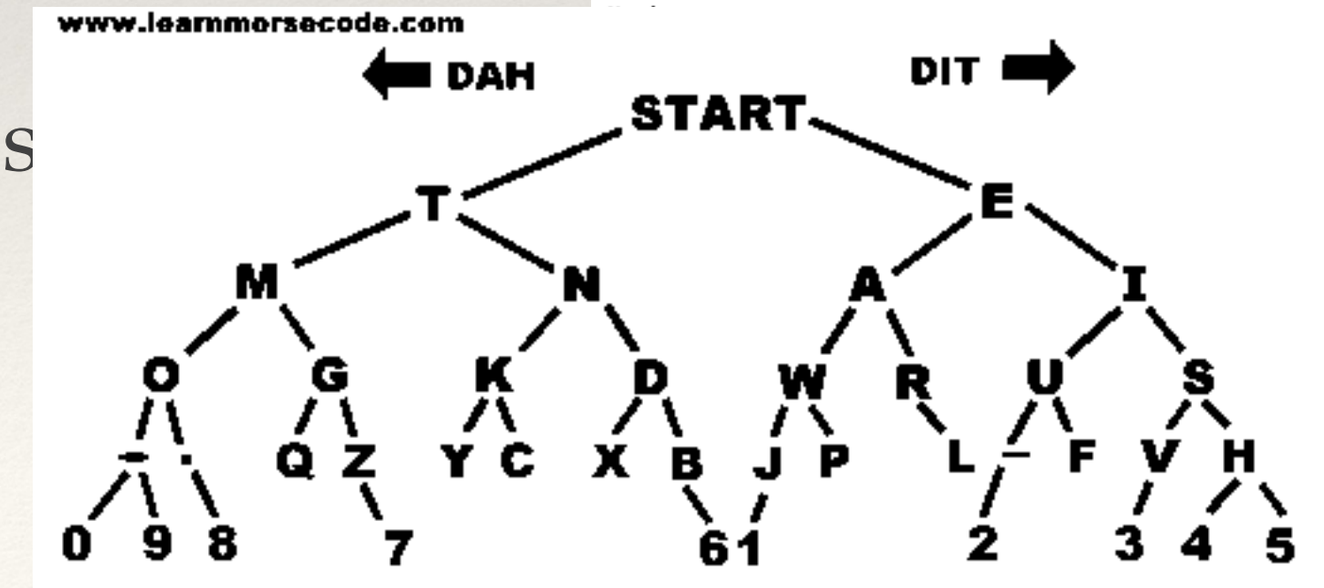
10 9 8 7 6 5



# Network Precursors

- Dyar
  - US mid-1820's prototype
- Weber and Gauß
  - Germany early 1830's
- Wheatstone and Cooke
  - Britain mid-1830's-1860's
- Morse, Henry, and Vail
  - US mid-1830's-1850's

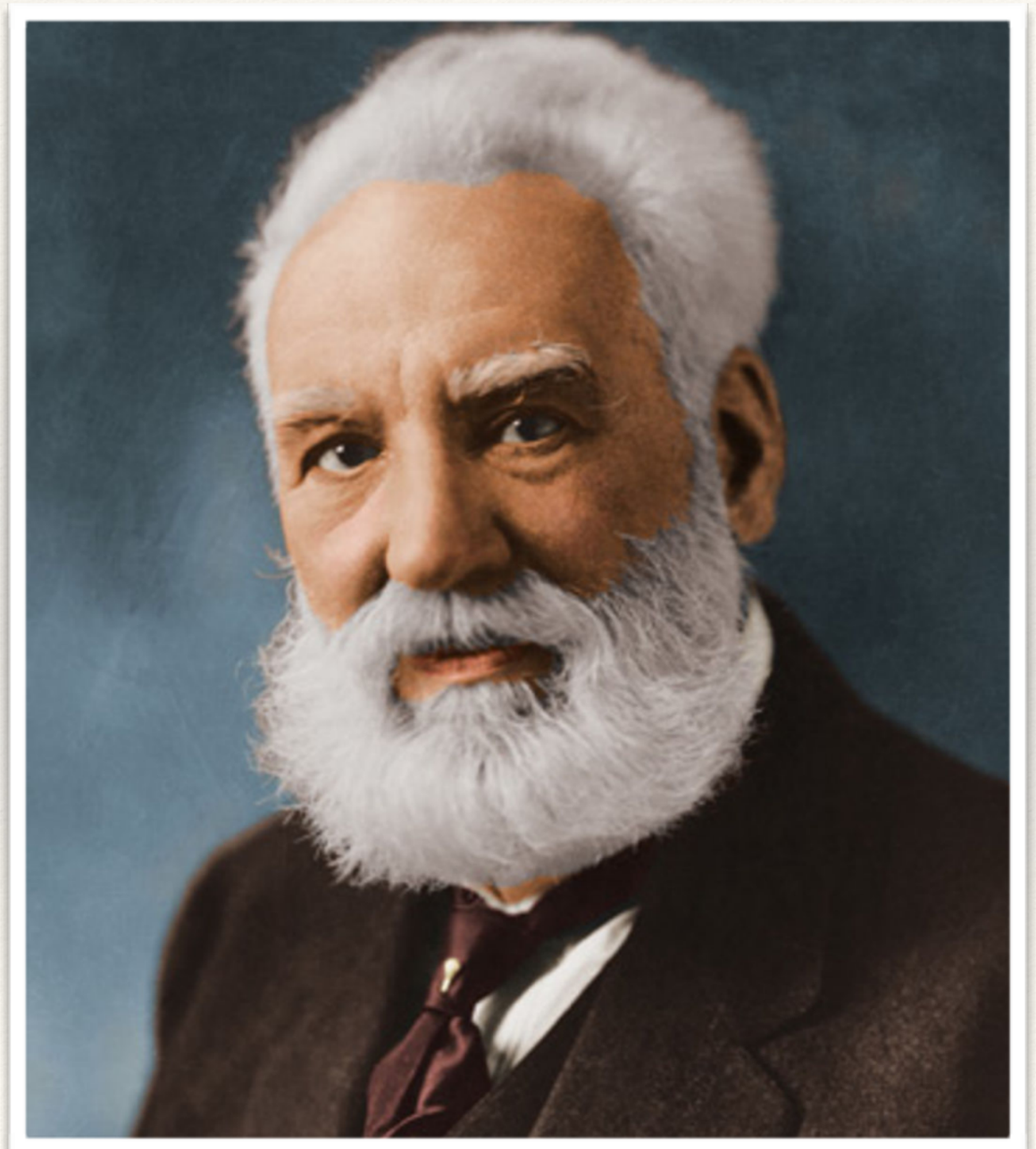


# Telegraph as Networking

- Wire medium
- Sending and receiving devices
  - Baudot teletypewriter
- Morse code (et al.) “protocol”
- Initially unicast
- “Fire and Forget”
- Initially single-transmission
  - Baudot multiplexing 1870’s

# Telephone as Networking



- Initially a wire medium, now fiber, VOIP, etc.
- Sending and receiving handsets (or modems …)
- Switched network
  - Initially literally: operators!
- Full-duplex communication
- Statefull connection

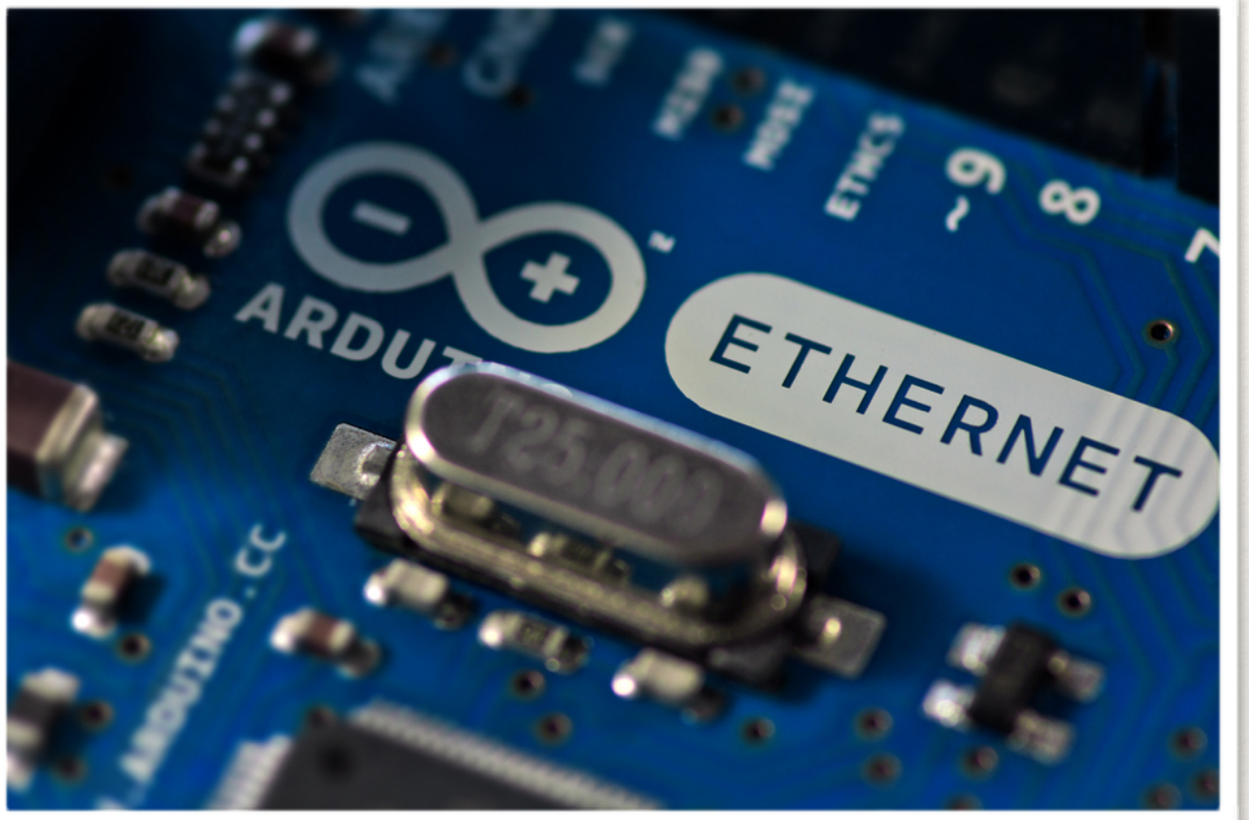

# Internet Architecture


- Connected independent heterogeneous networks
- Information travels through networks in packets
- Networks connected by gateways/routers
- Each router/gateway passes packet closer to destination
- No sense of central network control
  - ICANN: central authority for registering unique resources
    - IP addresses, domain names, country codes, …

ICANN

# Ethernet

- Metcalfe & Boggs '72 PARC 1973
- Local Area connection of devices
  - Hosts have unique 48-bit IDs
  - Data sent in "packets"
    - source address
    - destination address
    - payload
    - error checking
  - CSMA/CD

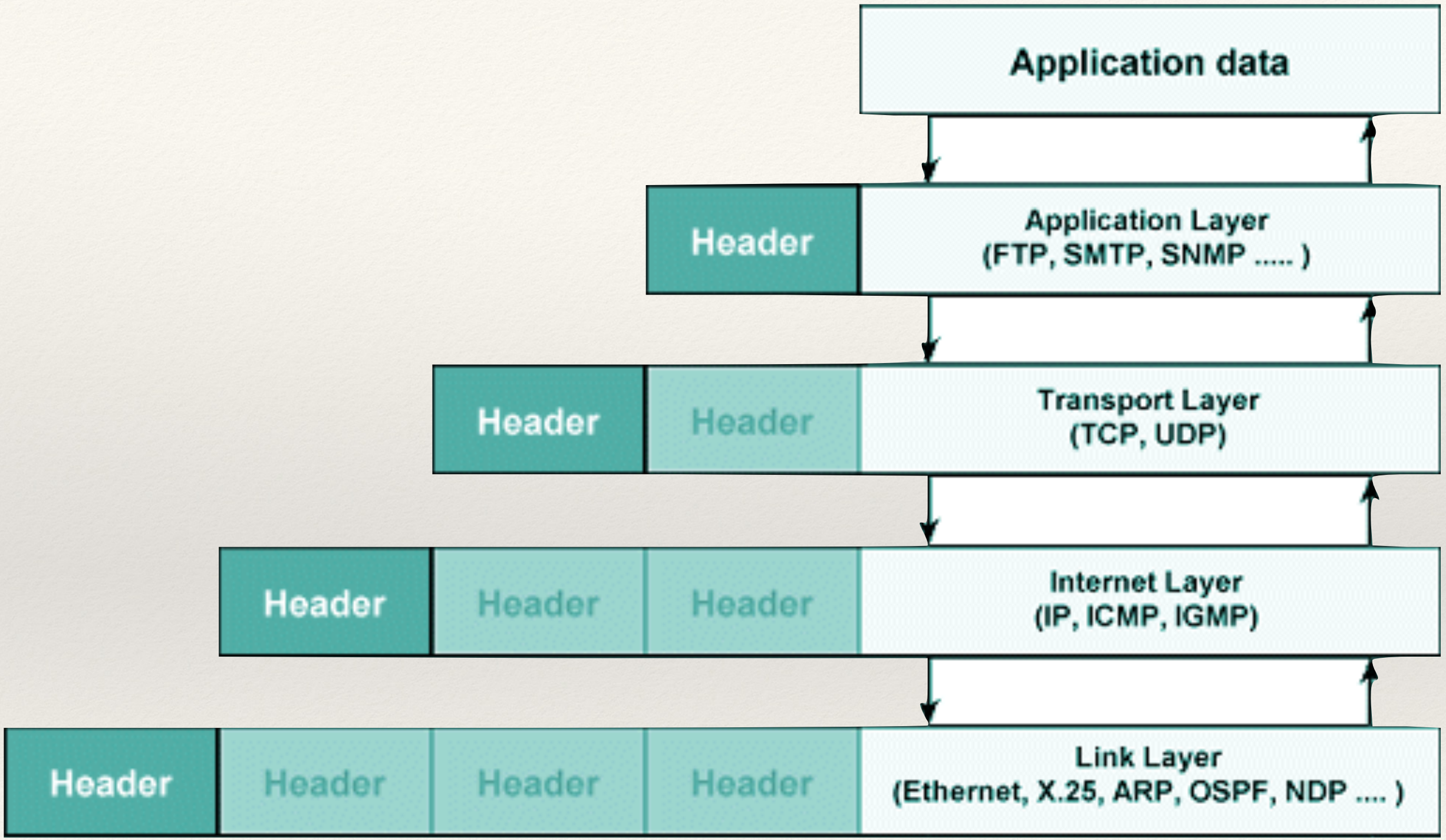



| Preamble | Dest Addr | Source Addr | Type | Info | FCS |
|---|---|---|---|---|---|
| 8 bytes | 6 bytes | 6 bytes | 2 bytes | 46<=N<=1500 bytes | 4 bytes |

Ethernet

# Network Protocols

- Precise rules governing communication
- Usually though of as a “stack”:
  - Physical (e.g. bits on a wire)
  - Data/Link (e.g. ethernet)
  - Internet (e.g. IP, ICMP)
  - Transport (e.g. TCP, UDP)
  - Application (e.g. HTTP, SSH)




http://www.technologyuk.net/the_internet/

# IP

- Unreliable connectionless packet delivery service
- Packets have:
  - Headers: source, destination, TTL, checksum
  - Payloads: up to 65 KB of data
    - long messages have to be split and reassembled

# Internet (IP) addresses

- each network and each connected computer has an IP address
- IP address: a unique 32-bit number in IPv4 (IPv6 is 128 bits)
  - 1st part is network id, assigned centrally in blocks
    (Internet Assigned Numbers Authority -> ISP -> you)

| net part | host on that net |
|---|---|

  - 2nd part is host id within that network
    assigned locally, often dynamically
- written in "dotted decimal" notation: each byte in decimal
  - e.g., 128.112.128.81 = www.princeton.edu

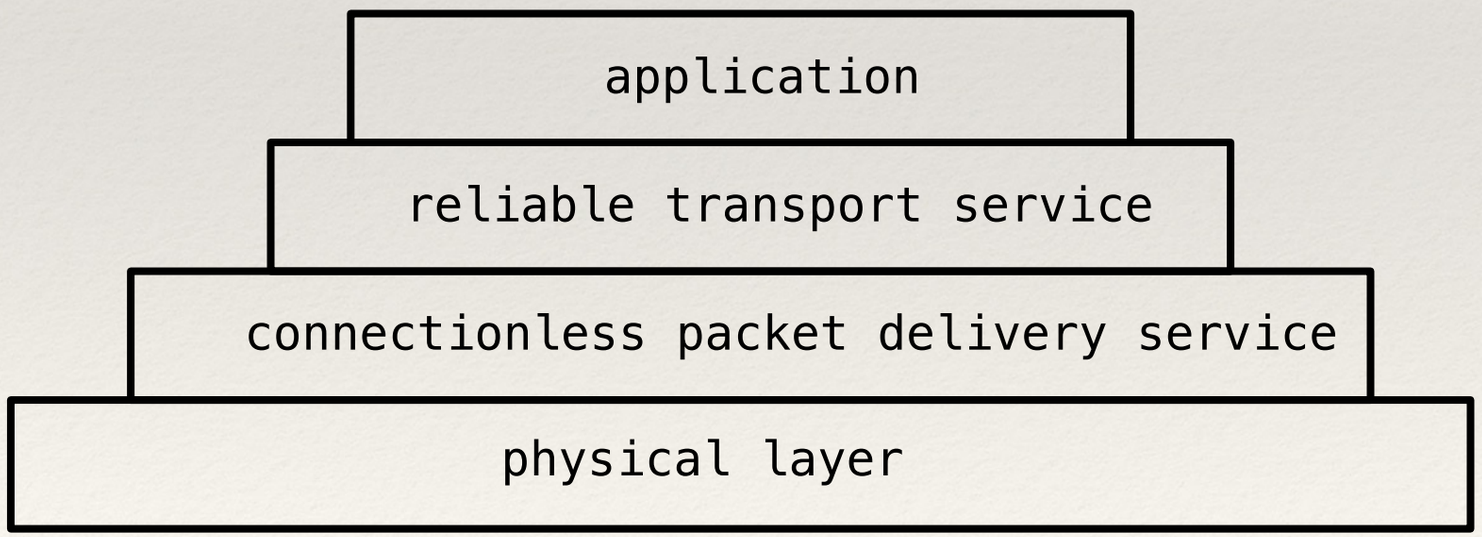
| 128 | 112 | 128 | 81 |
|---|---|---|---|
| 10000000 | 01110000 | 10000000 | 01010001 |

# TCP: Transmission Control Protocol

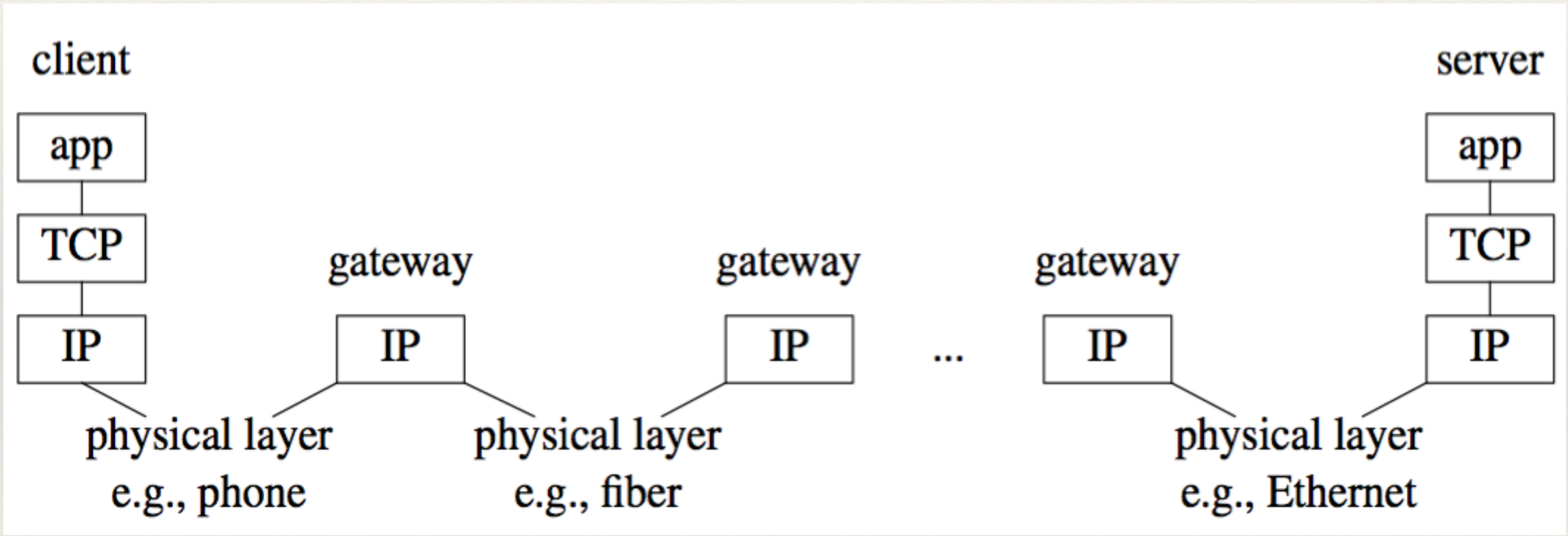
- reliable connection-oriented 2-way byte stream
  - no record boundaries
    if needed, create your own by agreement
- a message is broken into 1 or more packets
- each TCP packet has a header (20 bytes) + data
  - header includes checksum for error detection,
  - sequence number for preserving proper order, detecting missing or duplicates
- each TCP packet is wrapped in an IP packet
  - has to be positively acknowledged to ensure that it arrived safely
    otherwise, re-send it after a time interval
- a TCP connection is established to a specific host
  - and a specific "port" at that host
- each port provides a specific service
  - see /etc/services
  - FTP = 21, SSH = 22, SMTP = 25, HTTP = 80
- TCP is basis of most higher-level protocols

# Protocol Layering

- A single protocol can't do everything
- Build higher-level protocols out of simpler ones, each using only the services of the one directly below



# Protocol Stack Information Flow

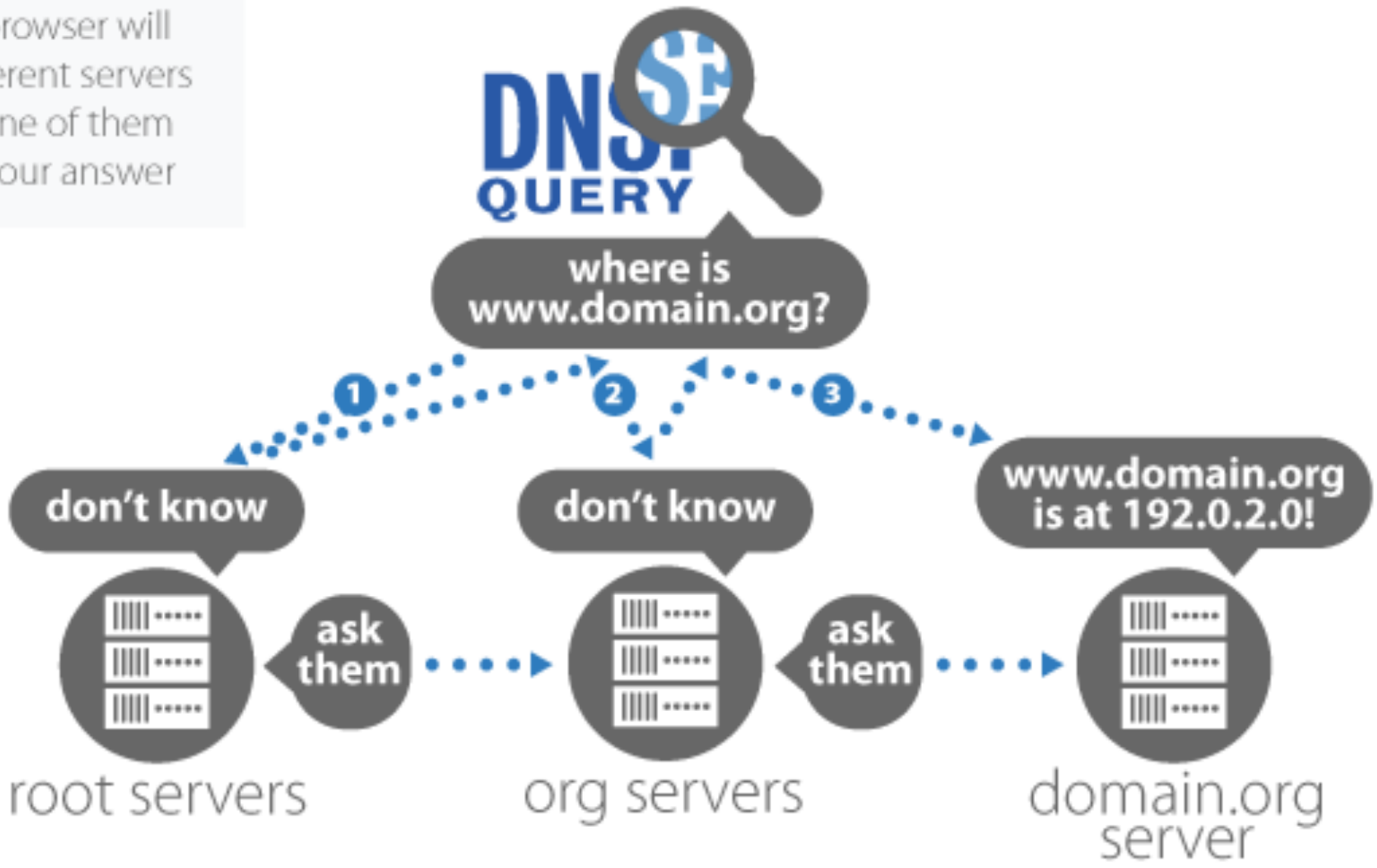



Greenland
Iceland
Suomi
Finland
Norge
Norway
Россия
Russia
Kazakhstan
Mongolia
中国
China
Japan
North
Atlantic
Ocean
Algeria
Libya
Egypt
Iraq
Saudi
Arabia
India
Mauritania
Mali
Niger
Chad
Sudan
Nigeria
Ethiopia
Kenya
DR
Congo
Tanzania
Angola
Namibia
Botswana
Madagascar
South
Africa
Indian
Ocean
Thailand
Indonesia
Papua New
Guinea
Australia
Venezuela
Colombia
Perú
Brasil
Brazil
Bolivia
Argentina
South
Atlantic
Ocean
Spain
Italy

http://bit.ly/1Sxpfls

# DNS

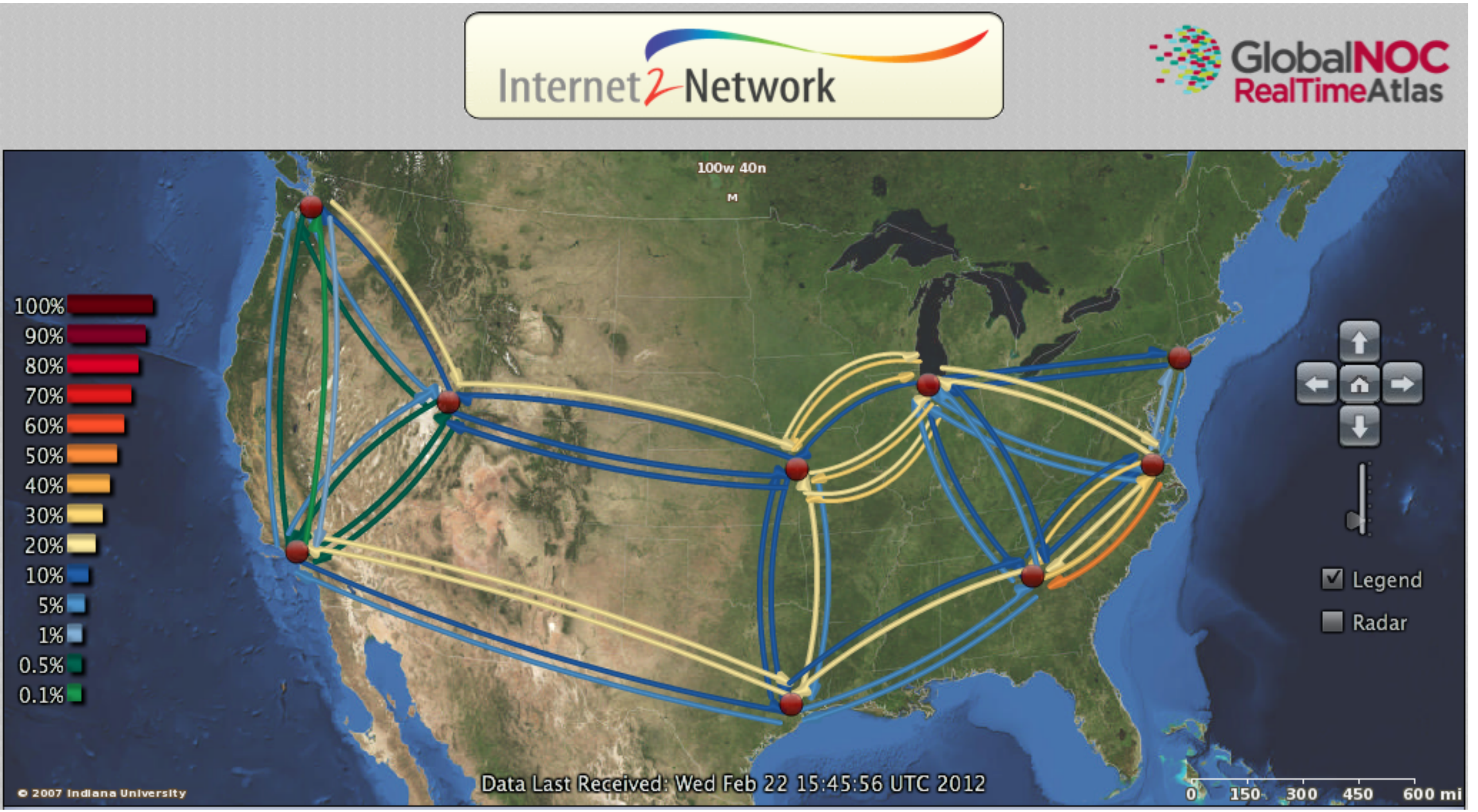
your browser will
ask different servers
until one of them
finds your answer




DNS query what happens when you enter a domain name into your browser?

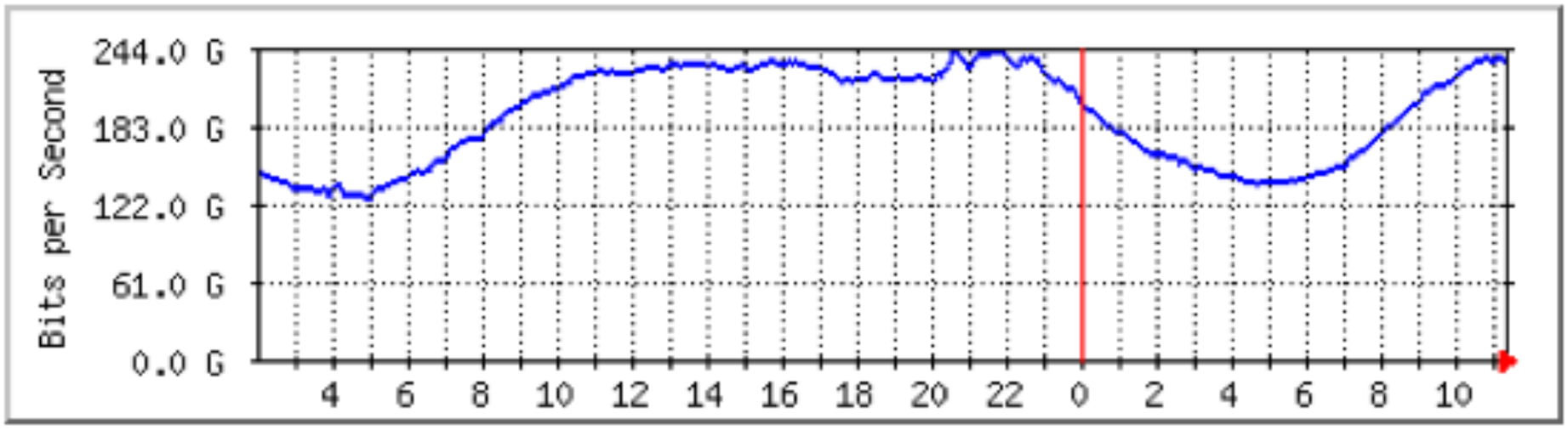
# Network Measuring & Monitoring

# http://atlas.grnoc.iu.edu/atlas.cgi?map_name=Internet2%20IP%20Layer



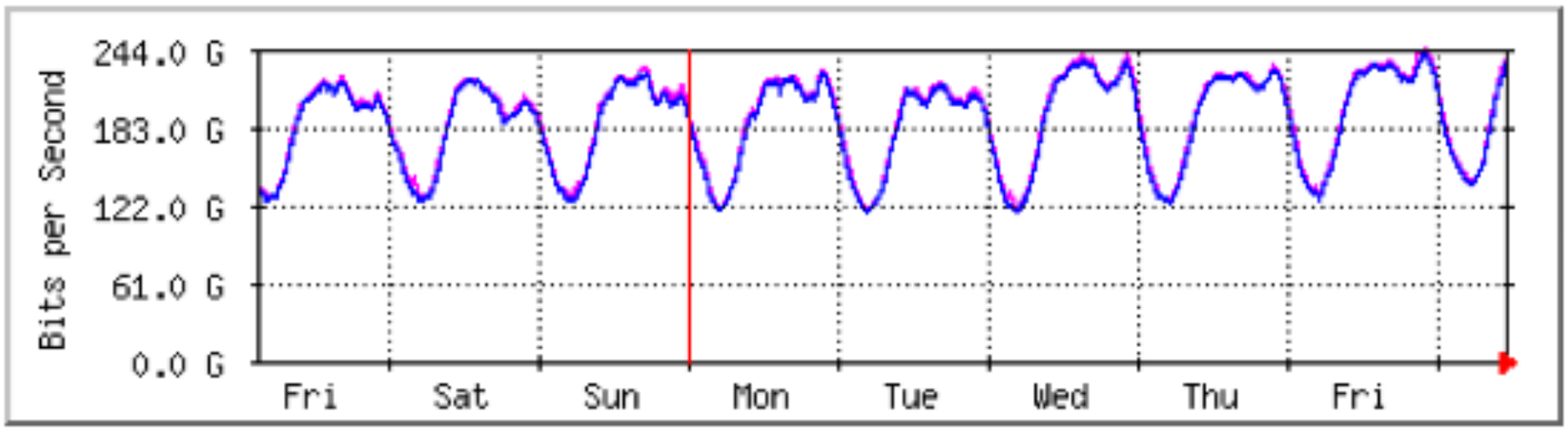
The statistics were last updated **Saturday, 8 March 2014 at 11:20**

## `Daily' Graph (5 Minute Average)




| | Max | Average | Current |
|---|---|---|---|
| **In** | 241.4 Gb/s (24.1%) | 190.9 Gb/s (19.1%) | 234.7 Gb/s (23.5%) |

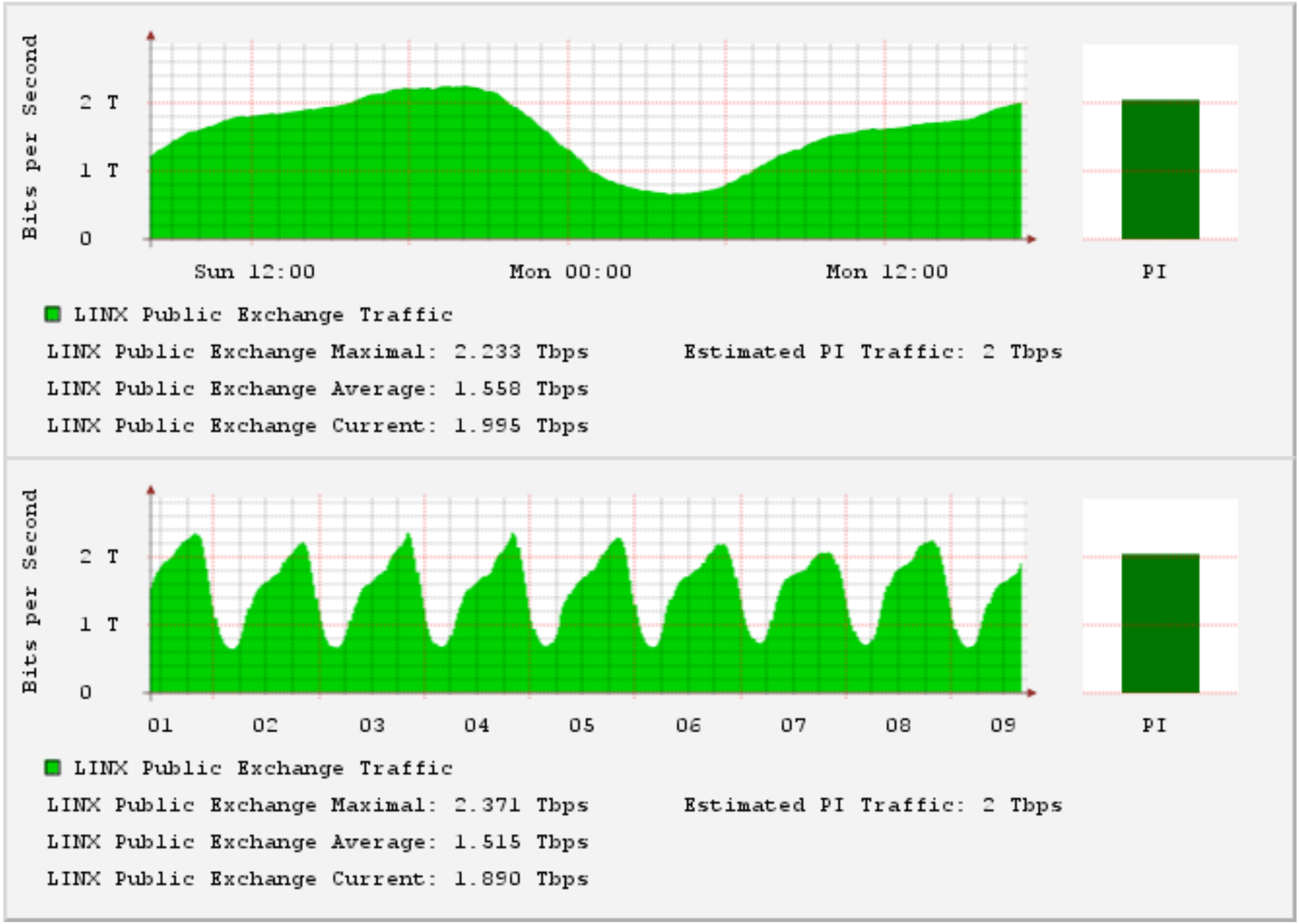
## `Weekly' Graph (30 Minute Average)




| | Max | Average | Current |
|---|---|---|---|
| **In** | 241.9 Gb/s (24.2%) | 185.3 Gb/s (18.5%) | 233.4 Gb/s (23.3%) |

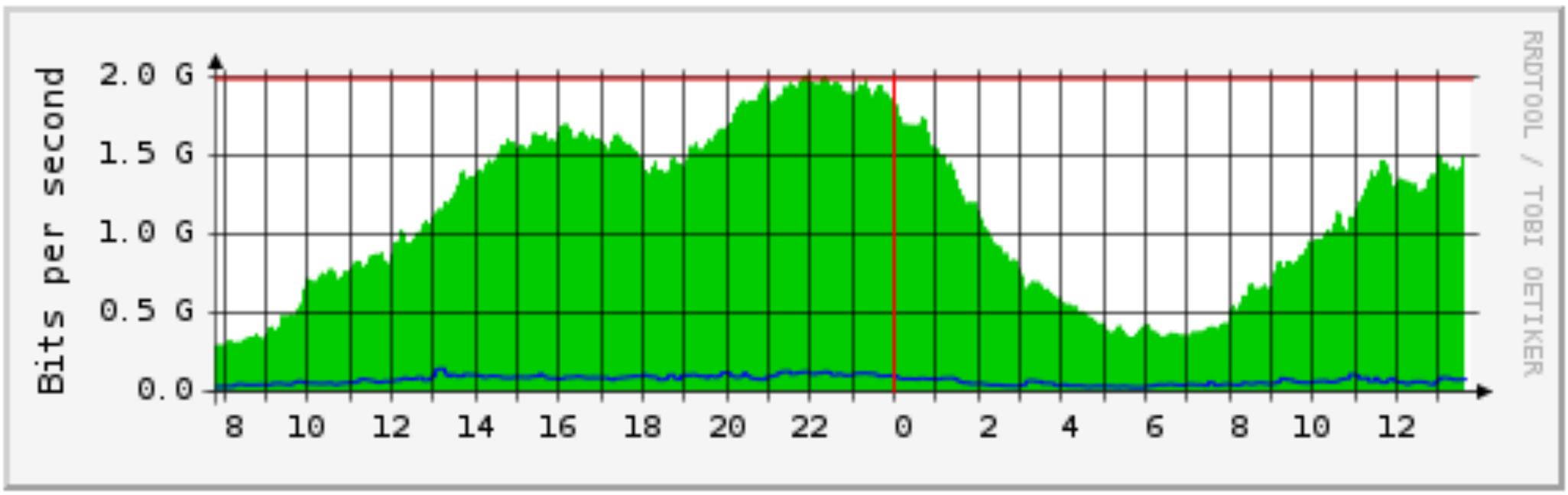
www.nyiix.net/mrtg/sum.html

# London Internet Exchange (linx.net)



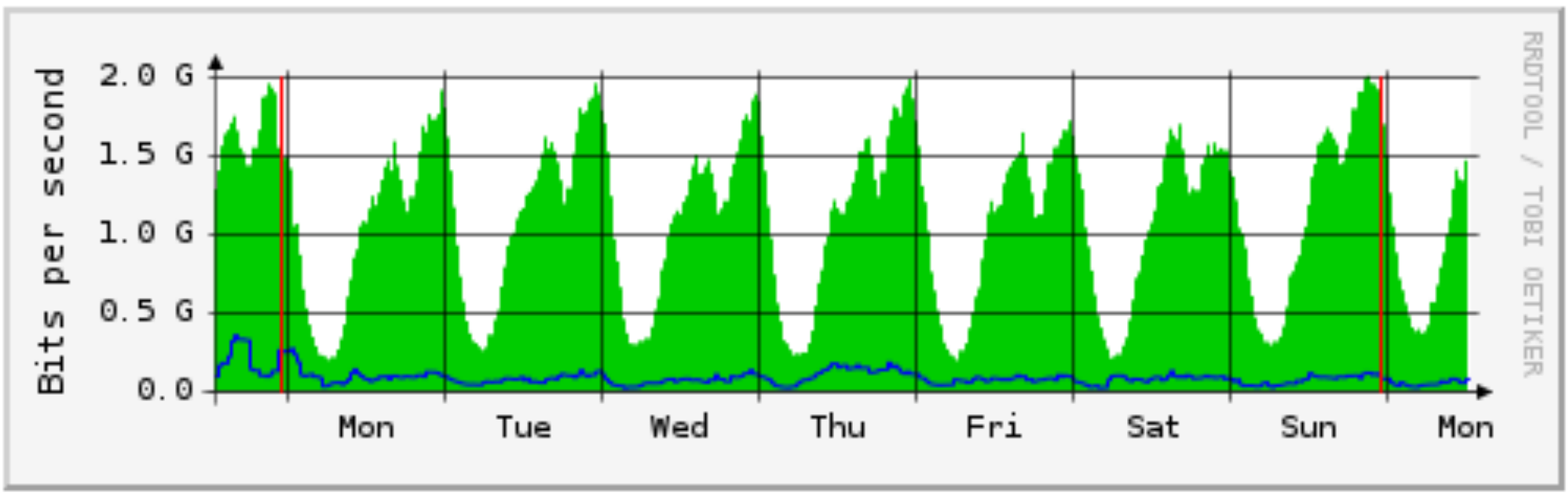
# What About Here?

## `Daily' Graph (5 Minute Average)



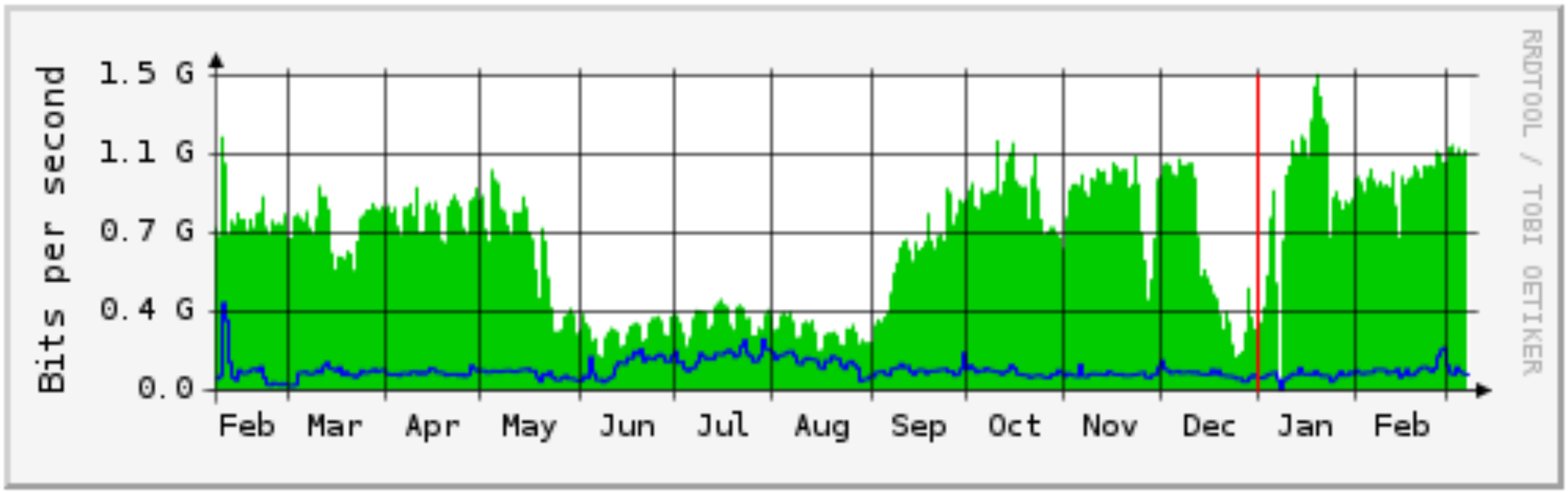

Max In: 2009.3 Mb/s (100.5%) Average In: 1136.7 Mb/s (56.8%) Current In: 1490.2 Mb/s (74.5%)
Max Out: 145.8 Mb/s (7.3%) Average Out: 72.9 Mb/s (3.6%) Current Out: 77.7 Mb/s (3.9%)

## `Weekly' Graph (30 Minute Average)




Max In: 1984.3 Mb/s (99.2%) Average In: 1109.5 Mb/s (55.5%) Current In: 1449.2 Mb/s (72.5%)
Max Out: 355.8 Mb/s (17.8%) Average Out: 90.7 Mb/s (4.5%) Current Out: 80.3 Mb/s (4.0%)

## `Yearly' Graph (1 Day Average)




Max In: 1451.1 Mb/s (72.6%) Average In: 677.3 Mb/s (33.9%) Current In: 1099.9 Mb/s (55.0%)
Max Out: 402.1 Mb/s (20.1%) Average Out: 93.7 Mb/s (4.7%) Current Out: 73.2 Mb/s (3.7%)

# A Logical View Of The Princeton University Fiber Optic Network

## FHS #01
1879/Marx
Admin Network

**End Buildings**

| | |
|---|---|
| Architecture | McCosh Infirm |
| Chapel | McCosh Hall |
| Dickinson | Marx |
| Eno | Moffett |
| Erist | Schultz |
| Frist(Palmer) | Woolworth |
| Guyot | |
| Jones | 1879 Hall |

## FHS#03
200 Elm
Admin Network
DormNet

**End Buildings**

| | |
|---|---|
| Baker Rink | 126 Alexander |
| Chilled Water | 130 University |
| Hibbon Magid | 180 Alexander |
| Lawrence Apts | 200 Elm Drive |
| McCarter | 228 Alexander |
| MacMillan | 262 Alexander |
| New South | 294/306 Alexander |
| Lot 7 Park | 330 Alexander |
| 106 Alexander | 350 Alexander |
| 120 Alexander | 701 Carnegie |

## FHS#04
Dillon Gym
Admin Network
DormNet

**End Buildings**

| | |
|---|---|
| Dillon Court | Lockhart |
| Dillon Gym | Pyne |
| Dod | Spelman |
| Edwards | Whitman |
| Forbes | 2 Dickinson |
| Foulke | 26CollegeParkRd |
| Grad.College | 1901 |
| Henry | 48 University |
| Laughlin | 71 University |
| Little | 99 Alexander |

## FHS#05
Frick
Admin Network

**End Buildings**

| | |
|---|---|
| Aaron Burr | 70 Washington |
| Corwin | Wallace |
| Fisher/Benheim | Friend Ctr |
| 20 Washington | 159/169 Nassau |
| Green Hall | 179 Nassau |
| Hoyt | 185 Nassau |
| Sherrerd Halll | 199 Nassau |
| 21 Prospect | 201 Nassau |
| Robertson | 221 Nassau |
| Princeton University Press | 5 Ivy Lane |

## 87 Prospect Fiber Central

*Networking Devices:*

Vgate1(InterNet)

Gigagate6(HPCRC)

ppn-87

Swch-ppn,Swch-ppn2,Swch-ppn3

Gigagate5 (NewSouth)

Core-87

EIS Consoles

CWDM Transport to HPCRC

Core Ancillary Switch

VOIPGate,VOIPGate2,VOIPGate3

border-87

Gigagate2

Core-ns (NewSouthDataCenter)

## FHS#02
87 Prospect G-07
Admin Network
DormNet

**End Buildings**

| | |
|---|---|
| Bowen Hall | 221 Nassau |
| Computer Science | PPPL |
| Engineering Quadrangle | |
| Mudd Library | 116 Prospect |
| | 22 Chambers |
| | 171 Broadmead |
| 693/755 Alexander | |
| Forrestal(Sayre) | |
| 701 Carnegie | Butler Tract |

## FHS#06
Jadwin Hall
Admin Network

**End Buildings**

| | |
|---|---|
| Architectural Lab | |
| Caldwell | ElemPartLab |
| Carl Icahn Lab | |
| Denuzio Pool | Lewis Library |
| Fine Hall | Lewis Thomas |
| Jadwin Gym | Frick Laboritory |
| Jadwin Hall | McDonnell Hall |
| Peyton | Princeton Stadium |

## FHS#07
Murray Dodge
Admin Network

**End Buildings**

| | |
|---|---|
| Alexander Hall | Nassau Hall |
| Clio | Prospect House |
| Dodge | Stanhope |
| East Pyne | West College |
| Firestone Library | Whig |
| Henry House | |
| MacLean | |
| McCormick | |
| Murray | |

## FHS#08
Campbell Hall
Admin Network
DormNet

**End Buildings**

| | |
|---|---|
| Blair | Madison Cafe |
| Campbell | |
| Hamilton | |
| Holder | |
| Joline | |
| Witherspoon | |

## FHS#09
Butler
Admin Network
DormNet

**End Buildings**

| | |
|---|---|
| Butler Hall Complex | |
| Clapp | |
| Dodge-Osborn | |
| Ellipse Dorm | |
| Scully | |
| 1927 | 1952 Fieldhouse (Stadium) |
| 1938 | Roberts Soccer Stadium |

## FHS#10
1939 Hall
Admin Network
DormNet

**End Buildings**

| | |
|---|---|
| Brown | 1903 |
| Cuyler | 1915 |
| Feinberg | 1937 |
| Gauss | 1939 |
| Patton | |
| Walker | |
| Wilcox | |
| Wu | |

## FHS#11
83 Prospect
Admin Network
DormNet

**End Buildings**

| | |
|---|---|
| Campus | Ivy |
| Cap&Gown | Quadrangle |
| Charter | Terrace |
| Cloister | Tiger |
| Colonial | Tower |
| Cottage | Bobst |
| BenHeim Ctr | 91 Prospect |
| 58Prospect | (Eating Clubs) |

**OIT Network Systems**

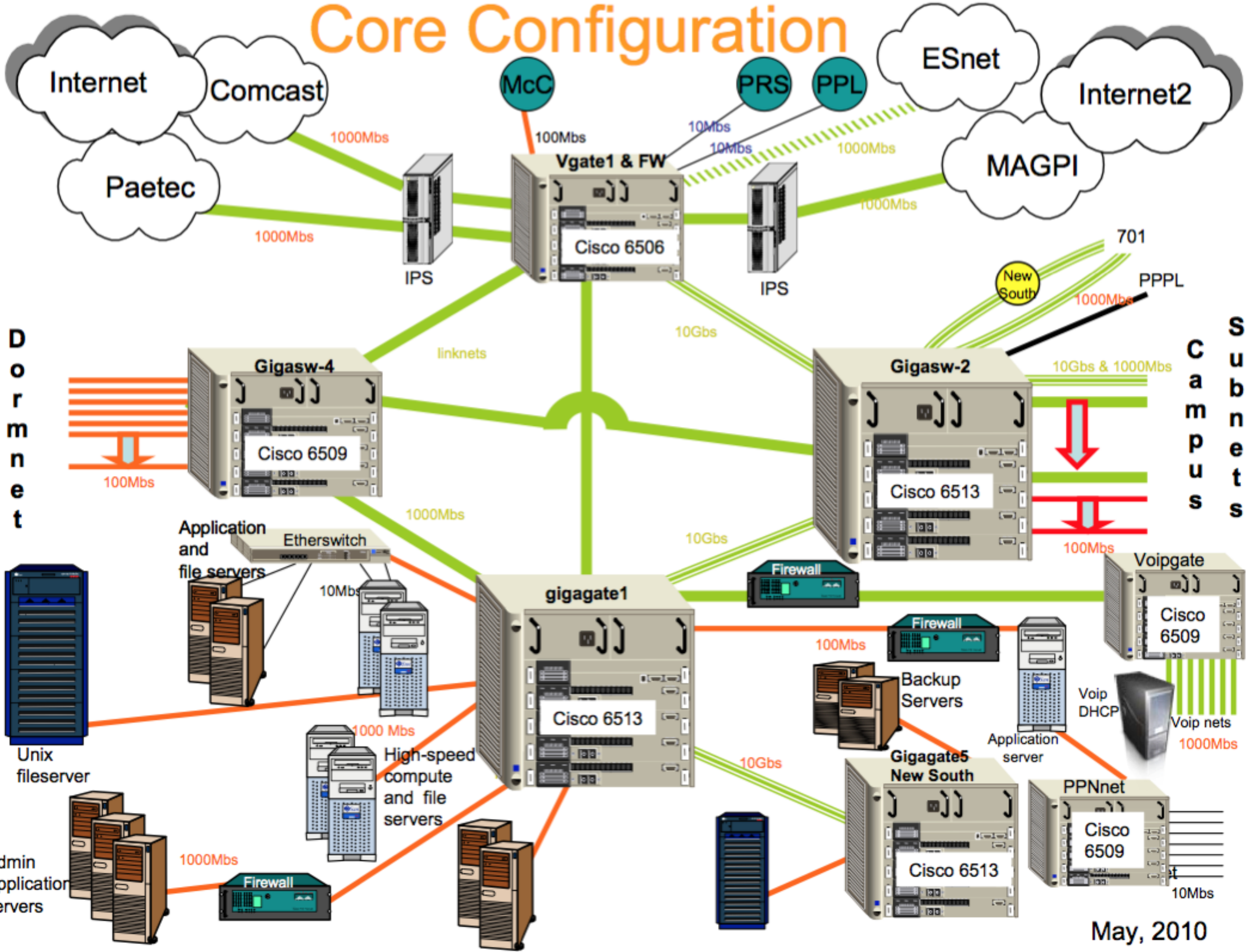
Administrative Network 128.112. *Office of Information Technology, Princeton University*

DormNet Network 140.180.

FHS = Fiber Hub Site

01/15/2014

# Core Configuration



# Network Programming

- C: processes, inetd, sockets:
  - `socket, connect, bind, accept, listen write, read, close`
- Java: `import java.net.*` for `Socket, ServerSocket, Datagram`
  - Python: `import socket, SocketServer`
  - General pattern:

```
server:
      fd = socket(protocol)
      bind(fd, port)
      listen(fd)
      fd2 = accept(fd, port)
      while (...)
          read(fd2, buf, len)
          write(fd2, buf, len)
      close(fd2)
```

```
client:
    fd = socket(protocol)
    connect(fd, server IP address, port)
    while (...)
       write(fd, buf, len)
       read(fd, buf, len)
    close(fd)
```

# C Server

```
struct protoent *ptrp;          /* protocol table entry */
struct sockaddr_in sad;         /* server adr */
struct sockaddr_in cad;         /* client adr */
memset((char *) &sad, 0, sizeof(sad));
sad.sin_family = AF_INET;   /* internet */
sad.sin_addr.s_addr = INADDR_ANY; /* local IP adr */
sad.sin_port = htons((u_short) port);
ptrp = getprotobyname("tcp");
fd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
bind(fd, (struct sockaddr *) &sad, sizeof(sad));
listen(fd, QLEN);

while (1) {
   fd2 = accept(sd, (struct sockaddr *) &cad, &alen));
   while (1) {
      read(fd2, buf, N);
      write(fd2, buf, N);
   }
   close(fd2);
}
```

# C Client

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

struct hostent *ptrh;  /* host table entry */
struct protoent *ptrp;         /* protocol table entry */
struct sockaddr_in sad;        /* server adr */
sad.sin_family = AF_INET;      /* internet */
sad.sin_port = htons((u_short) port);
ptrh = gethostbyname(host);  /* IP address of server /
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
ptrp = getprotobyname("tcp");
fd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
connect(sd, (struct sockaddr *) &sad, sizeof(sad));
while (...) {
   write(fd, buf, strlen(buf)); /* write to server */
   n = read(fd, buf, N);          /* read reply from server */
}
close(fd);
```

# Java Server (1)

```
public class srv {
 static String port = "33333";
 public srv(int port) {
  try {
     ServerSocket ss = new ServerSocket(Integer.parseInt(port));
     while (true) {
       Socket sock = ss.accept();
       System.err.println("server socket " + sock);
       new echo(sock);
     }
   } catch (IOException e) {
     e.printStackTrace();
   }
 }
 public static void main(String[] argv) {
   if (argv.length == 0)
     new srv(port);
   else
     new srv(argv[0]);
 }
}
```

# Java Server (2)

```
public class echo {
 echo(Socket sock) throws IOException {
   BufferedReader in = new BufferedReader(
     new InputStreamReader(sock.getInputStream())); // from socket
   BufferedWriter out = new BufferedWriter(
     new OutputStreamWriter(sock.getOutputStream())); // to socket
   String s;
   while ((s = in.readLine()) != null) {
      out.write(s);
      out.newLine();
      out.flush();
      if (s.equals("exit"))
         break;
   }
   sock.close();
 }
}
```

# Java Client (1)

```
import java.net.*;
import java.io.*;


public class cli {

 static String host = "localhost";
 static String port = "33333";

 public static void main(String[] argv) {
    if (argv.length > 0)
        host = argv[0];
    if (argv.length > 1)
        port = argv[1];
    new cli(host, port);
 }

 // actual meaty code comes next
```

# Java Client (2)

```
cli(String host, String port)
{
    try {
        BufferedReader stdin = new BufferedReader(
            new InputStreamReader(System.in));
        Socket sock = new Socket(host, Integer.parseInt(port));
        System.err.println("java client " + sock);
        BufferedReader sin = new BufferedReader(
            new InputStreamReader(sock.getInputStream()));
        BufferedWriter sout = new BufferedWriter(
            new OutputStreamWriter(sock.getOutputStream()));

        String s;
        while ((s = stdin.readLine()) != null) { // read cmd
            sout.write(s);  // write to socket
            sout.newLine();
            sout.flush();   // needed
            String r = sin.readLine(); // read reply
            System.out.println("cli.java got [" + r + "] from " + host);
            if (s.equals("exit"))
                break;
        }
        sock.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

# Serving Multiple Requests

- Real servers need to serve multiple clients at a time
- In C using *nix, typically use fork/exec
  - separate independent process for each “conversation”
- In Java, use Thread interface
  - all run in same process, address space
  - two primary methods: `start` `run`
  - Subclass (`extends Thread`) for multithreading

# Java Multithreaded Server

```
class echo1 extends Thread {
 echo1(Socket sock) {    this.sock = sock; start(); }
 public void run() {
    try {
        BufferedReader in = new BufferedReader(new
             InputStreamReader(sock.getInputStream()));
        BufferedWriter out = new BufferedWriter(new
          OutputStreamWriter(sock.getOutputStream()));
        String s;
        while ((s = in.readLine()) != null) {
            out.write(s);
            out.newLine();
            out.flush();
            System.err.println(sock.getInetAddress() + " " + s);
            if (s.equals("exit"))        // end this conversation
                break;
            if (s.equals("die!"))        // kill the server
              System.exit(0);
        }
        sock.close();
    } catch (IOException e) {
        System.err.println("server exception " + e);
    }
}
```

# Multi-threaded Python Server

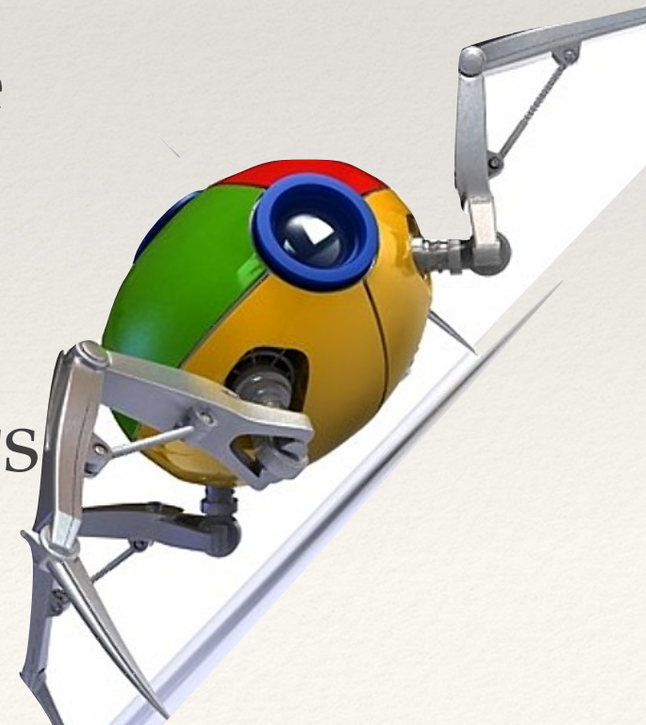
```
#!/usr/bin/python

import SocketServer
import socket
import string

class Srv(SocketServer.StreamRequestHandler):
   def handle(self):
      print "Python server called by %s" % (self.client_address,)
      while 1:
         line = self.rfile.readline()
         print "server got " + line.strip()
         self.wfile.write(line)
         if line.strip() == "exit":
            break

srv = SocketServer.ThreadingTCPServer(("",33333), Srv)
srv.serve_forever()
```

# Simple Node.js Server

```
var net = require('net');
var server = net.createServer(function(c){
                                    //'connection' listener
  console.log('server connected');
  c.on('end', function() {
    console.log('server disconnected');
  });
  c.pipe(c);
});
server.listen(33333, function() { //'listening' listener
  console.log('server bound');
});
```

# Another Client: Web Crawler

- Get a set of webpages (e.g. to see how big they are)
  - `crawler.py`
- Findings:
  - Latency and/or bandwidth are "slow" for some
  - Wasted time while waiting for a response
- Solution: parallelism!
  - asynchronous requests, threaded handlers
  - `crawler-threads.py`



# Sequential Version

```
import urllib2, time, sys

def main():
  start = time.time()
  for url in sys.argv[1:]:
    count("http://" + url)
  dt = time.time() - start
  print "\ntotal: %.2fs" % (dt)

def count(url):
  start = time.time()
  n = len(urllib2.urlopen(url).read())
  dt = time.time() - start
  print "%6d  %6.2fs   %s" % (n, dt, url)
main()
```

# Threaded version

```
import urllib2, time, sys, threading

global_lock = threading.Lock()

class Counter(threading.Thread):
  def __init__(self, url):
    super(Counter, self).__init__()
    self.url = url

  def count(self, url):
    start = time.time()
    n = len(urllib2.urlopen(url).read())
    dt = time.time() - start
    with global_lock:
      print "%6d  %6.2fs   %s" % \
(n, dt, url)

  def run(self):
    self.count(self.url)
```

```
def main():
  threads = []
  start = time.time()
  for url in sys.argv[1:]:
    # one thread each
    w = Counter("http://" + url)
    threads.append(w)
    w.start()

  for w in threads:
    w.join()
  dt = time.time() - start
  print "\ntotal: %.2fs" % (dt)

main()
```

# DDoS Experiment

- `ssh` into nobel.princeton.edu or arizona.princeton.edu
- copy a client with a command like these:
  - wget http://www.cs.princeton.edu/~cmoretti/cos333/net/cli.[py,java,c, or pl]
    - curl http://www.cs.princeton.edu/~cmoretti/cos333/net/cli.[py,java,c, or pl] > cli.[whichever]
- compile (if necessary)
- run client with argument yuma.princeton.edu
  - e.g. `python cli.py` yuma.princeton.edu
- type messages!