CSE 461: Introduction to Computer Communications Networks
Winter 2009

Module 1
Course Introduction

John Zahorjan
zahorjan@cs.washington.edu
534 Allen Center

Today's agenda

- Course Administration
  - course overview
    - course staff
    - general structure
    - the text
    - policies
- Introduction to Course Content

Course Admin

- Everything you need to know will be on the course web page:

  http://www.cs.washington.edu/461/

- Most everything (lecture schedule, reading, assignments, section materials, …) is linked off the schedule

- Quick overview now…

Course staff

- Who:
    - John Zahorjan
    - Alper Sarikaya
    - Emad Soroush
- What:
  - we're all just people
    - people learn in different ways – help is available in different ways
      - class reading materials
        - » it's easy to find lots more online
      - assignments/exams are a kind of help
      - person-to-person to person dialog is valuable (to both sides)
        - » questions in class/sections
        - » office hours / appointments
        - » email
    - questions in class/sections are more than welcome

Course goals

- Communication is more important than smarts for many applications
- Our primary goal is to understand how today’s networks are built
- This involves a mixture of:
  - science: Is there an algorithm that meets some goal?
  - engineering: How cost effective are various alternatives likely to be?
  - experience: what has worked, what hasn’t, and why?
  - measurement: are current networks working as intended? how are people using them?

Course goals (cont.)

- What is likely to be of lasting value to you?
  - Specific information: Many (most? all?) real applications involve networks.
  - General lesson: engineering a large, dynamic system
- The hope, as always, is to make all minutes you spend on the course worth your while
  - We won't be doing the substantial, multi-part project often associated with this course
- Instead:
  - reading text, answering questions from text, taking exams
  - reading additional important papers, writing short analyses of them
  - Still some implementation…

Last bit of course admin

– the text
  - Peterson & Davie, *Computer networks: a systems approach* (4th edition)

– other resources
  - many online; some of them are required reading

– Policies
  - email
  - late policy
  - grading
  - collaboration vs. cheating

Introduction to Networking

- Understanding networking involves thinking in a way you're almost certainly not accustomed to
  - networks are *distributed*:
    - concurrent: there is more than one program/computer involved
    - possibly strange failure semantics
      - sure, part of the "application" can crash and others stay up, but…
      - part can operate incorrectly, or
      - part can go down and come back up while app is running…
  - network architectures are deeply layered:
    - not just 2 levels (process/kernel)
      - it can get confusing what is happening at each level, and why
    - actual implementations favor function and efficiency over blind respect for layering
  - networks can have immense scale and heterogeneity
    - our most prominent network, the Internet, is so large and dynamic, and operated by so many distinct, entities that no one knows just exactly what it looks like, how it's being used, or how well it's working
  - networks *must* work correctly
    - is "five nine's" (99.999%) correctness enough?
    - Estimate: by 2015, one zettabyte/year traffic (one million million billion bytes)

Today

- Two examples, with the goal of helping us starting thinking “in a network way”
- First, a familiar example that shows us that what networks do is actually pretty simple
- Second, an unfamiliar example that shows that seemingly simple problems can be knotty in this arena

The Familiar Example

- Suppose you've amassed a 1TB (1024 GB) collection of "home movies" and you want to communicate a copy of them to a friend living in Walla Walla

- Q: What networking technology should you use?
  - A: A 1.5TB disk and the US Postal System

- This is not a joke…

TeraScale SneakerNet: Using Inexpensive Disks for Backup, Archiving, and Data Exchange.

Jim Gray, Wyman Chong, Tom Barclay, Alex Szalay, Jan Vandenberg

May 2002, Technical Report, MS-TR-02-54

Table 2: The raw price of bandwidth, the true price is more than twice this when staff, router, and support costs are included. Raw prices are higher in some parts of the world.

Context	Speed Mbps	Rent $/month	Raw $/Mbps	Raw $/TB sent	Time/TB days
home phone	0.04	40	1,000	3,086	6 years
home DSL	0.6	70	117	360	5 months
T1	1.5	1,200	800	2,469	2 months
T3	43	28,000	651	2,010	2 days
OC3	155	49,000	316	976	14 hours
100 Mpbs	100				1 day
Gbps	1000				2.2 hours
OC192	9600	1,920,000	200	617	14 minutes

Table 3: The relative cost of sneaker-net, using various media. The analysis assumes 6MBps tape, 10MBps CD/DVD and robots at each end to handle the media. Note that the price of media is less than the fixed robot cost.

	Media	Robot$	Media$	TB read + write time	ship time	TotalTime /TB	Mbps	Cost (10 TB)	$/TB shipped
CD	1500	2x800	240	60 hrs	24 hrs	6 days	28	$2,080	$208
DVD	200	2x8000	400	60 hrs	24 hrs	6 days	28	$20,000	$2,000
Tape	25	2x15,000	1000	92 hrs	24 hrs	5 days	18	$31,000	$3,100
DiskBrick	7	1,000	1,400	19 hrs	24 hrs	2 days	52	$2,600	$260

“Storage Brick”

Huh? Why not just send the disks?

"We began sending raw disks to one another, but that has the nasty problem that disks do not plug right into the network. So the recipient had to have exactly the right kind of disk reader (an ATA path that could read an NTFS file system with an SQL database). At a minimum, this excludes our Macintosh, Solaris, MVS, and Linux colleagues. Even within the select group that can read our favored disk format, we had many problems about master-slave settings on the ATA bus, about recognizing dynamic disks, and innumerable other details. So, sending magnetic disks around is problematic."

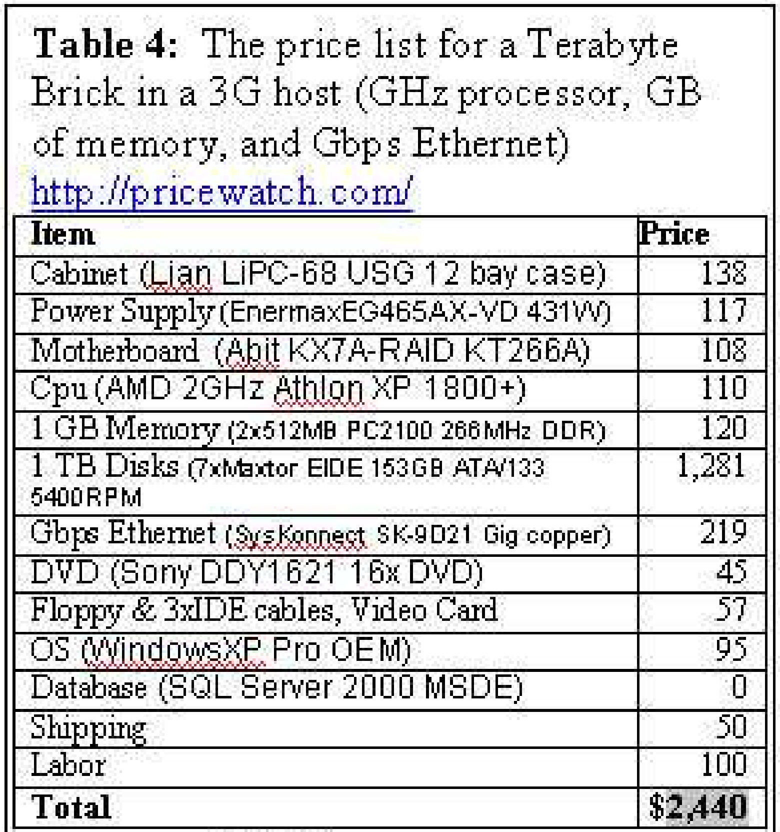
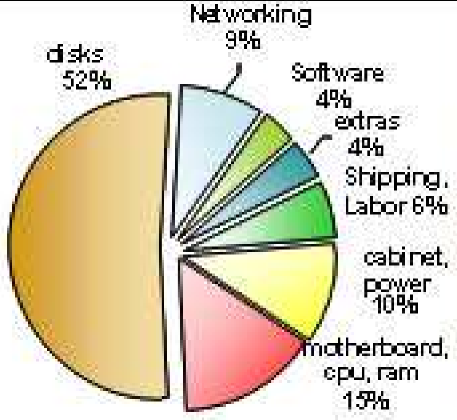


Table 4: The price list for a Terabyte Brick in a 3G host (GHz processor, GB of memory, and Gbps Ethernet)
http://pricewatch.com/

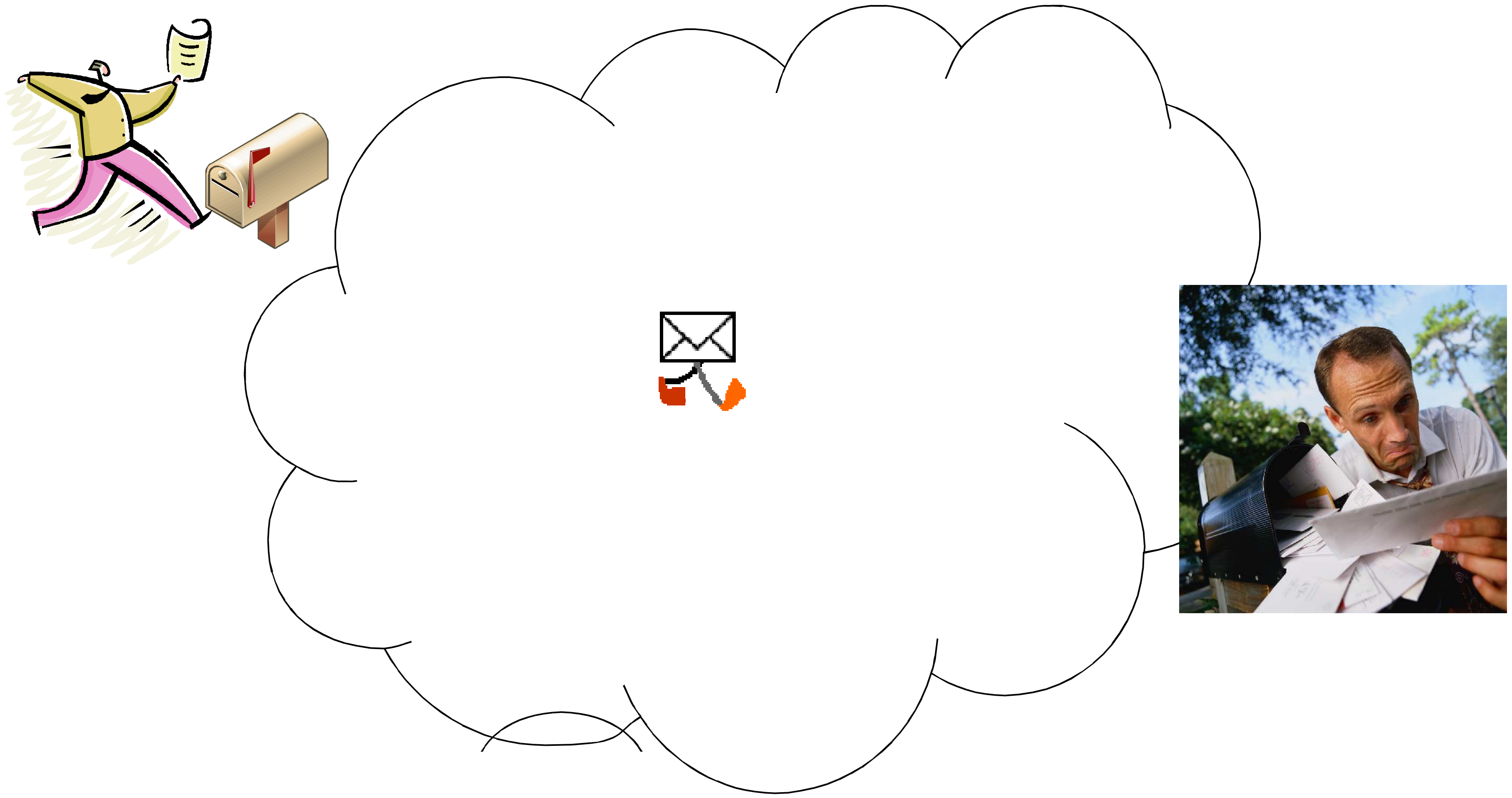
Item	Price
Cabinet (Lian LiPC-68 USG 12 bay case)	138
Power Supply (EnermaxEG465AX-VD 431W)	117
Motherboard (Abit KX7A-RAID KT266A)	108
Cpu (AMD 2GHz Athlon XP 1800+)	110
1 GB Memory (2x512MB PC2100 266MHz DDR)	120
1 TB Disks (7xMaxtor EIDE 153GB ATA/133 5400RPM	1,281
Gbps Ethernet (SysKonnect SK-9D21 Gig copper)	219
DVD (Sony DDY1621 16x DVD)	45
Floppy & 3xIDE cables, Video Card	57
OS (WindowsXP Pro OEM)	95
Database (SQL Server 2000 MSDE)	0
Shipping	50
Labor	100
Total	**$2,440**



Back to the Example: USPS as a Network

- There is a *client* at each end of the connection
- USPS itself is a networking *service*
- The service exports an *API* that tells clients how to use it:
  - To send:
    - wrap your data in an envelope
    - put an address on the envelope in a format that we (USPS) dictate
    - enter your packet into our system
  - To receive:
    - check your mailbox every so often
    - remove the envelope from whatever you find
    - voila, the data that was sent to you

USPS



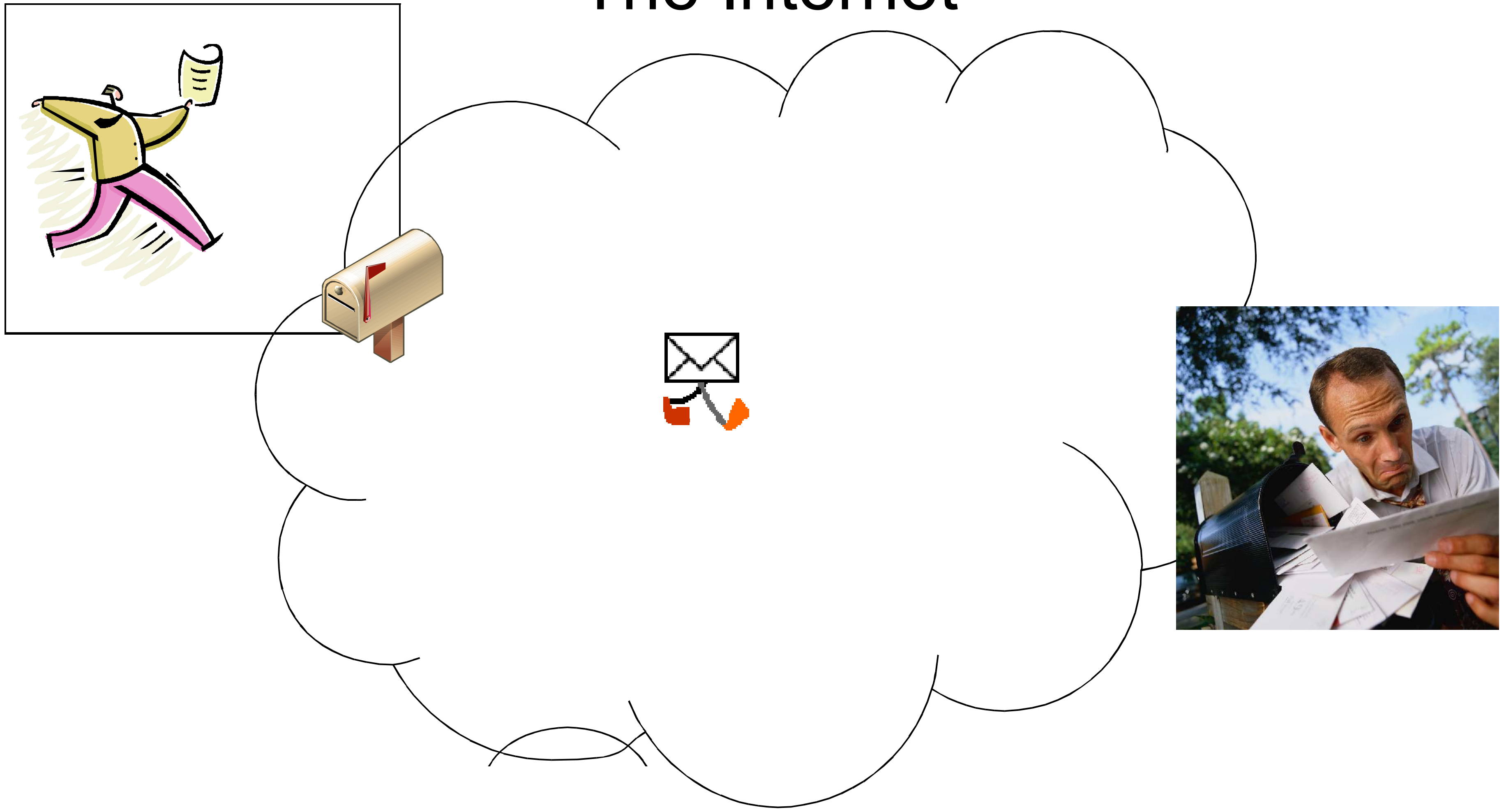
How does the USPS network work?

- *To be honest, I have almost no idea.*
- *Fortunately, for our purposes it doesn't matter – it's enough that our hypothetical explanation is plausible and could work*

1. All mail deposited into a mail box, no matter what its final destination, is first encapsulated in a new container (a mail truck) and routed to a local sorting facility.
2. The mail is unencapsulated (taken out of the truck). The destination address is examined, and it is re-encapsulated in a new container (e.g., mail headed to zip codes 993** is placed in a bag)
3. The bag of mail is un- and then re-encapsulated a number of times as it is transported over different physical media – a truck to Boeing Field, then an airplane to SFO, a truck to a sorting facility there, a truck back to SFO, a plane to Walla Walla, a truck to sorting facility there, a jeep/mailbag on the delivery route. At each stop, a decision is made about where to send it next.
4. As it nears its final destination, routing is based on its actual street address. It's eventually stuffed into the mailbox for your friend's address.
5. Your friend's roommates don't open it because part of the address includes his name. (Note that his name portion of the address was irrelevant up to this point.)

This is a lot like (but not exactly like) what happens in the Internet

The Internet



Parallels to the Internet

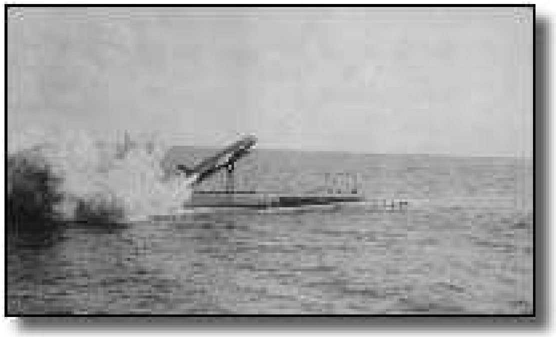
- Division of responsibilities/capabilities intersecting at an API
  - At the highest level, you don't much care *how* USPS delivers your mail, all you care about is the API. (We'll look at what the API is in just a moment.)
  - Similarly, USPS doesn't care or know what your data is.
  - Moreover, you don't really have much control over how USPS delivers your mail.
- Delivery involves a number of hops, with *routing* decisions made at each.
- A number of different physical media (trucks, planes, feet) are used, with the lowest capacity media typically found near the sender and the destination and the fastest media in the middle (of the route).
- Addresses are places, not people
  - "1600 Pennsylvania Avenue NW, Washington, DC 20500"
    not
    "Barack Obama"
  - Why? Why do we care?

Parallels to the Internet (cont.)

- There is a loose hierarchy involved in choosing a delivery route – more precise location information is needed as the mail gets closer to its destination.
- Correspondingly, the useful part of the address changes at various stages of delivery, for example:
  - None of it is relevant in the first step (truck to local sorting facility)
  - The first three digits of the zip are relevant through a lot of the middle stages
  - The full street address is important in the second last stage
  - "Joe Smith" is relevant (only) once it has been delivered to the destination mailbox
- There is a maximum allowed size – if you want to send more than that, put whatever it is in multiple boxes, each not too big
- If it's Christmas, expect more problems than usual
  - The system capacity is set to give good performance most of the time, but can suffer during periods of unusual load

Parallels to the Internet (cont.)

- USPS is able to make use of new delivery technologies as they arise, without altering its API


Missile Mail Launch, 1959

Missle Mail

Throughout its history, the Postal Service enthusiastically has explored faster, more efficient forms of mail transportation. Technologies now commonplace -- railroads, automobiles, and airplanes -- were embraced by the Post Office Department at their radical birth, when they were considered new-fangled, unworkable contraptions by many. One such technology, however, remains only a footnote in the history of mail delivery. On June 8, 1959, in a move a postal official heralded as "of historic significance to the peoples of the entire world," the Navy submarine U.S.S. *Barbero* fired a guided missile carrying 3,000 letters at the Naval Auxiliary Air Station in Mayport, Florida. "Before man reaches the moon," the official was quoted as saying, "mail will be delivered within hours from New York to California, to Britain, to India or Australia by guided missiles."

History proved differently, but this experiment with missile mail exemplifies the pioneering spirit of the Post Office Department when it came to developing faster, better ways of moving the mail.

copied without permission from http://www.usps.com/history/his2_75.htm

One Last Parallel

- 1963: USPS rolls out zip codes
  - 5 digits => 100,000 different zips, that's plenty
- 1983: USPS rolls out zip+4
  - 9 digits => 1,000,000,000 different zips; this time we mean it

- early 1970's: IP developed
  - 32-bit address fields => 4 billion distinct addresses, that's plenty
- circa 1995: IPv6 standarized
  - 128-bit address fields => about 10^{38} distinct addresses (about 10^{24} per square meter of the earth)
- (circa 2009: IPv6 still not widely adopted)

The USPS API

- We know how to send/receive mail. But what are the semantics of those operations (i.e., what properties are guaranteed)?
  - Reliability?
    - Is everything sent eventually delivered?
    - Is it received? (What's the difference?)
    - Is it received by the person named in the address?
  - Failure notification?
    - Does USPS let me know if it got there or not?
  - Integrity?
    - If it arrives, are the contents undamaged? *(Does it arrive only once?)*
  - Latency (delay)
    - Is there a guaranteed upper bound? How much does it vary from one letter to the next?
  - Ordering?
    - If I send a letter a day to a single destination address, do they arrive in the order I sent them?
  - Security?
    - Is anyone reading your mail in transit?
    - Can you be sure who sent the mail?
    - Can you avoid having so much junk mail sent to you that there's no room for the mail you want?

USPS as Engineering

- Want the service to be widely useful
  - Has to accommodate lots of different client “decisions”
    - Most anything as an envelope (not a USPS designed one)
    - Any old handwriting (not printed in some specific font)
    - Contents of envelope are irrelevant to its delivery
  - Has to be “scalable” – able to deliver mail from any and to any of an ever increasing number of addresses
  - Has to be cheap enough that people will use it
    - So, has to be cheap to provide
- To be cheap, the API provides almost no guarantees
  - It’ll probably get there, it’ll probably take around a week, and it’ll probably not be damaged
  - We can’t tell you who sent it or how it got there or if the check is actually in the mail
  - We can’t guarantee the carrier won’t read your postcard

USPS Engineering Decisions

- This may sound as though the design is “the cheapest thing to implement possible.” It’s not.
  - “Once a day, everyone take mail you find in your mailbox that’s not for you and put it in the next mailbox to the right…”
- Hopefully, it’s an appropriate tradeoff:
  - to be any cheaper, some property would have to get a lot worse (e.g., delivery times in months, not days)
  - to provide any stronger properties (e.g., we can verify who the sender was) it would have to get a lot more expensive
  - the properties it does provide are good enough in practice for lots of uses
- A desirable feature: if in some cases it’s worth it to the sender to have stronger properties, those properties can be built on top of the generic service at additional expense
  - If you want to know when the mail you sent arrives, phone the person you sent it to daily
  - if you don’t want the postman reading your postcards, put them in envelopes

Moral of First Example

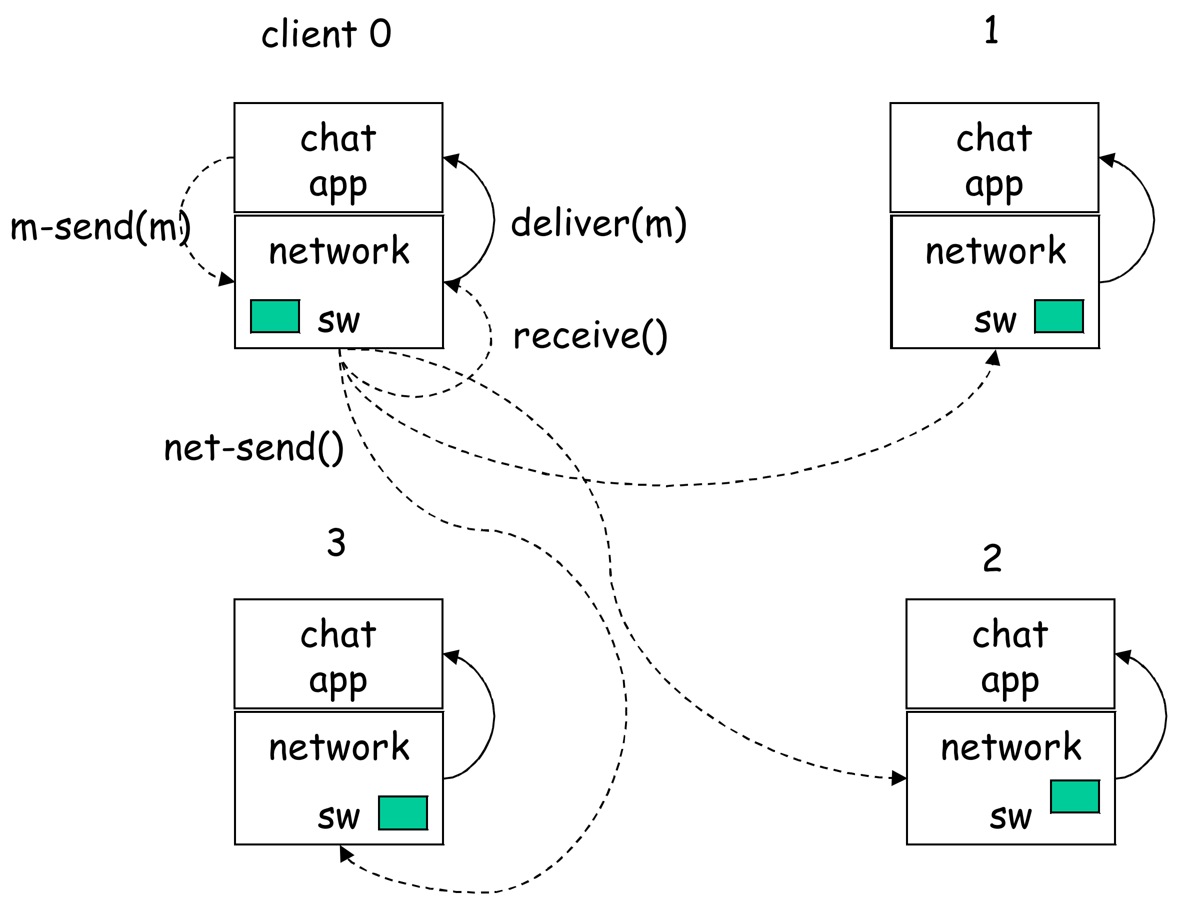
- What the Internet is doing isn't really all that complicated
- The fact that it isn't all that complicated is a stunning engineering accomplishment
  - Not that it could be built, but that they decided to build it this way
- If the pieces of the Internet architecture get a bit confusing, think about USPS – it's a pretty good analogy

A Second Example

- Suppose you want to build chat room software
- You want all messages typed by all participants to show up on everyone’s screen in the same order
- Division of responsibilities:
  - Your software: most everything, except for…
  - <u>Multicast</u>
    - a single send(m) call causes message m to be delivered to multiple destinations

```
forever {
   read local user’s input;
   multicast user input;
}
```
	```
	forever {
	   read network message;
	   print network message;
	}
	```

The Chat Room Application



Reliable, Totally Ordered Multicast

- multicast: a single send(m) call causes message m to be delivered to multiple destinations
- totally ordered: roughly, there is a unique sorted order to the messages (less roughly, the ordering is determined by an antisymmetric, transitive, and total relation)
- reliable: if a correctly operating client displays message m before displaying message m’, then any other correctly operating client that displays m’ will first display m
- (Also want liveness: all messages are eventually displayed)

First Try: The Straightforward Implementation

- When m-send(m) is invoked, immediately send it to each client (including yourself):

```
foreach client c {
    net-send(c,m);
}
```

- When a message m is received from the network, hand it up to the app (to display):

```
deliver(m);
```

- What can (will) go wrong?

Second Try

- Assume net-send() is reliable, and that no client crashes or has bugs
- On m-send(m) :

```
t = localClockTime();
foreach client c {
    net-send(c,m,t);
}
```

- When a message (m,t) is received from client s:

```
put (m,t) in a sorted queue;
while (there is a message in the queue) {
  deliver(the message with the lowest timestamp);
  remove the delivered message from the queue;
}
```

- Does it work?

Third Try

- Assume net-send() is reliable, and that no client crashes or has bugs
- On m-send(m) :

```
t = localClockTime();
foreach client c {
    net-send(c,m,t);
}
```

- When a message (m,t) is received from client s:

```
put (m,t) in a sorted queue;
while (there is a message in the queue from each client) {
  deliver(the message with the lowest timestamp);
  remove the delivered message from the queue;
}
```

- Does it work?
  - What assumption about what net-send() guarantees is it making?
  - What other assumption is it making?
  - Why isn't it an acceptable solution in practice?

Last Try: Lamport clocks

- First, we need to define *Lamport clocks…*
- Each client has its own Lamport clock, with monotonically increasing timestamp t_c
- Every event is tagged with a “timestamp”
  - For us, events are m-send() invocations and message receptions
- When m-send(m) on c is invoked:
  - $t_c = t_c + 1$
- When a message with timestamp t_s is received:
  - $t_c = \max(t_c, t_s) + 1$

Last Try: Implementation

- On m-send(m) at client s:

```
$t_s$ = $t_s$ + 1;
foreach client c {
    net-send(c,m,$t_s$);
}
```

- When (m,t_s) is received at c:

```
$t_c$ = max($t_c$, $t_s$) + 1;

// broadcast an acknowledgement of m to everyone else
if (the message received is not itself an ACK) {
    foreach client q {
        net-send(q,ACK(m),$t_c$);
    }
}

put (m,$t_s$) in a sorted queue;
while (the first non-ACK message in the queue has been ACK'ed by all clients) {
   deliver(that first non-ACK message);
   remove that message and its ACKs from the queue;
}
```

For next time

- Assignment 1 is out (linked from syllabus page)
  - Try out some useful, standard tools
  - Read a key paper, think about it, and write two paragraphs
  - Due a week from now
- An implementation homework will be out on Wednesday (or thereabouts)
- Sections Thursday intended to help with the programming assignment
- For next class, please have read Chapter 1
- We'll return to a (more standard) overview of networks