CS 126 Lecture S5: Networking

Outline

- **Introductions**
- Connectivity
- Naming and addressing
- Abstractions and layering
- Example: socket programming
- Conclusions

Review: Technology Advances

	1981	1999	Factor
MIPS	1	1000	1,000
$/MIPS	$100K	$5	20,000
DRAM Capacity	128KB	256MB	2,000
Disk Capacity	10MB	50GB	5,000
Network B/W	9600b/s	155Mb/s	15,000
Address Bits	16	64	4
Users/Machine	10s	<= 1	< 0.1

- Expensive machines, cheap humans
- Cheap machines, expensive humans
- (Almost) free machines, really expensive humans, and communities

The Network *is* the Computer

- Relentless decentalization trend
  - Over the years, machines have and will continue to become smaller, cheaper, and more numerous: mainframes **->** mini computers **->** personal computers **->** palm computers **->** ubiquitous (embedded, not necessarily general-purpose) computers
  - Computers are intrinsically social animals: as we have more of them, their need to “talk” to each other becomes more imperative
- Why do computers need to talk to each other? The focus has shifted over the years
  - Efficient use of machine resources
  - Sharing of data
  - Use parallelism to solve bigger problems faster
  - Ultimate killer app: enabling humans to communicate with each other

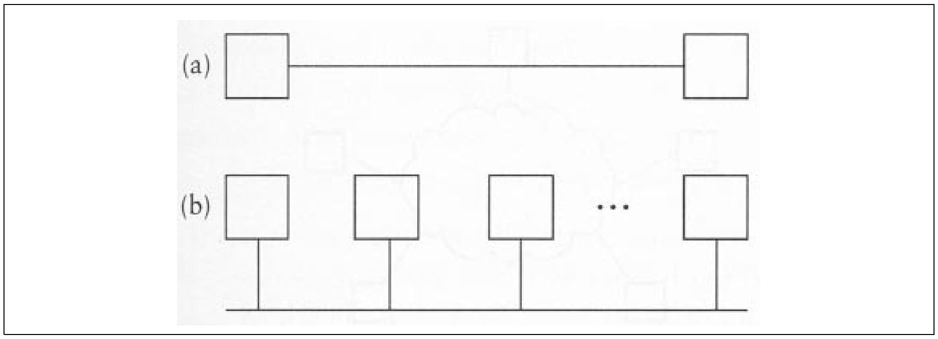
The Network *is* the Computer (cont.)

- Networks are everywhere and they are converging
  - SAN, LAN, MAN, WAN
  - All converging towards a similar switched technology
- New chapter of every aspect of computer science
  - Re-examine virtually all the issues that we looked at in this class in the context of distributed systems or parallel systems
- This is only the beginning!

Outline

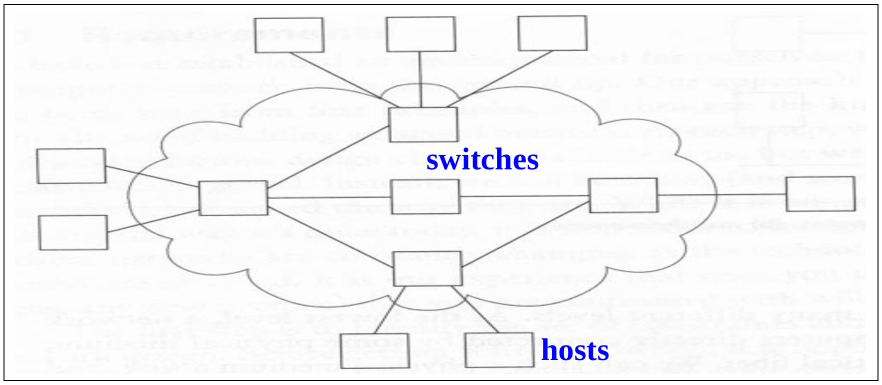
- ~~Introductions~~
- **Connectivity**
- Naming and addressing
- Abstractions and layering
- Example: socket programming
- Conclusions

Directly Connected



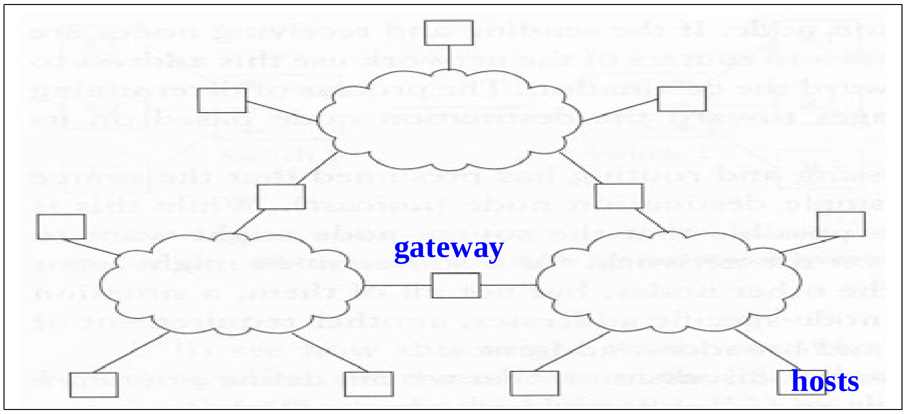
- (a) point-to-point: ATM
- (b) multiple-access: ethernet, FDDI
- Can’t build a network by requiring all nodes are directly connected to each other: scalability in terms of the number of wires or the number of nodes that can attach to a shared media

Switched Network



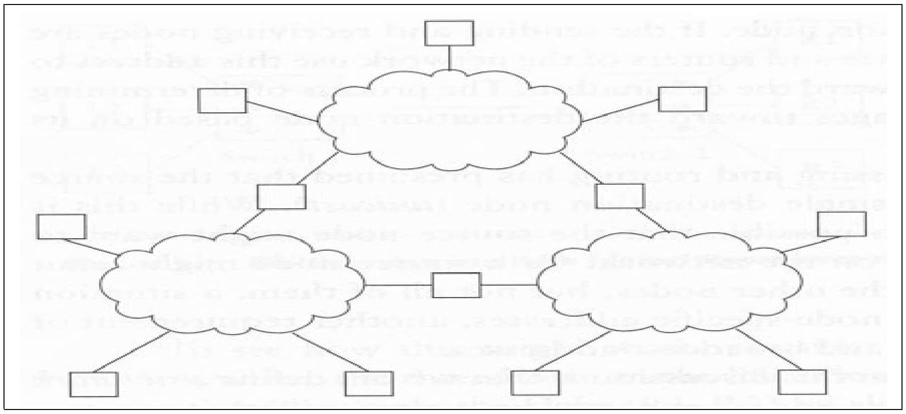
- Circuit switching vs. packet switching
- hosts vs. “the network”, which are made of switches
- Nice property: scalable aggregate throughput

Interconnection of Networks



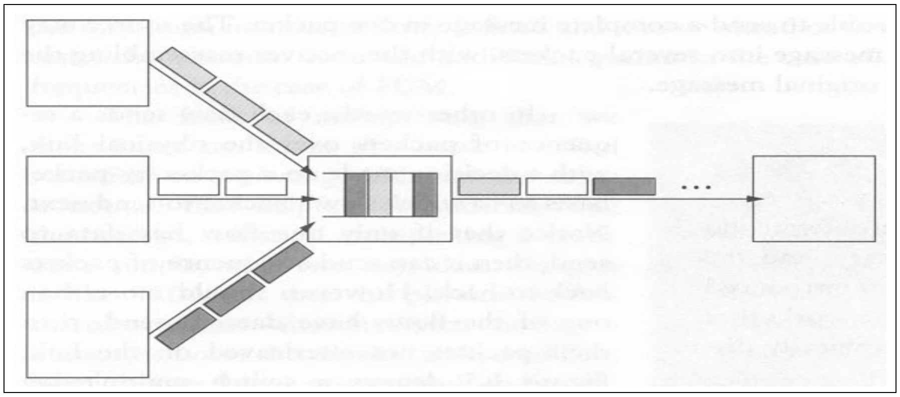
- Recursively building larger networks

Some Hard Questions



- How do hosts share links?
- How do you name and address hosts?
- Routing: given a destination address, how do you get to it?

Sharing a Link



- Chop up each “flow” into “packets” (packet switching)
- Bandwidth is allocated to each flow on-demand at the granularity of packets
- Key challenges: fairness, congestion control, QoS

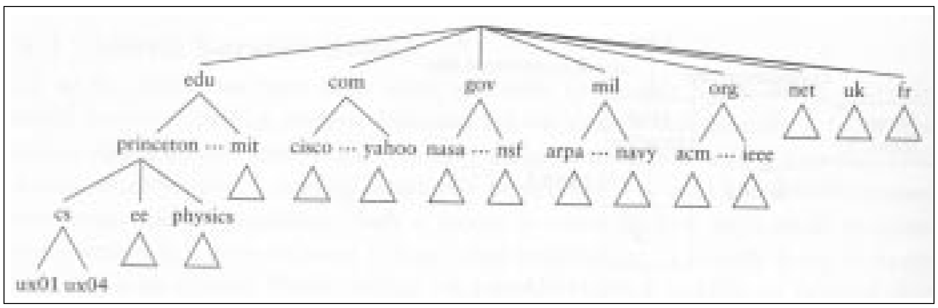
Outline

- ~~Introductions~~
- ~~Connectivity~~
- **Naming and addressing**
  - **DNS: Domain Name System**
- Abstractions and layering
- Example: socket programming
- Conclusions

IP Addresses and Host Names

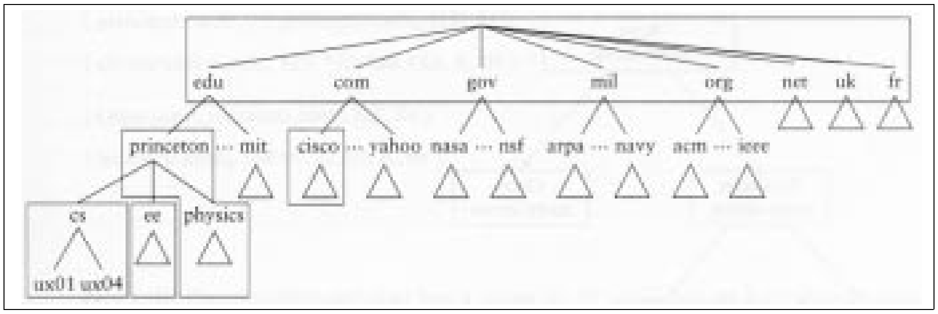
- Each machine is addressed by a 32-bit integer: IP address
  - We will tell you what “IP” is later
  - Ran out of numbers and there’re schemes to extend that
- An IP address
  - Written down in a “dot notation” for “ease” of reading such as “128.112.136.19”
  - Consists of a network address and a host ID
- IP addresses are the universal IDs that are used for everything, including routing
- For convenience, each host also has a human-friendly host name: for example “128.112.136.19” is “sequim.cs.princeton.edu”
- Question: how do you translate names to IP addresses

Domain Hierarchy



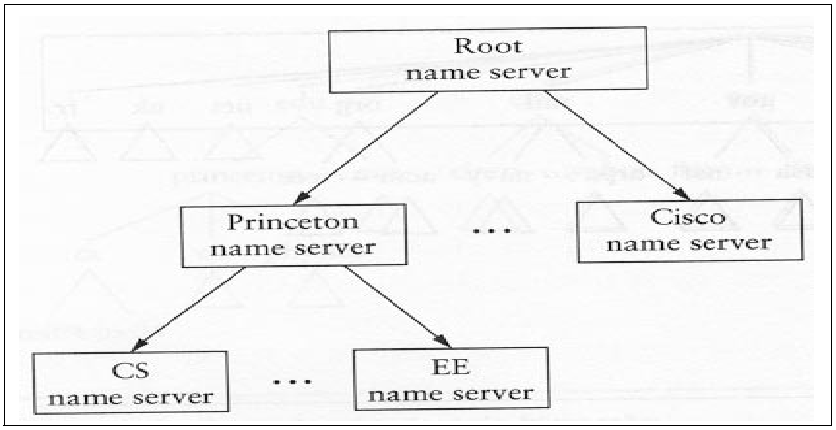
- Initially name-to-address mapping was a flat file mailed out to all the machines on the internet!
- Now we have a hierarchical name space, just like a Unix file system tree
- Top level names: historical influence: heavily US centric, government centric, and military centric view of the world

DNS Zones and Name Servers



- Divide up the name hierarchy into zones
- Each zone corresponds to one or more name servers under a single administrative control

Hierarchy of Name Servers

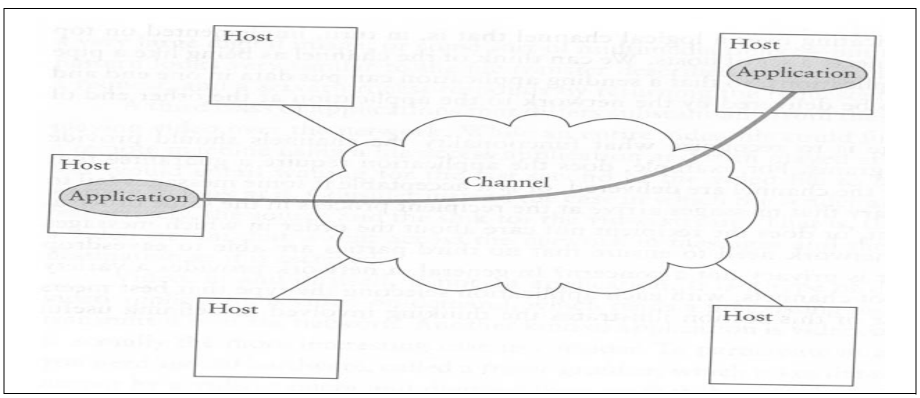


- Clients send queries to name servers
- Name servers reply with answer, or forward requests to other name servers
- Most name servers also perform lookup caching

Outline

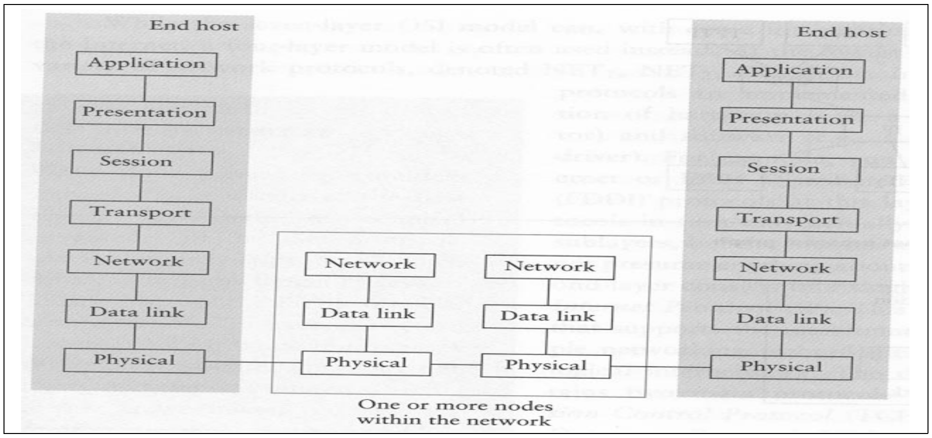
- ~~Introductions~~
- ~~Connectivity~~
- ~~Naming and addressing~~
- **Abstractions and layering**
- Example: socket programming
- Conclusions

Application-Level Abstraction



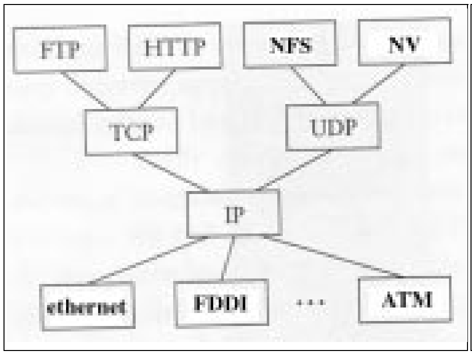
- What you have: hop-to-hop links, multiple routes, packets, can be potentially lost, can be potentially delivered out-of-order
- What you may want: application-to-application (end-to-end) channel, communication stream, reliable, in-order delivery

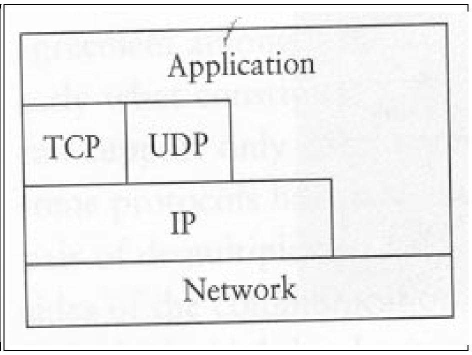
OSI Architecture



- Physical: handles **bits**
- Data link: provides "**frames**" abstraction
- Network: handles hop-to-hop routing, at the unit of **packets**
- Transport: provides process-to-process semantics such as in-order-delivery and reliability, at the unit of **messages**
- Top three layers are not well-defined, all have to do with application level abstractions such as transformation of different data formats

Reality: the "Internet" Architecture



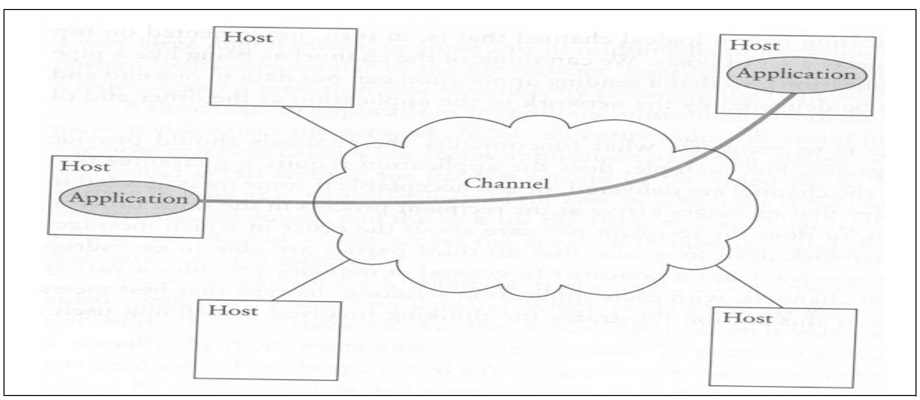


- Protocols: abstract objects that makeup a layer
- Lowest level: hardware specific, implemented by a combination of network adaptors and OS device drivers
- IP (Internet Protocol): focal point of the architecture, provides host-to-host connection, defines common method of exchanging packets
- TCP (Transmission Control Protocol): reliable, in-order stream
- UDP (User Datagram Protocol): unreliable messages (maybe faster)
- On top of those are the application protocols
- Not-strictly layered, "hour-glass shape", implementation-centric

Outline

- ~~Introductions~~
- ~~Connectivity~~
- ~~Naming and addressing~~
- ~~Abstractions and layering~~
- **Example: socket programming**
- Conclusions

Socket Interface



- Originated in BSD Unix
- One of the most widely-supported internet programming interfaces today
- A socket is an application-to-application channel
- Example program: half of a chat program

Java Chat

- To compile:
  `javac Client.java Server.java`
- To run:
  - In a window of the (any) server machine, type:
    `java Server`
  - In a window of the (any) client machine, type:
    `java Client name_of_the_server_machine`
- In the client window, just type away and you should see the messages echoed on the server window
- I'm going to omit some details having to do with error handling, real code on the lecture web page
- I will also put up an equivalent C version, which exposes a little more detail that the Java version hides

Client Code

```
Socket socket; String host;
BufferReader stdIn; String fromUser;    1
...
socket = new Socket(host, 5432);    2
out = new PrintWriter(socket.getOutputStream(), true);    3
BufferedReader stdIn = new BufferedReader(new
            InputStreamReader(System.in));    4
while ((fromUser = stdIn.readLine()) != null)    5
    out.println(fromUser);
```

1. Variable declarations
2. Creating a socket to the machine named by "host", at port number 5432
3. Makes an output stream from the socket so now the socket behaves just like the terminal screen
4. "stdIn" is an input stream from the keyboard
5. As long as the user types some input, it gets written to the socket

Server Code

```
ServerSocket serverSocket = null, clientSocket = null;
BufferedReader in = null; String inputLine;1
serverSocket = new ServerSocket(5432);2
...
while (true) {
   clientSocket = serverSocket.accept();3
   in = new BufferedReader(new InputStreamReader
              (clientSocket.getInputStream()));4
   while ((inputLine = in.readLine()) != null)5
      System.out.println(inputLine);
   clientSocket.close();6
}
```

1. Variable declarations
2. Creates a server socket that patiently waits for a client to connect
3. When a client connects to the server, this line returns a new socket for communicating to the client, leaving the old server socket waiting for other clients
4. Makes an input stream out of the client socket so it's just like a keyboard
5. As long as we get inputs from the client socket, we print it out
6. Destroy the client socket

Outline

- ~~Introductions~~
- ~~Connectivity~~
- ~~Naming and addressing~~
- ~~Abstractions and layering~~
- ~~Example: socket programming~~
- **Conclusions**

Closing Thoughts

- Historical perspective
  - Review: we said that the stuff we talked about in the OS lecture is nothing more than a point in time,
  - well same is true with the networking lecture
- Trends
  - Faster: bandwidth is increasing enormously, but latency is insurmountable--speed of light constraint
  - Ubiquitous: peripheral devices, backplane of super-computers
  - Ubiquitous: cheap personal and embedded devices
  - Ubiquitous: universal and global connectivity
  - Wireless
  - Mobile
- Challenges: parallel and distributed systems add a new dimension to everything in CS: programming, hardware, theory, systems, and applications
- It's an exciting time to be in computer science!