CS 162 Operating Systems and Systems Programming

Professor: Anthony D. Joseph

Spring 2003

Lecture 20: Networks and Distributed Systems

20.0 Main Points

Motivation for distributed vs. centralized systems

Survey of network technologies

Technology trends

Decade	Technology	$ per machine	Sales volume	Users per machine
50's	–	$10M	100	1000's
60's	Mainframes	$1M	10K	100's
70's	Mini-computers	$100K	1M	10's
80's	PC's	$10K	100M	1
90's	Handhelds	$100 – $1K	10B	1/10

20.1 Centralized vs. Distributed Systems

Distributed systems: physically separate computers working together

Why do we need distributed systems?

- Cheaper and easier to build lots of simple computers
- Easier to add power incrementally
- Principle of bureaucracy avoidance

The added promise of distributed systems:

- Higher availability – one machine goes down, use another
- Better reliability – store data in multiple locations
- More security – each piece easier to make secure

Reality has been disappointing:

- Worse availability – depend on every machine being up
- Worse reliability – can lose data if any machine crashes

- Worse security – anyone in world can break into system

Key idea: Coordination is more difficult, because we have to coordinate multiple copies of shared state information. (using only the network). What would be easy in a centralized system becomes a lot more difficult.

20.2 Goals: Transparencies and "ilities"

One important goal behind much distributed systems design is "Transparency". That is, distributed systems are complex, so you would like to shield the user from much of this complexity. (Or would you???). Various types of transparency are:

- **Location** – Can't tell where resources are located
- **Migration** – Resources may move throughout the system without the user knowing.
- **Replication** – Can't tell how many copies of a resource exist.
- **Concurrency** – Can't tell how many users there are
- **Parallelism** – System may be able to speed up large jobs by splitting them into smaller pieces that run at the same time.
- **Failure** – System may hide various things that go wrong in the system.

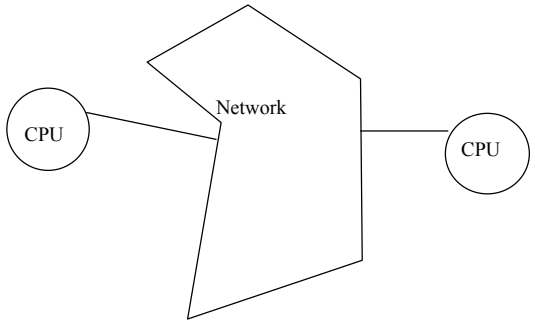
In addition, there are certain properties that are desirable of a distributed system. These include:

- **Availability –** what is the probability of the system being able to accept and process requests?
- **Reliability –** This is usually stronger than simply availability. It means not only is the system up, but that it is working correctly. Thus, it includes availability, security and fault tolerance.
- **Performance (not really an "ility" but should be) –** various metrics here: latency, throughput, type to first byte, time to last byte, scalability, etc. The key to performance in a distributed system is to avoid "bottlenecks". Possible bottlenecks include: centralized components, centralized state (e.g., system tables), centralized control, limited connectivity, and need for consistent "global" state.
- **Manageability**
- **Flexibility**

A great example of a system that meets many of these criteria is the Internet as it has evolved into its current form.

20.3 Definitions

Network: physical connection that allows two computers to communicate.



Packet: unit of transfer, sequence of bits carried over network

Network carries packets from one CPU to another. Destination gets interrupt when packet arrives.

Protocol: agreement between two parties as to how information is to be transmitted.

20.4 Broadcast Networks

Broadcast networks: shared communication medium.

For example, shared medium can be a wire – all hosts listen to wire. Inside a computer, this is called a "bus" – a shared set of wires between the CPU and the memory modules

Ethernet is an example broadcast network (10 Mbits/sec – 1Gbit/s)
More examples: cellular phones, Metricom packet radios (10Kbit/s – 100Kbit/sec).

20.4.1 Delivery

When you broadcast a packet, how does receiver know who it is for?

Put header on front of packet:

Destination | Packet

For example, header would contain unique machine # (network address) of target. Everyone gets packet, discards if not the target. In Ethernet, this check is done in hardware; no OS interrupt if not for you.

This is an example of layering: we're going to build up complex network protocols by layering more and more stuff on top of the packet.

20.4.2 Arbitration

How do your machines arbitrate for use of shared medium?

Aloha network (70's) – packet radio between Hawaiian islands

Arbitration: blind broadcast, with checksum at end of packet. If received ok (not garbled), send back an acknowledgement. If not received ok, discard.

Need checksum anyway, in case airplane flies overhead (or maybe a surfer goes by), and packet gets garbled.

Sender waits for a while, and if doesn't get an acknowledgement, re-transmits. So if two senders try to send at same time, both get garbled, both simply re-send later.

Problem: stability. What if load increases?

Ethernet – early 80's, first practical local area network (Xerox PARC). 10 Mb/s. Most common LAN for UNIX environments, becoming more common in PC's. What we have in the department.

Used wire instead of radio, but still broadcast: all machines tap into single wire, listen to all packets.

Key advance was in a new way to do arbitration called CSMA/CD: Carrier sense multiple access/ collision detection. Three pieces:

1. Carrier sense – don't send unless idle
2. Collision detect – sender checks if packet is trampled. If so, abort, wait, and retry.
   How long should I wait, after trying to send and failing? What if everyone waits the same length of time? We'd all keep colliding forever.
   Also, what if load increases? More likely to get collisions.
3. Adaptive randomized waiting strategy – if try to send, and get collision, means there are a number of people trying to send, so pick bigger mean wait time. If collide again, pick even bigger wait.

20.5 Point-to-point networks

Here's a different way of thinking about all this – why have a shared bus for Ethernet at all, why not simplify and only have point-to-point links, plus routers/switches?

Central idea behind ATM (asynchronous transfer mode), the first commercial point-to-point LAN. Inspiration for ATM taken from telephone network.

A bunch of advantages:

1. Higher link performance – can drive point-to-point connection faster than broadcast link
2. Greater aggregate bandwidth than single link
3. Can add capacity incrementally – add more links/switches to get more capacity
4. Better fault tolerance (as in Internet)
5. Lower latency (no need for arbitration to send, although you do need a buffer in the switch)

Disadvantage: more expensive than having everyone share one bus.

But, technology has been relentlessly driving the costs down!

As a result, point-to-point communications are starting to be used in everything from workstations, to local area networks, to the Internet.

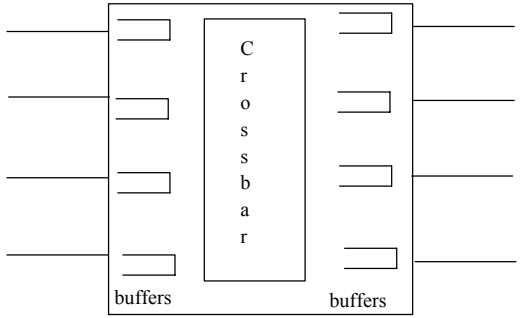
Workstations: In a few years, open up a workstation, will see its CPU connected to memory and graphics engine by a switched network, instead of a bus.

Multiprocessors are already connected by hooking together lots of small-scale switches. For instance, in a 2-D mesh, or in a hypercube.

In LAN's, not only ATM, but now a version of Ethernet called "switched Ethernet" – uses same packet format, analog signaling as Ethernet, but only one machine on each Ethernet.

20.5.1 Issues in point-to-point networks

Switches look just like computers: inputs, memory, and outputs.



Switch

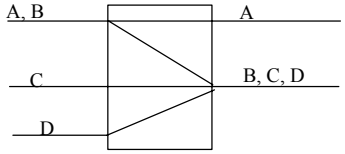
What if everyone sends to the same output? Congestion. What happens when buffers fill up?

Option 1: no flow control. Packets get dropped if they arrive and there's no space.

If I send a lot, I'll grab the buffers, and then everyone else is hosed.

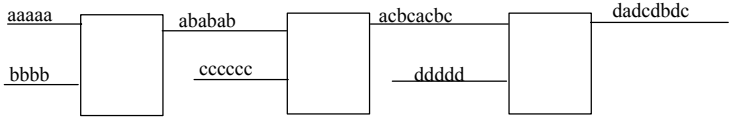
Option 2: flow control between switches. Don't send me more, unless I have room.

Problem: cross-traffic. What if path from source to destination is completely unused, but goes through some switch that has buffers filled up with unrelated traffic?



Option 3: per flow flow control. Allocate a separate set of buffers to each end-to-end stream, and use "don't send me more" on each separate end-to-end stream.

Problem: fairness



Throughput of each stream is entirely dependent on topology, and relationship to bottleneck.

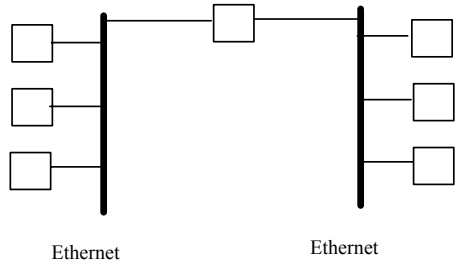
20.6 The Internet

What happens if you need more bandwidth than a single Ethernet?

For example, SUN has > 10000 workstations.

Buy two Ethernets? If so, how do machines on each network talk to each other?

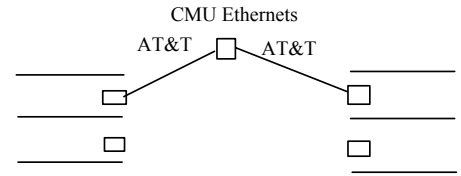
Put machine that straddles both networks. Lots of different words for this function: router, gateway, bridge, repeater. But basically, acts like a switch. Machine watches all packets on each Ethernet, and if packet is for machine on other one, then copies packet over.



The Internet is just a generalization of this.

Internet = interconnecting local area networks.

Local networks can be anything – Ethernet, AppleTalk, FDDI, even phone company wires, but building blocks are machines that straddle two or more networks.



20.6.1 Routing

How do packets get to their destination? Simple if there's a single machine that straddles all networks, but that won't work in the Internet!

If packet has to go several hops before it gets to destination, and router straddles several networks, how do routers know how to forward packets?

To answer this, some definitions:

- **Name** – Mom, Soda
- **Address** – phone #, network address
- **Route** – how do we get there from here

Internet solution: routing tables. Each router looks at packet header, does table lookup to decide which link to use to get it closer to destination.

Also, all machines on the same local area network, have common portion of address/machine id.

Routing table contains:

Destination LAN –> output link that gets closer to destination

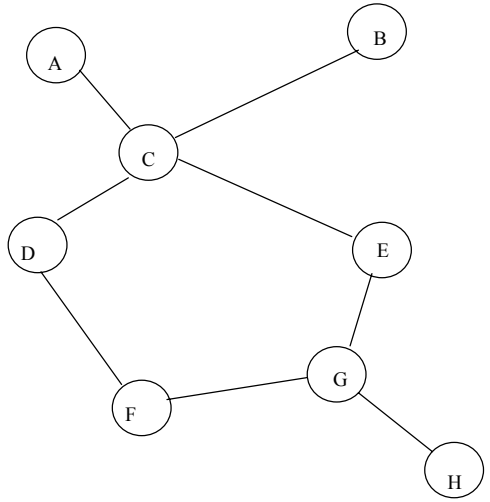
How do you set up the table? Internet has no centralized state! No single machine knows entire topology, and topology is constantly changing!

Instead:

1. Routing table has "cost" – number of hops to destination (in practice, also considers how heavily used each link is)
2. Neighbors periodically exchange routing tables
3. If neighbor has cheaper route, use that one

So:

Initially, routers don't know about any destination
Loop
    Get routing table from neighbors
    Update routing table



Note that since the Internet is made up of many individual networks, it's routing is similar but more complicated. Basically these types of algorithms are performed at different levels of the network, such as within a subnet and then among the routers that route between the subnets.