Initializing `sockaddr_in` from hostname

```c
void mksin (struct_sockaddr *sinp, const char *host, int port)
{
    struct hostent *hp;

    bzero (sinp, sizeof (*sinp));
    sinp->sin_family = AF_INET;
    sinp->sin_port = htons (port);
    if (!(hp = gethostbyname (host))) {
        fprintf (stderr, "%s: bad host name\n", host);
        exit (1);
    }
    sinp->sin_addr = *(in_addr *) hp->h_addr;
    // or: *
    memset (&sinp->sin_addr, hp->h_addr, sizeof (sinp->sin_addr));
}
```
Goal: deliver packets from input to output ports

Three potential performance concerns:
- Throughput in terms of bytes/time
- Throughput in terms of packets/time
- Latency
Shared bus switch

- **Shared bus – like your PC**
  - NIC DMAs packet to memory over I/O bus
  - CPU examines pkt header, sends to dest NIC over bus
  - I/O bus is serious bottleneck
  - For small packets, CPU may be limited, too

- **Shared memory – similar, has memory bottleneck**
Crossbar switch

- Can connect any input to any output
  - Trivially allows any input→output permutation
  - More than one input to same output requires trickery
Self-routing switches

- Idea: Build up switch out of $2 \times 2$ elements
- Each packet contains a “self-routing header”
  - For each switch along the way, specifies the output
- Must somehow compute a path when introducing packet
  - Is there more than one path to choose from?
  - Will path collide with another packet?
- Easy to implement stages once path computed
Banyan networks

- A Banyan network has exactly one path from any input port to a given output port
  - Example: Each stage can flip one bit of the port number

- Easy to compute paths

- Problem: Not all permutations can be routed
  - Might want 1 → 0 and 7 → 1, but both paths use same link

- But: Can always route packets if sorted
  - Leads to batcher banyan networks
  - Batcher phase sorts packets before banyan
Example: Banyan network
Beneš

- Two back to back Banyan networks
  - Can route any permutation of inputs → outputs
  - (Can be proven by induction on size of network)

- Unfortunately, figuring schedule is global problem
Bisection bandwidth

• Can speak of the bandwidth between sets of ports
  - Bandwidth is maximum achievable aggregate bandwidth between the two sets

• **Bisection bandwidth is important property of network**
  - Lowest possible bandwidth between equal-sized sets of ports
  - Or almost equal-sized if odd number of ports

• A network with bad bisection bandwidth may offer poor behavior
  - Even if no conflict between input and output link utilization, may have internal bottlenecks reducing throughput
Example: Poor bisection bandwidth

- Connect two Ethernet switches with Ethernet
  - Suppose all clients on left, and all servers on right...
  - Aggregate bandwidth between all clients and servers only 100Mbit/s
Example: Poor bisection bandwidth 2

- Remember it’s *worst case* cut
  - Even with one fat link, don’t have to slice down middle
  - Put fat link in one partition, and bisection b/w very small