Switching

- **Switch**
  - Forwards packets from input port to output port
  - Port selected based on address in packet header

- **Advantages**
  - Cover large geographic area (tolerate latency)
  - Support large numbers of hosts (scalable bandwidth)
Source routing
Virtual Circuit Switching

- Explicit connection setup (and tear-down) phase
  - Establishes virtual-circuit ID on each link
  - Each switch maintains VC table

- Subsequent packets follow same circuit
  - Switch maps \( \langle \text{in-link, in-VCI} \rangle \rightarrow \langle \text{out-link, out-VCI} \rangle \)

- Sometimes called connection-oriented model
Datagram switching

- No connection setup phase
  - Switches have routing table based on node addresses
- Each packet forwarded independently
- Sometimes called connectionless model
Virtual Circuit Model

- Typically wait full RTT for connection setup before sending first data packet
- While the connection request contains the full address for destination, each data packet contains only a small identifier, making the per-packet header overhead small
- If a switch or a link in a connection fails, the connection is broken and a new one needs to be established
- Connection setup provides an opportunity to reserve resources
Datagram Model

- There is no round trip time delay waiting for connection setup; a host can send data as soon as it is ready.
- Source host has no way of knowing if the network is capable of delivering a packet or if the destination host is even up.
- Since packets are treated independently, it is possible to route around link and node failures.
- Since every packet must carry the full address of the destination, the overhead per packet is higher than for the connection-oriented model.
Cut through vs. store and forward

- Two approaches to forwarding a packet
  - Receive a full packet, then send it on output port
  - Start retransmitting as soon as you know output port, before you have even received the full packet (*cut-through*)

- Cut-through routing can greatly decrease latency

- Disadvantage: Can’t always send useful packet
  - If packet corrupted, won’t check CRC till after you started transmitting
  - Or if Ethernet collision, may have to send run packet on output link, wasting bandwidth
Optical switches

- Already analog optical repeaters deployed
  - Will amplify any signal
  - Can change your low-level transmission protocol w/o replacing repeaters

- Could possibly do the same thing for switching
  - Microscopic mirrors can redirect light to different ports
  - (The ultimate cut-through routing)

- Technology exists, but not widely deployed
  - Optical switch will not see packet headers
  - Instructions on where to send packet need to be out-of-band
Bridges and extended LANs

- LANs have physical limitations (e.g., 2500m)
- Connect two or more LANs with a bridge
  - Operates at on Ethernet addresses
  - No encapsulation required
- Ethernet switch like a multi-way bridge
Learning bridges

- Idea: Don’t forward packet if not useful
  - If you know recipient is not on that port

- Learn host’s location based on source address
  - Switch builds a table when it receives packets

- Table says when *not* to forward packet
  - Does not need to be complete for correct behavior
Dealing w. loops

- Problem: People might form loops
  - Accidentally, or to provide redundancy if link fails
  - Don’t want to forward packets infinitely!
Spanning tree

- Need to disable ports, so that no loops in network
- Like creating a spanning tree in a graph
  - View switches and networks as nodes, ports as edges
Spanning tree algorithm

- Every bridge has a unique ID (Ethernet address)
- Let bridge w. smallest ID be the root
- Each segment has one designated bridge responsible for forwarding packets over it
  - Bridge closest to root is designated bridge
  - If there is tie, smallest ID is designated bridge

- Overview:
  - Each node assumes it is the root (and thus the dedicated bridge for all its ports)
  - Sends messages, may learn of a better root
  - Eventually links from nets to designated bridges form a spanning tree
Spanning tree protocol

- **Spanning tree messages contain:**
  1. ID of bridge sending message
  2. ID sender believes is the root
  3. Distance (in hops) from sender to root

- **Bridges remember best config message on each port**

- **Send message when you think you are the root**

- **Otherwise, forward messages from best known root**
  - Add one to distance before forwarding
  - Don’t forward if you know you aren’t dedicated bridge

- **In the end, only root is generating messages**
Multicast and Broadcast

- Forward all broadcast/multicast frames
  - current practice

- Could learn when no group members downstream
  - Have each member of group $G$ periodically send a frame to bridge multicast address with $G$ in source field
Limitations of bridges

- **Do not scale**
  - Spanning tree algorithm does not scale
  - Broadcast does not scale
  - No way to route around congested links

- **Caution: beware of transparency**
  - Some applications could get confused by bridge
  - Much more likely to drop packets (with congestion)
  - Makes latency between nodes very non-uniform
• Imagine network in an office building
  - Groups take over offices, move around
  - Don’t want to switch wires to change office → net mapping
  - Don’t want one big Ethernet (too much broadcast traffic)

• Solution: Virtual LANs
  - Encapsulate Ethernet, add 12-bit VLAN ID
  - Send encapsulated packets on backbone
ATM

- Connection-oriented, packet-switched network
  - Used by telco, often what goes over SONET

- Based of fixed-size, 53-byte packets
  - 5-byte header + 48-byte payload

- Signaling protocol (Q.2931) can make various Quality of Service guarantees

- Used for very high speed LAN backbones
  - For a while was fastest thing you could get
Why cells?

- **Easier to implement in hardware**
  - Implementing functionality in hardware is faster
  - Easier to build hardware for fixed packet lengths

- **Easier to have parallelism in implementation**
  - Simple building block to process packet in fixed time
  - Process parallel packets & complete simultaneously

- **Queue management is much easier**
  - Can switch between flows at very fine granularity
  - Especially important for controlling delay & jitter
  - Never get stuck waiting to finish sending long packet

- **Small delay from store & forward**
  - Consider voice samples @ 1 byte / 125 μs
Why 53 bytes?

- Packets have 48 bytes payload + 5 byte header

- The issue: echo cancelation needed if long delay
  - Echo cancelation equipment is expensive
  - Unnecessary w. 32B packetization delay + RTT across France
  - But U.S. needs equipment anyway, so wanted more efficient 64B

- The result: Compromise at 48 bytes
  - Less efficient and requires echo cancelation
  - Moral of the story: Design by committee sucks
## ATM cell format

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>3</th>
<th>1</th>
<th>8</th>
<th>384 (48 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFC</td>
<td>VPI</td>
<td>VCI</td>
<td>Type</td>
<td>CLP</td>
<td>HEC (CRC-8)</td>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>

- **GFC** – Generic flow control (mostly unused)
- **VPI/VCI** – Virtual path/circuit identifier
- **CLP** – Cell loss priority
- **HEC** – Header error correction (header only)
VPI vs. VCI

- Public networks care only about VPI
- Leaves VCI for individual sites
  - Another level of demultiplexing
  - Many machines on net A talk to many machines on net B, w. only one VPI
Segmentation and reassembly

- Recall *encapsulation* from first lecture
  - One layer (say IP) sends packet to lower layer (say Ethernet)
  - Ethernet prepends its header, IP header now Ethernet payload

- Doesn’t work so well w. 48-byte ATM cell payloads!

- Solution: **AAL** – *ATM adaptation layer*
  - Segments larger packets into cells, reassembles
Several different AAL layers

- Different formats for different needs
- **AAL 1/2** – for apps like voice needing guarantees
- **AAL 3** – connection-oriented packet services (X.25)
- **AAL 4** – datagram services (like IP)
  - Distinction between 3 & 4 turns out not to matter
- **AAL 5** – saner, simpler replacement for AAL 3/4
AAL 3/4 CS-PDU

- **CS-PDU** – *convergent sublayer protocol data unit*
  - Encapsulates higher-level protocol packets, e.g., IP
  - **CPI** – common part indicator (like version no.) always 0
  - **Btag/Etag** – must match (avoids framing errors)
  - **BASize** – buffer allocation size is hint, not length of packet
Packing AAL 3/4 CS-PDUs into cells

- Break CS-PDU into cells, each with AAL header/trailer
### AAL 3/4 header/trailer format

<table>
<thead>
<tr>
<th></th>
<th>40</th>
<th>2</th>
<th>4</th>
<th>10</th>
<th>352 (44 bytes)</th>
<th>6</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ATM header</td>
<td>Type</td>
<td>SEQ</td>
<td>MID</td>
<td>Payload</td>
<td>Length</td>
<td>CRC-10</td>
</tr>
</tbody>
</table>

- **Type** – first/last/middle cell, or single-cell CS-PDU
- **MID** – *multiplexing identifier*
  - Allows multiple CS-PDUs to be interleaved on one VC
- **SEQ** Sequence number, detects loss/reordering
- **CRC** Detects bit errors in cell
  - Is this good enough? (recall no CRC in CS-PDU)
AAL 5

- **AAL 3/4 uses up too many bits of payload**
  - Really just want to know when end of frame is…
  - Idea: Why not just co-opt one bit in 5-byte ATM header?

- **AAL 5 CS-PDU:**
  - Broken into 48 byte chunks and placed in cells
  - Stolen header bit indicates end of frame

- **What’s missing compared to AAL 3/4 cells?**
  - No MID (means you can’t interleave within VC)
  - No per-cell SEQ, CRC – but stronger CRC in CS-PDU will catch dropped/reordered packets