Switching

An Engineering Approach to Computer Networking
What is it all about?

- How do we move traffic from one part of the network to another?
- Connect end-systems to switches, and switches to each other
- Data arriving to an input port of a switch have to be moved to one or more of the output ports
Types of switching elements

- Telephone switches
  - switch samples
- Datagram routers
  - switch datagrams
- ATM switches
  - switch ATM cells
Classification

- **Packet vs. circuit switches**
  - packets have headers and samples don’t

- **Connectionless vs. connection oriented**
  - connection oriented switches need a call setup
  - setup is handled in *control plane* by *switch controller*
  - connectionless switches deal with *self-contained* datagrams

<table>
<thead>
<tr>
<th></th>
<th>Connectionless (router)</th>
<th>Connection-oriented (switching system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet switch</td>
<td>Internet router</td>
<td>ATM switching system</td>
</tr>
<tr>
<td>Circuit switch</td>
<td></td>
<td>Telephone switching system</td>
</tr>
</tbody>
</table>
Other switching element functions

- Participate in routing algorithms
  - to build routing tables
- Resolve contention for output trunks
  - scheduling
- Admission control
  - to guarantee resources to certain streams
- We’ll discuss these later
- Here we focus on pure data movement
Requirements

- Capacity of switch is the maximum rate at which it can move information, assuming all data paths are simultaneously active.
- Primary goal: \textit{maximize capacity}
  - subject to cost and reliability constraints.
- Circuit switch must reject call if can’t find a path for samples from input to output.
  - goal: \textit{minimize call blocking}.
- Packet switch must reject a packet if it can’t find a buffer to store it awaiting access to output trunk.
  - goal: \textit{minimize packet loss}.
- Don’t reorder packets.
A generic switch
Outline

- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches
Circuit switching

- Moving 8-bit samples from an input port to an output port
- Recall that samples have no headers
- Destination of sample depends on *time* at which it arrives at the switch
  - actually, relative order within a *frame*
- We’ll first study something simpler than a switch: a multiplexor
Multiplexors and demultiplexors

- Most trunks time division multiplex voice samples
- At a central office, trunk is demultiplexed and distributed to active circuits
- Synchronous multiplexor
  - N input lines
  - Output runs N times as fast as input
More on multiplexing

- Demultiplexor
  - one input line and N outputs that run N times slower
  - samples are placed in output buffer in round robin order
- Neither multiplexor nor demultiplexor needs addressing information (why?)
- Can cascade multiplexors
  - need a standard
  - example: DS hierarchy in the US and Japan
Inverse multiplexing

- Takes a high bit-rate stream and scatters it across multiple trunks
- At the other end, combines multiple streams
  - resequencing to accommodate variation in delays
- Allows high-speed virtual links using existing technology
A circuit switch

- A switch that can handle N calls has N logical inputs and N logical outputs
  - N up to 200,000
- In practice, input trunks are multiplexed
  - example: DS3 trunk carries 672 simultaneous calls
- Multiplexed trunks carry frames = set of samples
- Goal: extract samples from frame, and depending on position in frame, switch to output
  - each incoming sample has to get to the right output line and the right slot in the output frame
  - demultiplex, switch, multiplex
Call blocking

- Can’t find a path from input to output
- Internal blocking
  - slot in output frame exists, but no path
- Output blocking
  - no slot in output frame is available
- Output blocking is reduced in *transit* switches
  - need to put a sample in one of *several* slots going to the desired next hop
Time division switching

- Key idea: when demultiplexing, position in frame determines output trunk
- Time division switching interchanges sample position within a frame: time slot interchange (TSI)
How large a TSI can we build?

- Limit is time taken to read and write to memory
- For 120,000 circuits
  - need to read and write memory once every 125 microseconds
  - each operation takes around 0.5 ns => impossible with current technology
- Need to look to other techniques
Space division switching

- Each sample takes a different path through the switch, depending on its destination.
Crossbar

- Simplest possible space-division switch
- *Crosspoints* can be turned on or off
- For multiplexed inputs, need a switching *schedule* (why?)
- Internally nonblocking
  - but need $N^2$ crosspoints
  - time taken to set each crosspoint grows quadratically
  - vulnerable to single faults (why?)
Multistage crossbar

- In a crossbar during each switching time only one crosspoint per row or column is active
- Can save crosspoints if a crosspoint can attach to more than one input line (why?)
- This is done in a multistage crossbar
- Need to rearrange connections every switching time
Multistage crossbar

- Can suffer internal blocking
  - unless sufficient number of second-level stages
- Number of crosspoints < $N^2$
- Finding a path from input to output requires a depth-first-search
- Scales better than crossbar, but still not too well
  - 120,000 call switch needs ~250 million crosspoints
Time-space switching

- Precede each input trunk in a crossbar with a TSI
- Delay samples so that they arrive at the right time for the space division switch’s schedule
Time-space-time (TST) switching

- Allowed to flip samples both on input and output trunk
- Gives more flexibility => lowers call blocking probability
Outline

- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches
Packet switching

- In a circuit switch, path of a sample is determined at time of connection establishment
- No need for a sample header--position in frame is enough
- In a packet switch, packets carry a destination field
- Need to look up destination port on-the-fly
- Datagram
  - lookup based on entire destination address
- Cell
  - lookup based on VCI
- Other than that, very similar
Repeaters, bridges, routers, and gateways

- Repeaters: at physical level
- Bridges: at datalink level (based on MAC addresses) (L2)
  - discover attached stations by listening
- Routers: at network level (L3)
  - participate in routing protocols
- Application level gateways: at application level (L7)
  - treat entire network as a single hop
  - e.g mail gateways and transcoders
- Gain functionality at the expense of forwarding speed
  - for best performance, push functionality as low as possible
Port mappers

- Look up output port based on destination address
- Easy for VCI: just use a table
- Harder for datagrams:
  - need to find *longest prefix match*
    - e.g. packet with address 128.32.1.20
    - entries: (128.32.*, 3), (128.32.1.*, 4), (128.32.1.20, 2)
- A standard solution: trie
Tries

- Two ways to improve performance
  - cache recently used addresses in a CAM
  - move common entries up to a higher level (match longer strings)
Blocking in packet switches

- Can have both internal and output blocking
- Internal
  - no path to output
- Output
  - trunk unavailable
- Unlike a circuit switch, cannot predict if packets will block (why?)
- If packet is blocked, must either buffer or drop it
Dealing with blocking

- **Overprovisioning**
  - internal links much faster than inputs

- **Buffers**
  - at input or output

- **Backpressure**
  - if switch fabric doesn’t have buffers, prevent packet from entering until path is available

- **Parallel switch fabrics**
  - increases effective switching capacity
Outline

- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches
Three generations of packet switches

- Different trade-offs between cost and performance
- Represent evolution in switching capacity, rather than in technology
  - With same technology, a later generation switch achieves greater capacity, but at greater cost
- All three generations are represented in current products
First generation switch

- Most Ethernet switches and cheap packet routers
- Bottleneck can be CPU, host-adaptor or I/O bus, depending
Example

- First generation router built with 133 MHz Pentium
  - Mean packet size 500 bytes
  - Interrupt takes 10 microseconds, word access take 50 ns
  - Per-packet processing time takes 200 instructions = 1.504 µs

- Copy loop
  
  ```
  register <- memory[read_ptr]
  memory [write_ptr] <- register
  read_ptr <- read_ptr + 4
  write_ptr <- write_ptr + 4
  counter <- counter -1
  if (counter not 0) branch to top of loop
  ```

- 4 instructions + 2 memory accesses = 130.08 ns
- Copying packet takes 500/4 *130.08 = 16.26 µs; interrupt 10 µs
- Total time = 27.764 µs => speed is 144.1 Mbps

- Amortized interrupt cost balanced by routing protocol cost
Second generation switch

- Port mapping intelligence in line cards
- ATM switch guarantees hit in lookup cache
- Ipsilon *IP switching*
  - assume underlying ATM network
  - by default, assemble packets
  - if detect a flow, ask upstream to send on a particular VCI, and install entry in port mapper $\Rightarrow$ implicit signaling
Third generation switches

- Bottleneck in second generation switch is the bus (or ring)
- Third generation switch provides parallel paths (fabric)
Third generation (contd.)

Features
- self-routing fabric
- output buffer is a point of contention
  - unless we *arbitrate* access to fabric
- potential for unlimited scaling, as long as we can resolve contention for output buffer
Outline

- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches
Switch fabrics

- Transfer data from input to output, ignoring scheduling and buffering
- Usually consist of links and *switching elements*
Crossbar

- Simplest switch fabric
  - think of it as 2N buses in parallel
- Used here for *packet* routing: crosspoint is left open long enough to transfer a packet from an input to an output
- For fixed-size packets and known arrival pattern, can compute schedule in advance
- Otherwise, need to compute a schedule on-the-fly (what does the schedule depend on?)
Buffered crossbar

- What happens if packets at two inputs both want to go to same output?
- Can defer one at an input buffer
- Or, buffer crosspoints
Broadcast

- Packets are tagged with output port #
- Each output matches tags
- Need to match N addresses in parallel at each output
- Useful only for small switches, or as a stage in a large switch
Switch fabric element

- Can build complicated fabrics from a simple element
- Routing rule: if 0, send packet to upper output, else to lower output
- If both packets to same output, buffer or drop
Features of fabrics built with switching elements

- \( N \times N \) switch with \( b \times b \) elements has \( \left\lfloor \log_b N \right\rfloor \) elements with \( \left\lceil \frac{N}{b} \right\rceil \) elements per stage
- Fabric is *self routing*
- Recursive
- Can be synchronous or asynchronous
- Regular and suitable for VLSI implementation
Banyan

- Simplest self-routing recursive fabric

- (why does it work?)

- What if two packets both want to go to the same output?
  - output blocking
Blocking

- Can avoid with a buffered banyan switch
  - but this is too expensive
  - hard to achieve zero loss even with buffers
- Instead, can check if path is available before sending packet
  - three-phase scheme
  - send requests
  - inform winners
  - send packets
- Or, use several banyan fabrics in parallel
  - intentionally misroute and tag one of a colliding pair
  - divert tagged packets to a second banyan, and so on to k stages
  - expensive
  - can reorder packets
  - output buffers have to run k times faster than input
Sorting

- Can avoid blocking by choosing order in which packets appear at input ports

- If we can
  - present packets at inputs sorted by output
  - remove duplicates
  - remove gaps
  - precede banyan with a perfect shuffle stage
  - then no internal blocking


- Need sort, shuffle, and trap networks
Sorting

- Build sorters from merge networks
- Assume we can merge two sorted lists
- Sort pairwise, merge, recurse
Merging
Putting it together- Batcher Banyan

What about trapped duplicates?
- recirculate to beginning
- or run output of trap to multiple banyans (*dilation*)
Effect of packet size on switching fabrics

- A major motivation for small fixed packet size in ATM is ease of building large parallel fabrics
- In general, smaller size => more per-packet overhead, but more preemption points/sec
  - At high speeds, overhead dominates!
- Fixed size packets helps build synchronous switch
  - But we could fragment at entry and reassemble at exit
  - Or build an asynchronous fabric
  - Thus, variable size doesn’t hurt too much
- Maybe Internet routers can be almost as cost-effective as ATM switches
Outline

- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches
Buffering

- All packet switches need buffers to match input rate to service rate
  - or cause heavy packet losses
- Where should we place buffers?
  - input
  - in the fabric
  - output
  - shared
Input buffering (input queueing)

- No speedup in buffers or trunks (unlike output queued switch)
- Needs arbiter
- Problem: *head of line blocking*
  - with randomly distributed packets, utilization at most 58.6%
  - worse with *hot spots*
Dealing with HOL blocking

- Per-output queues at inputs
- Arbiter must choose one of the input ports for each output port
- How to select?
- Parallel Iterated Matching
  - inputs tell arbiter which outputs they are interested in
  - output selects one of the inputs
  - some inputs may get more than one grant, others may get none
  - if >1 grant, input picks one at random, and tells output
  - losing inputs and outputs try again
- Used in DEC Autonet 2 switch
Output queueing

- Don’t suffer from head-of-line blocking
- But output buffers need to run much faster than trunk speed (why?)
- Can reduce some of the cost by using the *knockout* principle
  - unlikely that all N inputs will have packets for the same output
  - drop extra packets, fairly distributing losses among inputs
Shared memory

- Route only the header to output port
- Bottleneck is time taken to read and write multiported memory
- Doesn’t scale to large switches
- But can form an element in a multistage switch
Datapath: clever shared memory design

- Reduces read/write cost by doing wide reads and writes
- 1.2 Gbps switch for $50 parts cost
Buffered fabric

- Buffers in each switch element

- Pros
  - Speed up is only as much as fan-in
  - Hardware backpressure reduces buffer requirements

- Cons
  - costly (unless using single-chip switches)
  - scheduling is hard
Hybrid solutions

- Buffers at more than one point
- Becomes hard to analyze and manage
- But common in practice
Outline

- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches
Multicasting

- Useful to do this in hardware
- Assume portmapper knows list of outputs
- Incoming packet must be copied to these output ports
- Two subproblems
  - generating and distributing copies
  - VCI translation for the copies
Generating and distributing copies

- Either implicit or explicit
  - Implicit
    - suitable for bus-based, ring-based, crossbar, or broadcast switches
    - multiple outputs enabled after placing packet on shared bus
    - used in Paris and Datapath switches
  - Explicit
    - need to copy a packet at switch elements
    - use a copy network
    - place # of copies in tag
    - element copies to both outputs and decrements count on one of them
    - collect copies at outputs
- Both schemes increase blocking probability
Header translation

- Normally, in-VCI to out-VCI translation can be done either at input or output
- With multicasting, translation easier at output port (why?)
- Use separate port mapping and translation tables
- Input maps a VCI to a set of output ports
- Output port swaps VCI
- Need to do two lookups per packet