Switches and Scale

- Recall Ethernet max. size is 1024 stations (< 500 recommended [100/segment]).
- Switches allow interconnection of links to provide larger networks.
- Interconnection of switches provides larger scale without necessarily reducing individual end-node performance.

Star Topologies

- Switches use star topologies (everyone gets a link to the switch).
- This provides:
  - per-station monitoring/control
  - the chance for many times the base bandwidth in aggregate
- Switching: mapping input to output

Switch Example

- Each runs at max link rate

Datagrams vs Circuits

- Yes, this decision once again...
- Virtual circuit switching:
  - connection-oriented
  - reserves switch resources
- Datagram model:
  - connectionless
  - may lead to congestion loss

Virtual Circuit Switching

- Virtual circuits:
  - connection setup establishes a path through switches
  - a virtual circuit ID (VCI) identifies path
  - uses packet switching, with packets containing VCI
  - (small) VCIs are often indices into per-switch connection tables; change at each hop
**Virtual Circuit Tables**

- VC switches contain VC tables which map incoming packets to outgoing (VCI, port):

<table>
<thead>
<tr>
<th>Input Port</th>
<th>Incoming Identifier</th>
<th>Output Port</th>
<th>Outgoing Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

**VCS Observations**

- Generally requires at least 1 RTT for connection establishment
- Small VCID unique to each link (small overhead, but requires VC tables)
- Failure requires end station to re-establish

**Datagram Switching**

- General idea: no connection establishment, but each packet contains enough info to specify destination
- Switches contain forwarding tables (but no per-connection "state")
- Forwarding tables contain info on which outgoing port to use for each destination

**Datagram Switching Observations**

- No initial RTT time required for connection setup
- No way of knowing if packet will arrive
- Failures do not require end-point re-connection if redundant paths exist
- Every packet has full destination address, implying more overhead per packet

**Switching Hardware**

- General-purpose systems can perform switching, but are typically slower due to memory and I/O bus limitations
- Design goals:
  - Throughput (# of packets switched/sec)
  - Scalability (# input/output ports supported)
- Terminology: an n x m switch has n input and m output ports (often n = m)

**Switch Throughput**

- Ideally with n inputs, each of speed s, we'd expect ns throughput (m >= n)
- Doesn't quite work this way due to contention (e.g., multiple inputs going to single output)
- During contention, data is queued or dropped, limiting throughput to below ns
Traffic Modeling

- so, we see that switch performance is a function of traffic dynamics:
  - when do packets arrive?
  - where are they going?
  - how big are they?
- traffic modeling is well-established in telephony; is very hard in data nets!

 Scalability and Ports

- Scalability measure is usually a (cost) function on the number of input and output ports
- so, \( n^3 \) is less scalable than \( n^2 \) cost
- Ports:
  - maintain VC or forwarding tables
  - provide buffering
  - electrical/photonic interface with "real world"

Switch Buffering

- where to hold packets during contention
  [note: can never fully escape output port contention]
- Common approaches:
  - input buffering (at input ports) [not popular]
  - output buffering (at output ports)
  - internal buffering (inside switching fabric)

Head of Line Blocking

- most buffering uses FIFOs
- if contention on output port for head of input queue packet, all others must wait behind (the head of the line)
- assuming uniform output port distribution, can reduce overall switch throughput to 59% of theoretical maximum!

Crossbar Switches

- every input port has a connection to every output port (\( n \) inputs to each of \( m \) outputs, will focus on common \( n \times n \) case)
- output port circuits must:
  - recognize packets destined for this output
  - deal with output port contention
- complexity grows at least as fast as \( n^2 \) [and even that is hard to achieve]

The Knockout Switch

- full crossbar requires each output port to handle up to \( n \) input packets
- \( n \) simultaneous inputs for same output is unlikely, especially in large switch
- instead implement port to accept \( l < n \) packets at the same time
- hard issue: what value of \( l \) to use?
Knockout Output Ports

- Components
  - packet filters (recognize packets destined for this output port)
  - concentrator (selects packets, “knocks out” others)
  - a queue with capacity \( \log n \) packets

Knockout Concentrator

- want some fairness: no single input should have its packets always “knocked out”
- essentially play a “knock out” tennis tournament with each “game” of 2 players (packets) chosen randomly
- overall winner is easy (just play \( \log n \) rounds and keep winner)

Determining \( n \) winners

- create \( n \) “sections”, each of which determines a winner
- winner of 1st section is standard knockout tourney winner
- all losers go over and compete for 2nd, losers there go for 3rd, etc...

Knockout Output Buffer

- must accept up to \( n \) packets in 1 cycle and send them ASAP (1 per cycle)
- rather than implement a FIFO that accepts data at a rate \( \log n \) times its drain rate, instead use an array of FIFOs w/shifter
- use round-robin among FIFOs

Evaluation

- \( n \) can remain fixed even for large \( n \)
- concentrator complexity scales with \( n \times \log n \) (so, linear in \( n \) given fixed \( \log n \))
- number of packet filters is just \( n \)
- so, complexity of output port is \( n \), and there are \( n \) of them \( \rightarrow \) order \( n^2 \) (not great, not awful)
- biggest weakness is traffic assumption

Self-Routing Fabrics

- the switch fabric decides where to send packets, based on a small header
- Banyan networks
  - can construct out of small 2x2 switching elements which look at 1 bit in header (e.g., up for 1, down for 0)
  - no collisions if packets presented in ascending order
**An 8x8 Banyan**

- Clever arrangement avoids collisions given ordered destinations
- Begins with “perfect shuffle” pattern
- Banyan with $n$ inputs needs: $\log_2 n$ stages, $n/2$ elements per stage
- Overall complexity is $n \log_2 n$

**Banyan Properties**

**Batcher Networks**

- So, Banyan is nifty, but needs ascending order to be non-blocking... so precede Banyan with a sorter (Batcher)
- Batcher Network
  - Provides sorting
  - May be combined to form Batcher-Banyan fabric
  - Non-blocking provided destinations are unique [no output port contention]

**An 8x8 Batcher**

- Each 2x2 element does complete comparison of routing header (not just one bit)
- Switch elements route higher tags “up” or “down” [random if equal]
- Provides a hardware merge-sort

**Observations on Batcher**

- Provides sorting
- May be combined to form Batcher-Banyan fabric
- Non-blocking provided destinations are unique [no output port contention]
**Batcher Complexity**
- $\log n (1+\log n) / 2$ stages
- $n/2$ switching elements per stage
- Total complexity on order $n \cdot \log n \cdot \log n$ (better than $n^2$)
- See [Batcher 1968] for development of the complexity based on bitonic sort.

**Output Contention**
- Batcher-Banyan solves internal contention if unique outputs
- This is a severe restriction...many approaches exist to address it.
- Sunshine Switch (see text) is one example.

**Sunshine Switch**
- Multiple Banyans mean output port logic must be able to handle 1 pkt per Banyan
- When > 1 packets to route, they are recirculated through the delay buffer
- Trap picks which exit, which recirculate
- Selector picks which Banyans to use