IV. PACKET SWITCH ARCHITECTURES

(a) General Concept

- as packet arrives at switch, destination (and possibly source) field in packet header is used as index into routing tables specifying next switch in path, i.e., which outgoing link from switch

- packet can then be switched from incoming port/link to appropriate outgoing port/link

General diagram of packet switch:

- port mapper used to read destination address or virtual circuit identifier and uses routing table to determine output

- typically packets can be switched either synchronously or asynchronously:

    - synchronous switching uses fixed size packets
        → packets at all inputs switched at same time
    - asynchronous switching can use variable size packets
        → packets switched from an input not synchronized with other input ports

(b) Generations of Packet Switches

1st Generation - Shared Memory

- CPU stores received packets in input queues in main memory

- packets routed to output queue by CPU according to packet header

- packets can be switched synchronously or asynchronously

- low cost but slow because of bottlenecks such as processor and memory access speeds

- typical for low cost Internet routers

2nd Generation - Shared Medium

- processors contend for shared medium → bottleneck

- line cards responsible for routing packets from input queue to appropriate output queue on outgoing line card

- central processor updates routing tables in front end processor (FEP) as appropriate

  → if route not found by FEP, packet can be forwarded to central processor which can make decision on routing tables and subsequently updating routing table of FEP

- packets can be switched synchronously and asynchronously

- typical for high end routers and low end ATM switches

3rd Generation - Switch Fabrics

- control processor can contain

    (1) functionality to establish connection through switch fabric

    (2) arbiter which is used to determine if packet can access required path in switch fabric and specified output port

- when packets at input port, destination port ID (sometimes called "routing tag") added to header by IPC

- typically at IPC packets queued until it is possible to switch it to OPC

- switch fabric routes packet to correct output port where it is typically buffered in output queue until link is available

- must have queues at IPC or IPC or both

- OPC would typically contain a scheduler in order to provide scheduling for different classes of service (eg. guaranteed or best-effort)

- bottlenecks can occur if packets not buffered quickly enough at output queue or if packets not dequeued rapidly enough from input queue by arbiter

- typical for ATM switches with fixed size packets and high end IP core routers
  → typically synchronous switching

(c) Blocking in Packet Switches

- in circuit switching (eg. telephone network), if blocking occurs such that path from input to output is not available at call setup, call is rejected

- in packet switching, if a path cannot be found for packet, it is buffered or dropped but session remains active

- "switching cycle" → time required to switch packets from input to output

3 types of blocking:

(1) Internal Blocking

- two packets contend for same link at same time inside the switch fabric

(2) Output Blocking

- two packets contend for same output of switch at same time

→ to overcome packets can be queued at output or queued at input with arbiter used to ensure no packets headed for same output in same switching cycle

(3) Head-of-Line (HOL) Blocking

- assume no output queuing:

→ when 2 input queues have HOL packets contending for same output, packets in blocked packet queue are delayed ⇒ hinders switch throughput

- also occurs if switch has output queues but potential internal blocking keeps a packet queued at input or when output queue full and packet kept at input queue

- which form of blocking might occur depends on nature of buffering and switching architecture

(d) Switch Fabrics

(i) Crossbar

- conceptual matrix of wires

- connections established under control of contral processor

- very complex task for control processor to control switching schedule (i.e., when connections are made) since this is changing on a per packet basis

- switch is internally non-blocking but if packets destined to same outputs, output blocking will occur (even if there are output queues)

→ could avoid by running switch *N* times faster than input rate or by using buffers at crosspoints

- arbiter for each output controls which packets are served

→ crossbar is expensive, does not scale well, complex to control

(ii) Broadcast

- incoming packet tagged for desired output and broadcast to all outputs

- each output port looks for address match and places matched packets in output queue

- drawbacks:

$\Rightarrow$ cost in address matching logic at output controller

$\rightarrow$ must do comparison at a rate of N times input

eg. for ATM link rate of 155 Mbps $\rightarrow$ one 53 byte cell every 2.73 μs

$\therefore$ comparison every 27.3 ns for 100x100 switch

$\Rightarrow$ large switches require a lot of lines resulting in routing problems in hardware design

Notes: $\rightarrow$ no input queues required for switch

$\rightarrow$ can be used as building blocks (eg. 8×8) to build larger switches

$\rightarrow$ good for multicast connections (i.e., one input to many outputs)

(iii) Self-Routing Multistage Interconnection Networks

- many switch fabric designs use $k \times k$ "switching element" as a building block in MIN

  eg. 2x2 switch element

- switch element examines header of incoming packet and switches packet to either output port based on some rule

  eg. address bit = 0 $\Rightarrow$ upper output
  address bit = 1 $\Rightarrow$ lower output

- if 2 packets arrive simultaneously and are to be switched to same output $\rightarrow$ packet buffered (in buffered SE) or dropped (in unbuffered SE)

- aside: can use $k \times k$ crossbars or $k \times k$ broadcast switches as $k \times k$ switching elements

- *self-routing* $\Rightarrow$ packet header labelled so that bits are used to make decisions at SE (compare to all connections being established by control processor as in crossbar)

- for $N \times N$ MIN switch composed of $k \times k$ SEs, switch has $\lceil \log_k N \rceil$ stages with $\lceil N / k \rceil$ SEs per stage

Banyan Switch Fabric

8×8 Banyan SF

- each packet tagged with binary representation of output

- SE routes packet as 0 $\Rightarrow$ upper, 1 $\Rightarrow$ lower

- first stage routing bit is most significant bit, last stage is least significant bit

- Banyan SF with $N = 2^n$ inputs/outputs $\Rightarrow$ n stages, 2^{n-1} SEs in each stage

$\therefore$ total # SEs = $n \cdot 2^{n-1} = \frac{1}{2} N \log_2 N$

- Banyan is internally blocking

  eg. 000 $\rightarrow$ 011 and 101 $\rightarrow$ 010 collide in top SE of 2nd stage

- to eliminate or mitigate internal blocking:

  (1) could add buffers inside SEs but expensive and still possible to have buffer overflows

  (2) could use 3-phase approach:
  - (A) send request to output (some lost due to blocking)
  - (B) output informs inputs which can be sent
  - (C) packets sent

(3) parallel switches → blocked packets get labelled as misrouted and sent to parallel switch which attempts to route
→ expensive since multiple SFs

(4) can reorder packets at input
→ if packets are sorted by output address, duplicates removed, and passed through shuffle-exchange connection, then Banyan will be internally non-blocking

Shuffle Exchange:

Example:

How to sort?

- consider a list: 3, 4, 6, 3, 7, 6, 4, 6

$\rightarrow$ can recursively divide into 2 lists which are sorted and then merge 2 sorted lists

- only sorting required is comparison of 2 elements but how to merge? in a network?

2×2 comparator (bitonic sorter):

4×4 Batcher merging network

8×8 Batcher merging network

Batcher Sorting Network:

bitonic sorters for $N \times N$ sorting network

=

Batcher-Banyan Network $\rightarrow$ internally non-blocking

- trapped packets can be recirculated to input of sorter causing 1/2 of inputs devoted to recirculation or can send trapped packets to alternate Banyan plane

See figure of 8×8 Batcher-Banyan network

Complexity of Switch Fabrics

Network	# crosspoints
single-stage crossbar	$O(N^2)$
Banyan routing	$O(N \log_2 N)$
Batcher sorting	$O(N(\log_2 N)^2)$
Batcher-Banyan SF	$O(N \log_2 N + N(\log_2 N)^2)$ $= O(N(\log_2 N)^2)$

(e) Buffering in Packet Switches

- can have any combination of input queuing, output queuing, internal queuing, or common input/output buffer

(i) Input Queuing Only

- in switch with only input queue, arbiter is used to determine whether a packet has access to SF and output

- arbiter must run fast enough to make decision on all N inputs in one switching cycle but internal links need only run at input rate

- HOL blocking means that input-queued switch with FCFS input queues can only achieve about 60% utilization (with uniform, random traffic)

- one approach to improve HOL blocking and increase utilization is to allocate N queues (one for each output) for each input port

  $\rightarrow$ "Virtual Output Queuing"

  $\rightarrow$ arbiter must select from all queues so that no 2 packets destined for same output

  $\rightarrow$ desirable to send as many packets in one switching cycle as possible

- one algorithm for determining which packets should be sent is "parallel iterated matching"

1st phase: - inputs request to send to destinations of queued packets

2nd phase: - several matching rounds to determine which packets should go
→ for each output randomly select input which has packet destined to that output and repeat several more times (should be sufficient to find best match, i.e., most packets should go)

3rd phase: - send selected packets

- fairness not guaranteed but randomness ensures it is unlikely that some inputs favoured

Example

(ii) Output Queuing Only

- no HOL blocking as with input queues but it is possible with some architectures (eg. broadcast) that N packets arrive at one output in one switching cycle
  → must store at N times rate of input or packets might be lost→ more expensive that input queuing

- can compromise by allowing $S < N$ packets to be stored at an output per cycle → probability of lost packets very small since traffic usually distributed among outputs
  → "knockout" principle

(iii) Shared Memory

- packets stored in common memory as they arrive, packet header extracted and routed to output

- output port scheduler extracts packet from memory when it is to be transmitted on link

- $N \times N$ switch must read and write N packets in one packet time

eg. 53-byte ATM cell arrives at 155 Mbps to 64x64 switch

What is memory access time required if memory accessed as 4-byte words?

- to read/write 64 cells in 2.735 μs as 4-byte words
  → one word read/written every 3.2 ns

(f) Performance of Networks

- generally very difficult to develop analytical models to represent network performance in response to various traffic patterns

→ instead, typically network performance is examined by simulation

- typical performance metrics (considered under different traffic loads and patterns)

(1) cell loss ratio - 10^{-9} good

(2) cell delay - 100 μsec for entire network

(3) throughput - # cells delivered per unit time

- traffic patterns considered often include:

→ *permutation traffic* - all requested output in a timeslot are distinct (not at all realistic)

→ *uniform random traffic* - each output has equal probability of being requested by a cell at input port (not very realistic)

→ *hotspot traffic* - one or more outputs receive larger fraction of incoming cells, remaining cells uniformly distributed

→ *community of interest* - fraction of cells at certain inputs go to certain outputs, remaining cells at other inputs are uniformly distributed

→ *bursty traffic* - inputs alternate between active and idle periods of random length, cells from one input during active period are destined to same output

→ *traffic mixes* - eg. URT + bursty (most realistic)