IP Router Architecture

OVS Bharadwaj

Outline

- Evolution
- IP Router functionality
- IP Router Architecture
    - Bus based
    - Switch fabric
- Queuing Mechanisms in a Switch fabric
- Crossbar Scheduling

Evolution

- Traditionally routers were implemented in Software.
    - High flexibility but
    - Performance limited by performance of the processor
- Hardware implementation (ASICs)
    - High performance
    - Low flexibility
- Need for both hardware and software for best overall performance

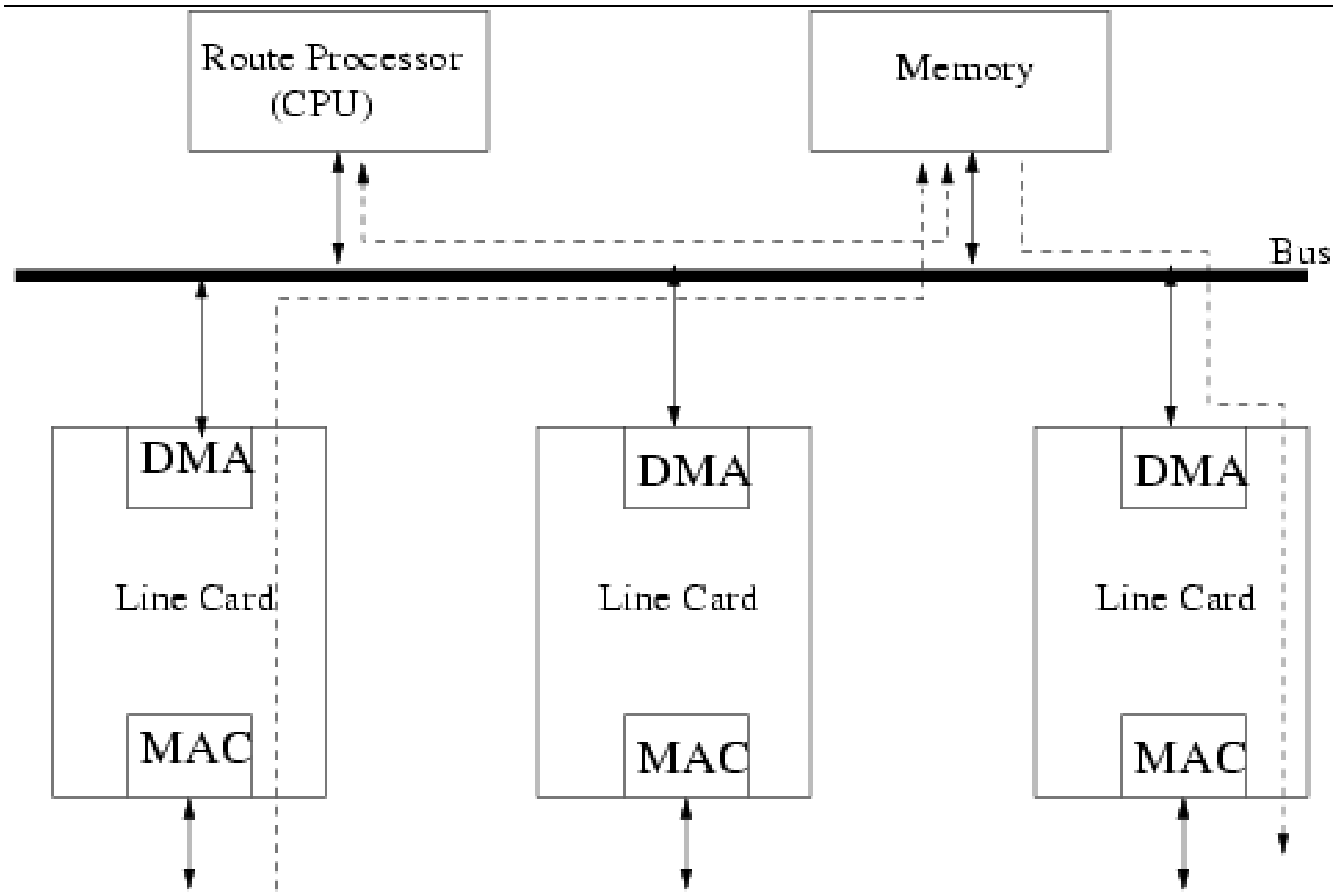
IP Router functionality

- Basic Forwarding: IP packet validation, Packet lifetime control, Checksum recalculation, fragmentation
- Complex forwarding: Packet translation, Traffic prioritization, Packet filtering
- Router-specific tasks: Routing protocols, System configuration and management
- Division into slow and fast path:
  - time critical (or fast path)
  - non-time critical (or slow path)

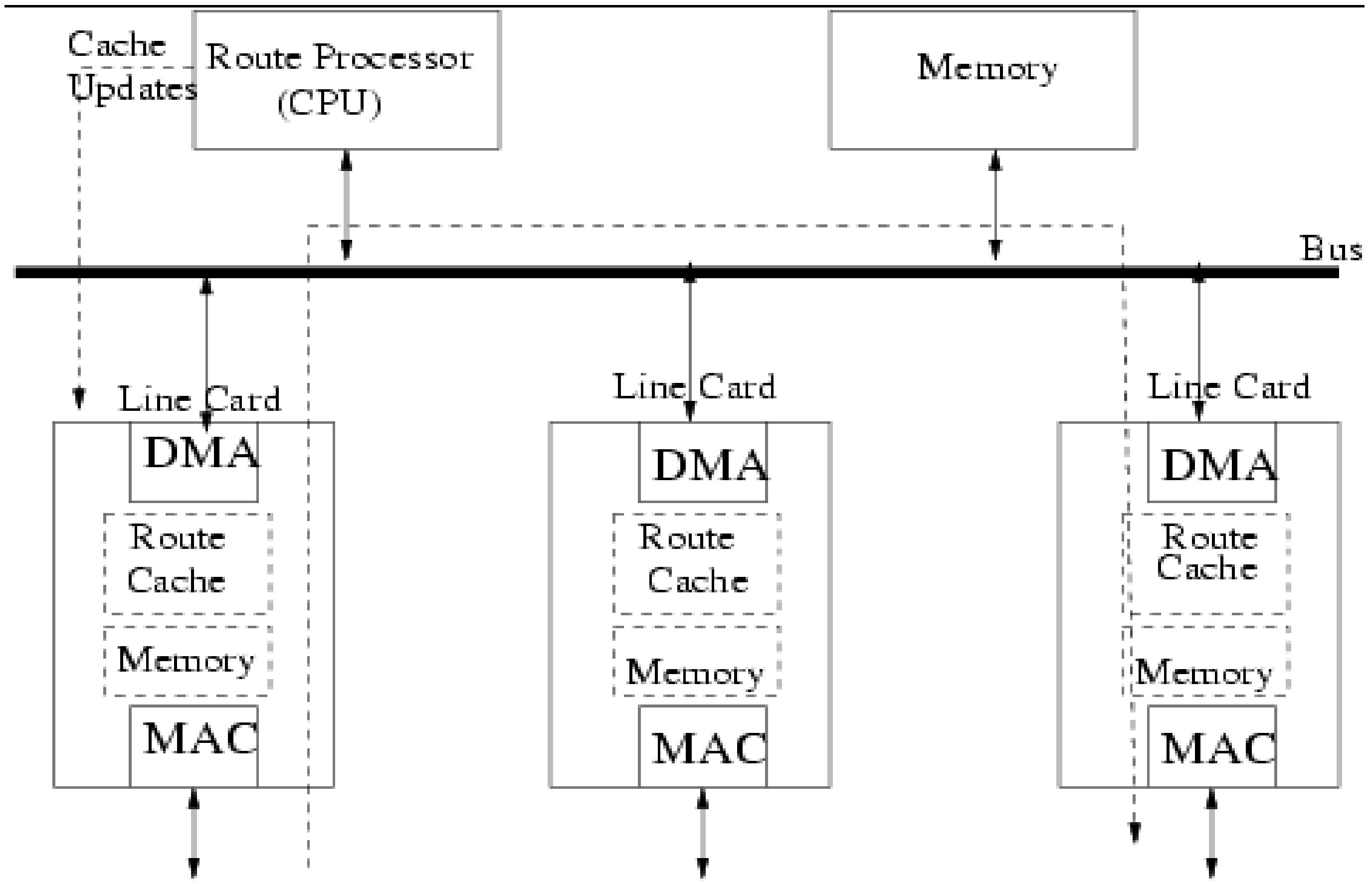
IP Router Architecture

- Network Interfaces
- Forwarding Engines
- General Processing Module
- Interconnection Unit

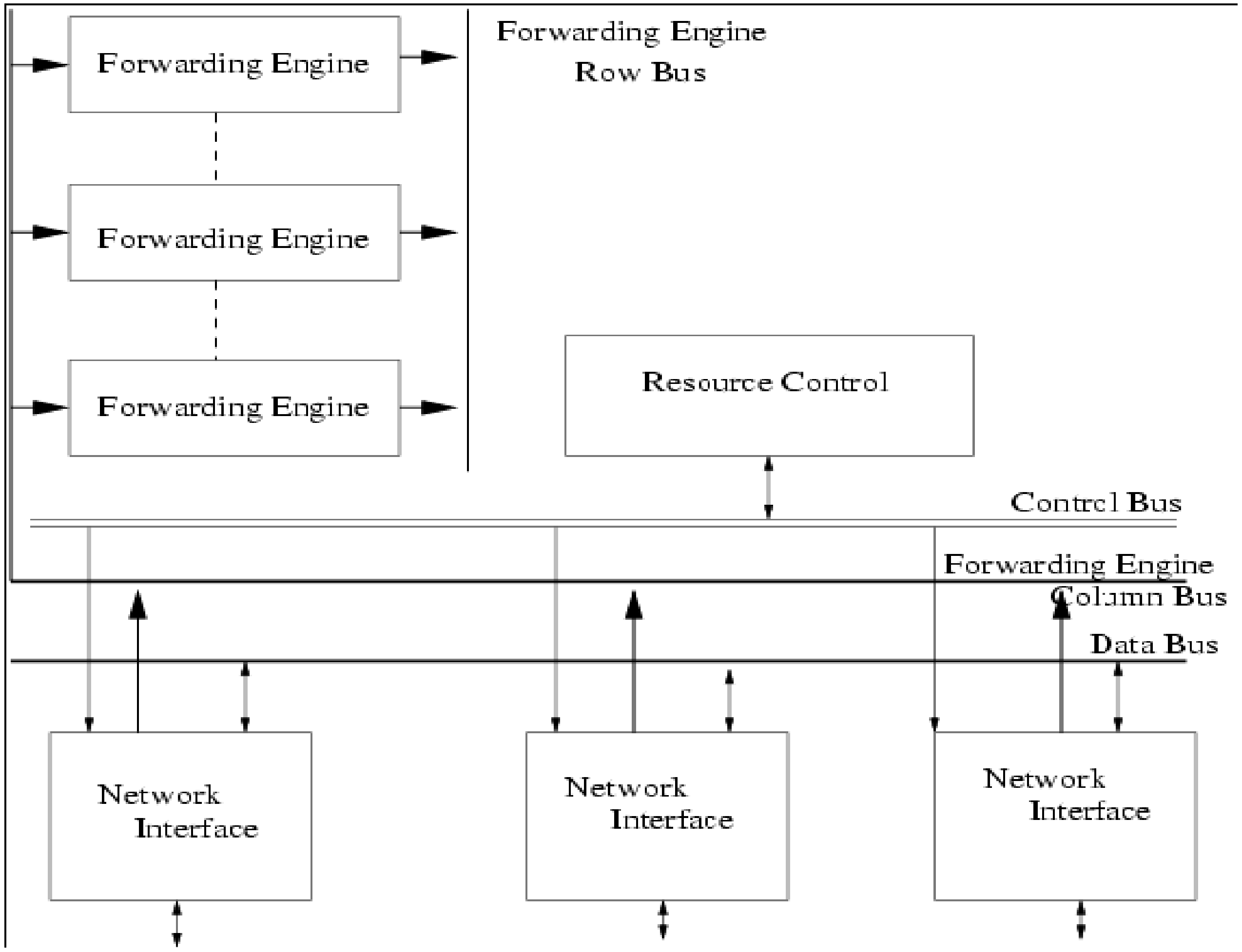
Bus-based Router Architectures with Single processor



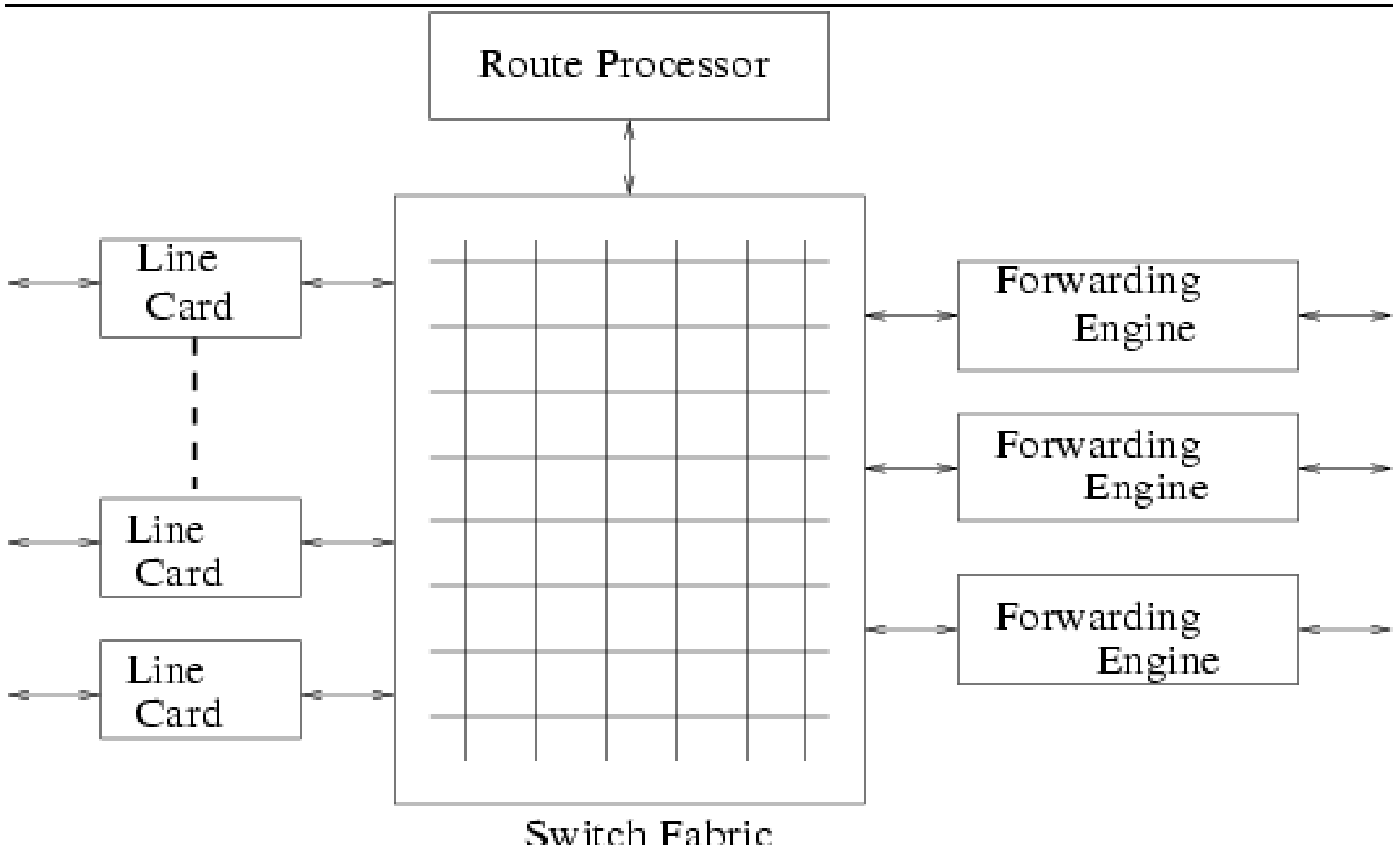
Architectures with Route Caching



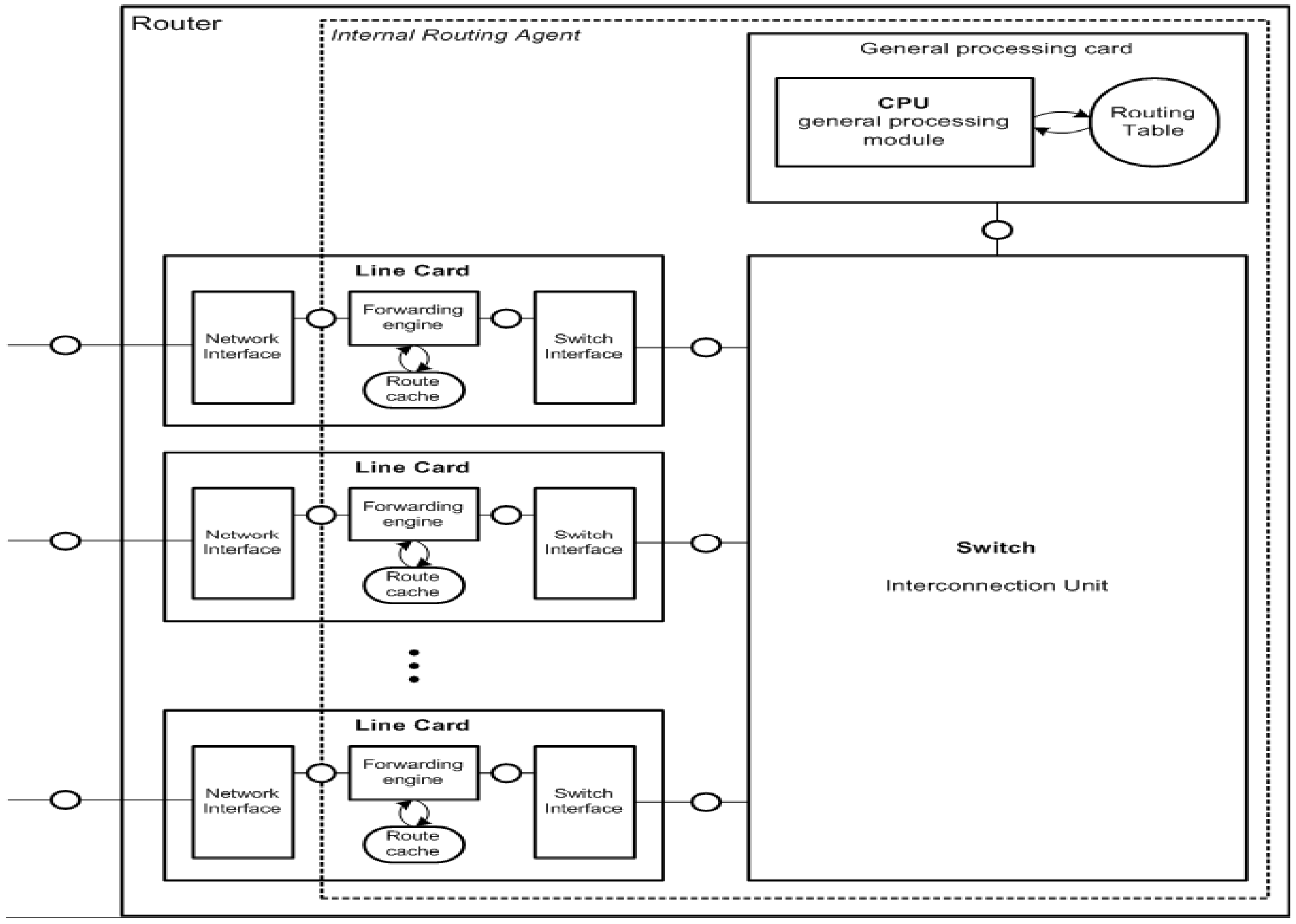
Multiple Parallel Forwarding Engines



Switch fabric



Modern Switch based architecture



Buffering and Queuing

- Need for Queuing
  - More than one packet can may arrive for the same output during the same time slot.

- Three basic types of queuing families
  - Output queuing
  - Input queuing
  - Shared Buffer

Output Queuing

- Filter selects all incoming packets destined for that output and places them in the output buffer
- Output links will never suffer from starvation, when there is at least more than one packet to be sent

Advantages of output queuing

- Multicasting
- Delaying of packets can be controlled
- QoS can be ensured by having multiples queues, one for each prioritization level

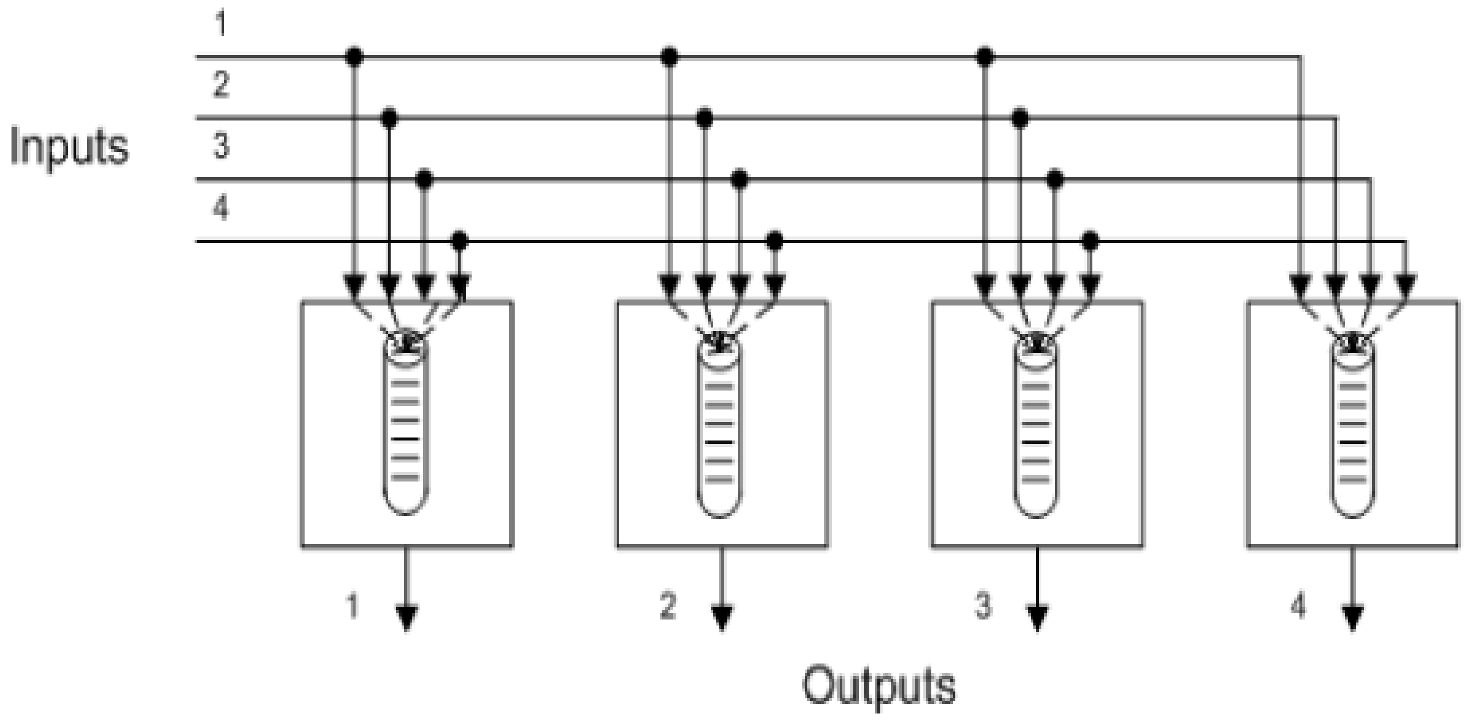
Disadvantage of output queuing

- High speedup (K=N) is required
- Memory must be working at (N+1).S speed
- The internal crossbar must work N times faster than the output links
- Example: N=16 ports, S=2.5 Gbits/sec
    - One line in the crossbar 40 Gbits/sec
    - The entire crossbar capacity needed 640 Gbits/sec
- These requirements are too high for implementation of both crossbars and memories

Shared output buffer

- Large memory is shared by all output links
- Better utilization of memory
- Packets distributed across the memory and only pointers to the packet locations must be stored in the queues
- Same performance under unicast traffic
- Better throughput under multicast traffic
- Large amount of required space and throughput can not be achieved with today’s memory technologies

Cross point or distributed output queuing



Cross point or distributed output queuing

- One queue for each input at each output
- Scheduler selects an appropriate packet from one of the N buffers and passes it to the output link
- No speed up is required
- Memory faces only two operations per cell time
- But distributing the output queues into N.N memories if inefficient
- Can reduce some of the cost by using the *knockout* principle
    - L (<N) queues for each output
    - Drop packets if more than L packets arrive

Output queuing

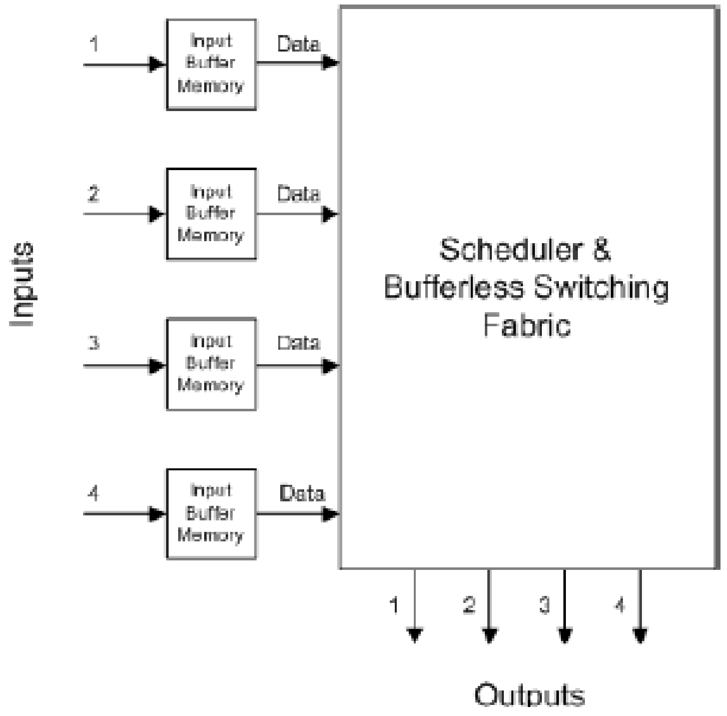
- Assume that packet arrivals on the N input trunks are governed by i.i.d Bernoulli processes.
- P=Probability that a packet will arrive on a particular input in that time slot.
- Each packet has equal probability 1/N of being addressed to any given output, and successive packets are independent.
- As N tends to infinity Pr[A=i] approaches possion probabilities
- Therefore throughput under heavy load can be calculated to be 63.2%

$$\Pr[A=i]=\binom{N}{i}(p/N)^i(1-p/N)^{N-i}$$

$$a_i \triangleq \Pr[A=i]=\frac{p^i e^{-p}}{i!} \qquad i=0, 1, 2, \cdots$$

Input queuing

- Buffer Memory at input ports
- No speed up required, internal switch fabric and memories only have to operate at the line rate S
- HOL blocking – limits throughput to 58%



Virtual input queuing

- Removes HOL blocking problem at the cost of the complexity of the scheduler

- Separate queues for each output port at each input port

Crossbar scheduling

- Problem- To find the configuration of the switch where each active input is connected to all necessary outputs in least time

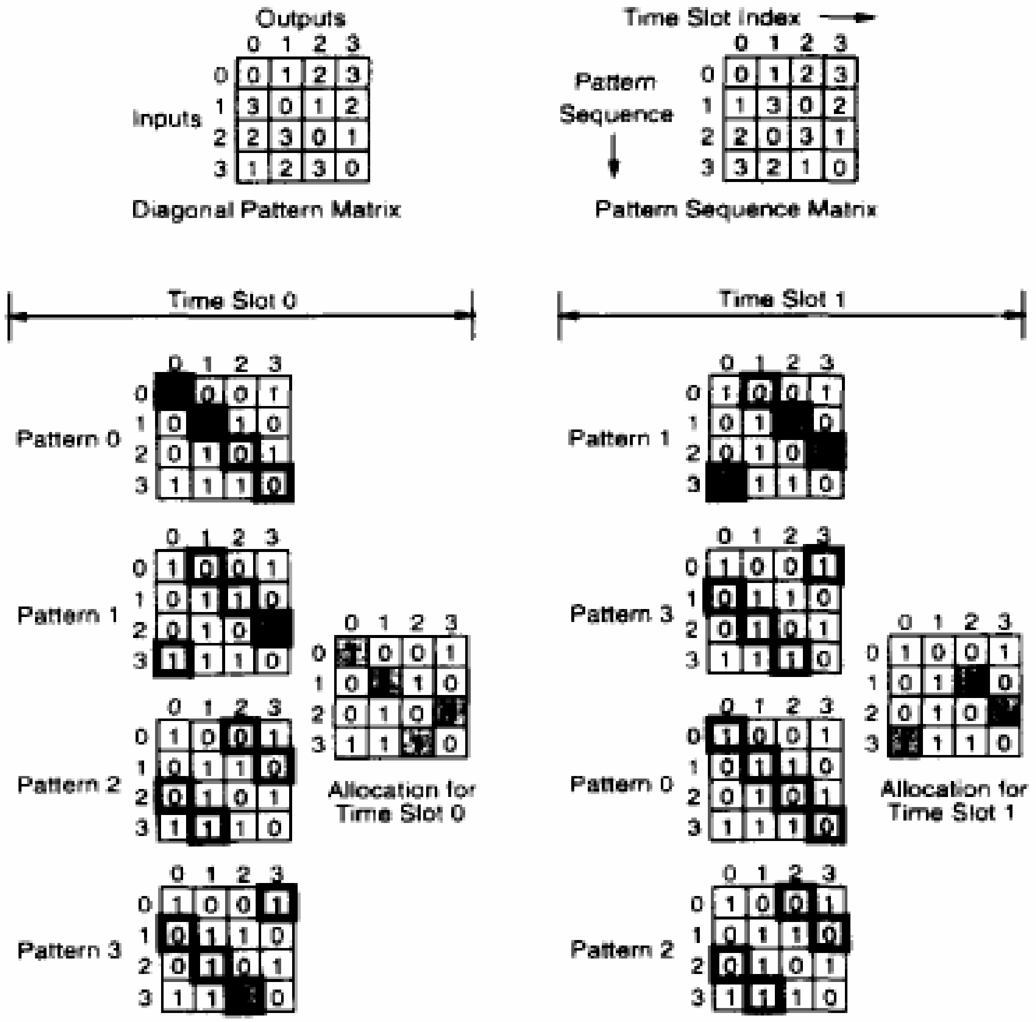
- Desirable properties of scheduling algorithms
  - High throughput
  - Starvation free
  - Fast
  - Simple to implement

2D Round Robin Scheduling

- Request Matrix

- Diagonal Pattern Matrix
  - DM[R, C] = (C – R) mod N.
  - If DM[R, C] = K, then RM[R, C] is covered by diagonal pattern K.
- Pattern Sequence Matrix
  - PM[I, J] = K implies that for time slot index J of a cycle, the I-th diagonal pattern applied is the one numbered K in the diagonal pattern matrix.
- Allocation Matrix

2D Round Robin Scheduling

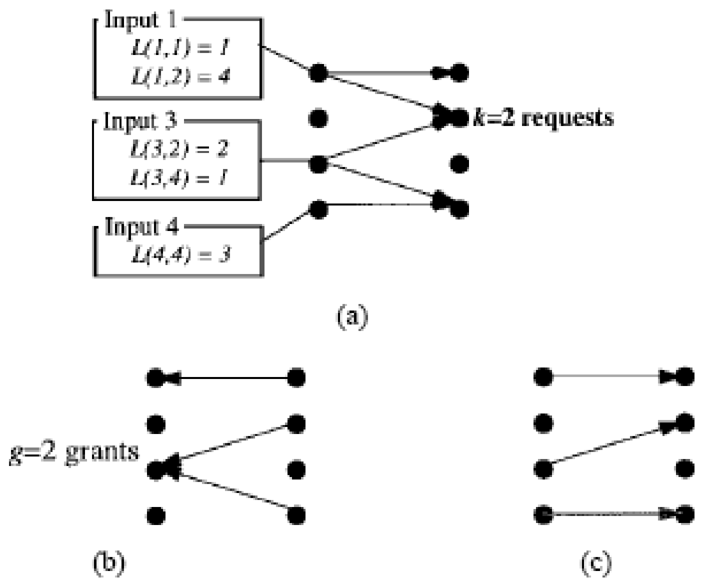


2D Round Robin Scheduling

- Fair : Guarantees that each of the N^2 requests will receive at least one opportunity for service during every cycle of N time slots.

Parallel Iterative Matching -PIM

- Request
- Grant
- Accept

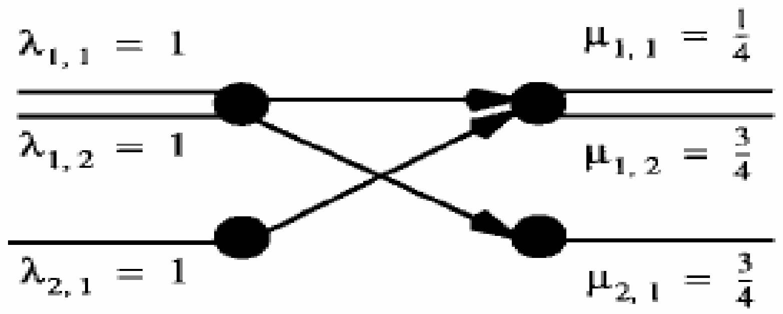


Parallel Iterative Matching -PIM

- Each iteration will match, on average, at least ¾ of the remaining possible connections, and thus, the algorithm will converge to a maximal match, on average, in O(log N) iterations.
- Randomness ensures that each request is eventually served, thus no input VOQ is starved.
- It uses no memory or state. At the beginning of each cell time, the match begins over, independently of the matches that were made in previous cell times.

Parallel Iterative Matching -PIM

- Random arbiters are difficult to implement at high speeds.
- Leads to unfairness under heavy loads.
- For single iteration – throughput is $1-(1/e)\approx 63\%$.

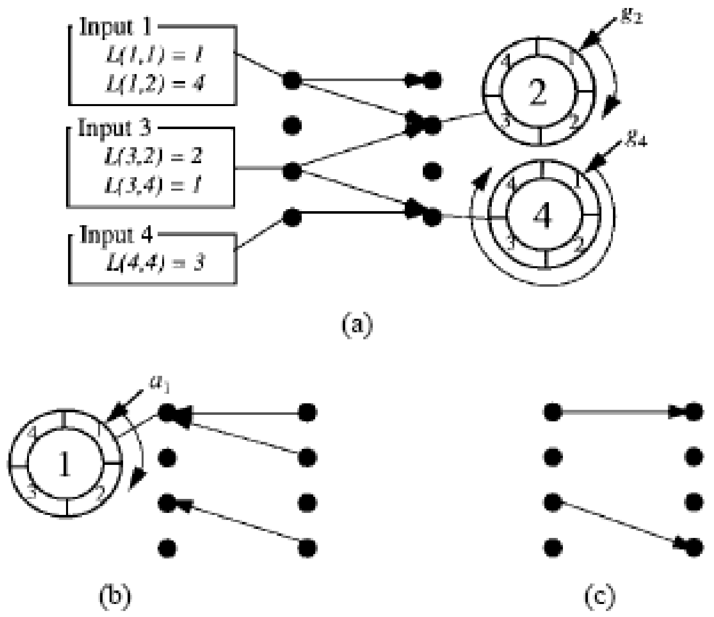


$\lambda_{1,1} = 1$
$\lambda_{1,2} = 1$
$\lambda_{2,1} = 1$
$\mu_{1,1} = \frac{1}{4}$
$\mu_{1,2} = \frac{3}{4}$
$\mu_{2,1} = \frac{3}{4}$

Fig. 3. Example of unfairness for PIM under heavy oversubscribed load with more than one iterations. Because of the random and independent selection by the arbiters, output 1 will grant to each input with probability 1/2, yet input 1 will only accept output 1 a quarter of the time. This leads to different rates at each output.

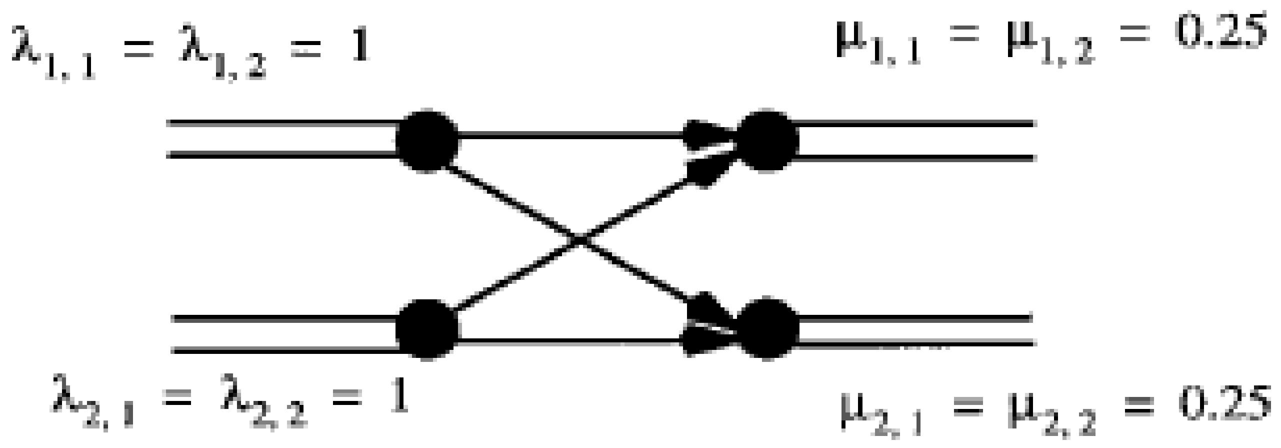
Basic Round Robin Matching algorithm

- RRM potentially overcomes two problems in PIM: *complexity* and *unfairness*.
- If an output receives any requests, it chooses the one that appears next in a fixed, round robin schedule starting from the highest priority element. The pointer is incremented to one location beyond the granted input.



Basic Round Robin Matching algorithm

- Fair
- Synchronization of RR output arbiters leads to a throughput of just 50%



$\lambda_{1,1} = \lambda_{1,2} = 1$
$\mu_{1,1} = \mu_{1,2} = 0.25$
$\lambda_{2,1} = \lambda_{2,2} = 1$
$\mu_{2,1} = \mu_{2,2} = 0.25$

iSLIP

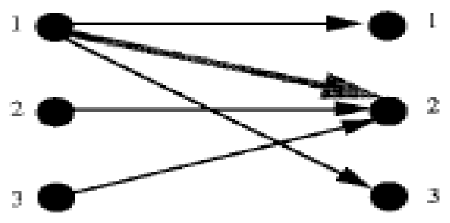
- Removes synchronization of the output arbiters.
- It achieves this by not moving the grant pointers unless the grant is accepted.
- The *Grant* step of RRM is changed to

  The pointer to the highest priority element of the round-robin schedule is incremented to one location beyond the granted input if, and only if, the grant is accepted in Step 3.

Properties of iSLIP

- Lowest priority is given to the most recently made connection.
- No connection is starved.
- Under heavy load, all queues with a common output have the same throughput.

iSLIP algorithm with multiple iterations



Cell 1, Iteration 1: $\begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ $\quad \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} \mathrm{R} \begin{bmatrix} j_1 & j_2 & j_3 \\ - & j_2 & - \\ - & j_2 & - \end{bmatrix} \rightarrow \begin{bmatrix} j_1 \\ j_2 \\ j_3 \end{bmatrix} G \begin{bmatrix} i_1 \\ i_1 \\ i_1 \end{bmatrix} \rightarrow \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} A \begin{bmatrix} j_1 \\ - \\ - \end{bmatrix}$

Cell 1, Iteration 2: $\begin{bmatrix} j_1 \\ j_2 \\ j_3 \end{bmatrix} G \begin{bmatrix} i_1 \\ i_2 \\ - \end{bmatrix} \rightarrow \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} A \begin{bmatrix} j_1 \\ j_2 \\ - \end{bmatrix}$

Cell 2, Iteration 1: $\begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}, \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$ $\quad \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} \mathrm{R} \begin{bmatrix} j_1 & j_2 & j_3 \\ - & j_2 & - \\ - & j_2 & - \end{bmatrix} \rightarrow \begin{bmatrix} j_1 \\ j_2 \\ j_3 \end{bmatrix} G \begin{bmatrix} i_1 \\ i_3 \\ i_1 \end{bmatrix} \rightarrow \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} A \begin{bmatrix} j_3 \\ - \\ j_2 \end{bmatrix}$

Cell 2, Iteration 2: $\begin{bmatrix} j_1 \\ j_2 \\ j_3 \end{bmatrix} G \begin{bmatrix} - \\ i_3 \\ i_1 \end{bmatrix} \rightarrow \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} A \begin{bmatrix} j_3 \\ - \\ j_2 \end{bmatrix}$

iSLIP algorithm with multiple iterations

- How Many Iterations?
  - Ideally N

- It takes $\log_2 N$ iterations for iSLIP to converge

iSLIP algorithm with multiple iterations

- Updating Pointers
  - Starvation is eliminated if the pointers are not updated after the first iteration.
  - In the previous example, output 2 would continue to grant to input 1 with highest priority until it is successful.

References

- The iSLIP Scheduling Algorithm for Input-Queued Switches Nick McKeown, Senior Member, IEEE
- Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queue, Richard O. LaMaire, Member, IEEE, and Dimitrios N. Serpanos, Member, IEEE
- J. Aweya. IP router architectures: An overview, 1999.
- Anatomy of a high performance ip router. Florian Brodersen and Alexander Klimetschek.
- Input Versus Output Queuing on a Space-Division Packet Switch, Mark J Karol and Samuel P Morgan
- An engineering approach to computer networking, S. Keshav

THANK YOU