INPUT BUFFERED SWITCHES

HIGH PERFORMANCE SWITCHES AND ROUTERS

Wiley

H. JONATHAN CHAO and BIN LIU

Instructor: Mansour Rousta Zadeh

INTRODUCTION

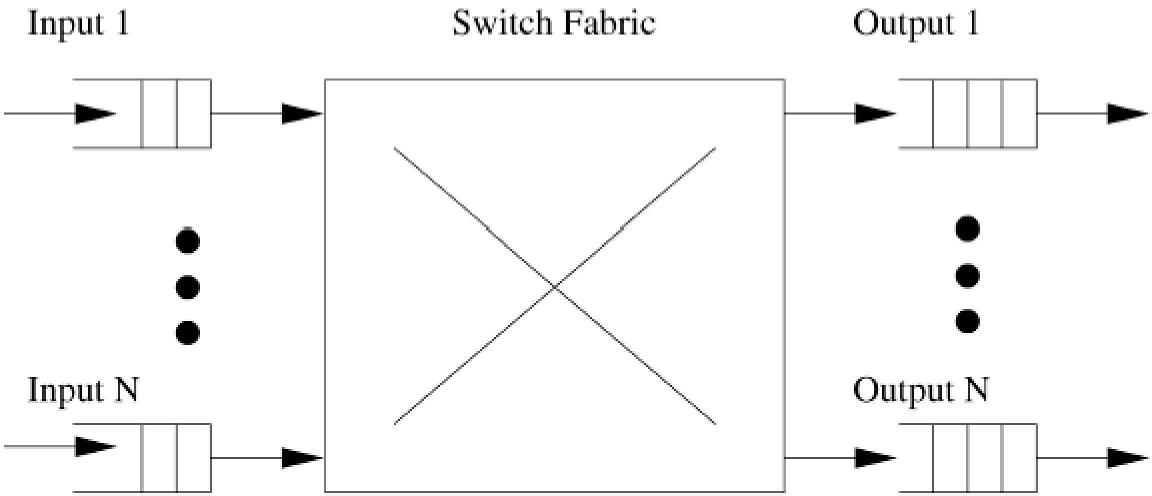
Input-buffered switches

- Two major problems
  - Throughput limitation
    - Because of head-of-line (HOL) blocking
    - Need for faster switch fabric
    - Need for more paths to output ports
  - Arbitration
    - Because of out port contentions
    - Need for fast scheduling mechanisms (this chapter)
- Factors to be considered in scheduling algorithm design
  - Throughput
  - Delay
  - Fairness
  - Implementation cost
  - Scalability
  - Per-flow scheduling

SWITCH MODEL

A simple switch model

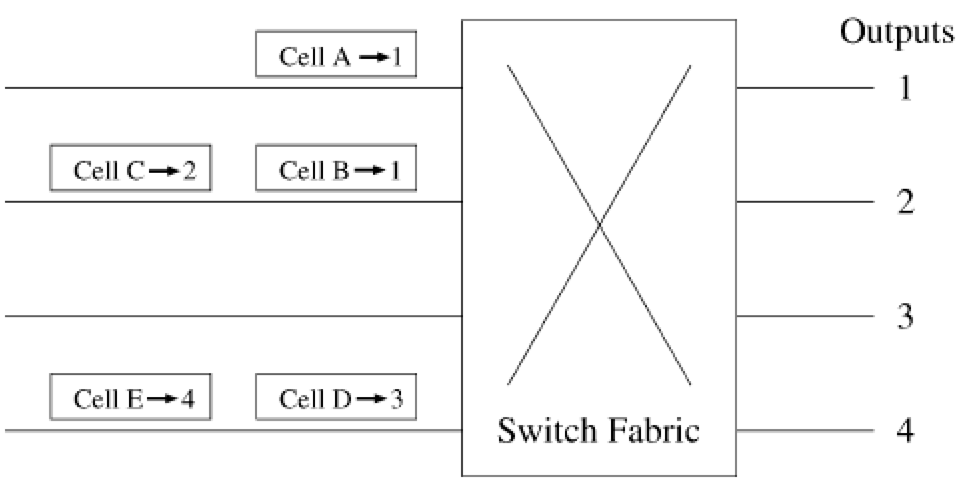
- Inputs and outputs have the same rate
- Switch fabric
  - Has a higher rate
    - This is for performance
    - Output buffer may be needed as a result
  - Is internally conflict-free
  - Has a constant delay



HOL BLOCKING

Head-of-line (HOL) blocking

- A cell whose intended out port is free may be blocked because another cell in front of it is already blocked
- When using FIFO policy
- Example
  - A and B have the same destination port
  - B is blocked in this time slot
  - C has to be blocked until B is cleared ALTHOUGH ITS DESTINED PORT IS FREE in this time slot



TRAFFIC MODELS

Bernoulli arrival process and random traffic

- Cells arrive at inputs slot-by-slot
- Cell arrival probability (offered load)
  - Equal for all inputs
  - Independent from other slots
- FIFO discipline
  - Consider k cells with the same destination out port at HOL
  - Only 1 cell is transferred
  - k-1 cells have to wait until next slot
  - Other cells behind those k-1 cells will be blocked too (HOL blocking)
- Small N -> Markov model
- Large N -> Poisson process
- For N->∞, Throughput->0.586

TRAFFIC MODELS

On-off model and bursty traffic

- Each input
  - Active period
  - Idle period
- Geometrical distribution
- p: probability o
- q: probability o

$$\Pr[\text{An active period} = i \text{ slots}] = p(1-p)^{i-1}, \quad i \geq 1,$$

$$\Pr[\text{An idle period} = j \text{ slots}] = q(1-q)^{j}, \quad j \geq 0.$$

- Mean burst length

$$b = \sum_{i=1}^{\infty} ip(1-p)^{i-1} = \frac{1}{p},$$

- Offered load

$$\rho = \frac{1/p}{1/p + \sum_{j=0}^{\infty} jq(1-q)^{j}} = \frac{q}{q+p-pq}.$$

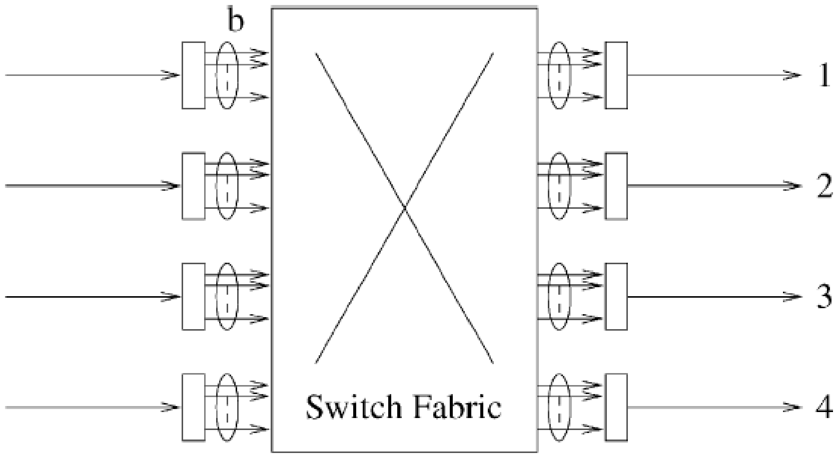
- Large N -> throughput=0.5..0.586 (depending on burstiness)

HOW TO IMPROVE PERFORMANCE

Methods for improving performance

- Increasing internal capacity
  - Multiline (input smoothing)
  - Speedup
  - Parallel switch
- Increasing scheduling efficiency
  - Window-based lookahead selection
  - VOQ-base matching

HOW TO IMPROVE PERFORMANCE



Increasing internal capacity

- Multiline (input smoothing)
  - b lines for each input
  - Nb*Nb switch fabric
  - High implementation cost
  - Out-of-sequence problem
- Speedup
  - Fabric is c times faster than ports
  - Time slot is divided to c cycles
  - Throughput for c=2
    - Bursty traffic: 82.8% … 88.5%
    - Random traffic: 100%
- Parallel switch
  - k identical switch planes
    - Individual input buffers
    - Shared output buffers
  - 100% throughput for k=2
  - The same problems as multiline scheme

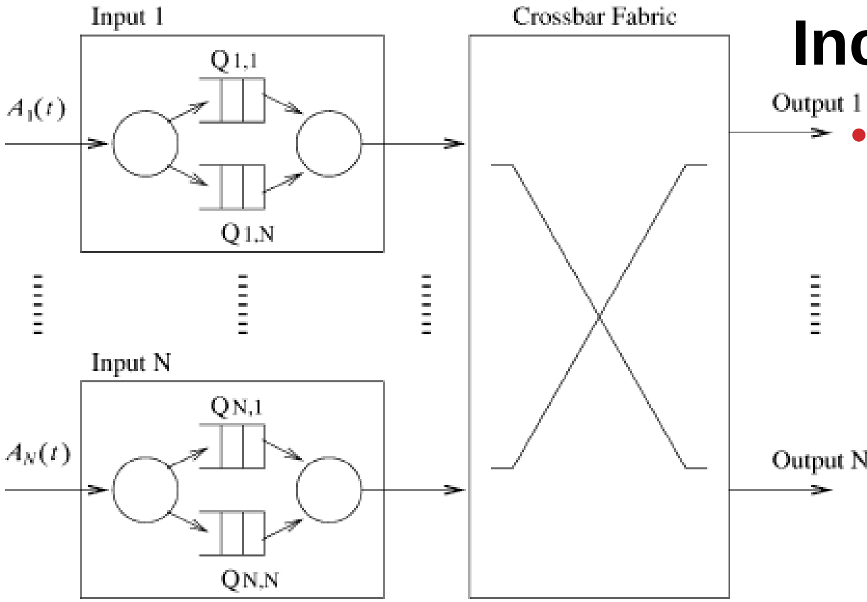
HOW TO IMPROVE ERFORMANCE

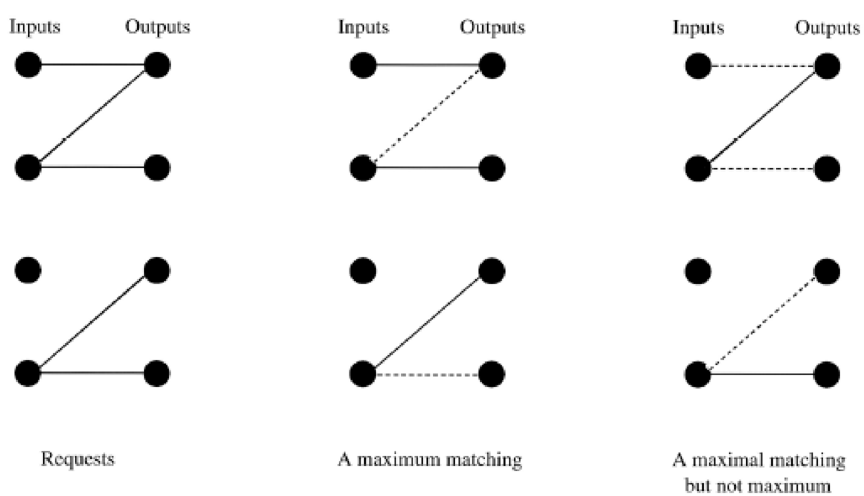
Increasing scheduling efficiency

- Window-based lookahead
  - Relax FIFO restriction
  - w cells in front of queue sequentially contend for access to outputs (w=window size)
  - Only one cell still can be selected at each time slot
  - Maximum throughput as a function of N and w
  - Si
  - A

	Window Size *w*							
N	1	2	3	4	5	6	7	8
2	0.75	0.84	0.89	0.92	0.93	0.94	0.95	0.96
4	0.66	0.76	0.81	0.85	0.87	0.89	0.91	0.92
8	0.62	0.72	0.78	0.82	0.85	0.87	0.88	0.89
16	0.60	0.71	0.77	0.81	0.84	0.86	0.87	0.88
32	0.59	0.70	0.76	0.80	0.83	0.85	0.87	0.88
64	0.59	0.70	0.76	0.80	0.83	0.85	0.86	0.88
128	0.59	0.70	0.76	0.80	0.83	0.85	0.86	0.88

HOW TO IMPROVE PERFORMANCE





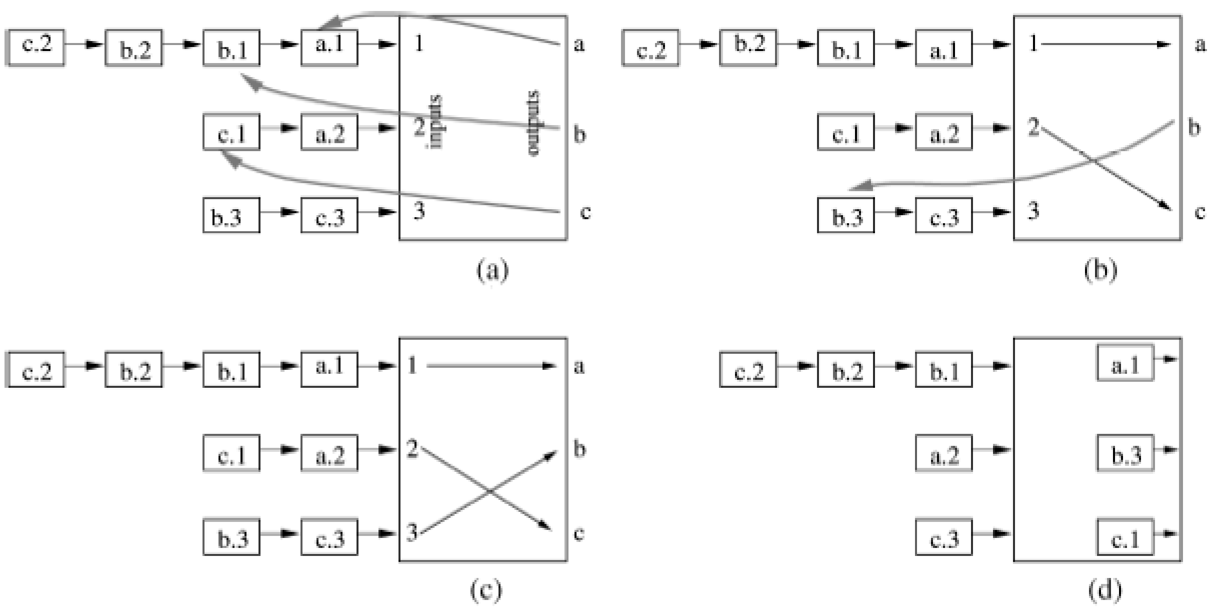
Increasing scheduling efficiency

- VOQ-based matching
  - Each input has a queue per output
    - Virtual output queue (VOQ)
    - $VOQ_{i,j}$ stores cells arriving at input port i and destined for output port j
    - Matching methods
      - Maximum matching: the maximum number ofinputs and outputs are matched
      - Maximal matching: no more matches can be made without modifying the existing matches
      - Stable matching: see next page

HOW TO IMPROVE PERFORMANCE

Stable matching

- A priority list for each input and each output
  - Input priority list: all the cells queued at the input
  - Output priority list: all the cells destined for that output port
  - A matching is stable for a waiting cell c if:
    - c is part of matching
    - A cell in front of c in input priority list is part of matching
    - A cell in front of c in output priority list is part of matching
    - (part of matching = will be transferred during this phase)



SCHEDULING ALGORITHMS

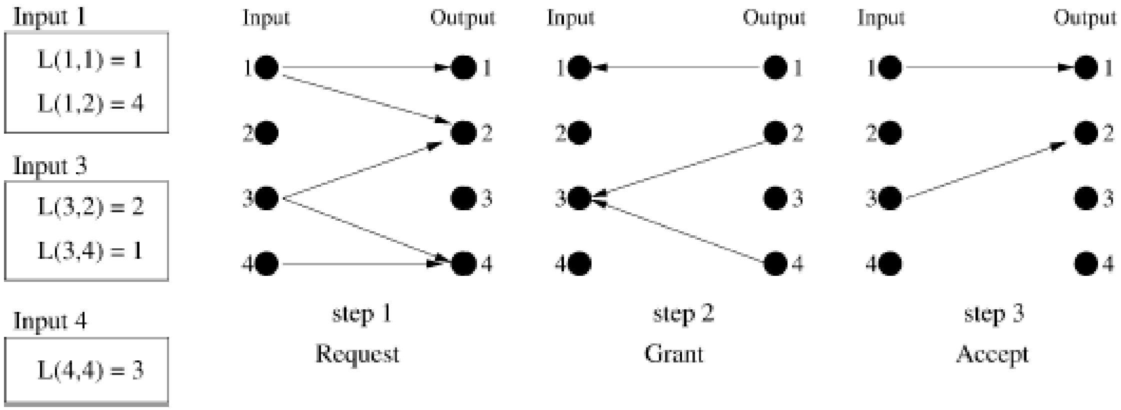
Scheduling algorithms

- Input buffer
  - Parallel iterative matching (PIM)
  - Iterative round robin matching (iRRM)
  - Iterative round robin with SLIP (iSLIP)
  - Dual round robin matching (DRRM)
  - Greedy round robin
- Output buffer emulation
  - Most-urgent cell first (MUCFA)
  - Critical cell first (CCF)
  - Last in highest priority (LIHP)
- Input-output buffer
  - Lowest-output-occupancy cell first

SCHEDULING ALGORITHMS

Parallel iterative matching (PIM)

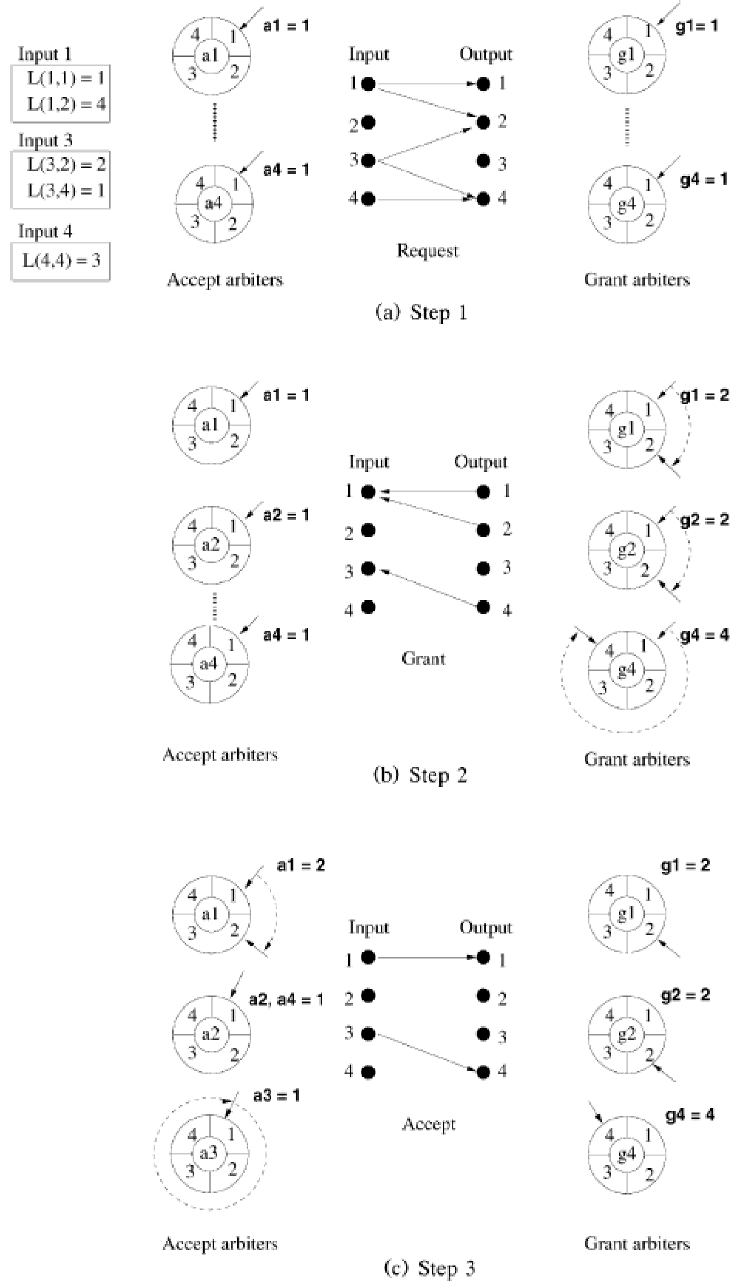
- Random selection
- Each iteration: 3 steps
  - Request: unmatched inputs send their requests to outputs
  - Grant: if an output receives more than one requests, selects one randomly
  - Accept: if an input receives more than one grants, selects one randomly
- 75% match completion in each iteration on average
- Converges at O(logN) iterations
- Throughput under uniform traffic
  - 63% for one iteration
  - 100% for N iterations
- Implementation cost of high speed random function



SCHEDULING ALGORITHMS - IRRM

Iterative round robin matching (iRRM)

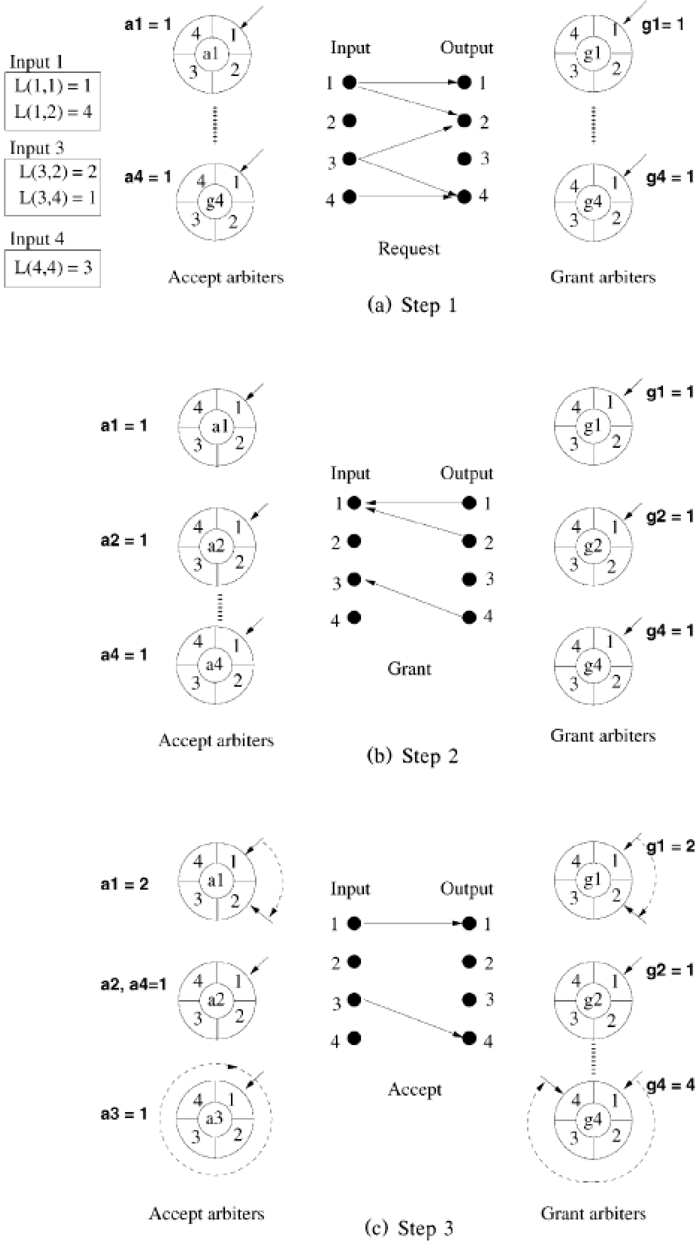
- Similar to PIM, but uses round robin selection instead of random selection
- A pointer to the port having the highest priority for each port
    - Accept pointer a_i
    - Grant pointer g_i
- Algorithm iteration:
    - Inputs send their requests
    - Each output i that receive multiple request, grants the one that g_i to grant the request according to its round robin schedule, and increments g_i
    - Each input i that receive multiple grants, refer to a_i to grant the request according to its round robin schedule, and increments a_i



SCHEDULING ALGORITHMS – ISLIP

Iterative round robin with SLIP (iSLIP)

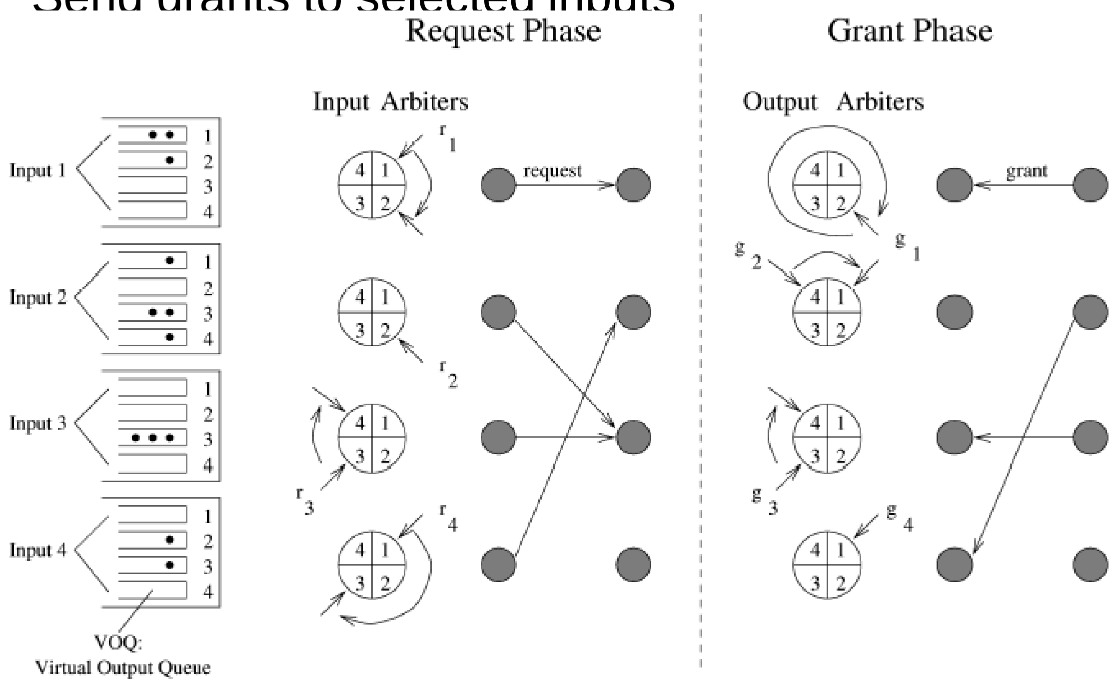
- Similar to iRRM, but g_i is incremented only when the grant is accepted
- No starvation: matched pairs get the lowest priority



SCHEDULING ALGORITHMS - DRRM

Dual round robin matching (DRRM)

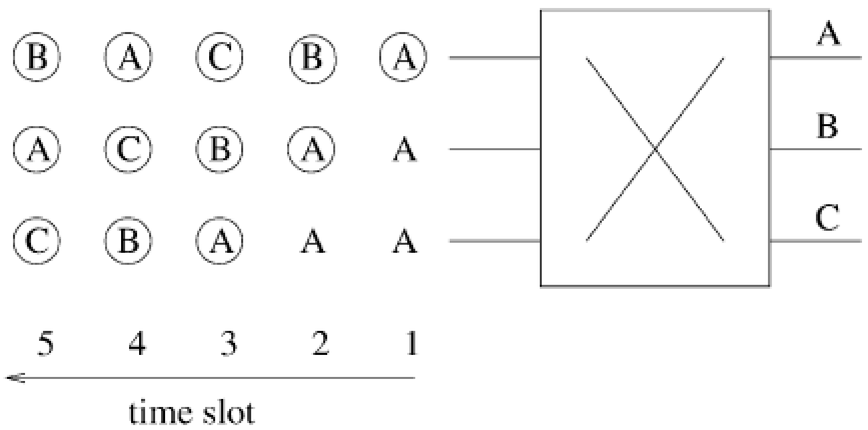
- Similar to iSLIP, but starts round robin at inputs
- Each input sends only one request
- Each iteration
    - Select one of requests at each input
    - Send selected requests to the outputs
    - Select one of requests at each output
    - Send grants to selected inputs



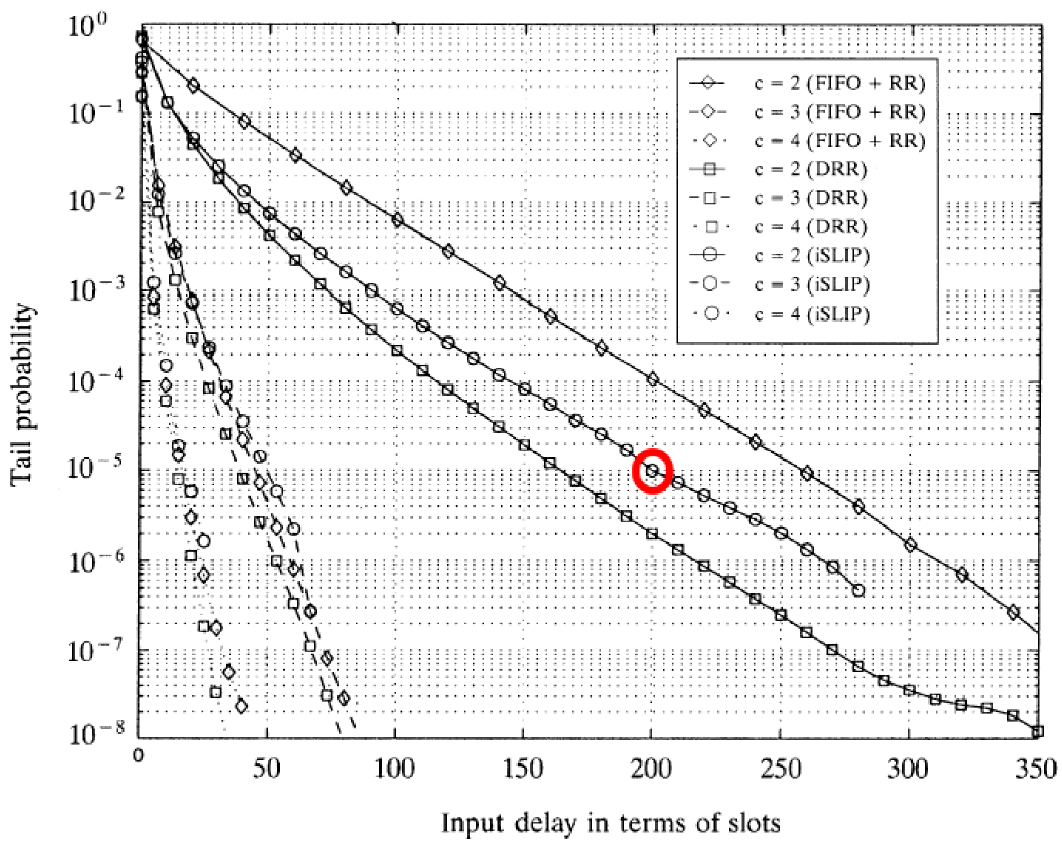
SCHEDULING ALGORITHMS

Dual round robin matching (DRRM)

- Desynchronization effect



- A comparison

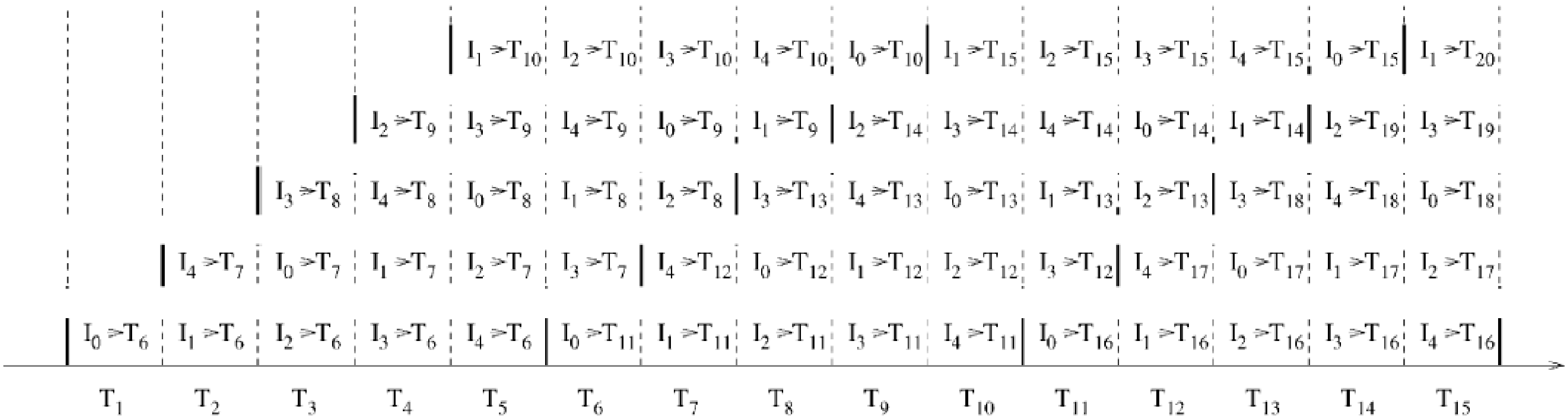


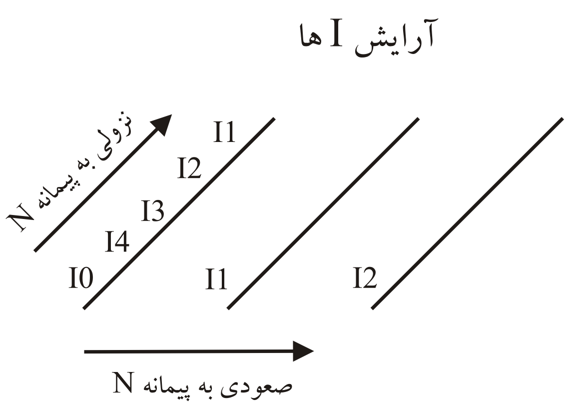
SCHEDULING ALGORITHMS - RRGS

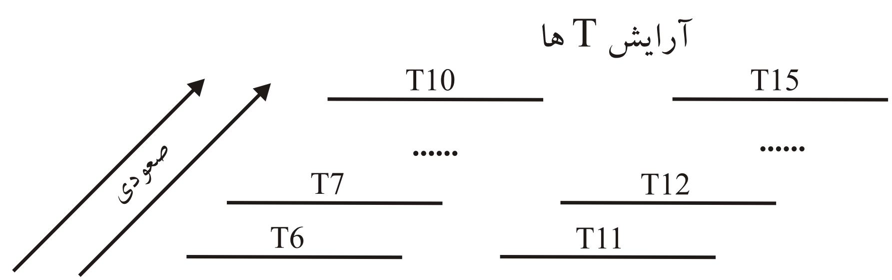
Round robin greedy scheduling (RRGS)

- A Bottleneck in iSLIP and DRRM:
  - Scheduling must be completed within one time slot
  - 64 bytes cells, 40 Gbits/S link -> 12.8 ns for computation!
- Pipelining can help
- RRGS Algorithm
- Nonpipelined version first:
  - *Step 1:* $I_k = \{0, 1, \ldots, N-1\}$ is the set of all inputs; $O_k = \{0, 1, \ldots, N-1\}$ is the set of all outputs. $i = (\text{const} - k) \bmod N$ (such choice of an input that starts a schedule will enable a simple implementation).
  - *Step 2:* If I_k is empty, stop; otherwise, choose the next input in a round-robin fashion according to $i = (i + 1) \bmod N$.
  - *Step 3:* Choose in a round-robin fashion the output j from O_k such that $(i, j) \in C_k$. If there is none, remove i from I_k and go to step 2.
  - *Step 4:* Remove input i from I_k, and output j from O_k. Add (i, j) to S_k. Go to step 2.

SCHEDULING ALGORITHMS - RRGS







SCHEDULING ALGORITHMS

Output-Queuing Emulation:

The major drawback of input queuing is that the queuing delay between inputs and outputs is variable, which makes delay control more difficult.

Question:

Can an input output-buffered switch with a certain speedup behave identically to an output-queued switch?

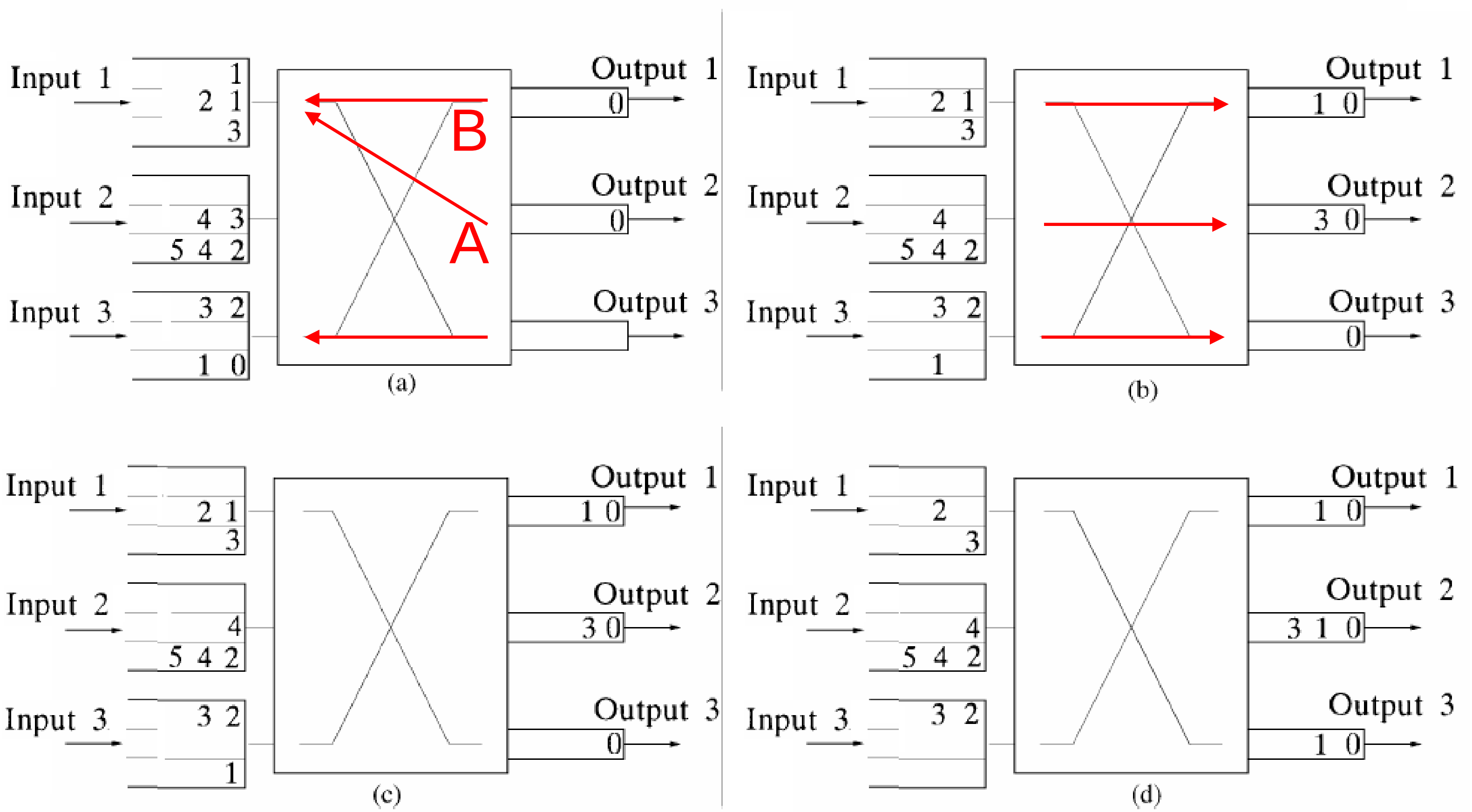
SCHEDULING ALGORITHMS - MUCFA

Most urgent cell first algorithm (MUCFA)

- Output queuing emulation
- TL: Time to leave (not time to live!)
- TL gets the number of cells ahead of this cell when entering
- Most urgent cell: the cell with the smallest TL
- The algorithm
  - Outputs send requests for the most urgent cells to the corresponding inputs
  - If an input gets multiple requests, selects the most urgent cell
  - Outputs which lose contention, request for next most urgent cell
  - These steps are repeated until no more matching is possible

SCHEDULING ALGORITHMS - MUCFA

An example



SCHEDULING ALGORITHMS – PRIORITY LISTS

Priority list base category of scheduling algorithms

- Input queues: Not FIFO
- Push-in queue
    - Insertion
        - According to a predefined priority, the cell goes somewhere in queue
        - Order of cells is unchanged after insertion
    - Removing
        - According to a predefined priority (push-in arbitrary out or PIAO)
        - From head of queue (push-in first out or PIFO)

SCHEDULING ALGORITHMS – PRIORITY LISTS

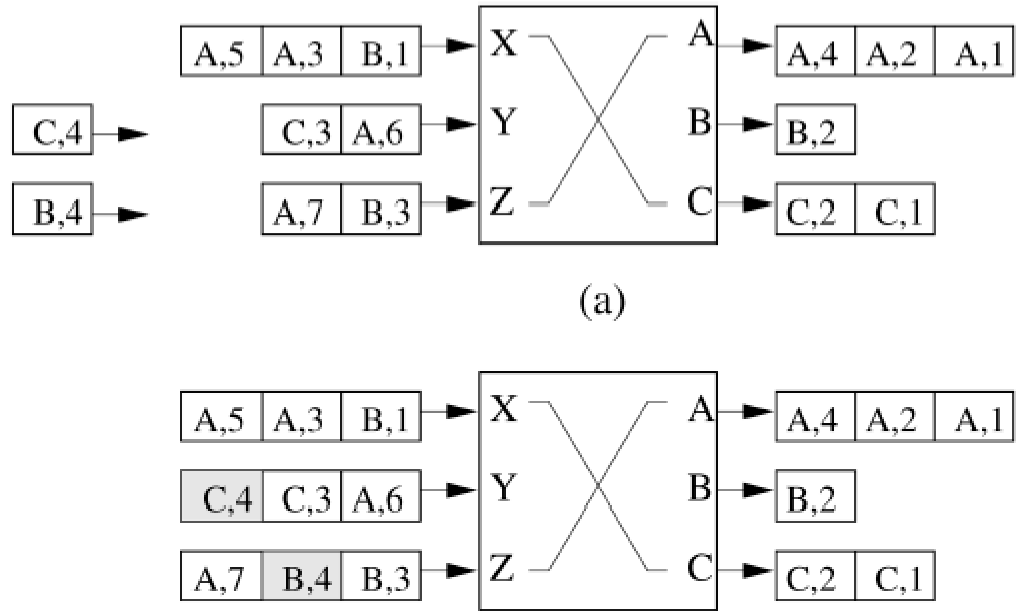
Some definitions

- Time to leave
    - TL(c)
    - The time slot in which cell c leaves the switch
- Output cushion
    - OC(c)
    - The number of cells waiting in output buffer at output port of cell c, having lower TL than c
- Input thread
    - IT(c)
    - The number of cells ahead of cell c in its input priority list
- Slackness
    - L(c) = OC(c) – IT(c)

SCHEDULING ALGORITHMS – PRIORITY LISTS

Critical cell first (CCF)

- Input queues: PIFO
- Position of insertion:
  - As far from the head as possible so that the slackness is positive



Last in, highest priority (LIHP)

- Position of insertion:
  - In front of the queue
  - IT(c) = 0

SCHEDULING ALGORITHMS – LOOFA

Lowest output occupancy cell first algorithm (LOOFA)

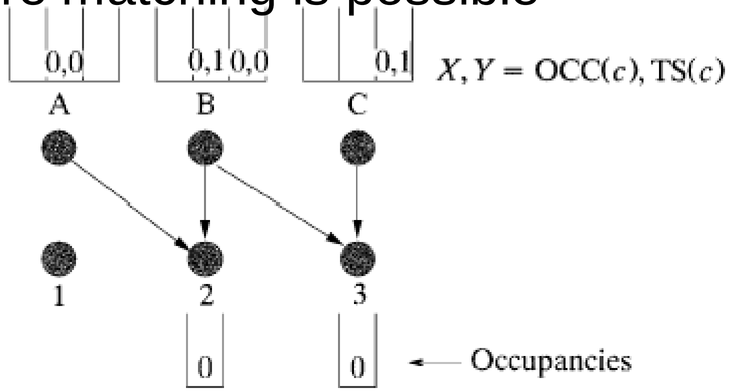
- 100% throughput
- Speedup of 2
- Bounded cell delay
- Two versions
  - Greedy
  - Best first
- Parameters associated with a cell c
  - Output occupancy: OCC(c)
    - The number of cells in output queue of destination port of c
  - Timestamp: TS(c)
    - Age of the cell c

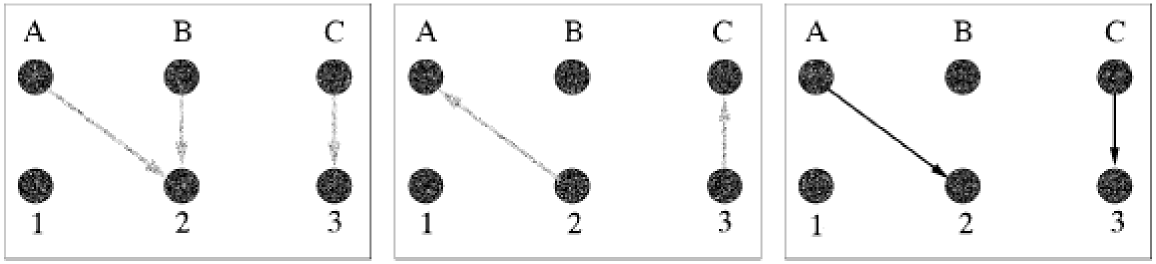
SCHEDULING ALGORITHMS – LOOFA

Greedy version of algorithm

- Initially, all inputs and outputs are unmatched
- Each unmatched input sends its request to the output with the lowest occupancy
- If an output gets multiple requests, grants the smallest TS
- Repeat from step 2, until no more matching is possible

An example





(a) (b) (c)

SCHEDULING ALGORITHMS – LOOFA

Best-first version of algorithm

- Initially, all inputs and outputs are unmatched
- Among unmatched outputs, the one having the smallest occupancy is selected. All inputs having a cell for it, send their request.
- The output, grants the request having the smallest timestamp
- Repeat from step 2, until no more matching is possible, or N iterations are completed