

Stable Round-Robin Scheduling Algorithms for High-Performance Input Queued Switches

Jing Liu, Mounir Hamdi,
Hung Chun Kit, and Chi Ying Tsui

Hong Kong University of Science
and Technology

1

Outline

- Introduction
- Randomized Algorithms
- Fully Desynchronized Round-Robin Algorithms
— SRR
- Derandomized RDSRR
- Simulation Results
- Hardware Implementation
- Conclusion
- Q&A

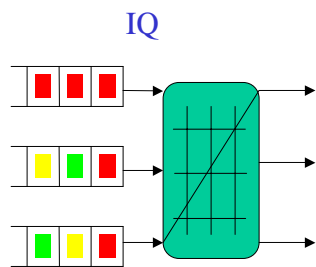
2

Introduction

- Input Queuing vs. Output Queuing
- Input Queuing Strategies
- Scheduling Problem
- Scheduling Algorithms

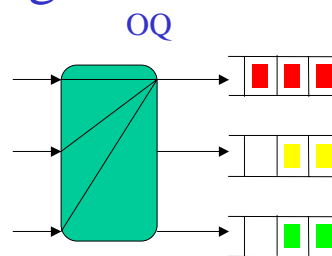
3

Input Queuing vs. Output Queuing



Usually a non-blocking switch fabric
(e.g. crossbar)

- Memory BW: $2R$ (R is link rate)
- HOL blocking limits throughput to 58.6% (But can be eliminated by different IQ strategies)



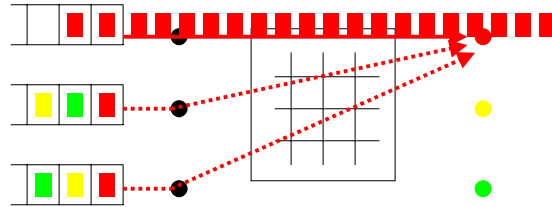
Usually a fast bus

- Memory BW: $(N+1)R$
- Provide guaranteed QoS
- Achieve 100% throughput

4

Input Queuing Strategies (1)

- Single FIFO and Head-of-Line blocking

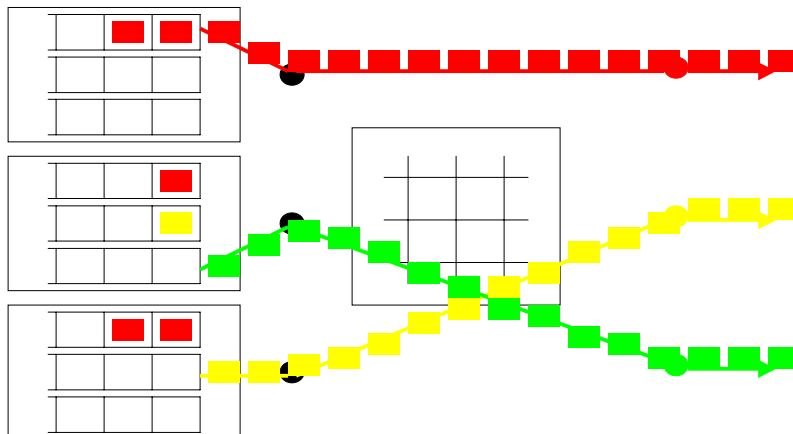


- A maximum throughput of 58.6% with a crossbar fabric

5

Input Queuing Strategies (2)

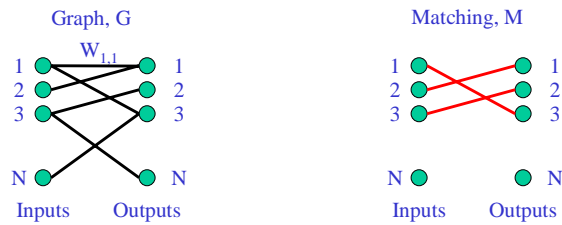
- Virtual Output Queueing (100% throughput)



6

Scheduling Problem

- The matching problem of a bipartite graph



7

Scheduling Algorithms

- Maximum weight/size matching algorithms
 - Stable under any admissible traffic, i.e. 100% throughput
 - Too complex to be of practical value
- Maximal size matching algorithms
 - Stable under uniform traffic, but less than 100% throughput under non-uniform traffic
 - Simple to implement in hardware

8

Outline

- Introduction
- ➔ Randomized Algorithms
- Fully Desynchronized Round-Robin Algorithms — SRR
- Derandomized RDSRR
- Simulation Results
- Hardware Implementation
- Conclusion
- Q&A

9

Randomized Algorithms

- Algo1 — A randomized scheme with memory (proposed by Tassiulus)
- Hamiltonian walk on the set of all matchings
- Algo2 — A derandomization of Algo1 (proposed by Paolo Giaccone)

10

Algo1 — A randomized scheme with memory

- Specification
 - (1) Let $S(t)$ be the schedule used at time t .
 - (2) At time $t + 1$, choose a matching $R(t + 1)$ uniformly at random from the set of all $N!$ possible matchings.
 - (3) Let $S(t + 1) = \arg \max_{S \in \{S(t), R(t+1)\}} \langle S, Q(t + 1) \rangle$
- Lemma 1 (Tassiulas) : Algo1 is stable under any Bernoulli. i.i.d. admissible input.

11

Hamiltonian walk on the set of all matchings

- A graph with $N!$ nodes and all possible edges between these nodes
- Each node denotes a distinct matching
- A Hamiltonian walk — visit each of the $N!$ nodes exactly once during time $t = 1, \dots, N!$. Extend $t > N!$ by defining $Z(t) = Z(t \bmod N!)$

12

Algo2 — A derandomization of Algo1

- Specification
 - (1) Let $S(t)$ be the schedule used at time t .
 - (2) At time $t + 1$, let $R(t + 1) = Z(t + 1)$, the matching visited by the Hamiltonian walk.
 - (3) Let $S(t + 1) = \arg \max_{S \in (S(t), R(t+1))} \langle S, Q(t + 1) \rangle$
- Lemma 2 (Paolo Giaccone) : Algo2 is stable under any Bernoulli. i.i.d. admissible input.

13

Outline

- Introduction
- Randomized Algorithms
- ➔ Fully Desynchronized Round-Robin Algorithms — SRR
- Derandomized RDSRR
- Simulation Results
- Hardware Implementation
- Conclusion
- Q&A

14

Fully Desynchronized Round-Robin Algorithms — SRR

- Basic idea: in the output and/or input arbiter, keep full pointer-desynchronization
- Different variations of SRR: SSRR, DSRR and RDSRR
- RDSRR performs the best

15

Specification of RDSRR

- *Initialization.* The output/input pointers are set to some initial pattern with no duplication among the pointers.
- The 3 steps of a single iteration:
 - Step 1:* Request.
 - Step 2:* Grant. The arbiters search by clockwise and counter-clockwise directions alternatively at each time slot. The pointer is always incremented by one (modulo N).
 - Step 3:* Accept. The pointer is always incremented by one (modulo N).

16

Achievements with RDSRR

- Lower delay than iSLIP and FIRM
- Fairness achieved by clockwise and counter-clockwise rotation scheme
- Simple hardware implementation
- Drawback: not stable under non-uniform traffic

17

Outline

- Introduction
- Randomized Algorithms
- Fully Desynchronized Round-Robin Algorithms
— SRR
- ➡ Derandomized RDSRR
- Simulation Results
- Hardware Implementation
- Conclusion
- Q&A

18

Derandomized RDSRR

- Specification
- Stability
- An improved version of DRDSRR
- Complexity analysis

19

Specification of DRDSRR

- *Initialization.* The output/input pointers are set to some initial pattern with no duplication among the pointers.
- *Step 1:* $S(t-1)$: the schedule used at previous time slot. $R(t) = Z(t)$, the matching visited by HW.
- *Step 2:* Do as usual RDSRR scheduler, resulting in a matching $M'(t)$ (unmatched inputs and outputs are matched to each other randomly).
- *Step 3:* Let $M(t) = \arg \max_{S \in \{S(t-1), R(t), M'(t)\}} \langle S, Q(t) \rangle$

20

Stability of DRDSRR

- DRDSRR uses the Hamiltonian walk with memory
- By Lemma 2, DRDSRR is stable under any Bernoulli. i.i.d. admissible traffic.

21

An improved version of DRDSRR — DRDSRR(v2)

- Make $M(t)$ a maximal matching
- Augment the matchings between k unmatched inputs and outputs randomly
- Time complexity is at most $O(k^2)$

22

Complexity analysis

- There is an algorithm for implementing the Hamiltonian matching with only $O(1)$ time and $O(1)$ space.
- Compute the weight of a matching: $O(\log N)$
- Time complexity is lower than iLQF and iLPF.

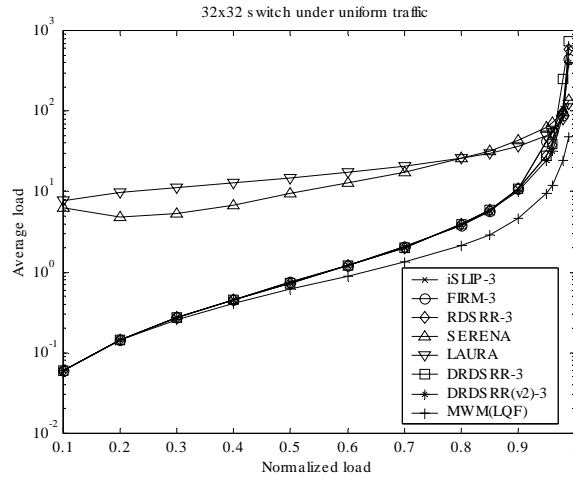
23

Outline

- Introduction
- Randomized Algorithms
- Fully Desynchronized Round-Robin Algorithms
— SRR
- Derandomized RDSRR
- ➡ Simulation Results
- Hardware Implementation
- Conclusion
- Q&A

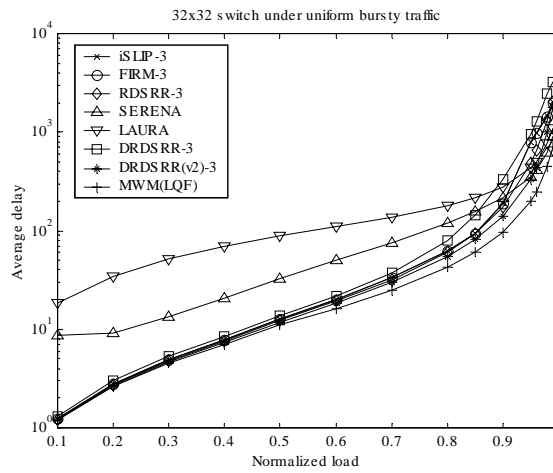
24

Average delay under uniform traffic



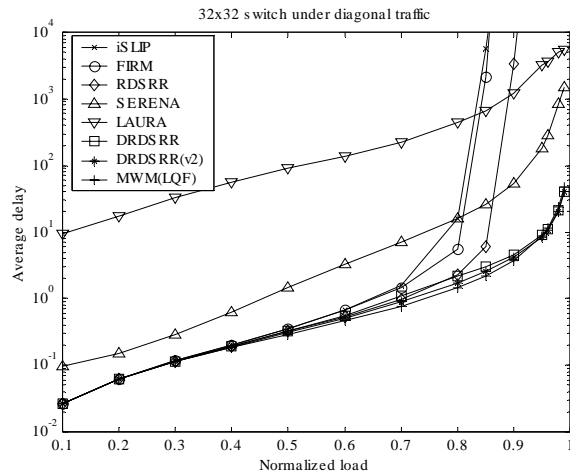
25

Average delay under uniform bursty traffic



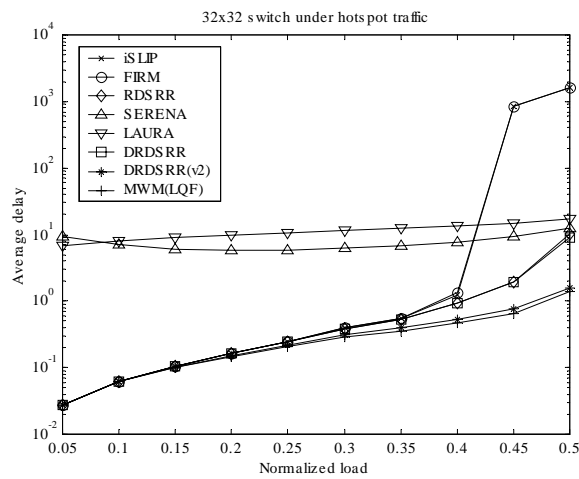
26

Average delay under diagonal traffic



27

Average delay under hotspot traffic



28

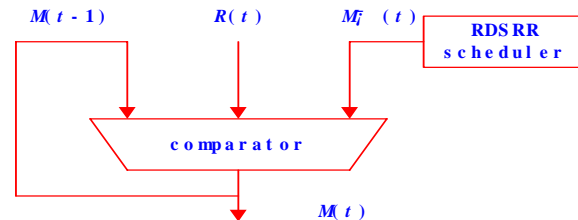
Outline

- Introduction
- Randomized Algorithms
- Fully Desynchronized Round-Robin Algorithms — SRR
- Derandomized RDSRR
- Simulation Results
- ➔ Hardware Implementation
- Conclusion
- Q&A

29

Implementation of DRDSRR

- Overall architecture



- Implementation of RDSRR scheduler
 - Optimized pipeline scheme
 - Achieve one more iteration within the same fixed amount of cycle time used in Tiny Tera
 - Can be implemented for a 256x256 crossbar

30

Conclusion

- DRDSRR combines RDSRR with randomized algorithms
- Achieve not only stability under any Bernoulli. i.i.d. admissible traffic but low delay
- Simple hardware implementation make it practical

31

Q&A

32