4. Time-Space Switching and the family of Input Queueing Architectures

4.1 Intro: Time-Space Sw.; Input Q'ing w/o VOQ's

4.2 Input Queueing with VOQ's: Crossbar Scheduling

4.3 Adv.Topics: Pipelined, Packet, Enveloppe Scheduling

4.4 Comb. Input-Output Q'ing (CIOQ) – Int. Speedup

4.5 Comb. Input-Crossp. Q'ing – Buffered Crossbars

4.6 Partitioned Crossbars

Manolis Katevenis

CS-534 – Univ. of Crete and FORTH, Greece

http://archvlsi.ics.forth.gr/~kateveni/534/

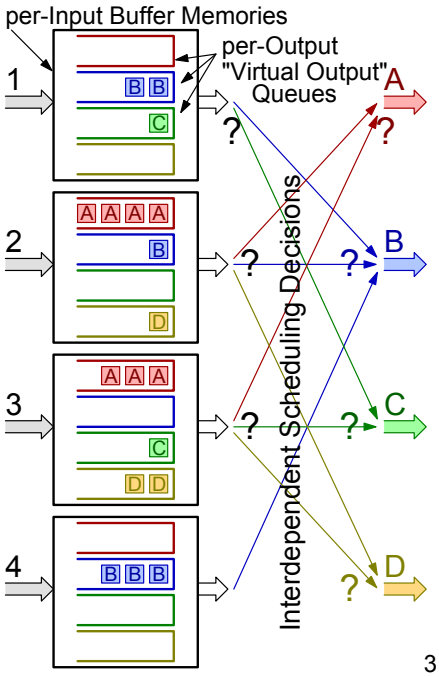
4.2 Input Q'ing with VOQ's: Crossbar Scheduling

Table of Contents:

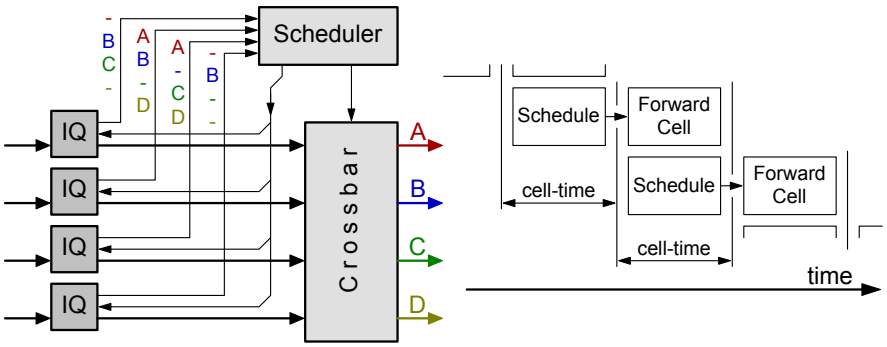
- Multiple (per-Output) Queues in each Input Buffer Memory
  - "Virtual Output Queues" – VOQ
- Crossbar Scheduling with VOQ's
  - Parallel Iterative Matching (PIM) – the general concept
  - Maximal versus Maximum Matchings
  - *iSLIP* – the practical version of PIM, the "slip" idiom
  - performance under uniformly-destined traffic
  - linear and circular (round-robin) priority circuits

Input Q'ing with VOQ's (Virtual Output Queues)

- Input Queueing: most promising *scalable* switch architecture
  - provided multiple queues
  - provided efficient scheduler
- "Virtual Output Queues" (VOQ): per-output queues at the inputs
  - N queues/input (for $N \times N$ sw.)
  - N^2 queues total in all inputs
- Interdependent Scheduling
  - can each output decide alone? … no: may create input conflicts
  - can each input decide by itself? … no: may create output confl.

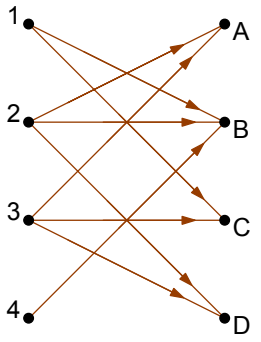


Crossbar Scheduling with VOQ's: the set-up



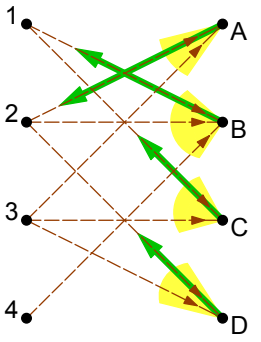
- Example: 64 Byte = 512 bit cells, 10 Gbit/s line rate ⇒ 51 ns cell-time
  - if speedup of 2 to 3, like usually (see §3.2) ⇒ cell-time below 20 ns
- Most popular scheduler: *iSLIP* – McKeown, IEEE/ACM ToN, Apr. 1999
- ... based on the earlier "Parallel Iterative Matching" (PIM) scheduler – Anderson, e.a.: ACM Tr. on Computer Systems, Nov. 1993

Crossbar Scheduling: Parallel Iterative Matching (PIM)



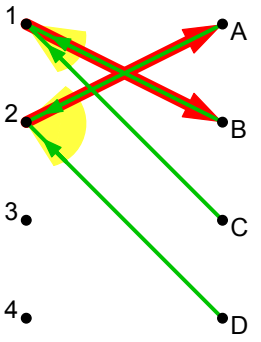
Request Phase:

All inputs send their requests in parallel



Grant Phase:

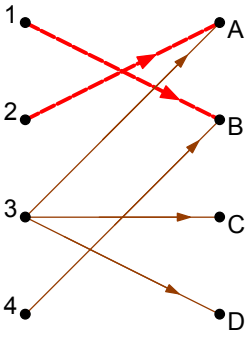
Each output, *independently*, grants to one of the requests that it received



Accept Phase:

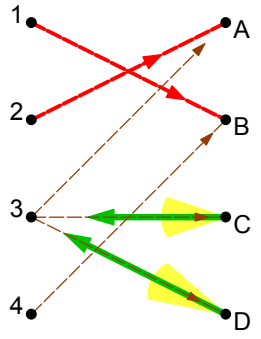
Each input accepts one of the grants that it received

First Iteration



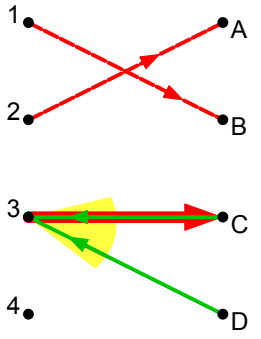
Request Phase:

Unmatched inputs (received no grant) resend their requests



Grant Phase:

Unmatched outputs (rcv'd no accept) grant to one of the received requests



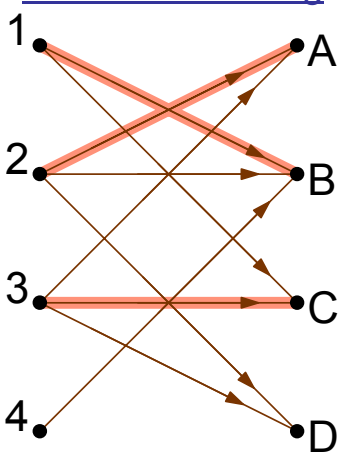
Accept Phase:

Unmatched inputs accept one of the received grants

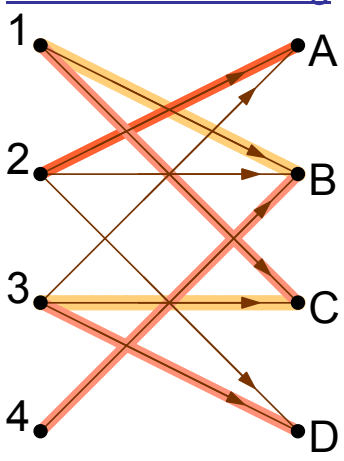
Second Iteration

- Original PIM proposal: outputs grant *randomly* among requesting inputs, inputs accept *randomly* among granting outputs

Maximal Matching – Maximum Matching



Cannot add any new connection *without breaking* some already made connection(s)

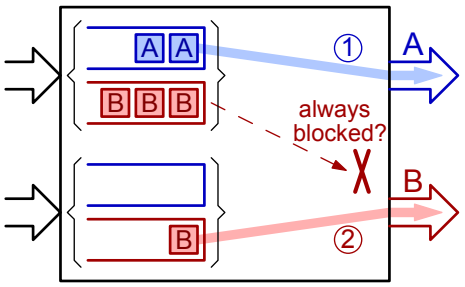


Maximum possible number of connections for the given request pattern

Maximum Matching is Complex

- *N*×*N* switch
- *R* requests; usually *R* is $O(N^2)$
- Deterministic Algorithm: $O(N \cdot (N+R)) \approx O(N^3)$
  - Tarjan: Data Structures & Network Algor., SIAM 1983
- Randomized Algorithm: $O(N+R) \approx O(N^2)$
  - Karp e.a: ACM STOC, 1990

… and may be unfair



- PIM progresses towards *maximal* matchings – not necessarily maximum.
- Original PIM, with random selections: each iteration resolves, on average, ≥3/4 of the remaining unresolved requests ⇒ $O(\log N)$ iterations.
- Hardware implementation of truly random selection, at high speed, with equal probability among (varying number of) choices, is unrealistic.

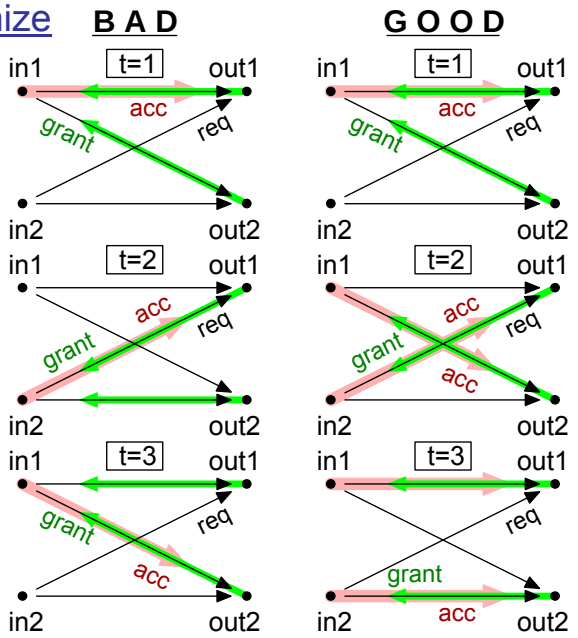
iSLIP: Practical, Popular Crossbar Scheduler

- Variation of PIM:
- Request Phase: same as PIM.
- Grant Phase:
  - PIM grants *randomly* to one of the requesting inputs.
  - iSLIP grants in *Round-Robin* order to the first requesting input after the previously marked input – careful which one that is: see next slide (the "Slip" idiom).
- Accept Phase:
  - PIM accepts *randomly* one of the granting outputs.
  - iSLIP accepts, in *Round-Robin* order, the first now-granting output after the output that was accepted last time.
- Relatively easy to implement, good fairness properties
- Was used in CISCO GSR-12000, Tiny Tera, Abrizio/PMC-Sierra, e.a.
  – Nick McKeown: IEEE/ACM ToN April 1999.

Slip to Desynchronize

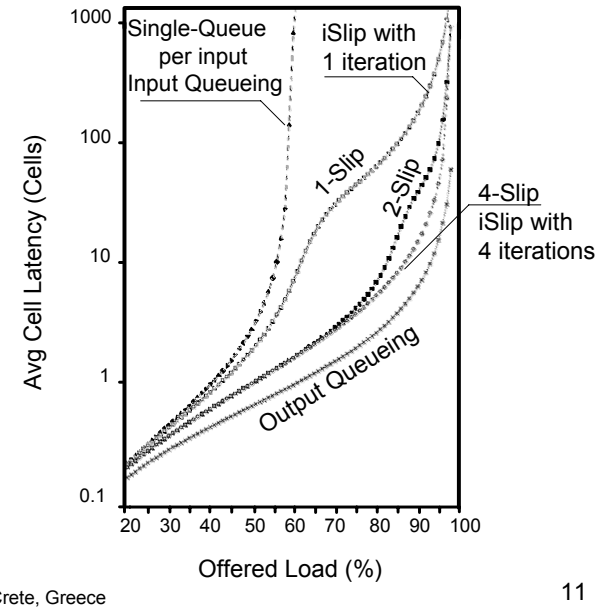
- Bad: grant round-robin to next request seen after last-time grant
- Good: grant rnd-robin to next request seen after last grant *that was accepted*

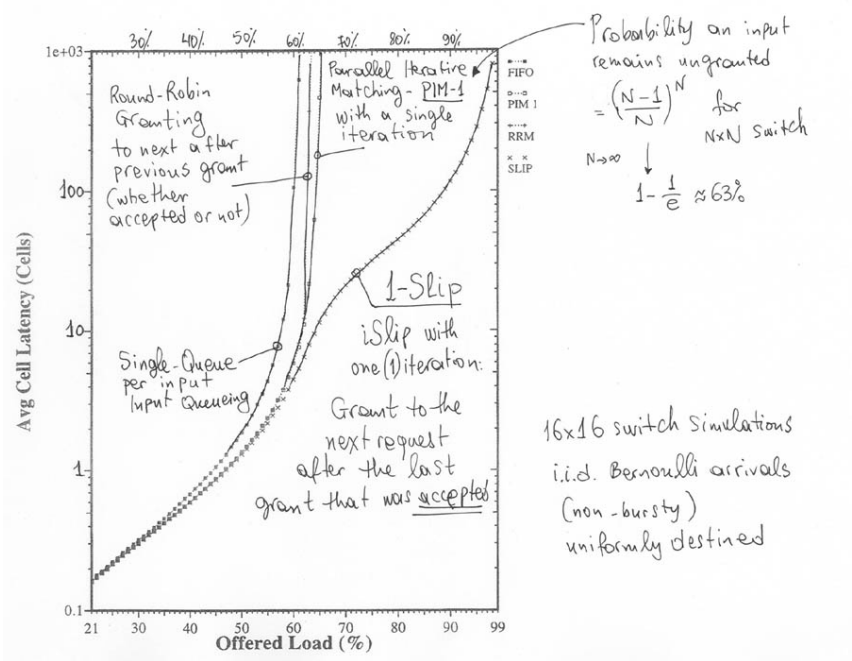
⇒ Under uniformly-destined traffic, when all VOQ's become non-empty, grant pointers get *desynchronized* and stay that way, moving all together, and yielding 100% output utilization

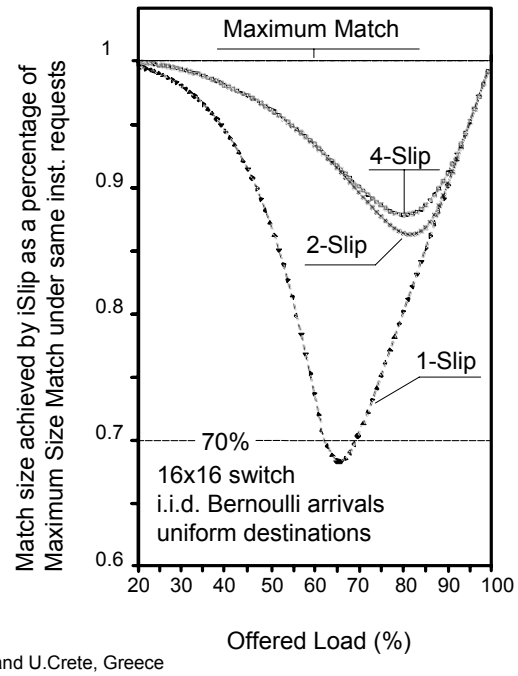


iSlip Performance under Uniform, Non-Bursty Traffic

- Source: McKeown, IEEE/ACM ToN, April 1999
- 16×16 switch
- i.i.d. Bernoulli arrivals (non-bursty traffic)
- uniformly distributed packet destinations
- *Beware!:* real traffic is very different from these assumptions – simulations under these assumptions are only useful to give the designer a "first, very rough idea"…

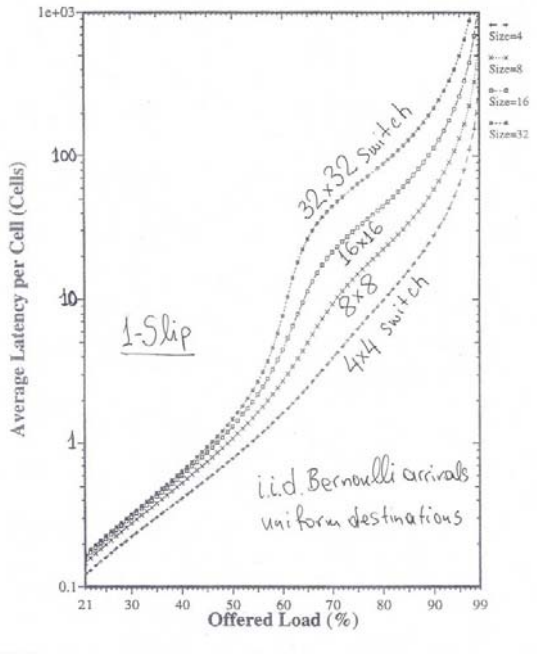






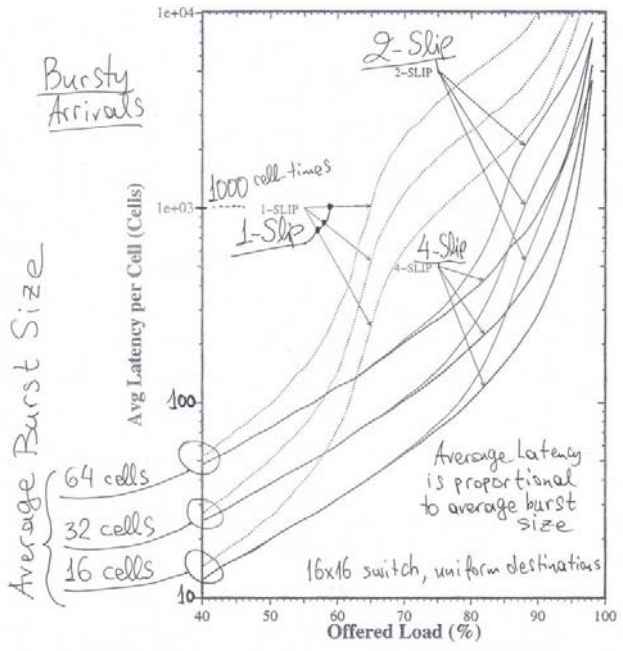
3.1 - M. Katevenis, FORTH and U.Crete, Greece

13



4.2 - U. Crete - M. Katevenis - CS-534

14



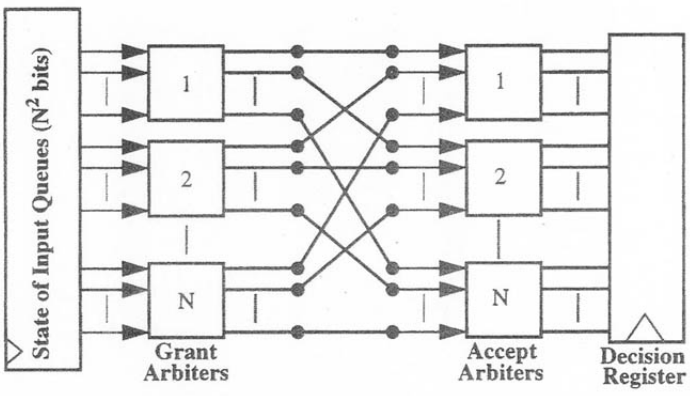
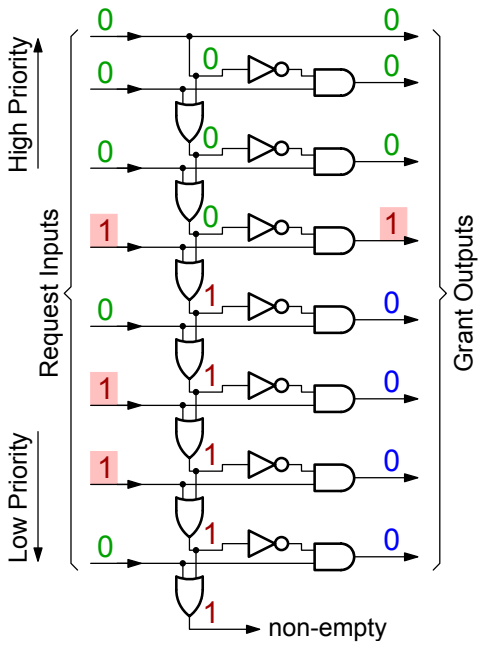


Fig. 21. Interconnection of $2N$ arbiters to implement iSLIP for an $N \times N$ switch.

Linear Priority Arbiter

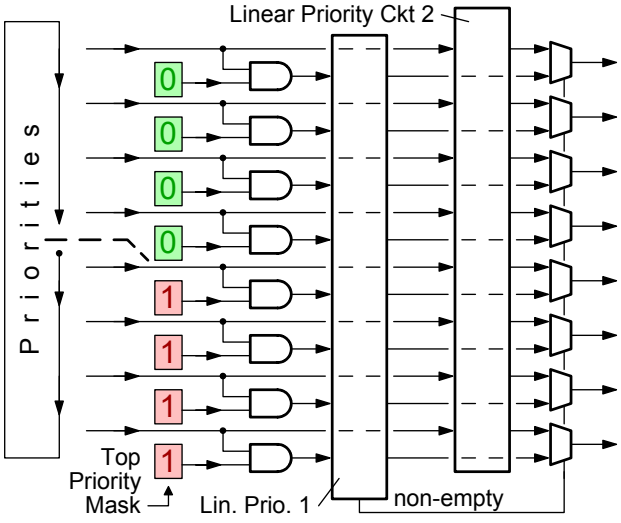
- *Input:* an ordered set of request flags
- *Output:* the same set of flags, where only the topmost asserted request remains ON, while all lower are cleared
- Output *i* is asserted iff input *i* is ON and none of the previous inputs, *0* to *i-1*, is ON:
  - Output[*i*] = Input[*i*] AND NOT ($OR_{k=0}^{i-1}$ Input[*k*])

⇒ *Delay* grows *logarithmicly* with the number of inputs
  - big OR functions built as trees of fixed-fan-in gates



Circular (RR) Priority Arbiter

- Bit mask defines the top-priority position.
- First priority circuit finds topmost request in "half" the input request set, if any.
- Second priority ckt finds topmost request in other "half" of input set.



- Chose the first answer, if any, else chose the second

⇒ Circular priority (round-robin) delay grows w. logarithm of num. of inputs