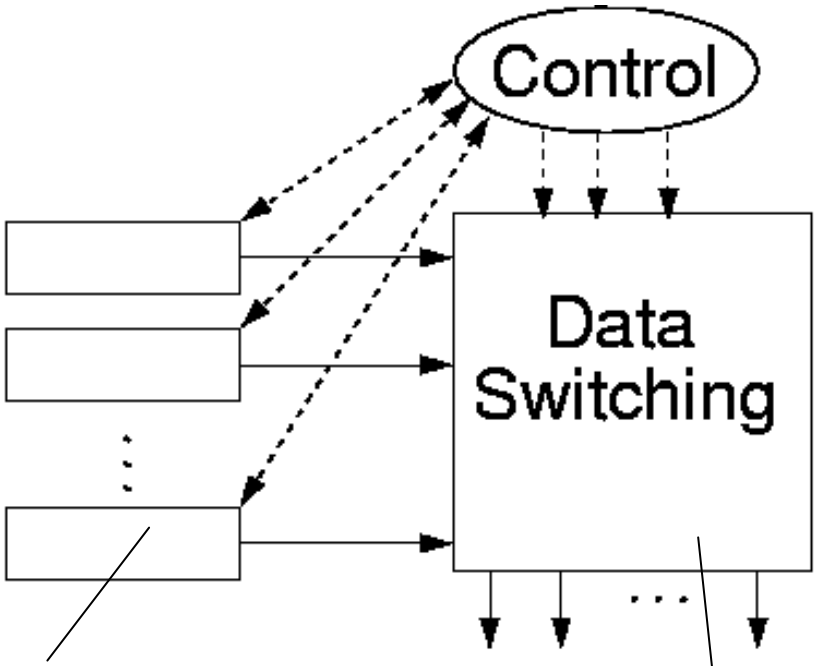
Congestion Management for Non-Blocking Clos Networks

Nikos Chrysos

Inst. of Computer Science (ICS) FORTH -Hellas

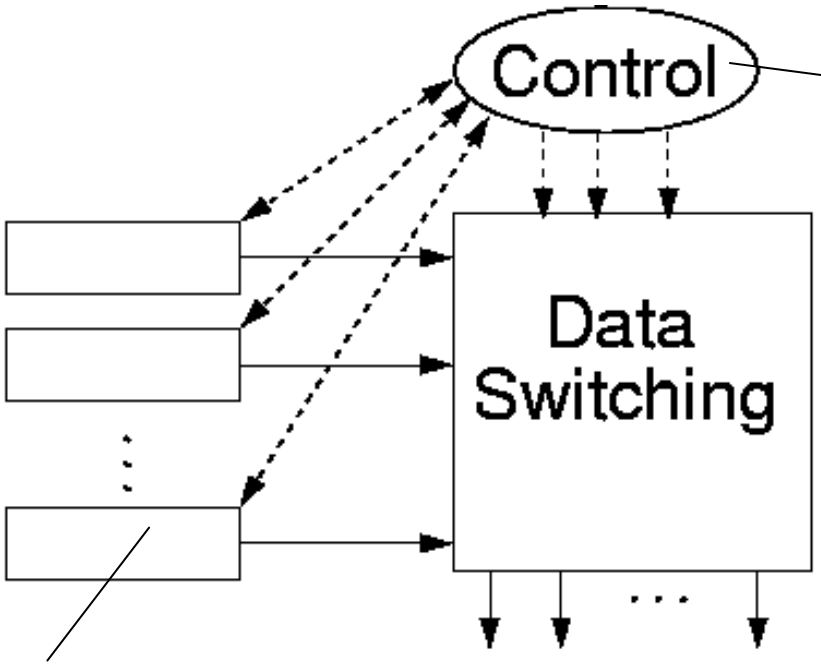
The way to Scalable Switches



Input Buffers
containing VOQ's

- Currently: bufferless Crossbar … $O(N^2)$ cost
- Next: Switching Fabric (e.g., Clos, Benes … *O(N *log(N)*) containing small buffers

The way to Scalable Switches

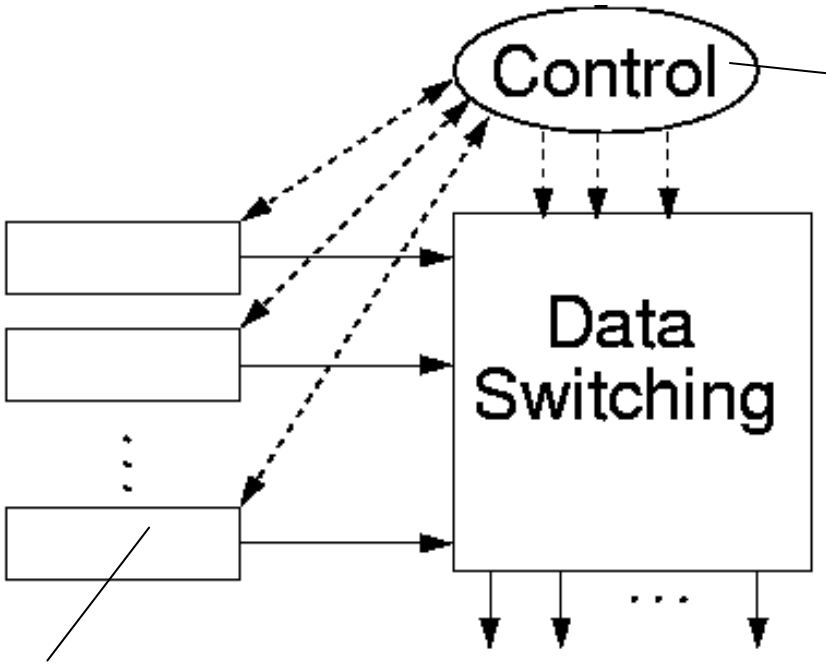


- Currently: Crossbar Scheduler
  - exact, one-to-one pairings
- Next: Congestion Management
  - approximate pairings owing to small buffers in the fabric

Input Buffers
containing VOQ's

- Currently: bufferless Crossbar … $O(N^2)$ cost
- Next: Switching Fabric (e.g., Clos, Benes … *O(N *log(N)*) containing small buffers

The way to Scalable Switches



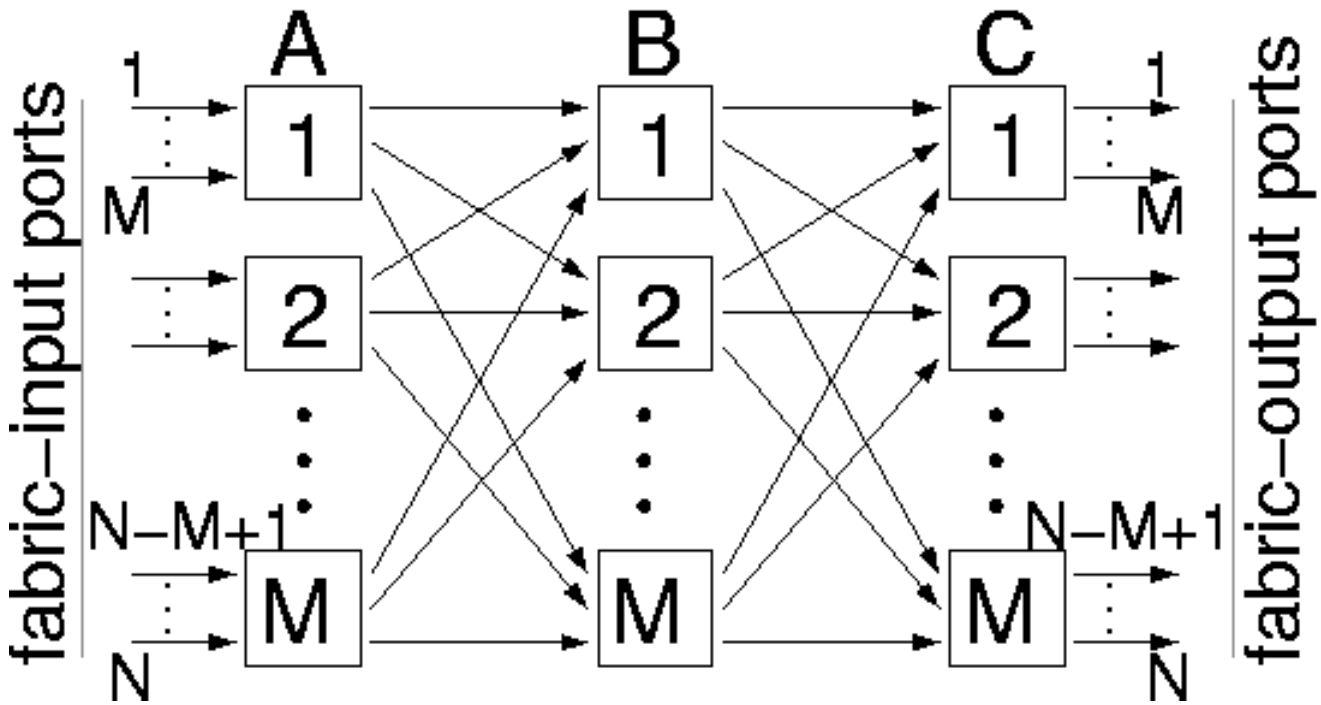
Input Buffers
containing VOQ's

- Currently: Crossbar Scheduler
  - exact, one-to-one pairings
- Next: Congestion Management
  - approximate pairings owing to small buffers in the fabric

exact: one cell/output/cell time

approximate: $\leq$ w+B cells / output / w time-window

3-Stage Clos/Benes Non-Blocking Fabric



- MxM buffered crossbar switch elements
  - $N = M^2$
- no internal speedup

Congestion Mgmt: Proposal Overview

Under Light Load (… lightly loaded destinations)

- transmit data without pre-notification
- minimize latency
- careful not to flood buffers:

  ⇒ Limited # of unacknowledged bytes

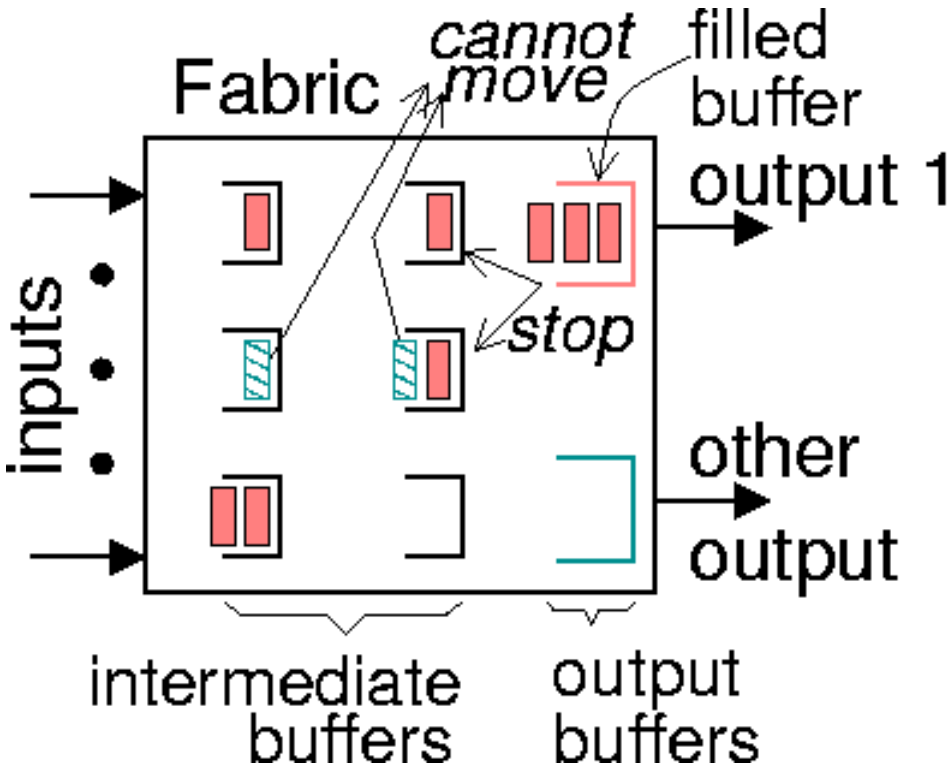
Under Heavy Load (… congested destinations)

- pre- approved transmissions
  - request *(to control)*
  - grant *(from control)*
  - transmit data

Alternative FC style is too expensive …

- per-flow queuing & backpressure
- more than N x M packet queues in one switch chip …

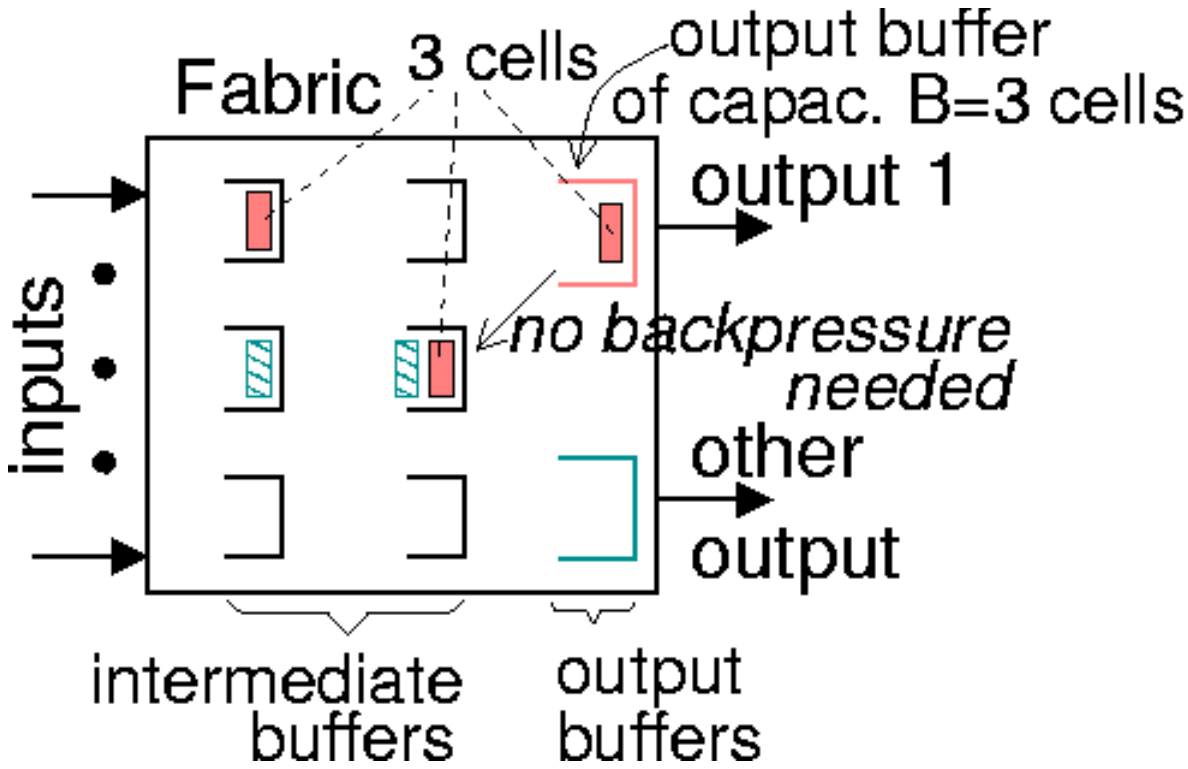
Problem: "Free" Injections w. Limited Buffers



Oversubscribed outputs delay other flows

- congested output buffer fill up, then …
- congested packets are stalled in intermed. buffers, then …
- other packets are delayed too

Our Congestion Control for High Load

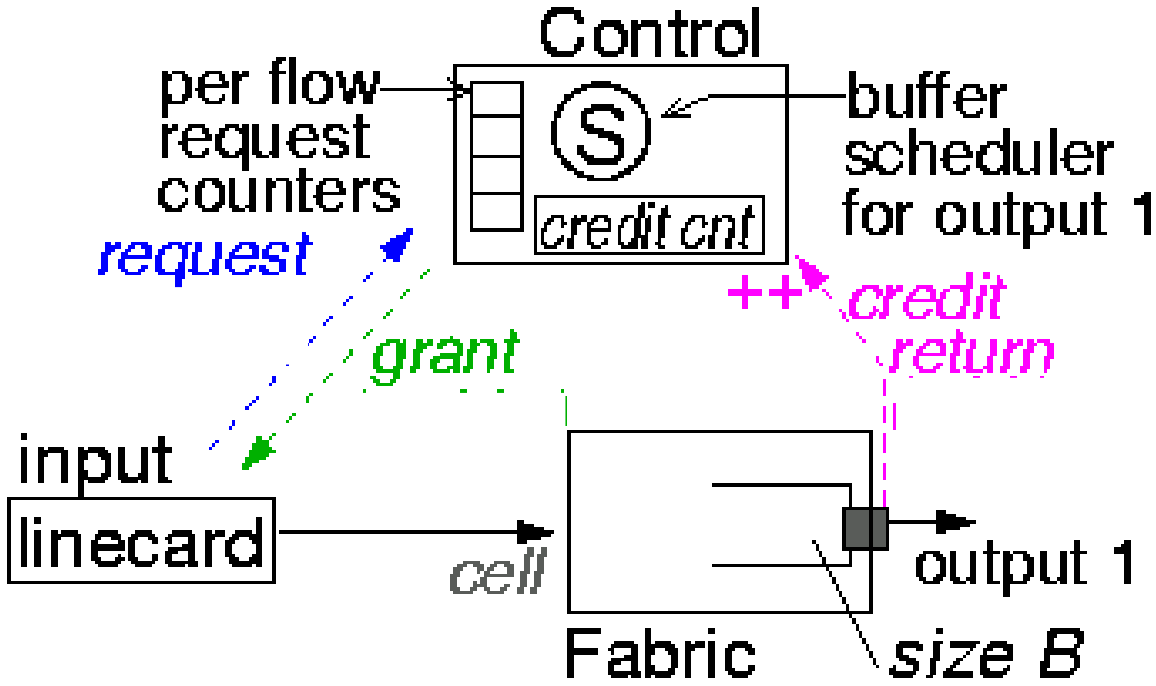


Approximate pairing constraint:

allow at most B cells destined to the same output inside the fabric

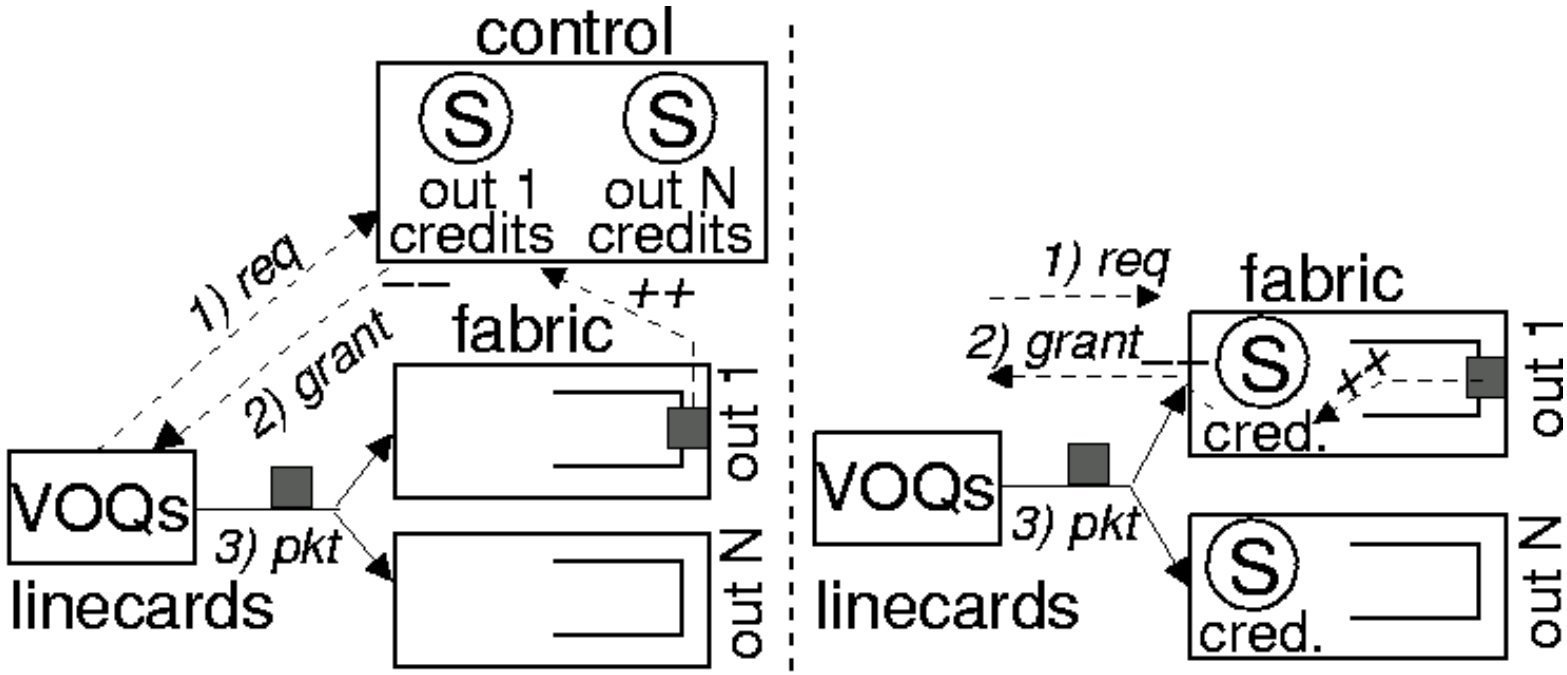
➔ can eliminate HOL blocking

Finding Approximate Pairings



per-output, independent (single-resource) schedulers ensure at most B cells destined to that output inside the fabric at any given time

Centralized vs. Distributed Control



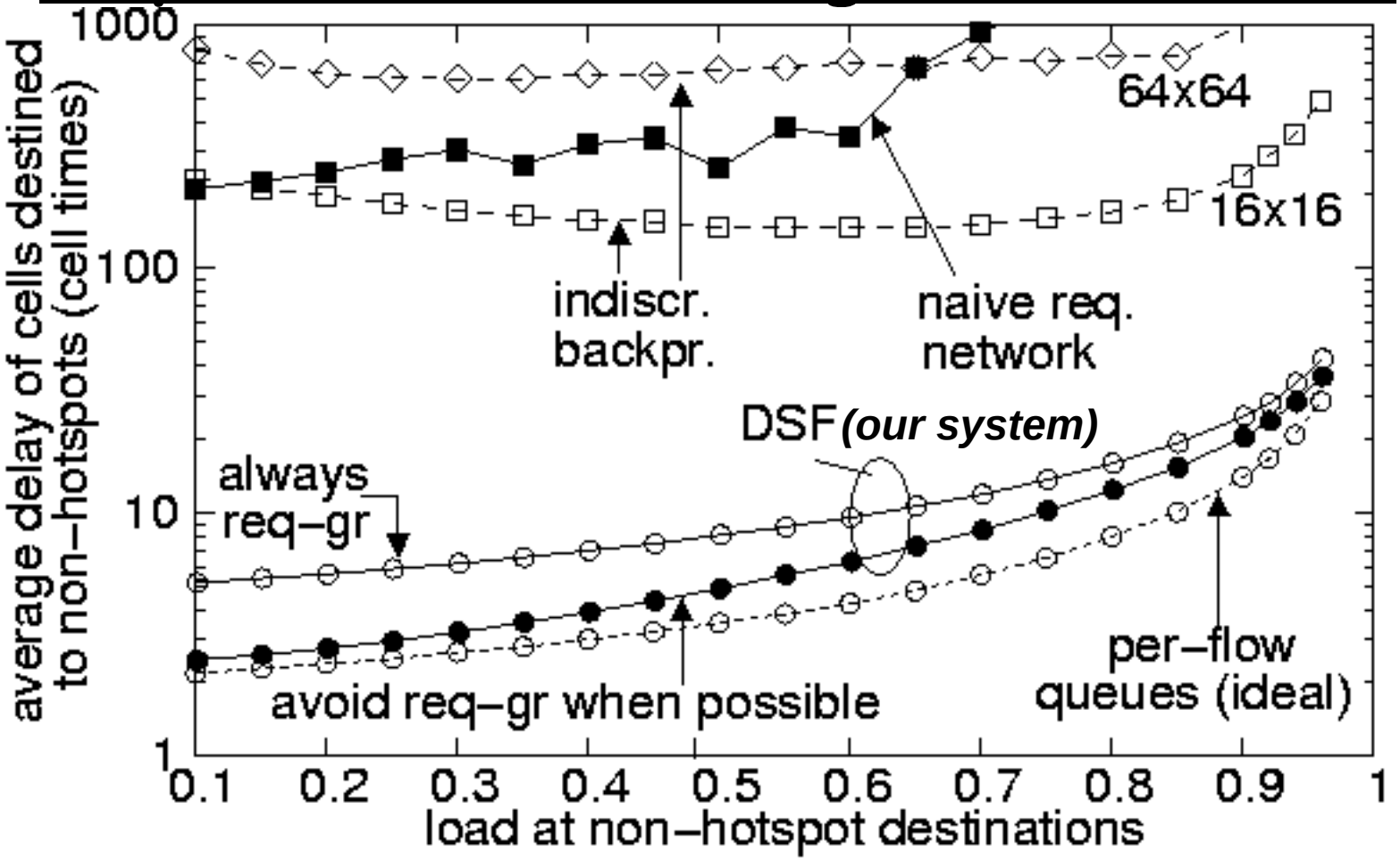
Central Control

- excellent performance
- feasible for 1024 (1K), 10 Gbs ports [ChrKatInf06]
- difficult to scale beyond that…

Distributed Control

- avoids central chip b/w and area constrains → more scalable
- needs careful "request contention" mgmt

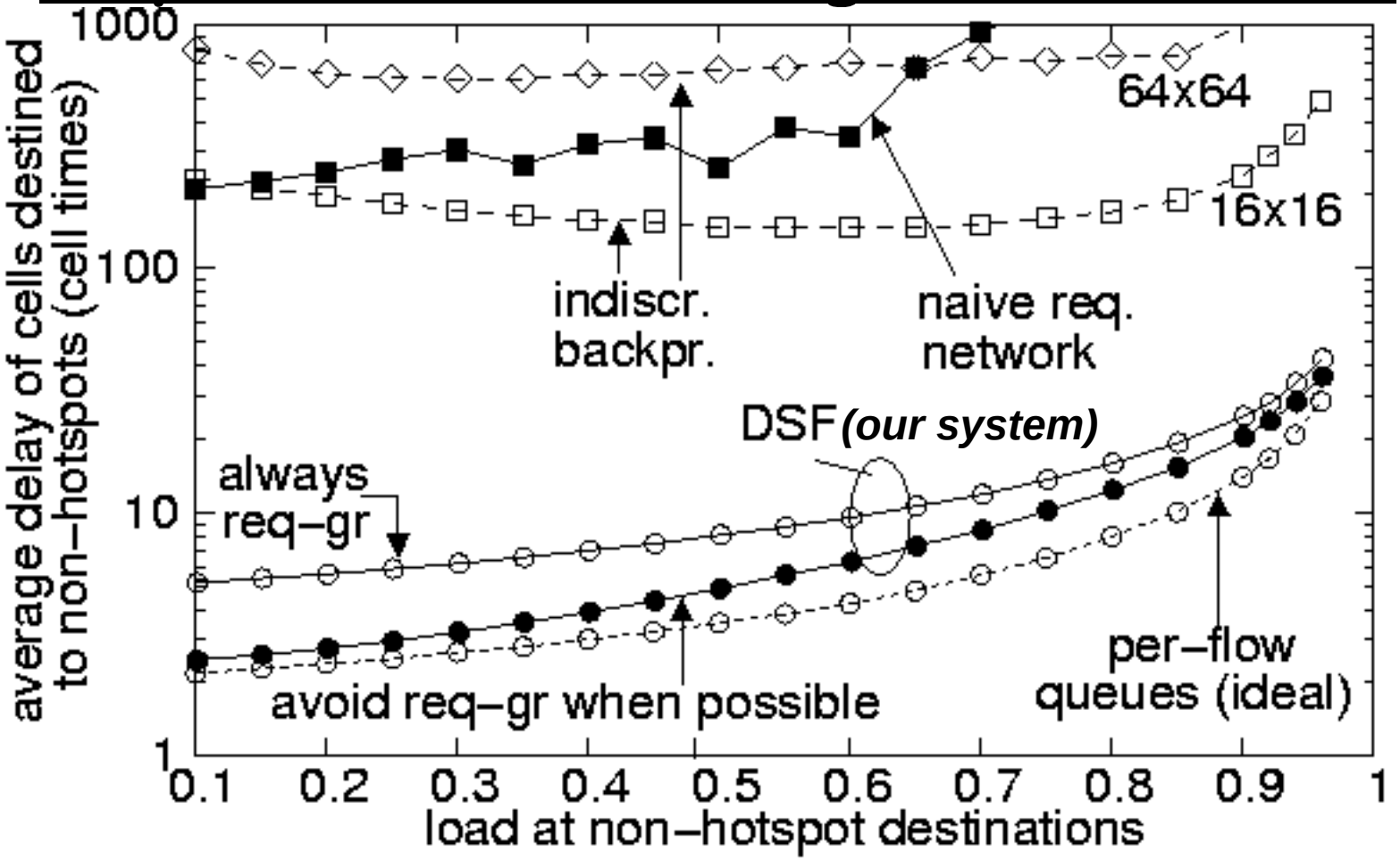
Request contention mgmt is needed…



2 hotspot (congested) outputs receive 1.1 cells /cell time

- "indiscriminate backpr.": plain hop-by-hop backpressure on data
- "naïve req. network": indiscriminate backpressure in req. channel
- "per-flow queues": ~N data queues per xpoint (way too expensive)

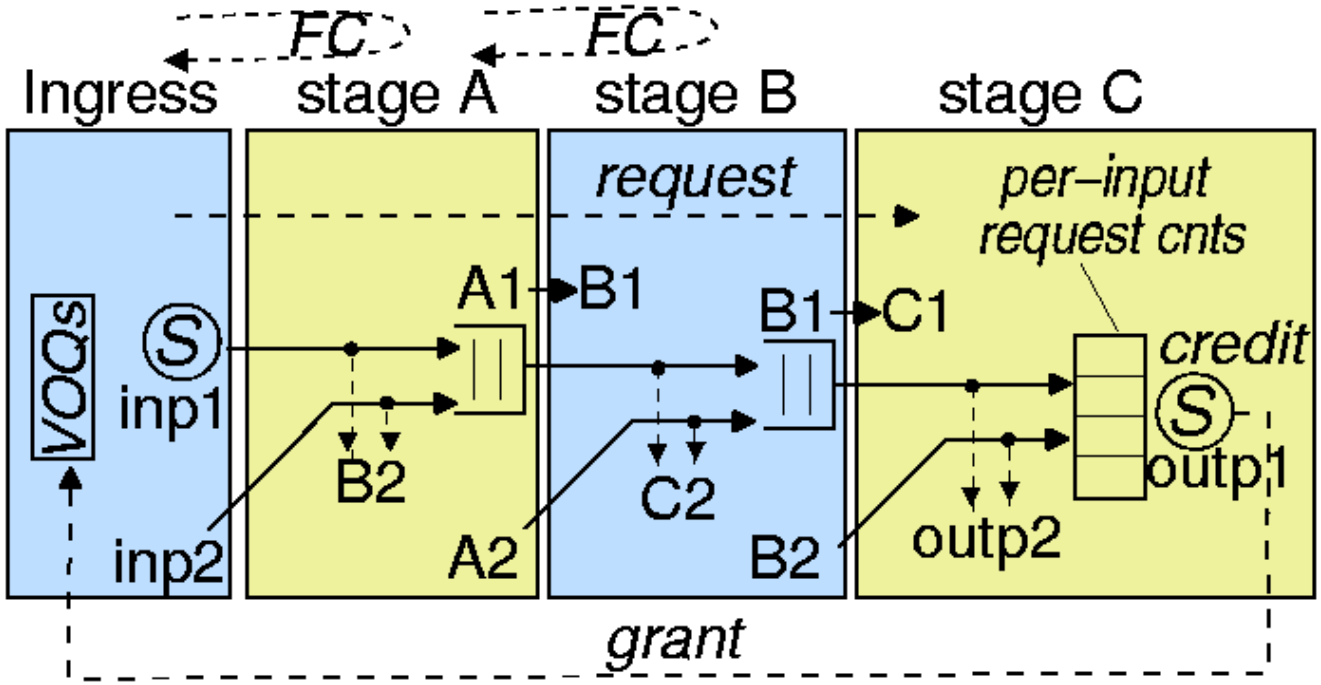
Request contention mgmt is needed…



2 hotspot (congested) outputs receive 1.1 cells /cell time

- not shown is the utilization at hotspot outputs
  - per-flow queues & our system: always 100%
  - indiscr. backpr.: ~ 75% at 0.1 x-axis, 100% at 1.0 x-axis

Distributed Scheduler: Request Channel

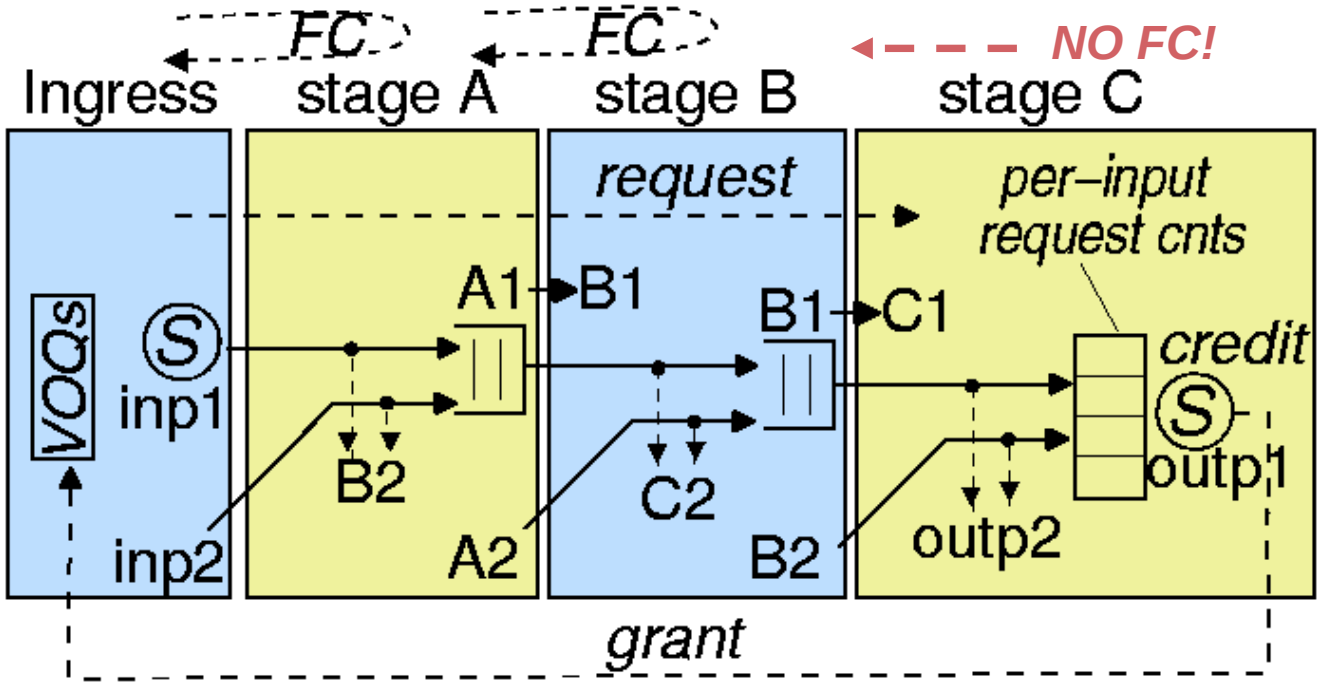


Each (output) credit scheduler at its corresponding C-switch

- request routed to *credit schedulers* via multipath routing
- grants travel the other way to linecards (grant chnl. not shown)
- pipelined operation of independent single-resource scheduler
- no problem with out-of-order delivery: per-flow request (or grants) interchangeable w. e. o.

Request Flow Control

Distributed Scheduler: Request Channel



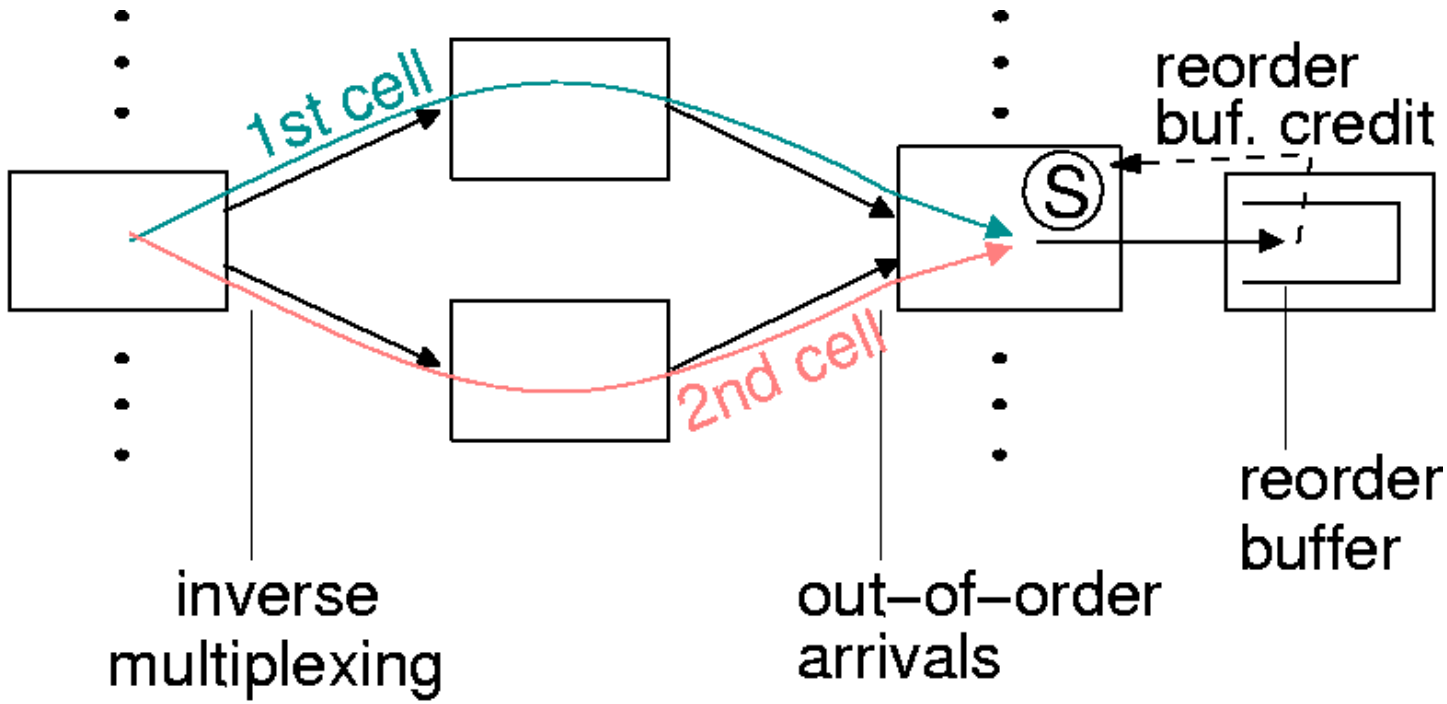
Each (output) credit scheduler at its corresponding C-switch

Request Flow Control

- indiscriminate hop-by-hop req. FC: ingress LC to A-st. / A- to B-st.
- hierarchical request FC: ingress LC to C-stage
  - pre-allocated C-stage request counter (space) upon req. injection

  → no backpressure upon B-C req. queues → no HOL in req. channel

Guaranteeing in-order cell delivery



Inverse-multiplexing (multipath routing) on data can yield out-of-order delivery

- each output credit scheduler (additionally) manages the reorder buffer space at its corresponding egress linecard:
  - issue new grants only if reorder buffer credits available

→ bounded reorder buffer size

Avoid request-grant latency overhead

Every linecard allowed to send ≤ *U* rush cells without having to first request & then wait for grant

- privilege renewed by ACKs received from rush cells
  - →low load: ACKs return fast & most cells send eagerly.
  - →high load: ACKs delay & request-grant cells dominate.
- at injection load, **ρ**, frequency of rush cells, **α(ρ)**

$$\alpha(\rho) \approx U / (\rho \cdot E[\text{ ACK-delay}(\rho)]),\ \text{where:}$$

E [ACK-delay (ρ)] = ACK_rtt *(fixed)* + E [queue_del(ρ)] *(variable)*

$$U = \text{ACK\_rtt},\ 1/\alpha(\rho) = \rho + \rho \cdot E[\text{ queue\_del}(\rho)] / \text{ACK\_RTT}$$

→ α(ρ) increases with ACK_rtt, decreases with load, ρ, and decreases w. queuing delays (e.g. from Bernoulli to bursty)

Prevent forwarding rush cells to congested outputs

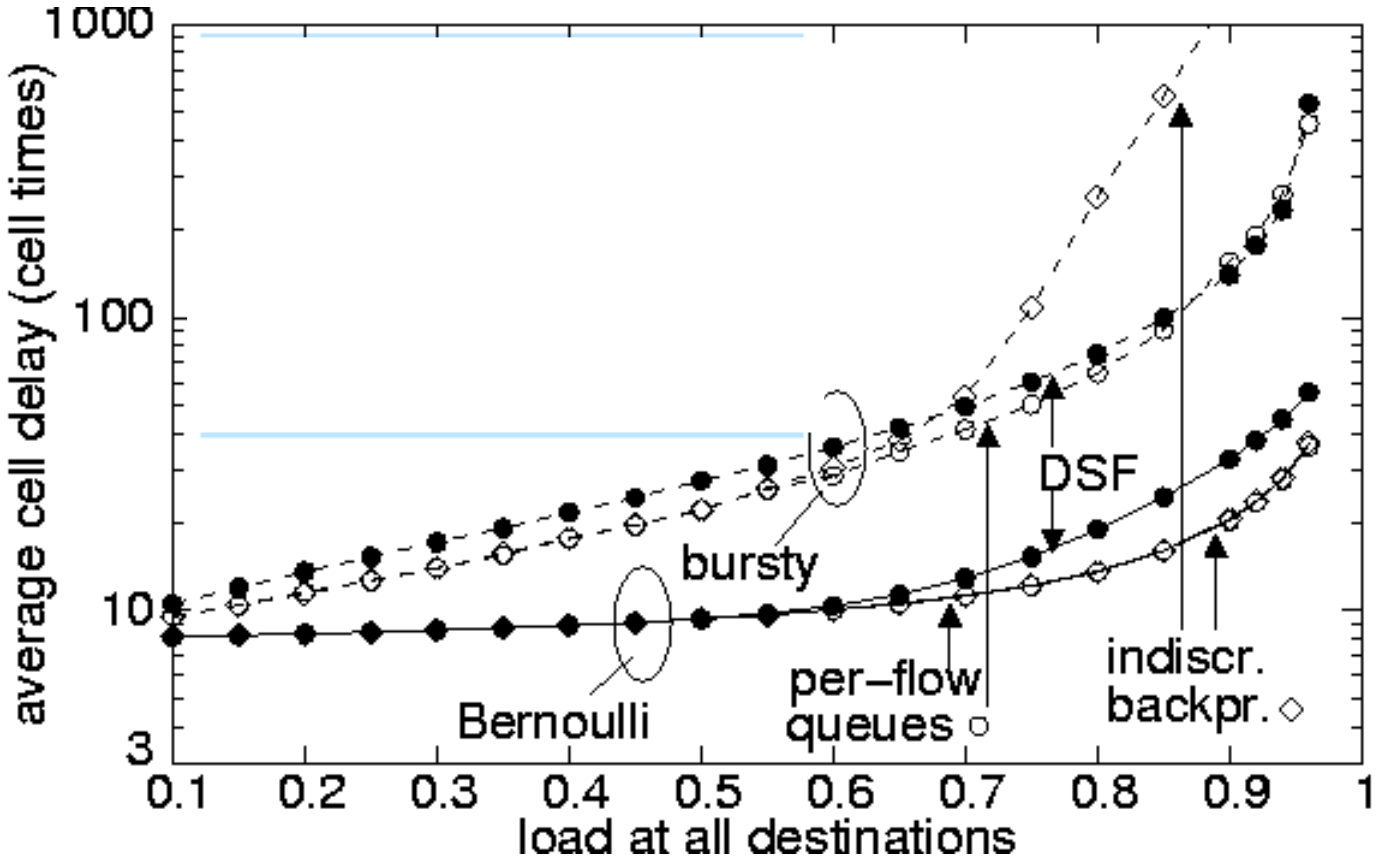
Avoid request-grant latency overhead

Every linecard allowed to send ≤ *U* rush cells without having to first request & then wait for grant

Prevent forwarding rush cells to congested outputs

- …congested outputs may needlessly use (and even worse "hog") rush-mode quota, depriving them from well-behaved cells
- So …
- outputs detect their congestion, using HIGH & LOW thresholds for the sum of their respective request counters & crosspoint buffer occupancies
- picky-back congested/non-congested information on ACK/grant messages to ingress linecards
- ingress linecards do not send rush cells to congested..

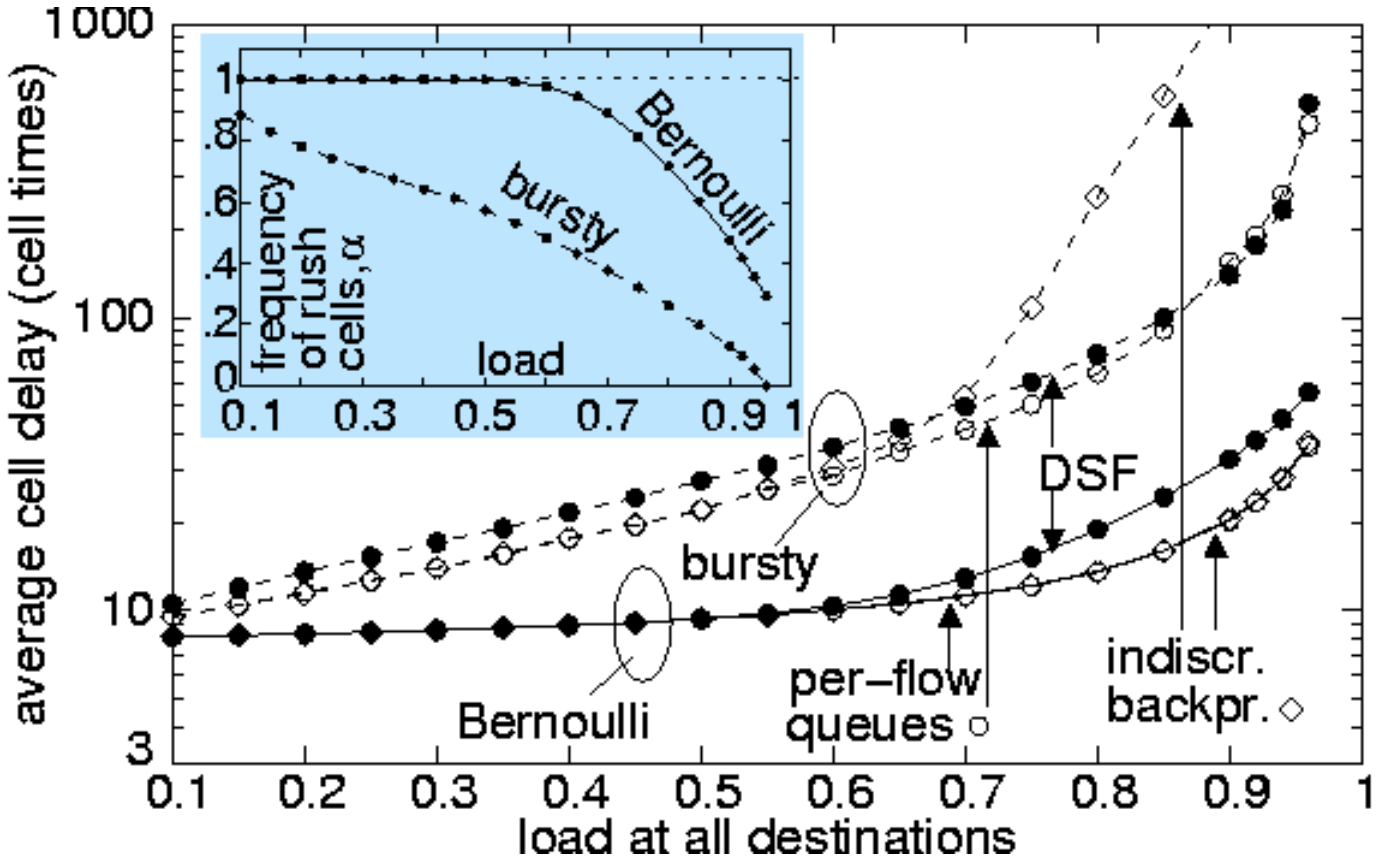
Delay minimization via rushy-mode



Uniform traffic (no hotspots); Bernoulli arrivals; bursty arrivals

- at low loads DSF delay very close to minimum possible
  - minimum rush-cell delay ~ 8
  - min. request-grant cell delay ~ 18

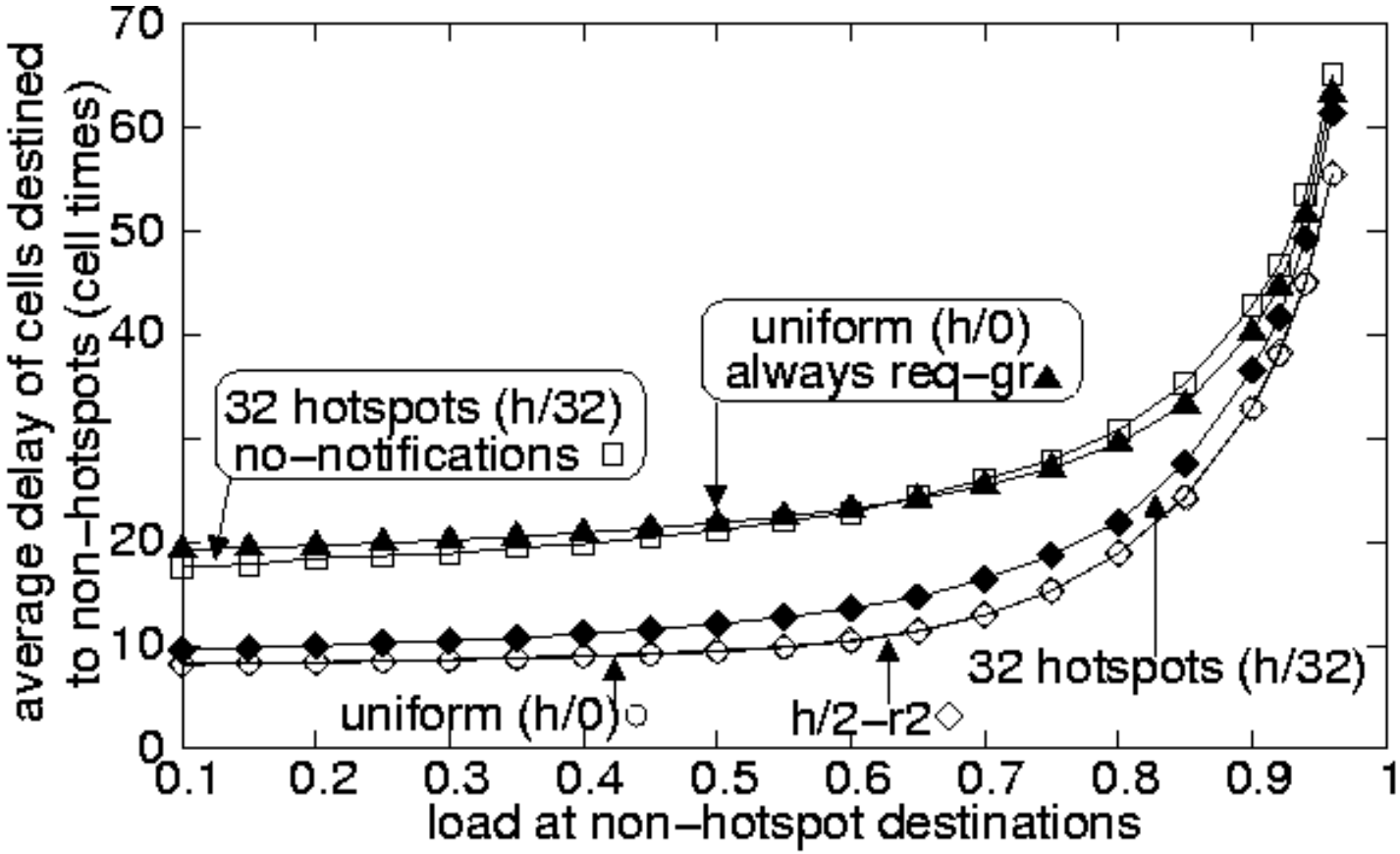
Delay minimization via rushy-mode



Uniform traffic (no hotspots); Bernoulli arrivals; bursty arrivals

- at low loads DSF delay very close to minimum possible
- the percentage of rush cells decreases as cell latency increases (i.e. faster when traffic is bursty)

Rushy-mode under hotspot traffic



- cells destined to non-hotspots use rushy-mode

  -> minimized delay (~ 10 cell times at low loads)

- cells to hotspots sent via request-grant transactions.

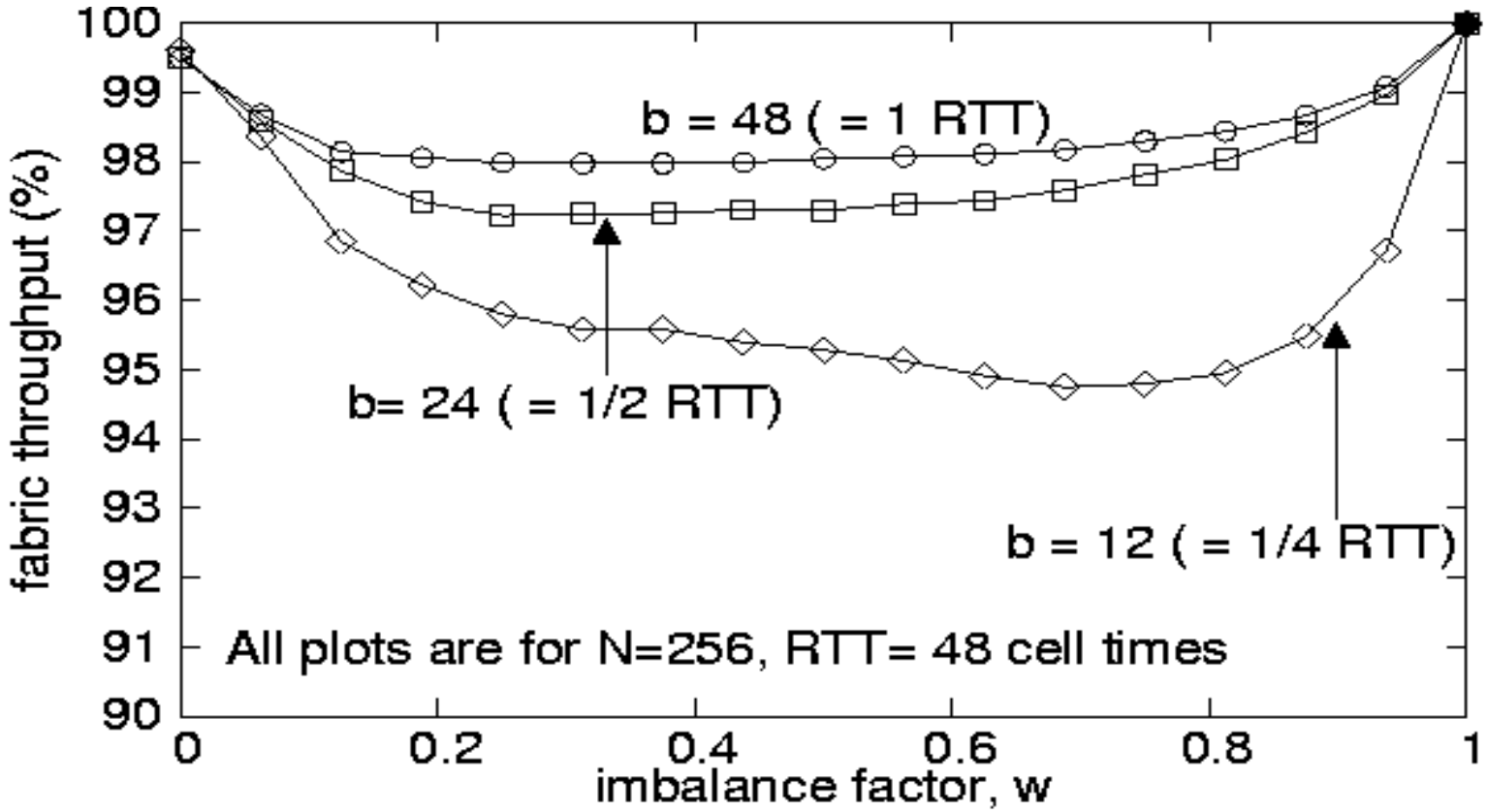
- "no-notifications": congested cells deprive rushy-quota from well behaved cells (➔ delay ≈ "always req.-grant", i.e. ≥ 18 cell times)

Sub-RTT crosspoint buffers

- ∃ M crosspoint buffers per output (this is a fact)
- 1 end-to-end RTT (linecard to C-stage credit scheduler and back) aggregate space in all these buffers suffices for req-gr

  → each crosspoint buffer needs space ≥ 1/M RTT
- practically B- ↔C-stage ("local") rtt will dictate the minimum required crosspoint buffer space

Thruput for sub-RTT buffers: N=256, M=16

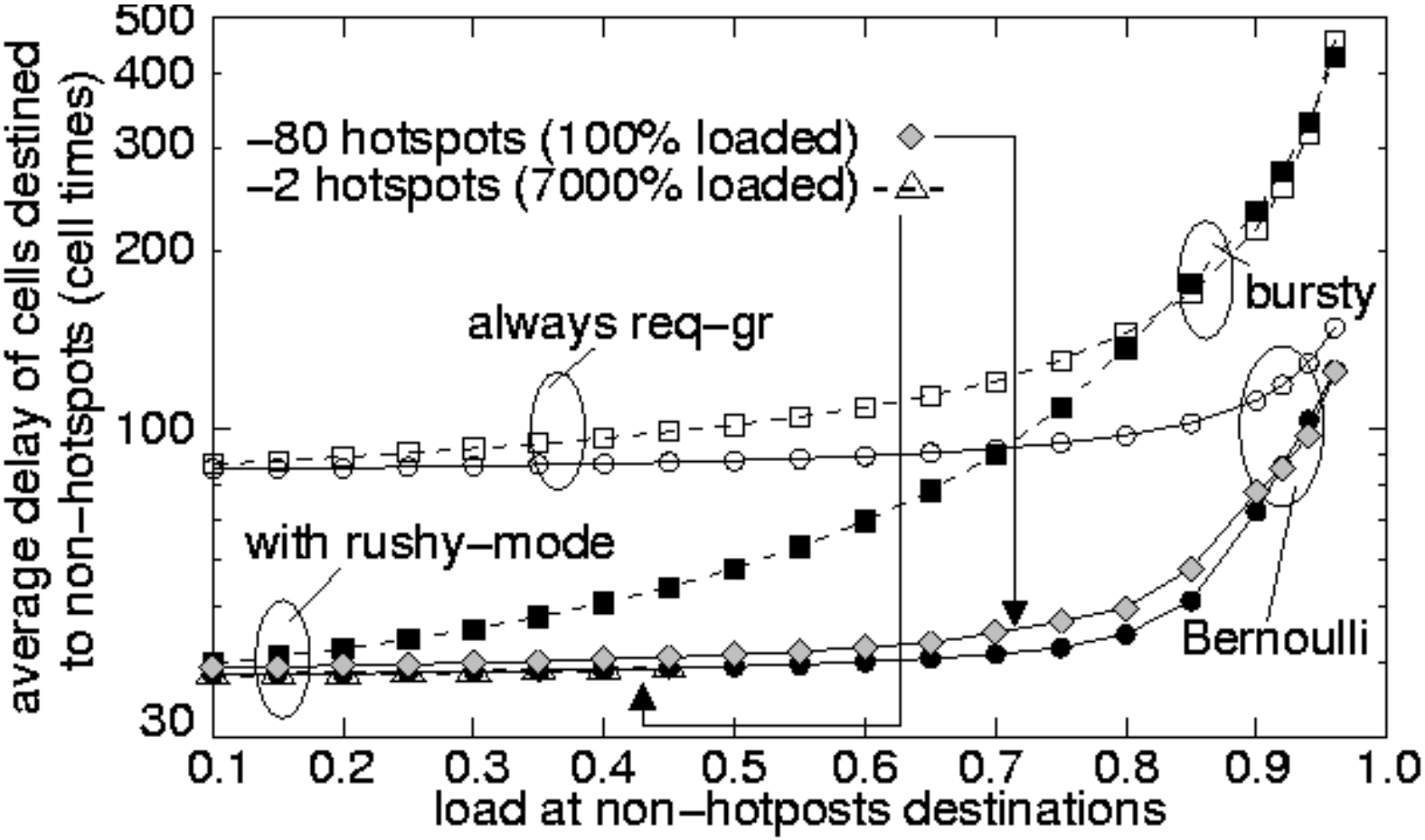


Fabric throughput for different imbalance factors, *w*

- w= 0:uniform traffic
- w= 1:totally imbalanced traf: non-conflicting input/output pairs
- almost 95% throughput, for any *"w"* with b= 12 = RTT /4

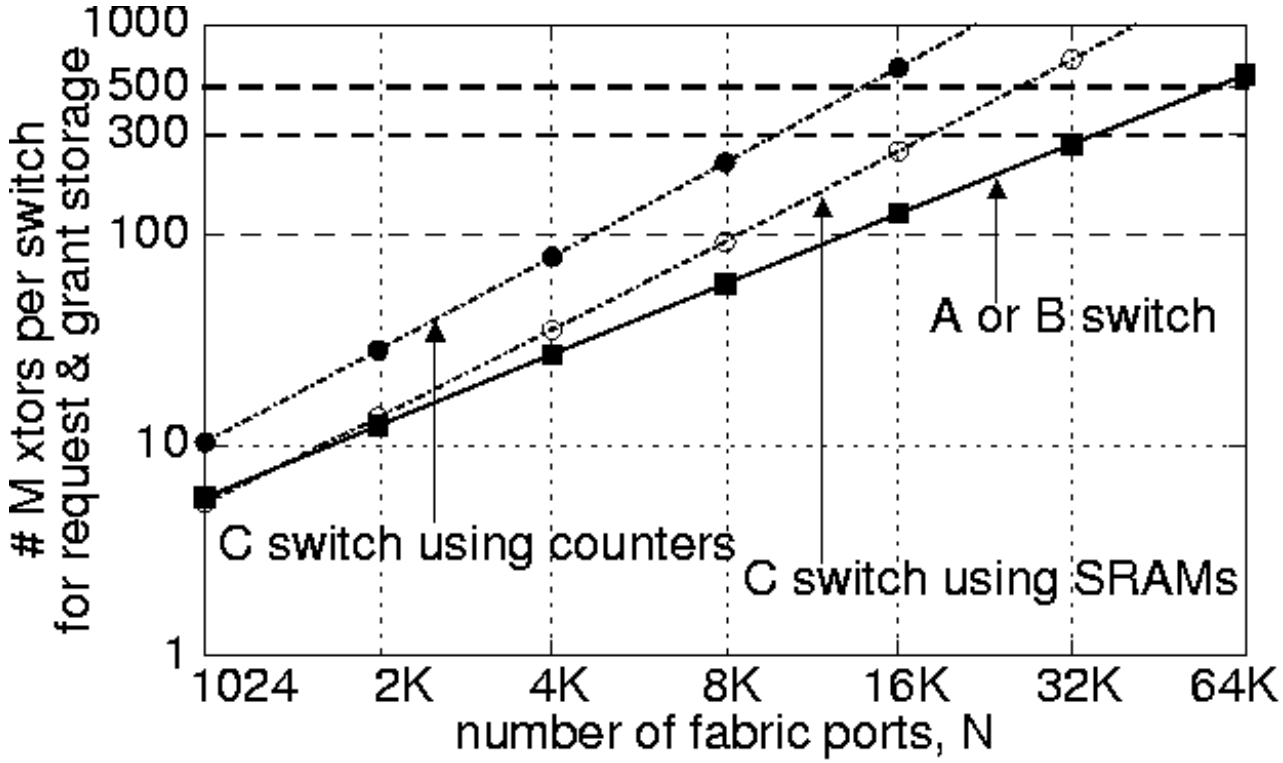
Buffer size b dictated by B-C "local" (11 cell times)

Delay performance with N=256, RTT=48 & sub-RTT buffers (b=12 cells)



- minimum delay of a req-grant cells ~ 84 cell times
- minimum delay of a rush-cells ~ 40 cell times.
- almost perfect performance even under severe traffic

Control Overheads



b/w overhead: O(log (N))

- ~ 10% for N= 16K & cell size = 64 bytes (log (N)
- 5 % for N= 16K & cell size= 128 bytes

area overhead

- request-grant storage +distribution pointers cost depicted in Figure (million transistors per switch-element (chip?))

Concluding points

Multi-Terabit switching feasible via distributed pipelined single-resource schedulers & multi-stage fabrics w. inverse multiplexing

- excellent flow isolation for any number of hotspots
- low latency, high throughput

The same architecture can work on variable-size segments

Need to study transient behavior

Use two queues (virtual channels)

Concluding points

Multi-Terabit switching feasible via distributed pipelined single-resource schedulers & multi-stage fabrics w. inverse multiplexing

The same architecture can work on variable-size segments

- eliminating padding overhead, which is a source of speedup

Need to study transient behavior

- Onset of congestion: well-behaved cells may suffer long delays due to many rush cells that target the same (hotspot) output

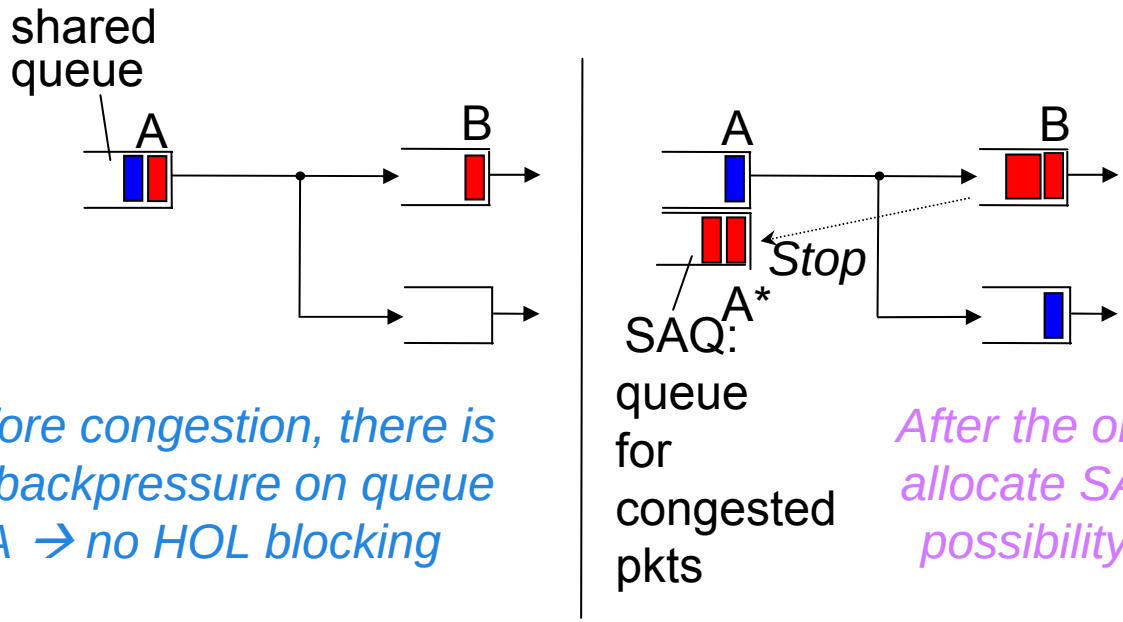
  → queues will temporarily fill w. congested rushy cells

- in the long term, congested flows will be throttled by req-grant

Use two queues (virtual channels)

- one for **all** congested cells; one for other

Reminds of RECN & SAQs



Before congestion, there is no backpressure on queue A → no HOL blocking

After the onset of congestion, allocate SAQ to deal with the possibility of backpressure.

With req-grant in the corner, no need for many SAQs as in RECN

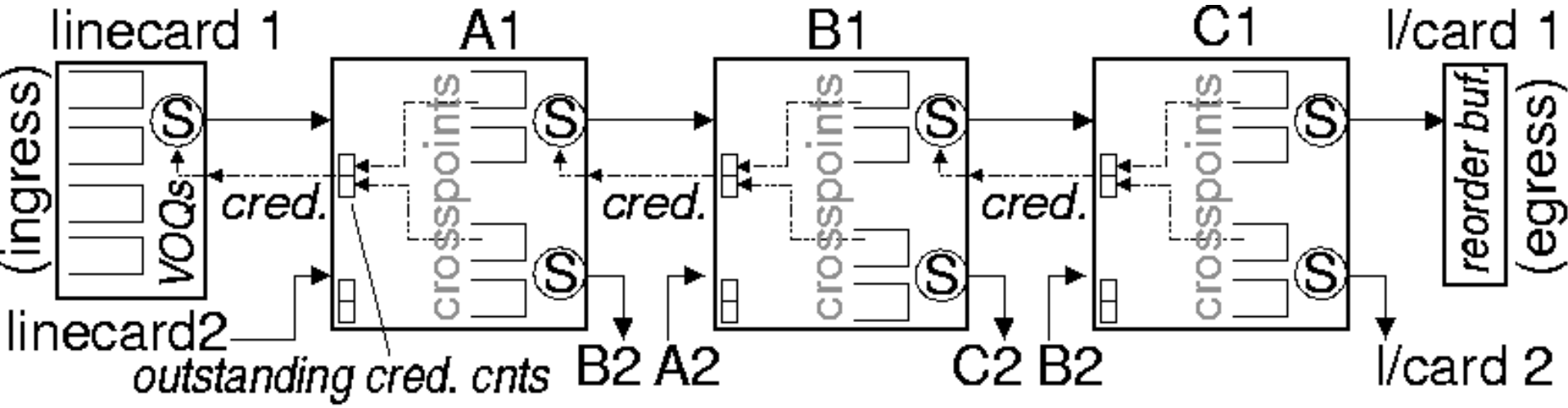
- SAQ used by all congested destination
  (no need to separate congested flows from each other)
  - request-grant will appropriately throttle (the long-term) congested flows rates
- just need a separate queue for well-behaved cells..

Thank you!

nchrysos@ics.forth.gr

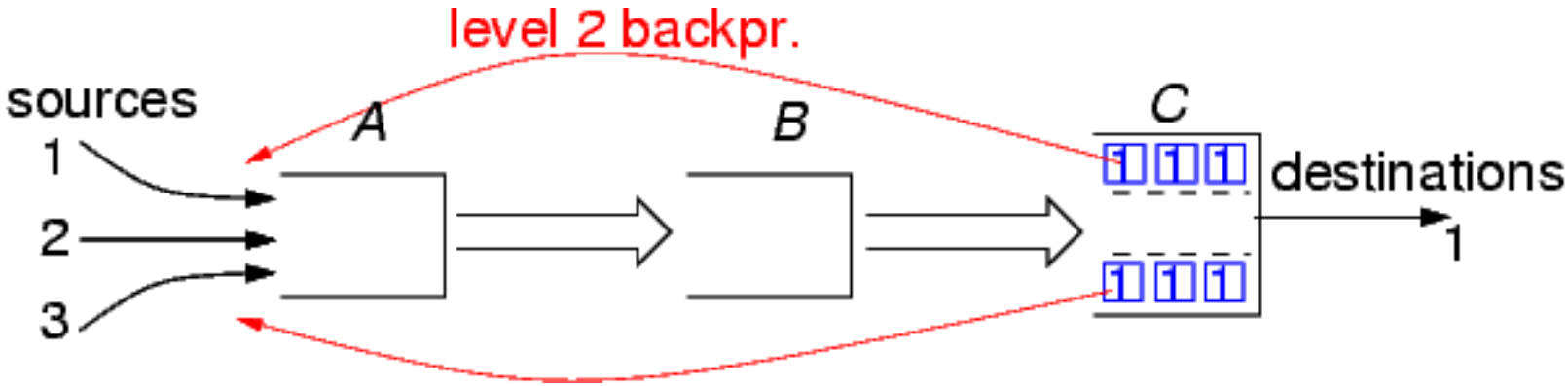
Inst. of Computer Science (ICS) FORTH -Hellas

3-Stage Clos Fabric Datapath



- Hop-by-hop indiscriminate backpressure
  - 1 queue per xpoint, i.e. N queues per switch element
  - 1 crosspoint buffer credit sent upstream per cell time
- B- to C-stage flow control used for rush cells
  - never exerted when all cells via request-grant

Hierarchical Backpressure



shared queues with multi-level backpressure

- conflicting cells accommodated in per-flow queues

  => other flows can progress

#queues (in front fabric-output) = N
…but for request implemented as counters