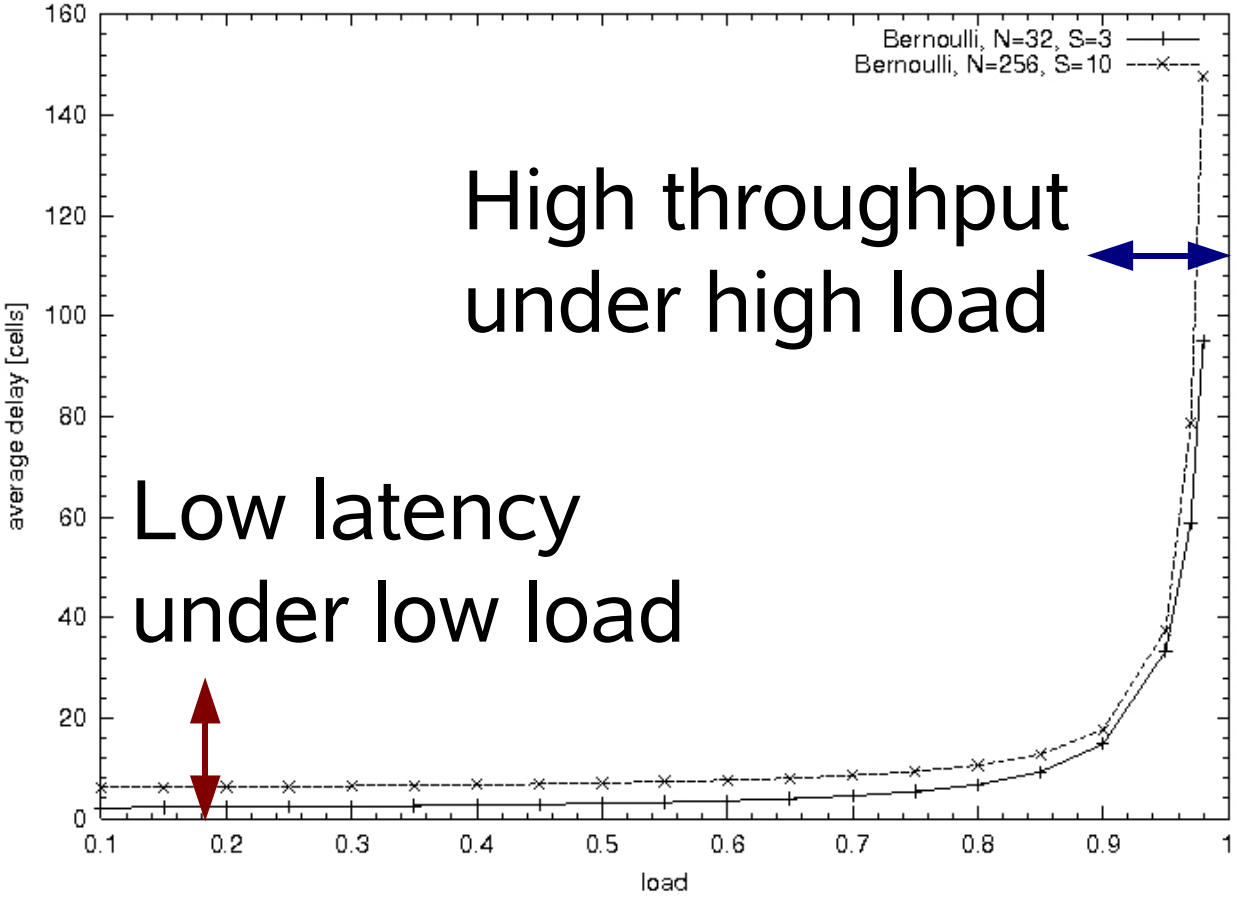
Low-Latency Scheduling in Large Switches

Wladek Olesinski
Hans Eberle
Nils Gura
Sun Microsystems Laboratories
Andres Mejia
Universidad Politecnica de Valencia

Context

- Large switch fabrics with hundreds of ports
- Designing single-stage switch architectures at Sun Labs based on a high-speed chip-to-chip I/O technology
- Scheduling is challenging
  - Most schemes exhibit quadratic growth in complexity (area/time/bandwidth) with #ports
  - High data rates (10Gb/s per port and up)
  - Short packets/cells

Conflicting Goals – Low Latency and High Throughput



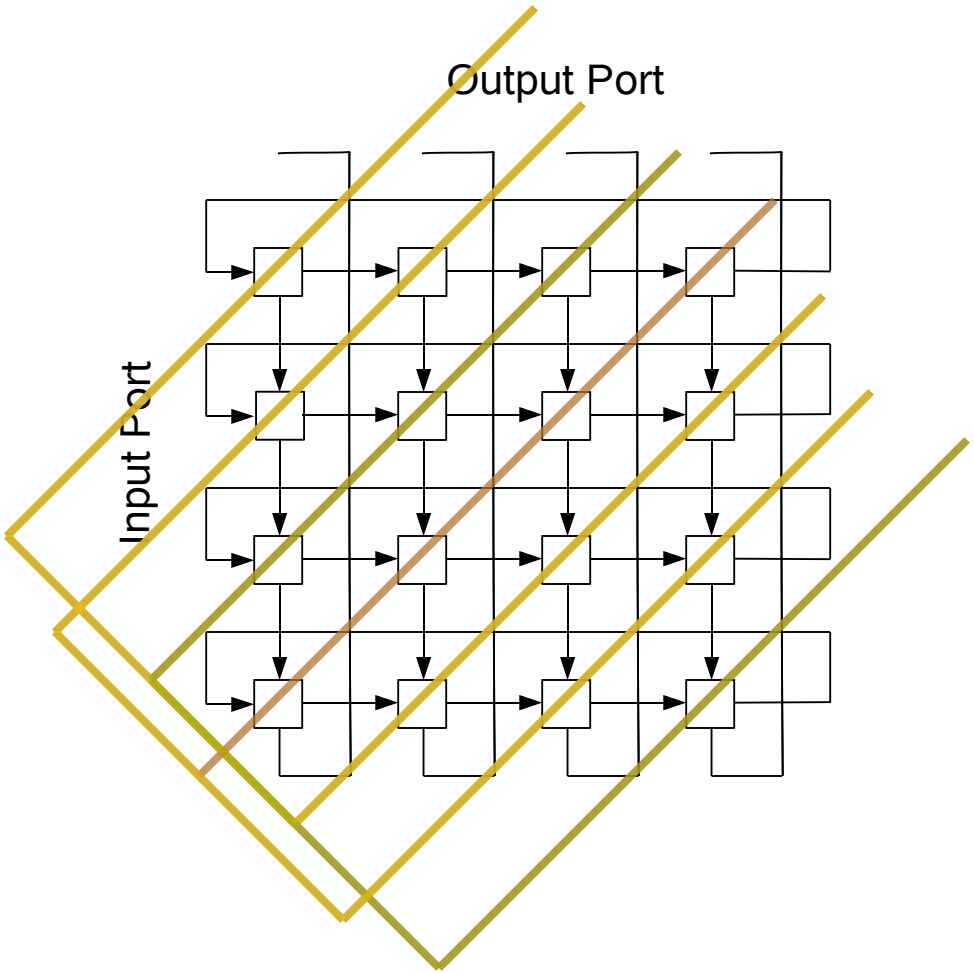
Existing Schemes

- Iterative algorithms (e.g., *PIM*, *iSLIP*, *DRRM*)
  - Multiple exchanges of requests and grants
  - Not sufficient for large switches because of
    - time and bandwidth
    - computational complexity
- Pipelining iterative schemes (e.g., *PMM*)
  - Subschedulers process several sets of requests concurrently
  - In every slot one of the subschedulers produces a schedule

PWWFA - High Throughput

- Parallel version of Tamir/Chi's wrapped wave front arbiter; PWWFA presented at HPSR 2007
- Scheduling time grows linearly with #ports
- Multiple subschedulers working in parallel to increase throughput
- Straightforward hardware implementation; $(\#ports)^2$ elements, but small and of regular structure

Parallel Wrapped Wave Front Arbiter

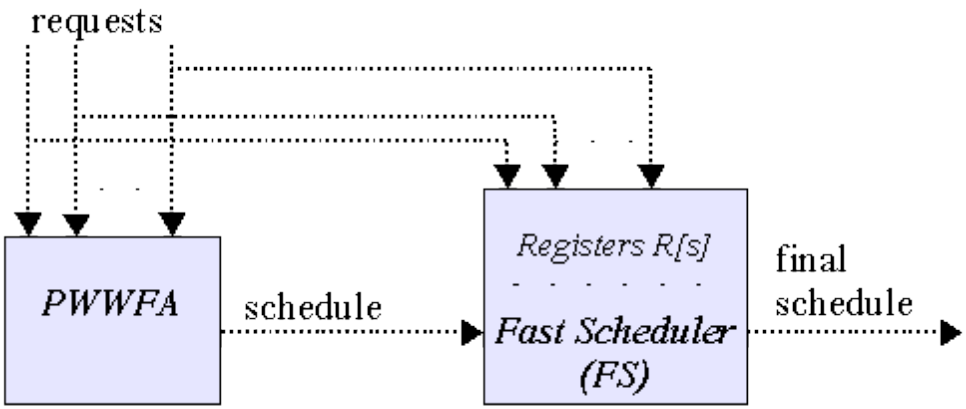


- Time *NT*: 1st schedule ready
- Time *NT*+*T*: 2nd schedule ready

...and so on. In the next 2 periods of *T*, the other 2 subschedulers produce schedules.

To provide fairness, subschedulers start at different waves in every scheduling cycle.

Fast Scheduler – Low Latency

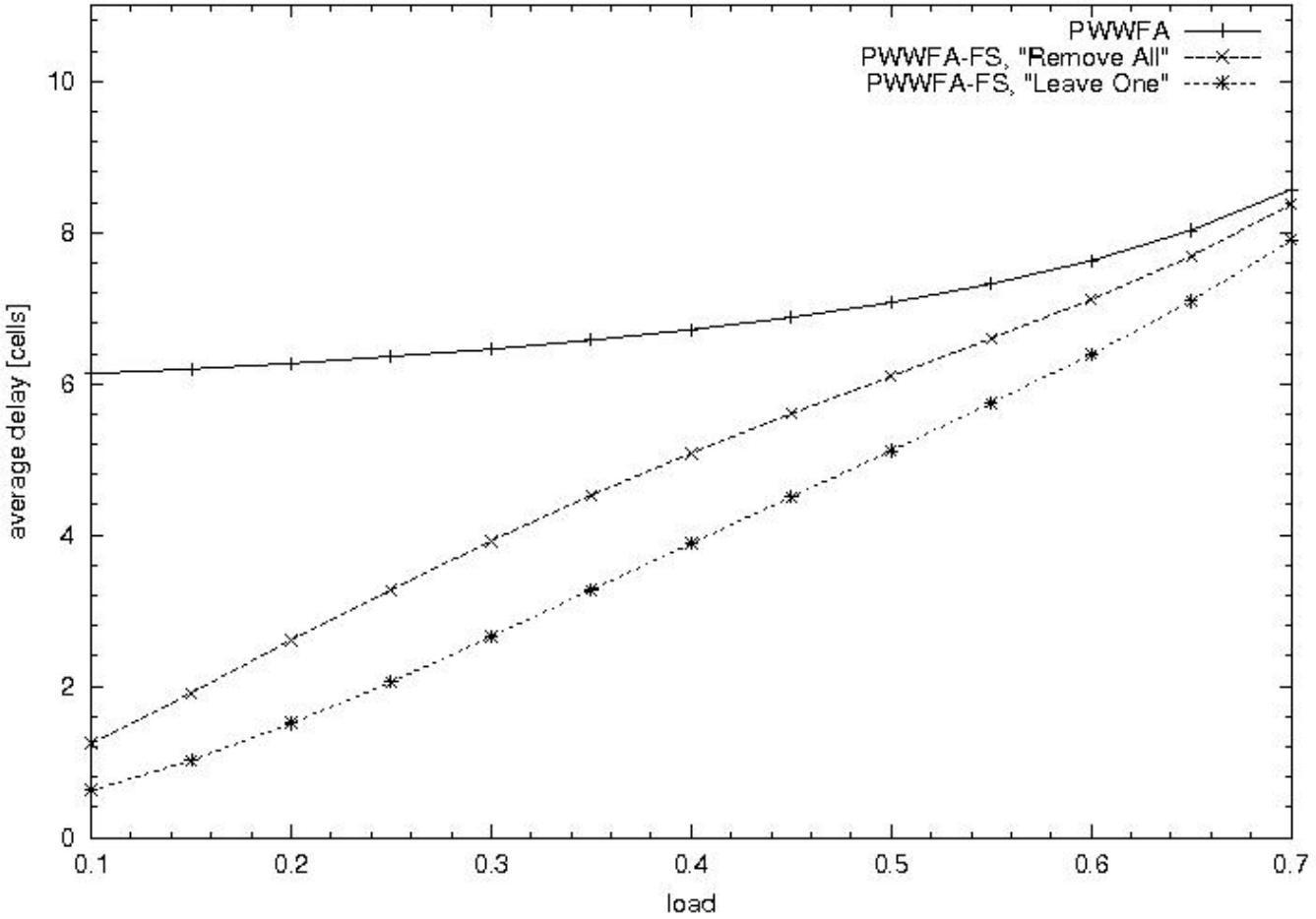


- Enhance PWWFA schedule with grants to most recent request
- Observed that scheduling conflicts are rare under low load
- Fast scheduler can thus be simple, but should have very low latency

Fast Scheduler – Two Schemes

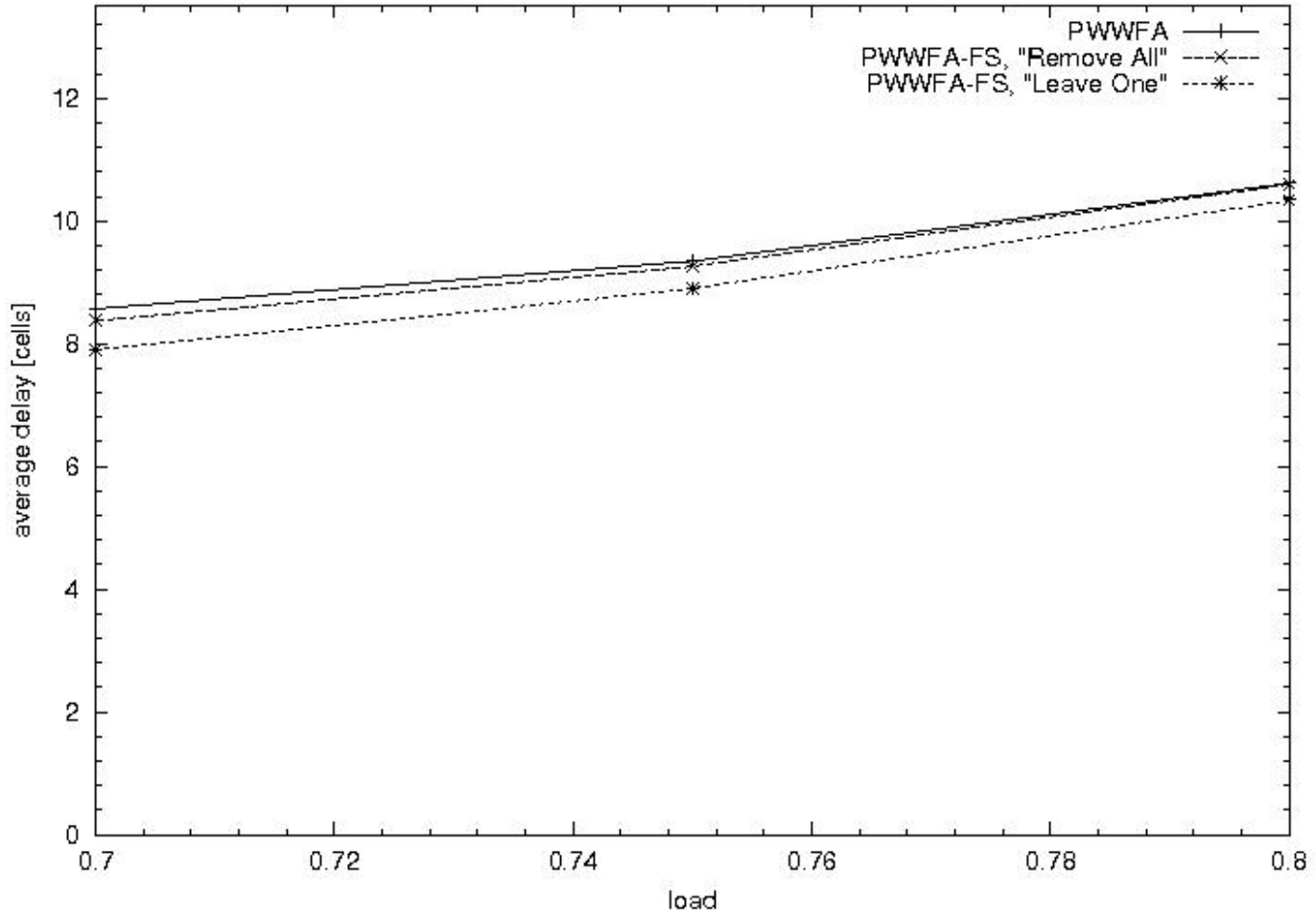
- Remove All
  - For a given output, give a grant if there is exactly one request
  - In case of conflict, defer all requests to PWWFA
- Leave one
  - In case of conflict, grant one of the conflicting requests and defer other requests to PWWFA

Performance – 10% to 70% Load



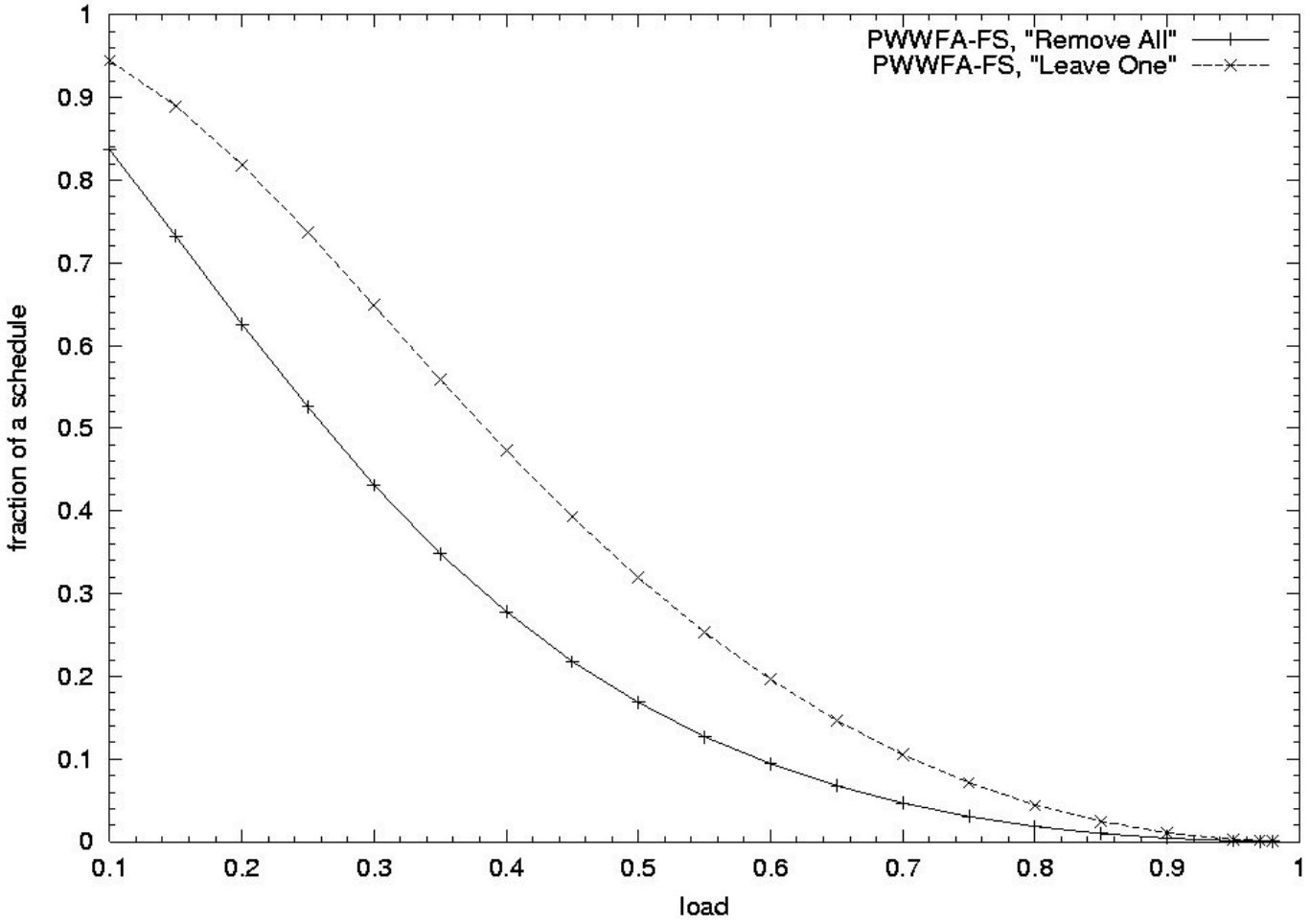
- Bernoulli traffic, N=256

Performance – 70% to 80% Load



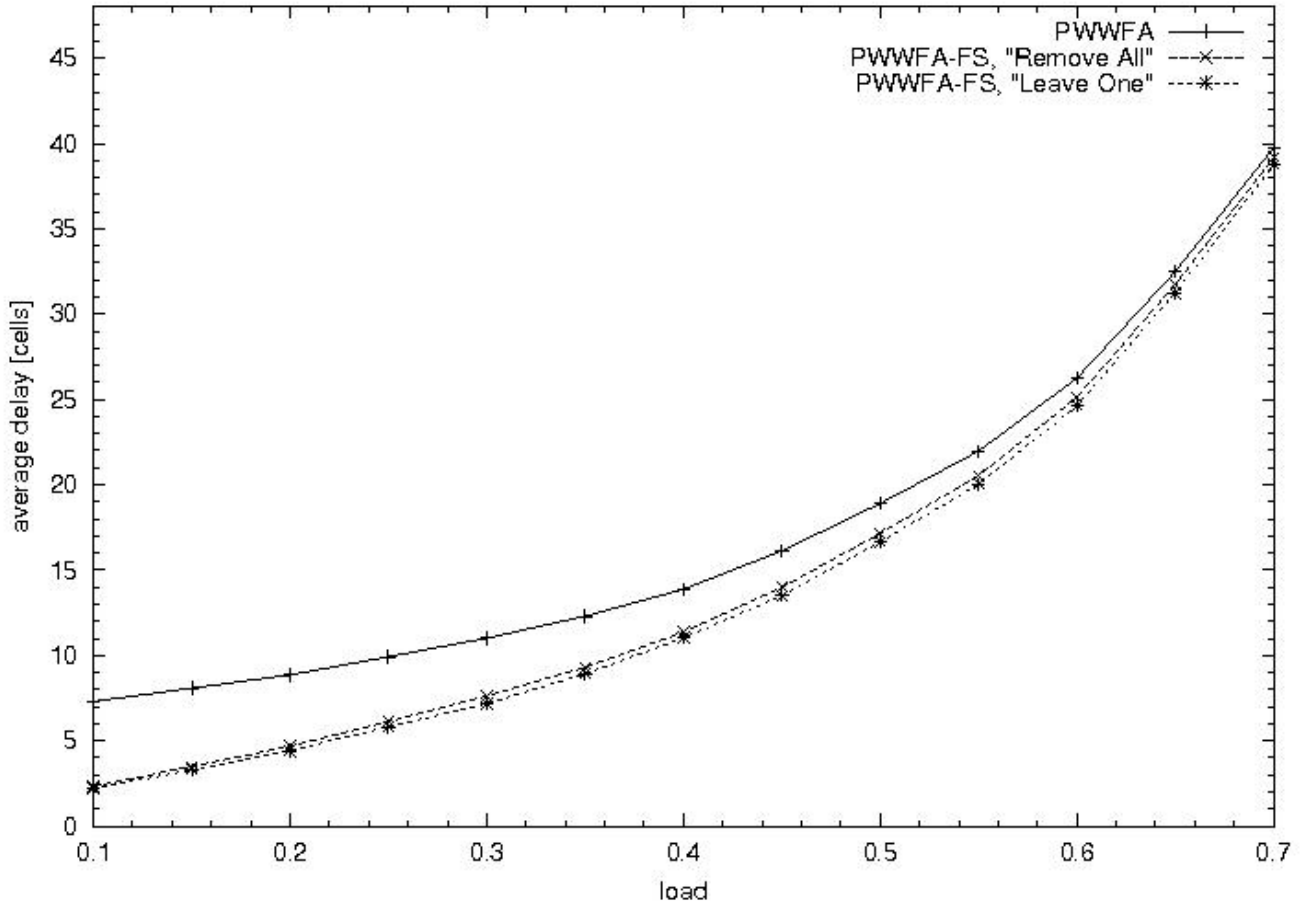
- Bernoulli traffic, N=256

Schedule fraction filled by FS



- Bernoulli traffic, N=256

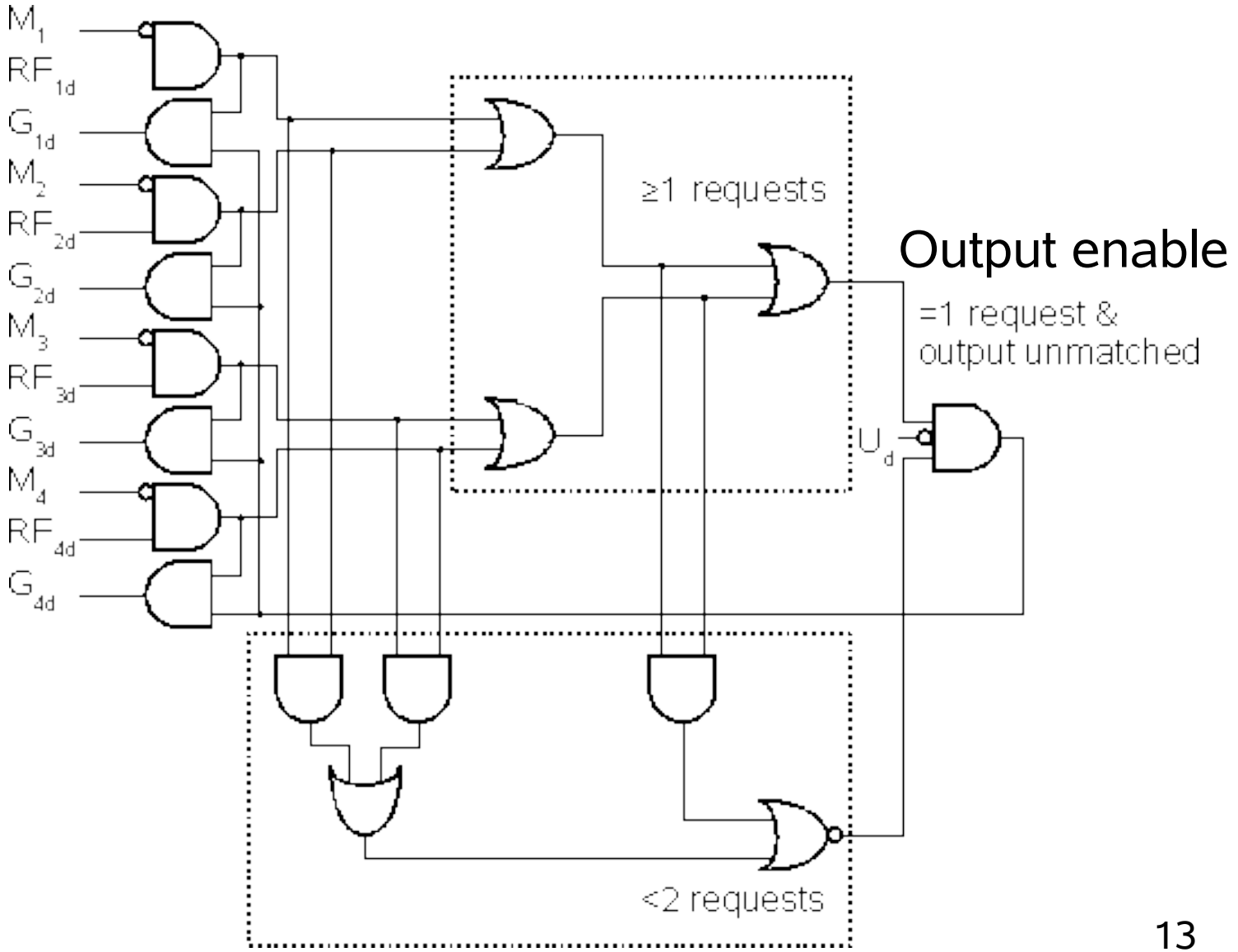
Performance – 10% to 70% Load



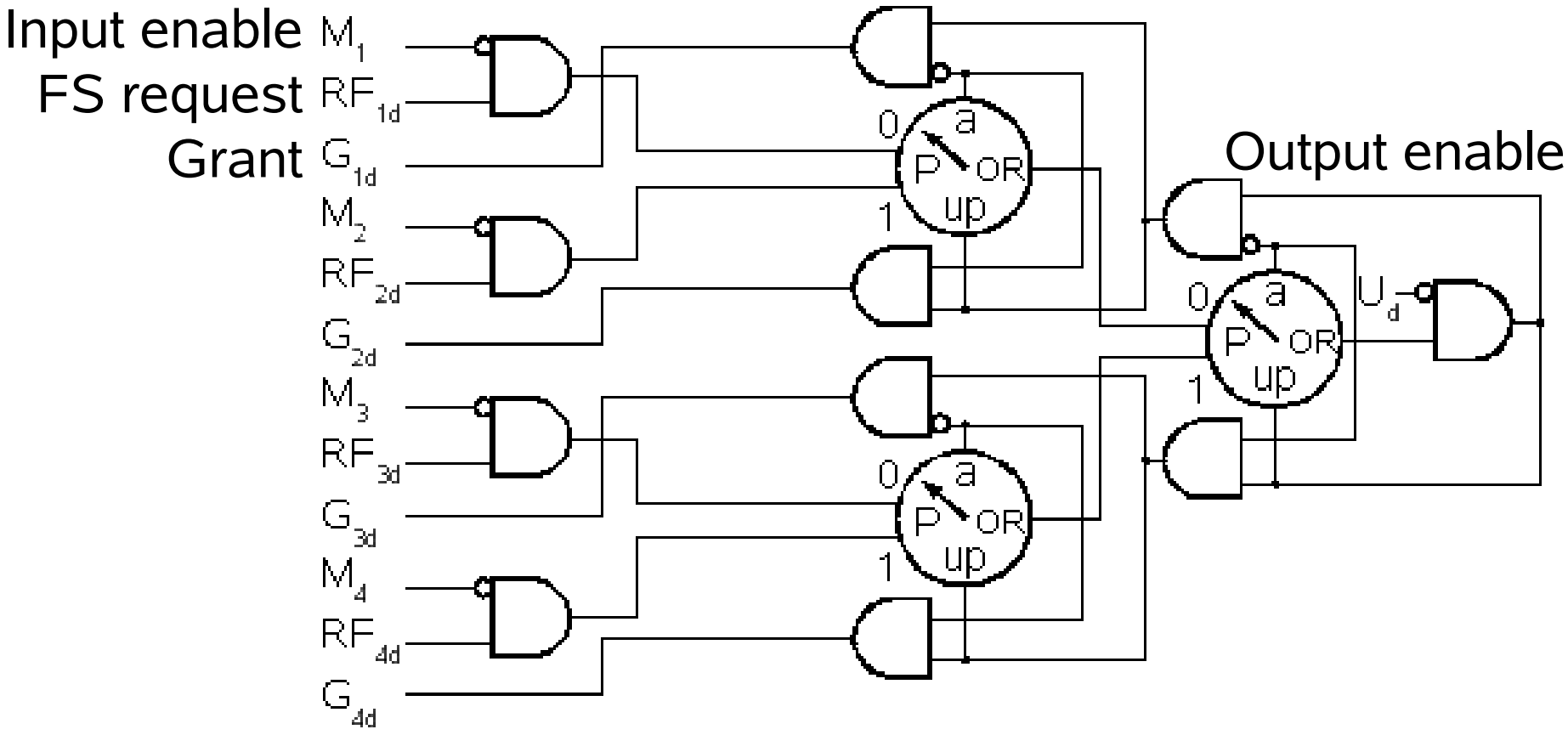
- On-off traffic, N=256

Implementation – Remove All

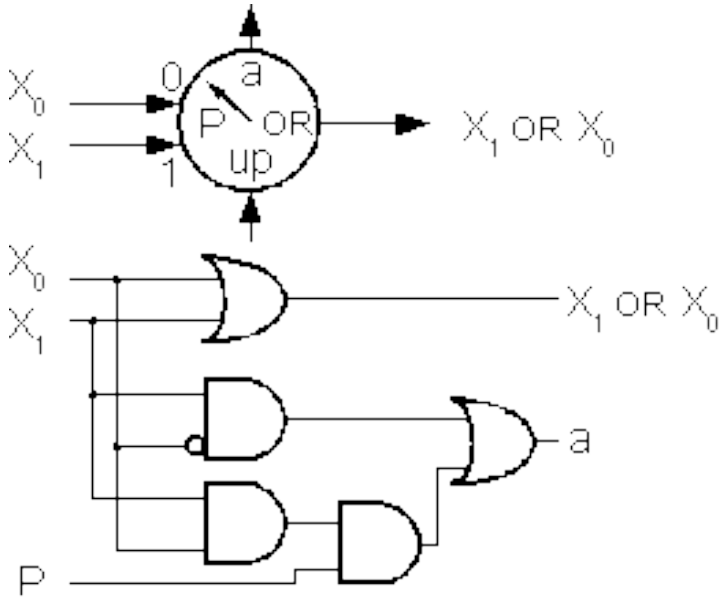
Input enable
FS request
Grant



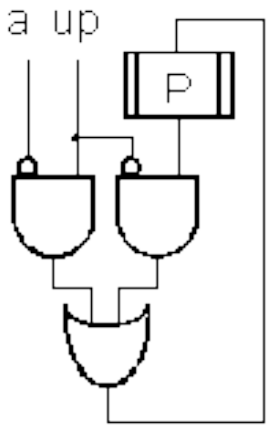
Implementation – Leave One



Implementation – Leave One



X_1	X_0	P	X_1 OR X_0	a
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1



up	a	P_{old}	P_{new}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Implementation - Complexity

- Remove All
  - > $6N^2$-3N 2-input gates for N ports
  - > Longest path 2 $\log_2(N)$+4, max. fanout N
  - > 392,448 gates for 256 ports
- Leave one
  - > $12N^2$-10N 2-input gates, N^2-N register bits for N ports
  - > Longest path 3 $\log_2(N)$+3, max. fanout 3
  - > 783,872 gates, 65,280 register bits for 256 ports

Conclusions and Future Work

- Conclusions
  - Presented hybrid PWWFA-FS scheduler that provides low latency and high throughput
  - Simulation results encouraging
  - Hardware implementation feasible with reasonable complexity
- Future work
  - Hardware implementation and characterization of the Fast Scheduler
  - Improve fairness

Nils Gura
Nils.Gura@sun.com