Using Memory and Random Sampling for Load Balancing in High-radix Switches

Soudeh Ghorbani Khaledi, Yashar Ganjali
Department of Computer Science, University of Toronto
{soudeh, yganjali}@cs.toronto.edu

Amin Firoozshahian
Hicamp Systems Inc.
aminf13@hicampsystems.com

***Abstract*—Dramatic increase in the size of communication networks as well as huge growth in data transmission rates have made designing high performance switches an essential part of building any interconnection network. Recent advances in fabrication technology have significantly increased the number of input and output ports in network switches. An interesting problem in designing such switches is how to balance the load among output ports of a switch when using adaptive routing algorithms. This problem is important for example in case of Clos networks and multi-stage load balanced switches; in the first stage of such a network/switch, packets from any input can be routed to any output port. Therefore, the decision of which output port to send the packets makes an important difference in the balance of the load in output ports and thus, in the throughput of the whole network. In practice, comparing the entire set of output queues to identify the best output port to direct a packet is prohibitively complex in the context of switches with a large number of ingress/egress ports (known as high-radix switches).**

In this paper we evaluate randomized schemes for balancing the load in such switches, both theoretically and via simulation. We prove that, despite being simple to implement, scheduling policies based on random sampling cannot guarantee stability when service rates are unequal. We also prove that simply adding one unit of memory to save the identity of least loaded queues in addition to few random samples can solve this problem: if no input port is oversubscribed, 100% throughput is achieved. We also present an extensive set of simulations to study the load distribution under different randomized scheduling algorithms with and without memory. We find that using as few as 2 or 3 random samples and one memory unit can result in balanced load in output queues regardless of low or high input loads and without any need for extra speedup.

I. Introduction

The size of communication networks and data transmission rates are constantly increasing. This makes designing high performance switches a must in building any interconnection network. In designing such switches, there are two main issues which need to be addressed: First, due to higher data transmission rates the time available for making routing decisions for each individual packet is very short. Therefore, routing decisions must be made very fast. Second, large networks require high-radix switches, *i.e.*, switches with many input/output ports. Coordination among all the ports in such a switch is very costly in terms of complexity and cycle time.

To remedy this situation the switching task can be parallelized by using multi-stage architectures that use lower speed and/or smaller switches operating independently and in parallel. Clos networks [1] are classical example of multi-stage network switches that are very attractive substitutes for crossbar switches since they have fewer cross-points than a complete crossbar and therefore scaling them is much easier. Although Clos networks were first designed for circuit switches, they have been widely used in packet switching networks because of their nice properties like scalability, path diversity, low diameter, and their simple structure. Parallel packet switch (PPS) architectures [2], Distributed Switch Architecture (ADSA) [3], and load balanced Birkhoff-von Neuman switches [4] are other examples of multi-stage architectures in which arriving traffic is demultiplexed over lower speed switches, switched to the correct output port, and then recombined before leaving the system.

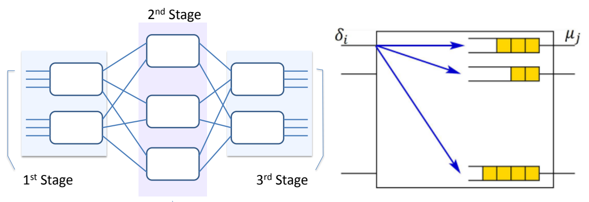
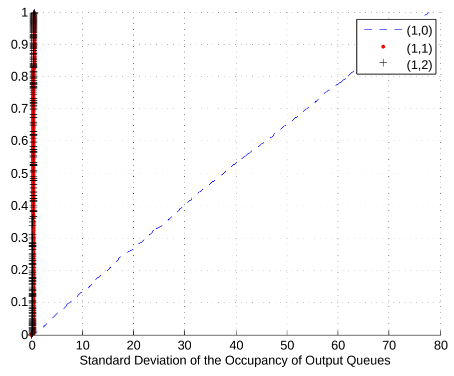


Fig. 1. Load balancing in multi-stage switches/Clos networks.

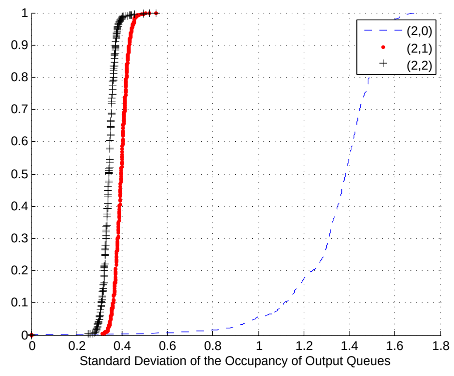
A critical issue to be addressed in distributed switch architectures is the design of simple, fast and efficient load-balancing scheduling algorithms for the first stage switches, so that they can balance the load by distributing the incoming packets evenly among the ports of the middle stage. The middle stage switches, then, route packets to their corresponding destination ports.

As an example, let us consider a packet p going from input port i to output port j. The switch in the first stage which is connected to the input port i can route the packet to any of the middle stage switches. No matter which middle-stage switch receives the packet p, it can route it to the switch in the third stage connected to the output port j. Since there are k middle stage switches, the switch has a path diversity of k. We note that the path from any middle stage switch to the output port is unique (Figure 1(left)).

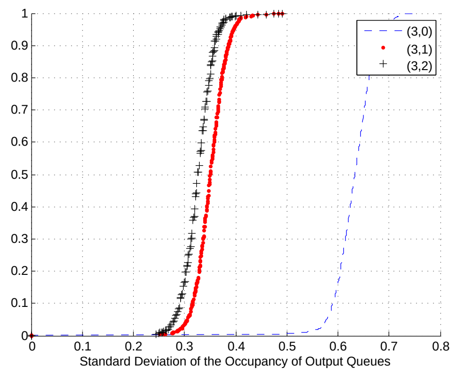
Clearly, performance of a multistage switch architecture highly depends on how packets are routed, *i.e.* which middle stage switch is used to route each packet. If choosing the



(a) d=1, m=0,1,2, 256*256 Switch



(b) d=2, m=0,1,2, 256*256 Switch



(c) d=3, m=0,1,2, 256*256 Switch

Fig. 2. CDF of Standard Deviation of Output Queues' Occupancy over Time for (d,m) Policies for Heavy Traffic and No Speedup For a 256*256 Switch.

middle stage is done inappropriately and too many packets are forwarded towards the same switch in the middle stage, the throughput of the network is reduced because of the congestion in the middle stage switch. To avoid this problem, each switch in the first stage of the network must try to distribute the load evenly on its output ports (Figure 1(right)).

If all switches in the first stage can balance the load among their outgoing ports, the total load will be balanced in the middle stage and the throughput of the whole network is maximized. Note that if the service rate is identical for all the output ports, a simple round robin approach could achieve balanced load. Hence, we address the more challenging case where the service rates can vary among the output ports. To have perfect load balancing, there are two intertwined problems to solve. First, each input port needs to choose among a large list of output ports (for N ports, this is of time complexity $O(N)$). Second, we need an explicit or implicit coordination mechanism among all input ports to avoid all inputs directing their packets to the same output, as this leads to huge bursts and load imbalance in the network.

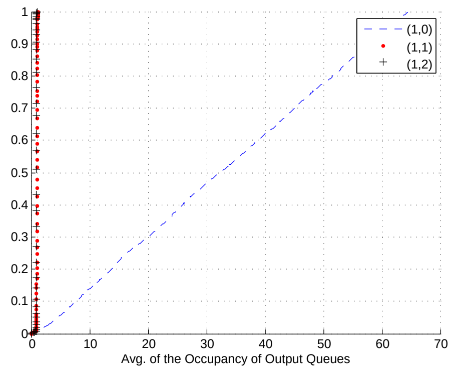
To mitigate the cost of comparing the length of all output queues and avoid communication overhead, randomized version of scheduling might be considered: whenever a packet is arrived at any input port, it is assigned to the least loaded of d randomly chosen output queues, where $d \ll N$.

The classic load balancing problem in the literature (usually referred to as the *supermarket* models), considers a single input and many output queues as opposed to multiple inputs in our case. For that problem, it is well-known that with a small amount of choice $d = 2$ load balancing performance greatly improves over random placement $d = 1$ [5]. However, as we prove in Theorem 1 in Section 2, in our problem, i.e., in the switches under Bernoulli i.i.d. packet arrival processes for admissible traffic and when no input port is oversubscribed, scheduling by using only randomization can lead to instability. This problem, alas, cannot be resolved by increasing the number of samples: even for $d = O(N)$, where N in the number of outputs; there exists admissible traffic for which the system becomes unstable, unless $d = N$.

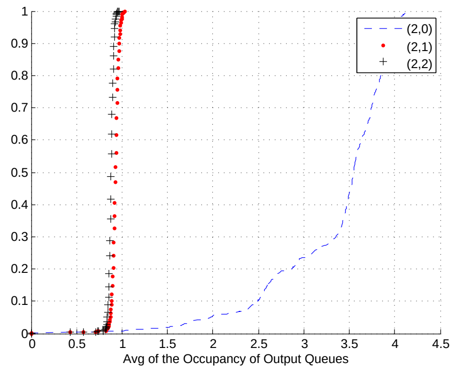
On the positive side, we show that there is a simple way around this problem to guarantee stability. We theoretically prove that deploying a single unit of memory for each input port in which that port saves the identity of the output port with the shortest queue length from the previous time slots makes the system stable.

We also present simulations that show random sampling with memory leads to nearly perfect load balance in system and can improve the performance of the system, e.g., it decreases the average and standard deviation of the occupancy of output queues. Our simulations show that regardless of the input load (light or heavy) and independent of the speedup of output queues (how many packets they can receive in each time slot) nearly perfect load balancing and acceptable latency is achievable with using very small number of samples, e.g., $d = 2 \; or \; 3$, and a single unit of memory for each input port.

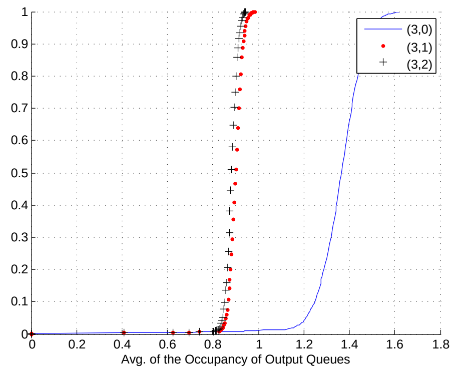
The organization of the paper is as follows. In Section



(a) d=1, m=0,1,2, 256*256 Switch



(b) d=2, m=0,1,2, 256*256 Switch



(c) d=3, m=0,1,2, 256*256 Switch

Fig. 3. CDF of Avg. of Output Queues' Occupancy over Time for (d,m) Policies for Heavy Traffic and No Speedup For a 256*256 Switch.

2, the model employed for the theoretical analysis of the system and random policies are described. We also prove that scheduling policies based only upon random samples are unstable. Moreover, for admissible i.i.d. arrival processes, stability is proved for (d, m) policies when $m \geq 1$. In Section 3, we explain how the policies are evaluated. The setup, components and results of our simulation are also presented. We provide an overview of related work in Section 4. The paper concludes in Section 5.

II. Load balancing in high-radix switches

We consider an $N \times N$ combined input output queued switch with FIFO queues in which the arrivals are independent i.e., arrivals at each input are independent and identically distributed(i.i.d) and arrival processes at each input are independent of arrivals at other inputs. We further assume that arriving packets are of fixed and equal length. Traffic is also assumed to be admissible, i.e., $\sum_{i=1}^{N} \delta_i \leq \sum_{i=1}^{N} \mu_i$, where δ_i is the arrival rate to input port i $(1 \leq i \leq N)$ and μ_j is the service rate of output queue j $(1 \leq j \leq N)$.

(d, m) *Scheduling Policy:* We consider randomized scheduling that at each time slot, every input port chooses d random outputs out of possible N queues, finds the queue with minimum occupancy between these d samples and m least loaded samples from previous time slot, and routes its packet to it. At the end of each time slot, an input port will update the content of its m memory units with the identity of the least loaded output queues. In case of the contention for any given output port, that port will randomly choose K of the input ports, where $1 \leq K \leq N$, and other input ports will be blocked. Such policies will be called (d, m) policies.

A. *Load balancing using random sampling without memory*

First, we consider $(d, 0)$ policy, i.e., the algorithm in which every input port chooses d random outputs out of possible N queues, finds the queue with minimum occupancy between them and routes its packet to it. By Theorem 1, we prove that such algorithm cannot guarantee stability.

Theorem 1: For unequal service rates and admissible i.i.d. arrival processes, $(d, 0)$ policy cannot guarantee stability for any arbitrary number of samples d, where d is less than the number of outputs.

Proof: Let δ_i be the arrival rate to input port i, and μ_j be the service rate of output queue j. Now consider output queue N. For any input port, the probability that it chooses output queue N as a sample is $\frac{d}{N}$. So, maximum arrival rate to queue N is $\frac{d}{N} \times \sum_{i}^{N} \delta_i$. Thus, the minimum arrival rate to the rest $N-1$ output queues is

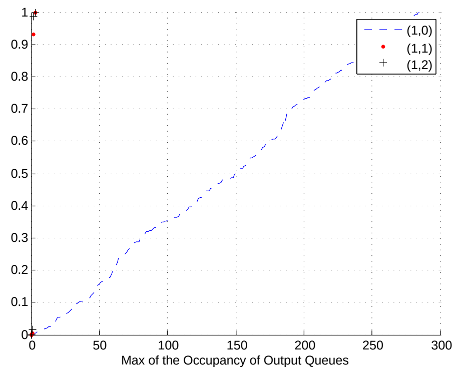
$$\delta_{N-1} = \sum_{i}^{N} \delta_i - \frac{d}{N} \times \sum_{i}^{N} \delta_i = \sum_{i}^{N} \delta_i \times (1 - \frac{d}{N}).$$

Clearly, if δ_{N-1} is larger than the sum of the service rates of these $N-1$ queues, the system will be unstable. ∎

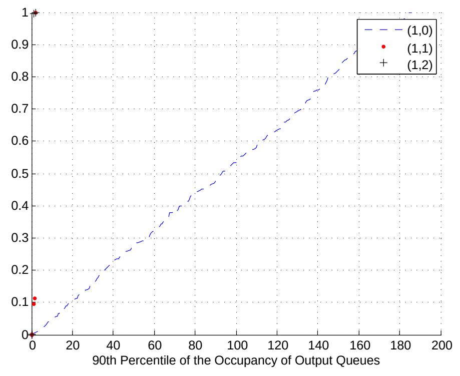
It should be noted that the argument does not hold

1) when there are some restriction regarding the service rates, like when the service rates are equal. Or,
2) when $d = N$.

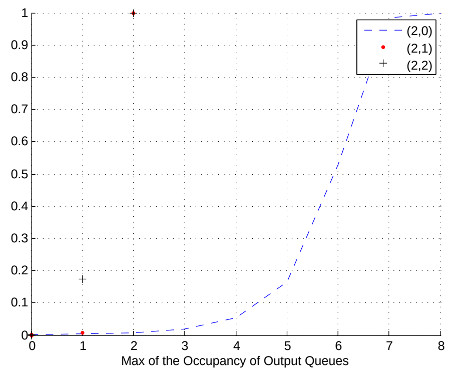
These two special cases, however, are of little interest. Since the former opts out some admissible patterns of traffic, and the latter will nullify the benefit of randomization. The results of our experiments suggest that the system will perform well with $d \ll N$.



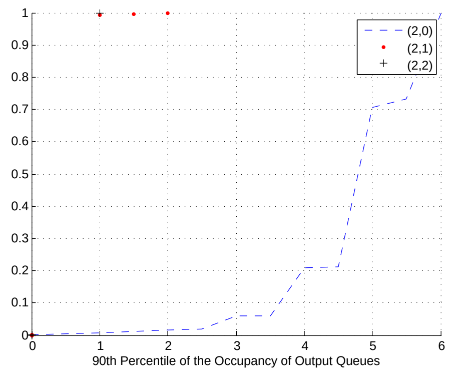
(a) d=1, m=0,1,2, 256*256 Switch



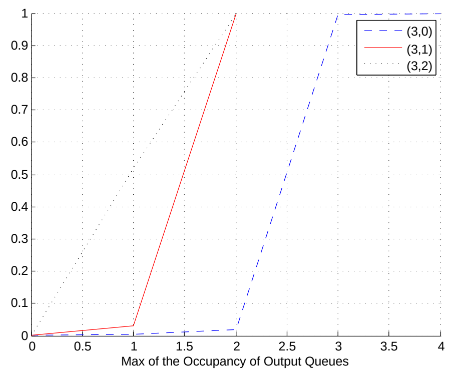
(a) d=1, m=0,1,2, 256*256 Switch



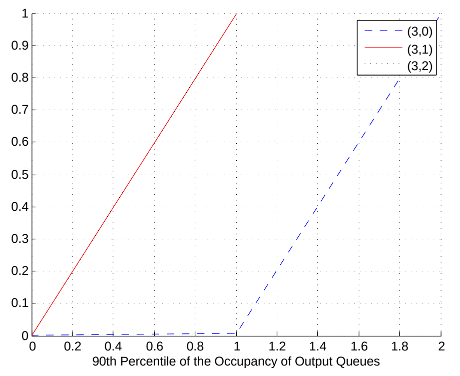
(b) d=2, m=0,1,2, 256*256 Switch



(b) d=2, m=0,1,2, 256*256 Switch



(c) d=3, m=0,1,2, 256*256 Switch



(c) d=3, m=0,1,2, 256*256 Switch

Fig. 4. CDF of Max of Output Queues' Occupancy over Time for (d,m) Policies for Heavy Traffic and No Speedup For a 256*256 Switch.

Fig. 5. CDF of 90th Percentile of Output Queues' Occupancy over Time for (d,m) Policies for Heavy Traffic and No Speedup For a 256*256 Switch.

B. Load balancing using random sampling with memory

It was shown that randomized policy cannot guarantee stability without using unit of memory. By Theorem 2, we prove that having only one sample and using a single memory for each input port ensures stability for all uniform and non-uniform independent arrival processes.

Theorem 2: $(1,1)$ policy is stable for all admissible i.i.d. arrival processes.

To prove that the algorithm is stable, using the result of Kumar and Meyn [6], we show that for an $N \times N$ switch scheduled using the $(1,1)$ policy, there is a negative expected single-step drift in a Lyapunov function, V. In other words,

$$E[V(n+1) - V(n)|V(n)] \leq \epsilon V(n) + k,$$

where $k > 0$ and $\epsilon > 0$.

We do so by defining $V(n)$ as:

$$V(n) = V_1(n) + V_2(n),$$

where

$$V_1(n) = \sum_{i=1}^{N} V_{1,i}(n),$$

$$V_{1,i}(n) = (\tilde{q}_i(n) - q^*(n))^2,$$

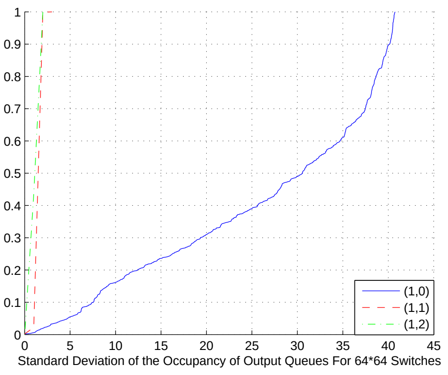
$$V_2(n) = \sum_{i=1}^{N} q_i^2(n).$$

And $q_k(n)$, $\tilde{q}_k(n)$ and $q^*(n)$, respectively, represent the length of the k-th output queue, the length of the output queue chosen by the input i, and the length of the shortest output queue in the system under policy $(1,1)$ at time instance n.

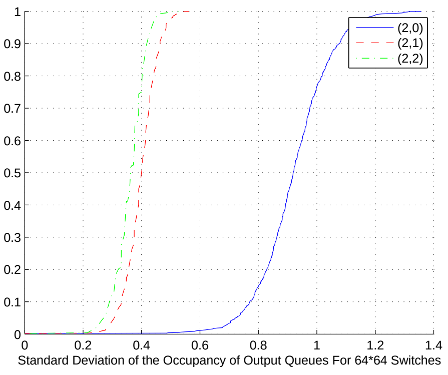
Details of the proof can be found in the Appendix.

III. Simulations

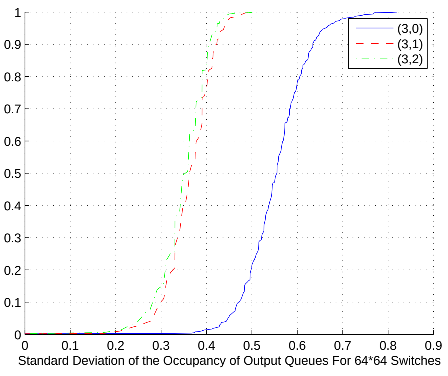
Stability of (d,m) policies, where $m > 0$ is proved in the previous section. Performance of different (d,m) policies are evaluated via simulation for different settings with different traffic load and speedup. In order to evaluate these policies, a reference scheme, hereafter called *ideal*, is used in which each input port will check the queue length of all of the output ports, and chooses the shortest one to send its packet to. It is worth emphasizing that this is done in parallel i.e. input ports cannot choose the output ports based on the results of the decision made by any other input ports. Moreover, for *ideal* policy, it is also assumed that any output queue can receive packets from all the input ports during each time slot, so that there will be no contention for output ports. It should be noted, however, that large number of packets that might be destined to a single output port may cause the so called *ideal* policy not to perform literally ideal in all scenarios.



(a) d=1, m=0,1,2, 64*64 Switch



(b) d=2, m=0,1,2, 64*64 Switch



(c) d=3, m=0,1,2, 64*64 Switch

Fig. 6. CDF of Standard Deviation of Output Queues' Occupancy over Time for (d,m) Policies for Heavy Traffic and No Speedup For a 64*64 Switch.

A. Setup

1) Components: To keep the simulation setup simple, we have chosen to model only one node of the Clos network. Each node is an $N \times N$ combined input output queue switch. For the simulation, we ran the experiments on 256×256 and 64×64 switches. Each node contains an arbitrator, which decides which input queues have the highest priority when contending for an output port. In the experiments, these queues are chosen randomly. The switch fabric in each node is a crossbar that routes packets from the input queues to the output queues. This is done in parallel for all input ports. the router in the node decides which output port to send the packet to for every packet that is at the head of the input queues.

We consider time to be slotted and that packets can arrive at the switch at the beginning of each time slot. Furthermore, we assume that at most one packet arrives at an input port and at most one packet departs from an output port during each time slot. Number of packets that can enter an output queue is also limited considering the speedup as explained in Speedup Section.

Packets may have to wait at an input port if they lose the contention for the output port they choose, and may have to wait at an output port before departing. We also consider the capacity of input and output buffers to be infinitive.

2) Simulated Traffic: We assume that sum of arrival rates is smaller or equal to sum of service rates, and no input port is oversubscribed. For evaluating the policies, we consider 2 cases: when the system is heavily loaded, e.g., $avg(\delta_i)/avg(\mu_j) = 0.98$, and when the system is lightly loaded, $avg(\delta_i)/avg(\mu_j) = 0.5$, where δ_i and μ_j represents the arrival rate at input port i and service rate at output port j, respectively. We also assume that packets arriving to input ports have independent Bernoulli distributions.

Since we are simulating a single node of a network, in order to get more realistic results we assign different service rates between zero and one. In a real system this is automatically done because of the backpressure from other switches in the system. Likewise, arrival processes are assumed to have unequal rates to mimic nonuniform traffic.

3) Speedup: In the randomized scheduling that we study, several input ports may compete for a specific output port. Our incentive for having faster internal fabrics actually stems from the need to be able to transfer more than a packet to each output port in a given time slot. Therefore, rather than enabling both input and output ports to respectively send and receive K packets during each time slot, which is feasible in switches with speedup of K, we are interested in the case that only output ports can receive up to K packets in each time slot, and akin to systems with no speedup, input ports are restricted to send out at most a single packet per time slot. Alternatively, by speedup of K, we mean that in each time slot, a given output port can grant permission to up to K of the input ports to destine their packets to it, while an input port can send out no more than a packet. Imposing the mentioned restriction on input ports causes us to only partially take advantage of potentials of switches with speedup. However, the results of our experiments show that this restricted speedup still results in great improvement in system performance in terms of average latency of packets when the system is heavily loaded.

In the experiments, we consider the case that the switch does not have speedup, and when it has speedup of 2.

B. Simulation results

1) Heavy Traffic and No Speedup: Instability of $(d, 0)$ policies makes exploiting memory inevitable. We dig closer into queue occupancy distributions when different number of memory units are used, and find that for small number of samples, d, taking advantage of a single memory unit considerably ameliorates balancing the load among output queues. As the number of samples increases, this effect gradually fades away. This is depicted in Figures 2, 3, 4, and 5 for $d = 1, 2, 3$, and for standard deviation, average, maximum and 90th percentile of output queues's occupancy for a 256*256 switch, and in Figure 6 for standard deviation of output queues' occupancy of a 64*64 switch. The figures suggest that the first memory unit causes the standard deviation, average, maximum, and 90th percentile of the policies with small number of sample to drop significantly which signals that balancing the load among the output ports is improved, but using the 2nd unit of memory does not have such strong effect. For other settings like light load or speedup of 2, and other values of d similar trend is observed.

These results suggest that one unit of memory for each input port should be adequate. To figure out the appropriate number of samples, we measured statistical metrics of input and output queues' occupancies for several $(d, 1)$ policies, $(256, 0)$ policy and *ideal* policy. Clearly, when the number of samples is equal to the number of input queues, 256 in this case, using memory will not alter the performance of the scheduling in any way, since at any time step the inputs will have all the information for determining the shortest output queues. Note that *ideal* and $(256, 0)$ policies differ in the speedup of system (which is N for the former and 1 for the latter in this setting).

As demonstrated in Figures 7 and 23 (a), the considered random policies all outperform the *ideal* policy by providing considerably lower average, standard deviation, max and 90th percentile of output queue occupancy. To have a better judgement of the random policies relative to each other, *ideal* policy is omitted from Figure 8 which demonstrates that nearly perfect load balancing in achievable with using as few as 2 or 3 samples and a single memory unit for each input port.

In this setting, for random policies with several samples and m=1, average latency of packets will drop to nearly half of the latency for *ideal* policy. A number of these policies and their corresponding latency are mentioned in Table 1.

2) Heavy Traffic and Speedup of 2: Same experiment is repeated for the setting that speedup of switch is equal to 2, and very similar results are obtained: as Figures 9 and 23 (b) show the random policies all outperform the *ideal* policy. Among the random policies, as demonstrated by Figure 9, using only 2 or 3 samples and a single memory unit for each input port is sufficient to have balanced load and very small average output queue size. Note that although average and standard deviation of different policies have slightly increased compared to the setting with heavy load and no speedup, as Table 1 shows, average latency falls sharply to less than 15% of its corresponding value when the system does not have speedup for policies $(2,1)$ and $(3,1)$. Moreover, standard deviations are still very low. For $(3,1)$, for instance, standard deviation of the lengths of output queues is always under 0.8 which suggests that load is perfectly balanced among output ports.

3) Light Traffic and No Speedup: When the system is not heavily loaded, balancing the load is not that much of a challenge. As $\rho = avg(\delta_i)/avg(\mu_i)$ diminishes, *ideal* policy approaches the optimum case for balancing the load among outputs. Yet, having the speedup of N and comparing the length of all queues make it impractical. Measured metrics related to output queues' occupancy in this case, depicted in Figures 11 and 23 (c) suggest that random policies will perform well in this setting as well when switch has no speedup. Also, as Table 1 shows, average latency for random policies is around 1/10 of its value for *ideal* case.

4) Light Traffic and Speedup of Two: Similar results are observed when the speedup of switch is 2, as shown by Figures 12 and 23 (d). In this case too, small number of samples and memory units guarantee low standard deviation of occupancy of output queues, and a relatively small number as the upper bound of queue occupancy. Average latency is only slightly diminished compared with the setting with no speedup. This should look trivial, considering that the switch is lightly loaded. The results suggests that when system is not heavily loaded, having speedup will not be of that much help in ameliorating the performance of switch.

An Observation of Occupancy of Input Queues:
For heavy traffic, when the system has no speedup, introducing memory increases the standard deviation, average, maximum and 90th percentile of input queues' occupancy, as shown in Figures 13,14,15 and 16. However, as the number of samples increases, this increase in average and maximum occupancy gradually decreases. Intuitively, when memory is used for small number of samples, the probability that more input ports identify and target the same output ports increases. This will result in contention for a limited number of output ports, and since the switch does not have speedup, many of these input ports will lose the contention and be blocked. Consequently, the average and maximum input queues' length increase in the system. However, these 2 numbers always remain under 100 and 275 packets, respectively, in our experiments (Figure 17).

For heavy traffic when the switch has the speedup of 2, the average and maximum input queues' occupancy are very small for small values of d, e.g., in (1,1), (2,1) and (3,1) policies (respectively, under 3 and 20 packets in all time for all these policies in our experiments, as shown by Figure 18). However, by increasing the number of samples these values increase. For light load (with or without speedup), as demonstrated in Figures 19 and 21, only for *ideal* policy input queues can become really large. As shown in zoomed Figures 20 and 22, all (d,m) policies (for small or large values of d) exhibit very small values for average and maximum input queues' occupancy, with no considerable difference between these values for different ms and ds.

In conclusion, in addition to the balanced load among output ports, $(d,1)$ policies with small number of samples, also provide vary small average and maximum input queues' occupancy for light traffic, and for heavy traffic when the switch has the speedup of 2, and reasonably small values for heavy traffic in switches without speedup.

IV. Related Work

The urgent need to build high speed switches has motivated many researchers to propose novel and high performance multi-stage switch architecture that scales up well. As an example of such attempts, Iyer *et al.* propose PPS architecture [2]. Iyer and McKeown show that the initial centralized approach used in PPS is impractical due to the large communication complexity that it requires [7]. They modify the PPS by allowing a small buffer, that runs at the line rate, in the multiplexer and the demultiplexer [7]. C.S. Chang *et al.*[4] have also proposed Birkhoff-von Neumann switch architecture with two-stage switching fabrics and one-stage buffering in which the 1st stage performs load balancing, while the 2nd stage is a Birkhoff-von Neumann input-buffered switch that performs switching for load balanced traffic. Although The on-line complexity of the switch is O(1), for the number of permutation matrices needed in their switch, the complexity is $O(N)$. ADSA architecture, proposed by Wang *et al.* [3] introduces a small communication overhead of $log(P) + 2log(N)$ extra bits per cell in addition to the cell payload, where P is the number of parallel switch elements and N is the number of communication overhead.

We believe that balancing the load in this context has a much simpler solution: randomization. While simplicity of implementation is clear, we proved the stability and evaluated the actual performance with simulation. The idea of using randomization and memory and the approach taken to theoretically prove the stability originate from previous work about load balancing.

Randomization in standard load balancing problem is generally considers placing n balls into n bins by choosing d samples for each ball independently, uniformly at random and sequentially, and placing each ball in the least loaded of its chosen bins [8]. Apart from the ball and bin problem, another model, called supermarket model, is defined for dynamic settings in which arrivals occur according to a rate $n\lambda(\lambda < 1)$ Poisson process at a bank of n independent rate exponential servers, with the ith server having service rate μ_i[8]. It is well established that for these problems, choosing a small number of samples significantly improves performance over random placement $d = 1$. Mitzenmacher *et al.* show that same performance gain can be achieved by deploying a small amount of memory [8]. Shah and Prabhakar introduce using a memory and prove the stability for the problems for $(1,1)$ policy [9]. Although supermarket problem is different

TABLE I
AVG. LATENCY EXPERIENCED BY EACH PACKET.

Policy	(1,1)	(2,1)	(3,1)	(100,1)	ideal/sp=256
heavy traffic/sp=1	0.1710	0.1648	0.1637	0.1674	0.2963
heavy traffic/sp=2	0.0277	0.0214	0.0228	0.0429	0.2963
light traffic/sp=1	0.0142	0.0140	0.0138	0.0139	0.1606
light traffic/sp=2	0.0140	0.0137	0.0137	0.0138	0.1606

from routing packets in high radix switches in that it has only a single arrival process and the arrivals will be served sequentially whereas in switches all arrivals in input ports are served simultaneously and in parallel, we take the same approach as Shah and Prabhakar [9] to show that the results similar to stability in supermarket problem hold valid in our switching problem as well.

V. CONCLUSION AND FUTURE WORK

In balancing the load in the output ports of a high-radix switch there are three main issues to be considered: First, in distributing the load from input ports to output queues, higher priorities must be given to shorter queues, i.e. more load must be forwarded to them. Second, the load-balancing scheme must not allow too many input ports to choose the same output queue since in this case blocking will occur. Finally, the load-balancing scheme must be very fast because of the high transmission rate of the switch and the large number of input/output ports. The randomized load-balancing methods studied in this paper have all of these properties. They are very fast since all the decisions are made based on the state of d randomly chosen output ports. They reduce the chance of blocking since output queues are chosen randomly and by choosing the shortest queue among these d random samples, they give higher priorities to shorter queues in the switch. We prove that only exploiting only random sampling can lead to instability, but also prove that using memory units for input ports can guarantee stability. Based on empirical results, we observe that using small values of samples, e.g., $d = 2$ *or* 3, and a single unit of memory for each input port results in perfect balance in the output queue lengths, and acceptable latency. The QoS guarantees that these policies may provide and their performance for different traffic patterns need to be investigated in future studies.

REFERENCES

[1] C. Clos, “A study of non-blocking switching networks,” *Bell System Technical Journal*, pp. 406–424, 1953.

[2] S. Iyer, A. A. Awadallah, and N. McKeown, “Analysis of a packet switch with memories running slower than the line rate,” in *Proc. IEEE INFOCOM*, 2000, pp. 529–537.

[3] W. Wang, L. Dong, and W. Wolf, “A distributed switch architecture with dynamic load-balancing and parallel input-queued crossbars for terabit switch fabrics,” in *Proc. IEEE INFOCOM*, 2002, pp. 352–361.

[4] C.-S. Chang, D.-S. Lee, and Y.-S. Jou, “Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering,” *Computer Communications*, vol. 25, no. 6, pp. 611–622, 2002.

[5] M. Mitzenmacher, “The power of two choices in randomized load balancing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, pp. 1094–1104, 2001.

[6] P. Kumar and S. Meyn, “The randomness in randomized load balancing,” in *Proc. of the 32nd IEEE Conference onDecision and Control*, 15-17 1993, pp. 2730–2735 vol.3.

[7] S. Iyer and N. McKeown, “Analysis of the parallel packet switch architecture,” *IEEE/ACM Transactions on Networking*, pp. 314–324, 2003.

[8] M. Mitzenmacher, B. Prabhakar, and D. Shah, “Load balancing with memory,” *Annual IEEE Symposium on Foundations of Computer Science*, vol. 0, p. 799, 2002.

[9] D. Shah and B. Prabhakar, “The use of memory in randomized load balancing,” in *Proc. ISIT*, 2002, p. 125.

APPENDIX

Theorem 1: $(1, 1)$ policy is stable for all admissible i.i.d. arrival processes.

Proof: Consider discrete instances of time when there is either an arrival or a departure, since these are the only times that the state of the system changes. We assume the speedup is K. We further assume at each time instance up to N packets arrive at the system according to N independent Bernoulli processes and the arrival rate to the input port i is δ_i $(1 \leq i \leq N)$. We denote by μ_i $(1 \leq i \leq N)$ the service rate of output port i $(1 \leq i \leq N)$. We assume that at most one packet can leave each queue at any given time instance (this assumption can be easily relaxed). Hence, the probability of having an arrival at any given time instant at input i is $\frac{\delta_i}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i}$. Similarly, the probability that a departure occurs from output port i at any instant of time is $\frac{\mu_i}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i}$.

Let $q_k(n)$, $\tilde{q}_k(n)$ and $q^*(n)$ represent the length of the k-th output queue, the length of the output queue chosen by the input i and length of the shortest output queue in the system under policy $(1, 1)$ at time instance n, respectively.

If n is an arrival instant, then the probability that under $(1, 1)$ policy input i chooses the shortest output queue, i.e., $\tilde{q}_i(n) = q^*(n)$, is at least 1/n. At each time instant, from up to N input ports that are contending for the same output port, at most K of them will be granted permission to direct their packets to that output. If by λ_i we refer to the arrival rate at output port i, then λ_i will be the summation of the arrival rate of these s input ports, and $\sum_{i=1}^N \lambda_i \leq \sum_{i=1}^N \delta_i$. The probability of having a packet forwarded to output port i is $\frac{\lambda_i}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i}$. Moreover, by Lemma 1, proved in the appendix, we will show that

$$\sum_{i=1}^{N} \lambda_i \times q_i(n) \leq \sum_{i=1}^{N} \delta_i \times \tilde{q}_i(n).$$

To prove that the algorithm is stable, using the result of Kumar and Meyn[6], we show that for an $N \times N$ switch scheduled using the $(1, 1)$ policy, there is a negative expected single-step drift in a Lyapunov function, V. In other words,

$$E[V(n+1) - V(n)|V(n)] \leq \epsilon V(n) + k,$$

where $k > 0$ and $\epsilon > 0$. Let $V(n)$ be:

$$V(n) = V_1(n) + V_2(n),$$

where

$$V_1(n) = \sum_{i=1}^{N} V_{1,i}(n),$$

$$V_{1,i}(n) = (\tilde{q}_i(n) - q^*(n))^2,$$

$$V_2(n) = \sum_{i=1}^{N} q_i^2(n).$$

Since at most K packets can be enqueued at time instance $n+1$ in $\tilde{q}_i$ when the speedup is K,

$$\tilde{q}_i(n+1) - \tilde{q}_i(n) \leq K.$$

And at most one packet can leave q^* at time instant n. So,

$$-q^*(n+1) + q^*(n) \leq 1.$$

Therefore,

$$\tilde{q}_i(n+1) - q^*(n+1) \leq \tilde{q}_i(n) - q^*(n) + K + 1.$$

Now consider:

$$E[V_1(n+1) - V_1(n)|V_1(n)] =$$

$$\frac{1}{N}\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times$$

$$\sum_{i=1}^{N} \delta_i \times ((\tilde{q}_i(n+1) - q^*(n+1))^2 - (\tilde{q}_i(n) - q^*(n))^2) +$$

$$\sum_{i=1}^{N} (1 - \frac{1}{N}\frac{\delta_i}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i}) \times$$

$$((\tilde{q}_i(n+1) - q^*(n+1))^2 - (\tilde{q}_i(n) - q^*(n))^2) \leq$$

$$-\frac{1}{N}\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times \sum_{i=1}^{N} \delta_i (\tilde{q}_i(n) - q^*(n))^2 +$$

$$\sum_{i=1}^{N} (2(\tilde{q}_i(n) - q^*(n)) + K + 1) \leq$$

$$-\frac{1}{N}\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times \sum_{i=1}^{N} \delta_i V_{1,i}(n) +$$

$$\sum_{i=1}^{N} (2\sqrt{V_{1,i}(n)} + K + 1)$$

So,

$$E[V_1(n+1) - V_1(n)|V_1(n)] \leq$$

$$-\frac{1}{N}\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times \sum_{i=1}^{N} \delta_i V_{1,i}(n) +$$

$$\sum_{i=1}^{N} 2\sqrt{V_{1,i}(n)} + N + NK.$$

And for V_2,

$$E[V_2(n+1) - V_2(n)|V_2(n)] =$$

$$\sum_{i=1}^{N} \frac{\lambda_i}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times (q_i(n+1) - q_i(n))(q_i(n+1) - q_i(n)) +$$

$$\sum_{i=1}^{N} \frac{\mu_i}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times (q_i(n+1) - q_i(n))(q_i(n+1) - q_i(n)) =$$

$$\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \sum_{i=1}^{N} \delta_i \times (2\tilde{q}_i(n) + 1) +$$

$$\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \sum_{i=1}^{N} \mu_i \times (-2q_i(n) + 1) =$$

$$\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times$$

$$(\sum_{i=1}^{N} \delta_i \times (1 + 2\sqrt{V_{1,i}} + 2q^*(n)) + \sum_{i=1}^{N} \mu_i - 2\sum_{i=1}^{N} \mu_i q_i(n)) =$$

$$\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times$$

$$[\sum_{i=1}^{N} \delta_i \times (1 + 2\sqrt{V_{1,i}}) +$$

$$2q_i^*(n)(\sum_{i=1}^{N} \delta_i - \sum_{i=1}^{N} \mu_i) +$$

$$2\sum_{i=1}^{N} \mu_i (q^*(n) - q_i(n))].$$

So,

$$E[V_2(n+1) - V_2(n)|V_2(n)] \leq$$

$$\frac{\sum_{i=1}^N \delta_i}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} +$$

$$\frac{2\sum_{i=1}^N \delta_i \sqrt{V_{1,i}(n)}}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} +$$

$$\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times 2q^*(n)(\sum_{i=1}^{N} \delta_i - \sum_{i=1}^{N} \mu_i) +$$

$$\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times 2\sum_{i=1}^{N} \mu_i (q^*(n) - q_i(n)).$$

Thus,

$$E[V(n+1) - V(n)|V(n)] \leq$$

$$-\frac{1}{N}\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times \sum_{i=1}^{N} \delta_i V_{1,i}(n) +$$

$$2\sum_{i=1}^{N} \sqrt{V_{1,i}(n)} (\frac{\delta_i}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} + 1) +$$

$$\frac{\sum_{i=1}^N \delta_i}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} + N + NK +$$

$$\frac{1}{\sum_{i=1}^N \delta_i + \sum_{i=1}^N \mu_i} \times 2q^*(n)(\sum_{i=1}^{N} \delta_i - \sum_{i=1}^{N} \mu_i) +$$

$$\frac{1}{\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i}\times 2\sum_{i=1}^{N}\mu_i(q^*(n)-q_i(n)).$$

Hence,

$$E[V(n+1)-V(n)|V(n)]\le$$

$$\sum_{i=1}^{N}\frac{-N(\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i)}{\delta_i}\times$$

$$\left(\frac{\delta_i\sqrt{(V_{1,i})}}{N(\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i)}-(\frac{\delta_i}{\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i}+1)\right)^2$$

$$+(N+1)\frac{\sum_{i=1}^{N}\delta_i}{\sum_{i=1}^{N}\delta_i\sum_{i=1}^{N}\mu_i}+N\frac{\sum_{i=1}^{N}\delta_i\sum_{i=1}^{N}\mu_i}{\sum_{i=1}^{N}\delta_i}+3N+$$

$$NK+\frac{1}{\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i}\times 2q^*(n)(\sum_{i=1}^{N}\delta_i-\sum_{i=1}^{N}\mu_i)$$

$$+\frac{1}{\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i}\times 2\sum_{i=1}^{N}\mu_i(q^*(n)-q_i(n)).$$

So if we define

$$A_i=\frac{\delta_i}{N(\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i)}$$

$$B_i=\frac{\delta_i}{\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i}+1$$

$$C=(N+1)\frac{\sum_{i=1}^{N}\delta_i}{\sum_{i=1}^{N}\delta_i\sum_{i=1}^{N}\mu_i}+$$

$$N\frac{\sum_{i=1}^{N}\delta_i\sum_{i=1}^{N}\mu_i}{\sum_{i=1}^{N}\delta_i}+3N+NK$$

Then,

$$A_i\ge 0\ and\ B_i\ge 0\ and\ C\ge 0$$

And,

$$E[V(n+1)-V(n)|V(n)]\le$$

$$\sum_{i=1}^{N}-(\frac{\sqrt{V_{1,i}}}{A_i}-\frac{B_i}{A_i^2})^2+C \qquad \text{(I)}$$

$$+\frac{1}{\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i}\times 2q^*(n)(\sum_{i=1}^{N}\delta_i-\sum_{i=1}^{N}\mu_i) \quad \text{(II)}$$

$$+\frac{1}{\sum_{i=1}^{N}\delta_i+\sum_{i=1}^{N}\mu_i}\times 2\sum_{i=1}^{N}\mu_i(q^*(n)-q_i(n)) \quad \text{(II)}$$

The following upper bounds are easily obtained: *(I)* $\le C$ *(II)* ≤ 0, since the traffic is admissible. *(III)* ≤ 0, by definition of $q^*(n)$. Suppose that $V(n)$ is very large. If $V_1(n)$ is very large, (I) will be negative, from which $E[V(n+1)-V(n)|V(n)] < -\epsilon_1$ for $V_1(n) > L_1$ follows. Otherwise, if $V_1(n)$ is not very large but $V(n)$ is, then $V_2(n)$ should be very large which implies that length of some output queue, $q_i(n)$, is very large. If $q^*(n)$ is not very large, then (III) will be less than $-C$ which is a bounded constant. If $q^*(n)$ is also large, then (II) will be less than $-C$. In both cases, it follows that $E[V(n+1)-V(n)|V(n)] < -\epsilon_2$ for $V_2(n) > L_2$. Hence, there exist L and ϵ such that $E[V(n+1)-V(n)|V(n)$ is very large$] < -\epsilon$ for $V(n) > L$.

■

Lemma 1: $\sum_{i=1}^{N}\lambda_i\times q_i(n)\le\sum_{i=1}^{N}\delta_i\times\tilde{q}_i(n)$.

Proof: Let us define $\rho_{i,j}$

$$\rho_{i,j} = \begin{cases}\delta_j & input\ j\ chooses\ output\ i\\ 0 & otherwise\end{cases}$$

It immediately follows that

$$\lambda_i\times q_i(n)\le\sum_{j=1}^{N}\rho_{i,j}\times\tilde{q}_j(n).$$

So,

$$\sum_{i=1}^{N}\lambda_i\times q_i(n)\le\sum_{i=1}^{N}\sum_{j=1}^{N}\rho_{i,j}\times\tilde{q}_j(n).$$

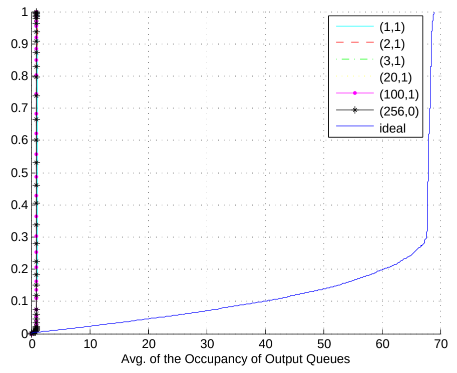
But since the input ports can compete for only a single output port at a time, the term $\rho_{i,j}$ can be non-zero only for at most N pairs of (i,j). It follows that

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\rho_{i,j}\times\tilde{q}_j(n)=\sum_{i=1}^{N}\delta_i\times\tilde{q}_i(n).$$

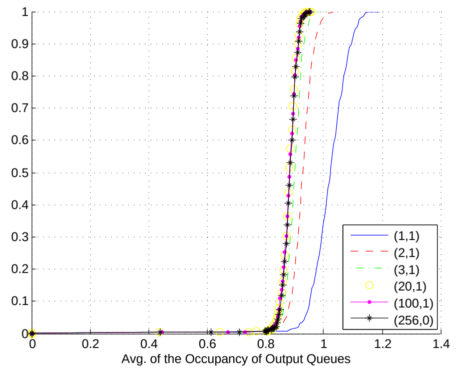
So,

$$\sum_{i=1}^{N}\lambda_i\times q_i(n)\le\sum_{i=1}^{N}\delta_i\times\tilde{q}_i(n).$$

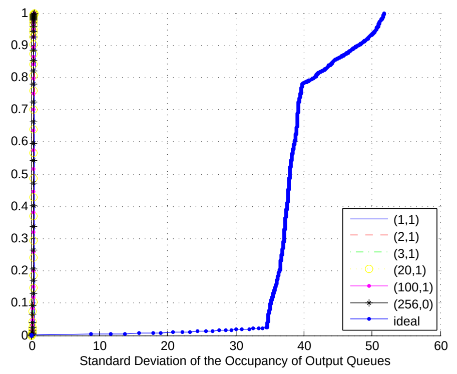
■



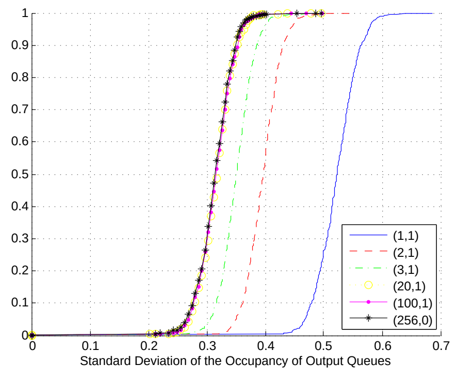
(a)



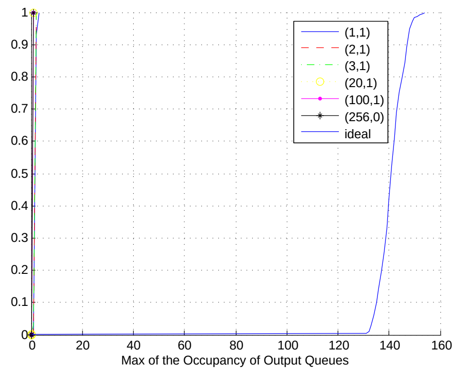
(a)



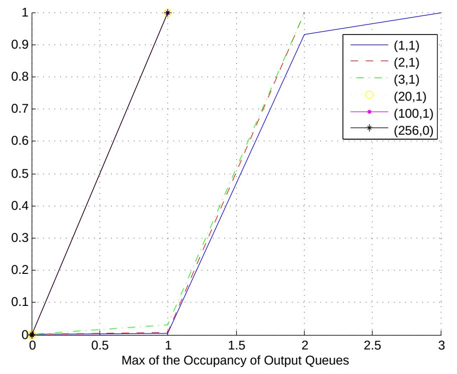
(b)



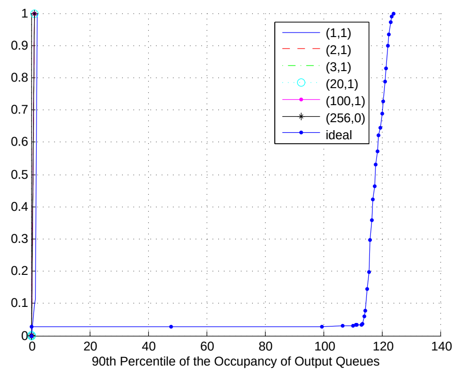
(b)



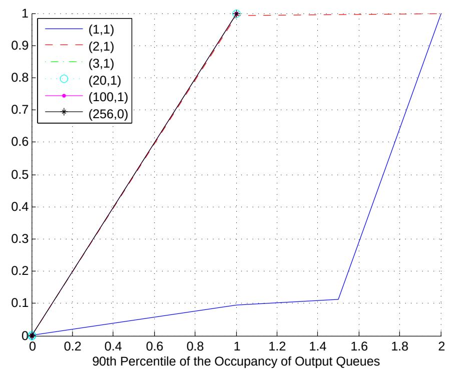
(c)



(c)



(d)



(d)

Fig. 7. CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Output Queues' Occupancy Over Time for Heavy Traffic and No Speedup for (d,m) Policies and *ideal* policy in a 256*256 Switch.

Fig. 8. Zoomed CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Output Queues' Occupancy Over Time for Heavy Traffic and No Speedup for a 256*256 Switch.

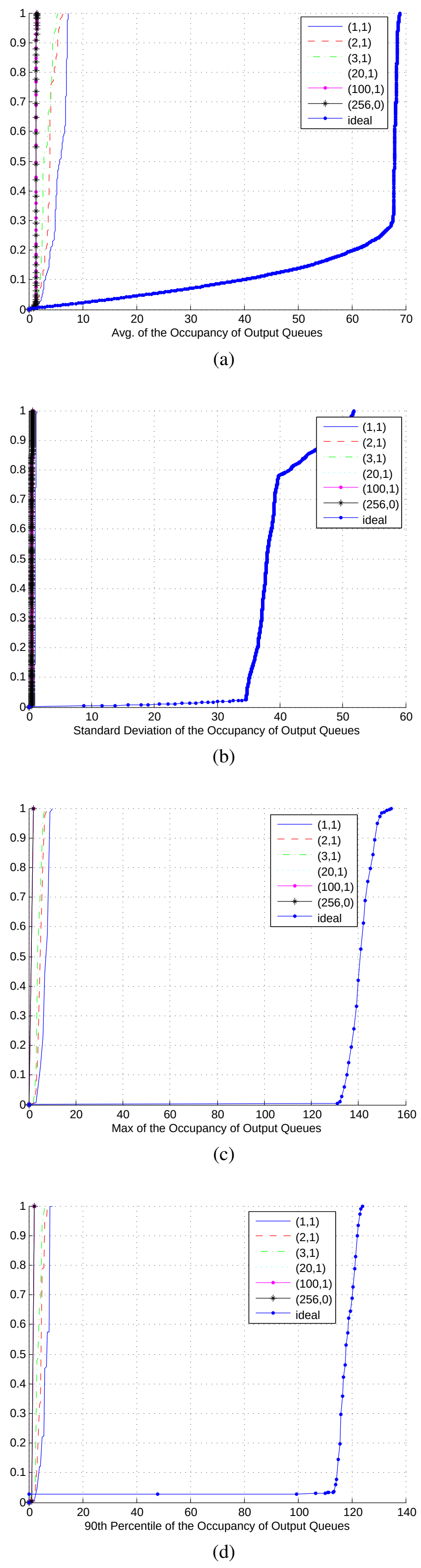


Fig. 9. CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Output Queues' Occupancy Over Time for Heavy Traffic and Speedup of 2 for (d,m) Policies and *ideal* policy in a 256*256 Switch.

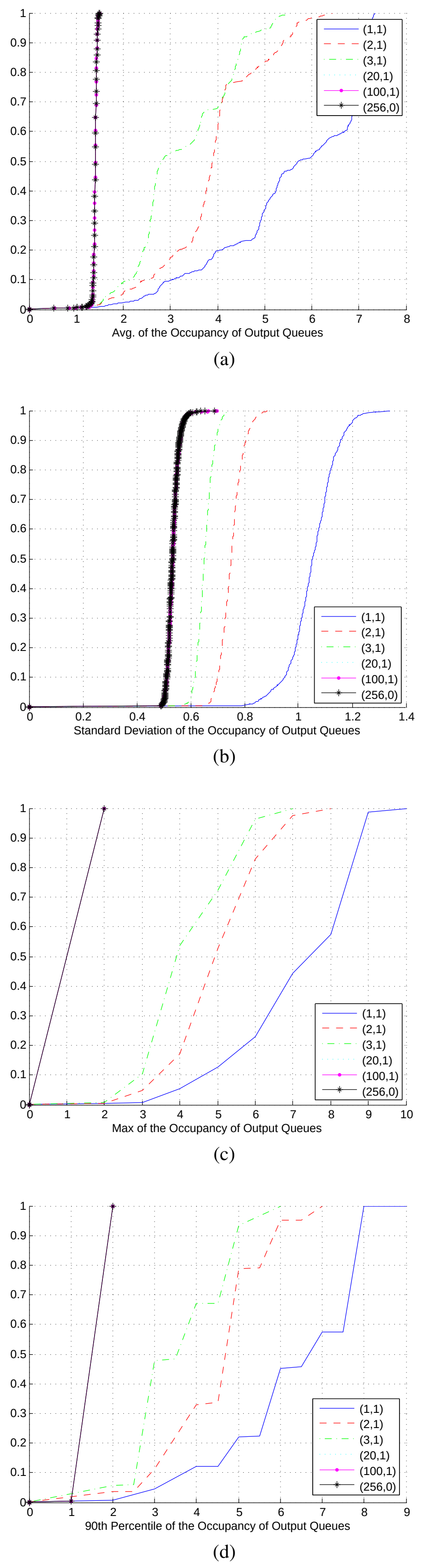
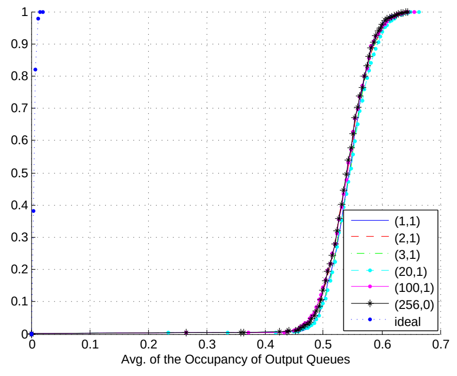
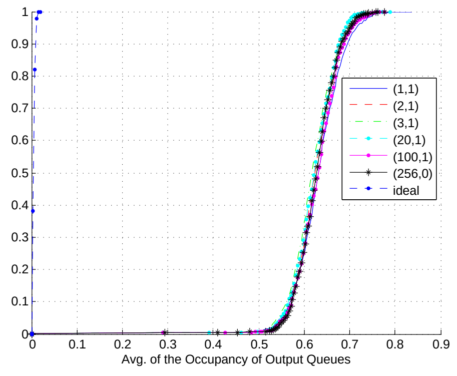
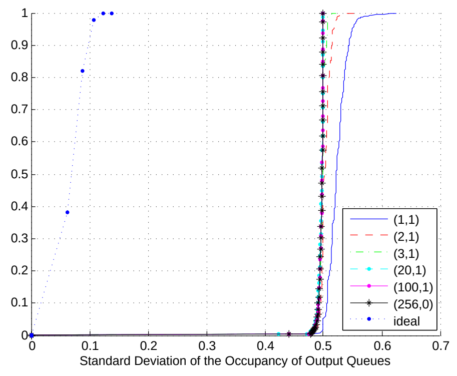
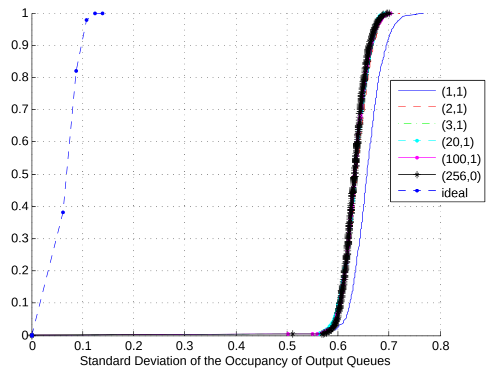


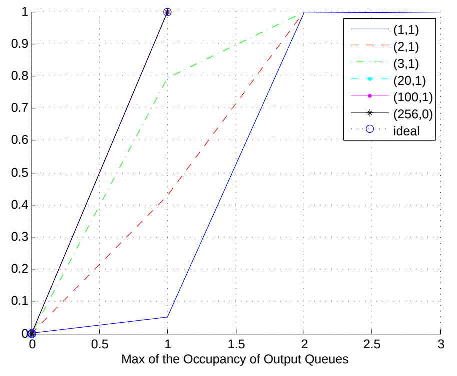
Fig. 10. Zoomed CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Output Queues' Occupancy Over Time for Heavy Traffic and Speedup of 2 for a 256*256 Switch.

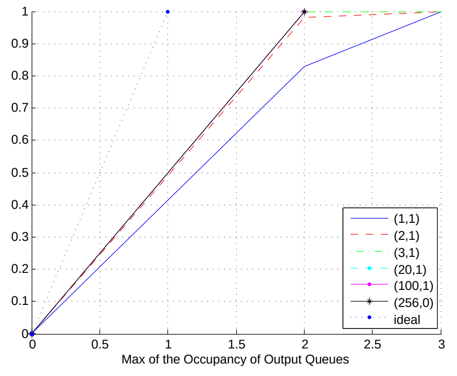

(a)

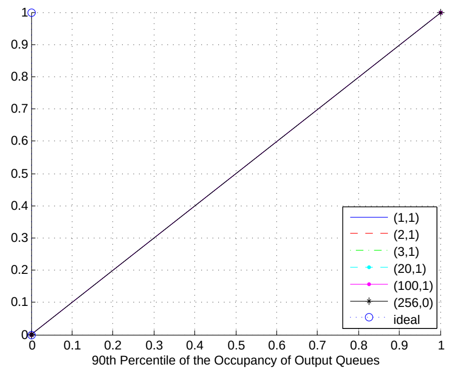

(a)


(b)


(b)


(c)


(c)


(d)

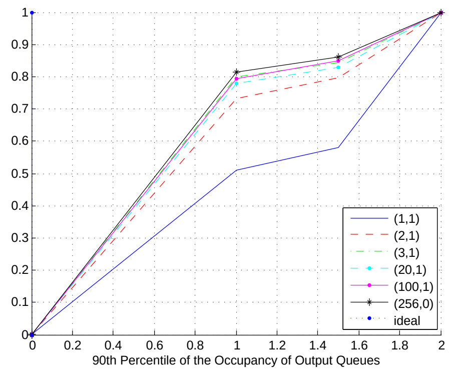
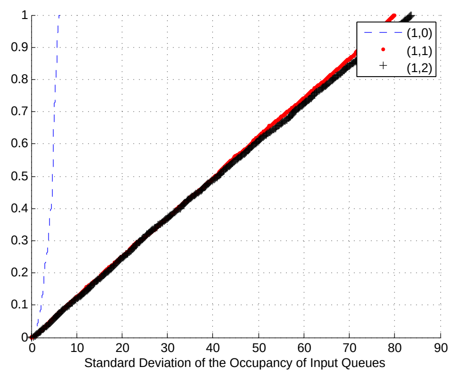
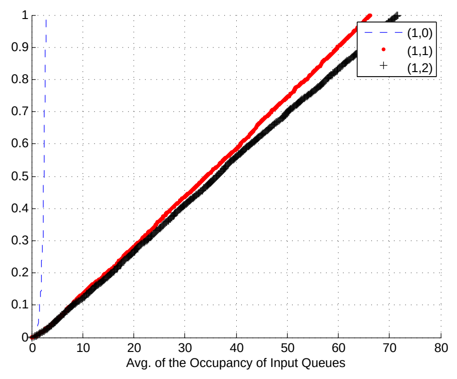

(d)

Fig. 11. CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Output Queues' Occupancy Over Time for Light Traffic and No Speedup for (d,m) Policies and *ideal* policy in a 256*256 Switch.

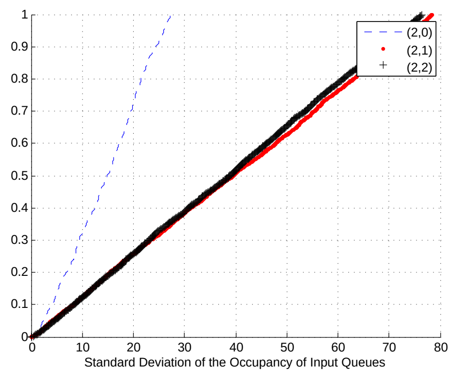
Fig. 12. CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Output Queues' Occupancy Over Time for Light Traffic and Speedup of 2 for (d,m) Policies and *ideal* policy in a 256*256 Switch.



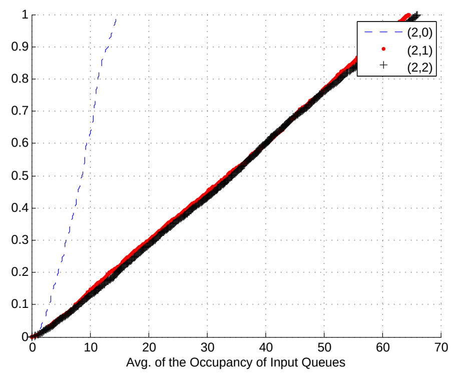
(a) d=1, m=0,1,2



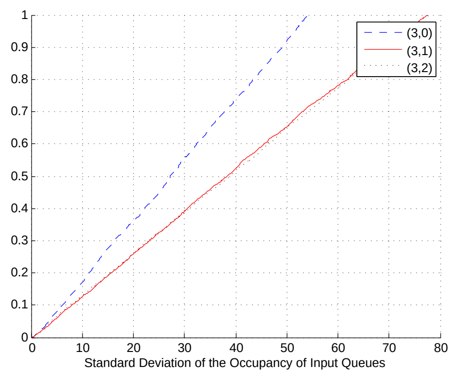
(a) d=1, m=0,1,2



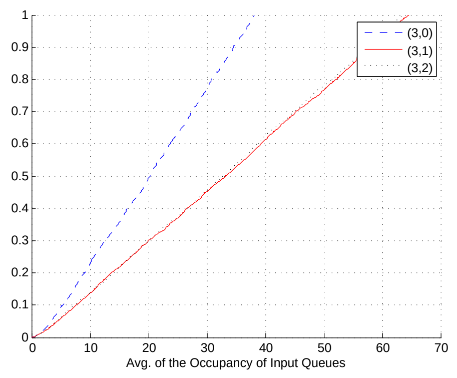
(b) d=2, m=0,1,2



(b) d=2, m=0,1,2



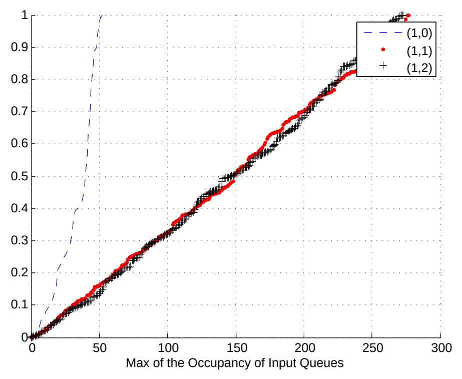
(c) d=3, m=0,1,2



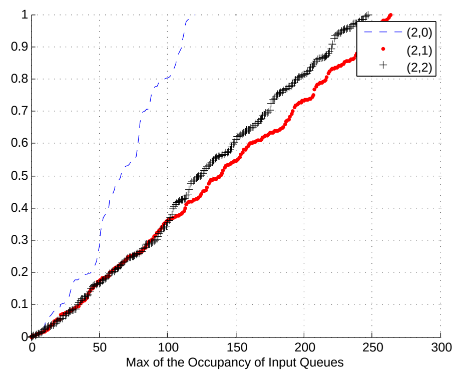
(c) d=3, m=0,1,2

Fig. 13. Effect of Memory on CDF of Standard Deviation of Input Queues' Occupancy over Time for (d,m) Policies for Heavy Traffic and No Speedup For 256*256 Switches.

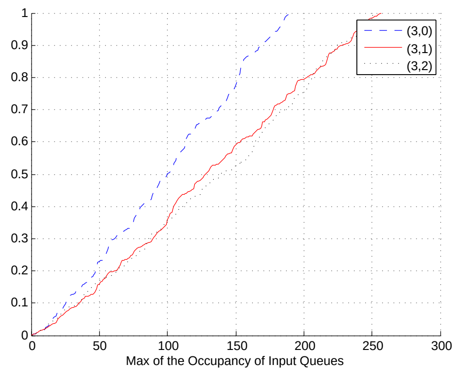
Fig. 14. Effect of Memory on CDF of Avg. of Input Queues' Occupancy over Time for (d,m) Policies for Heavy Traffic and No Speedup For 256*256 Switches.



(a) d=1, m=0,1,2

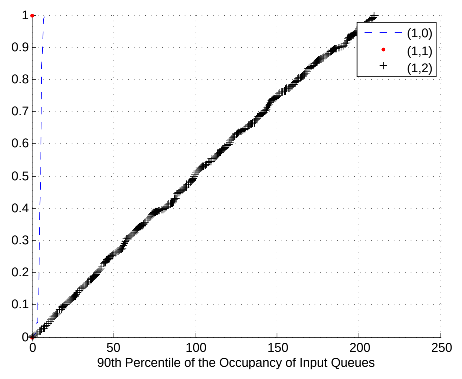


(b) d=2, m=0,1,2

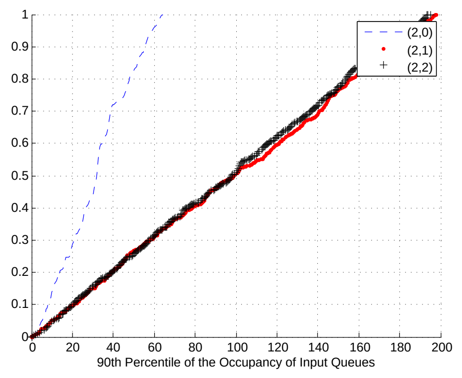


(c) d=3, m=0,1,2

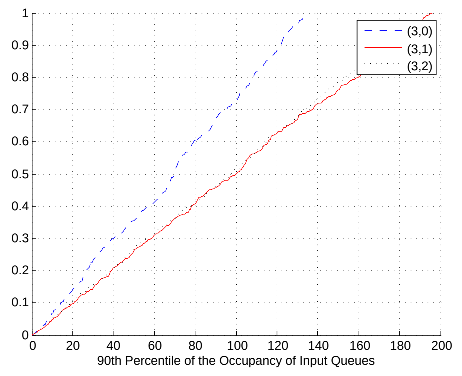
Fig. 15. Effect of Memory on CDF of Max of Input Queues' Occupancy over Time for (d,m) Policies for Heavy Traffic and No Speedup For 256*256 Switches.



(a) d=1, m=0,1,2

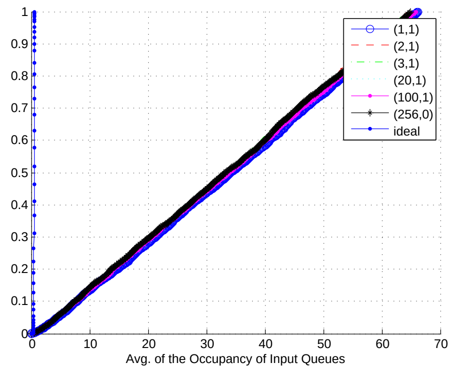


(b) d=2, m=0,1,2

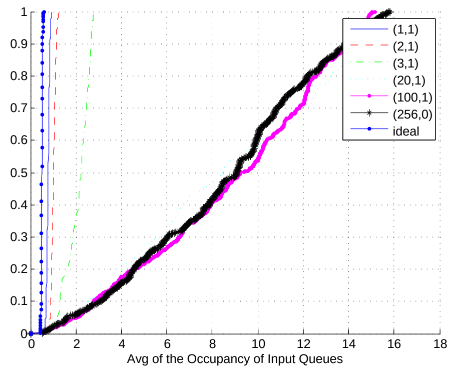


(c) d=3, m=0,1,2

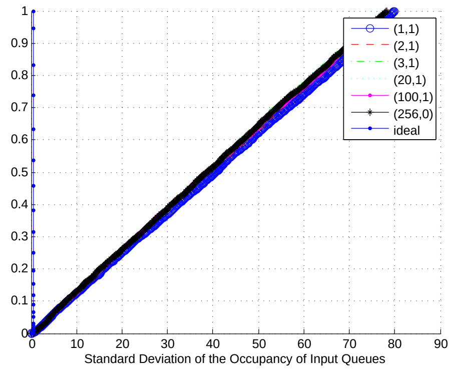
Fig. 16. Effect of Memory on CDF of 90th Percentile of Input Queues' Occupancy over Time for (d,m) Policies for Heavy Traffic and No Speedup For 256*256 Switches.



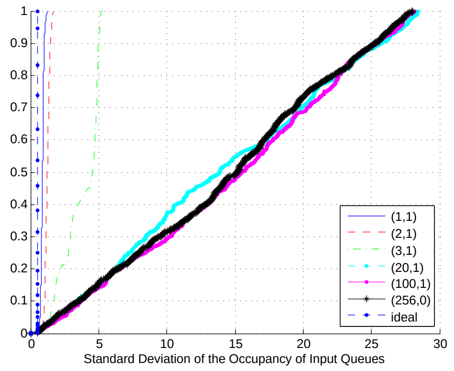
(a)



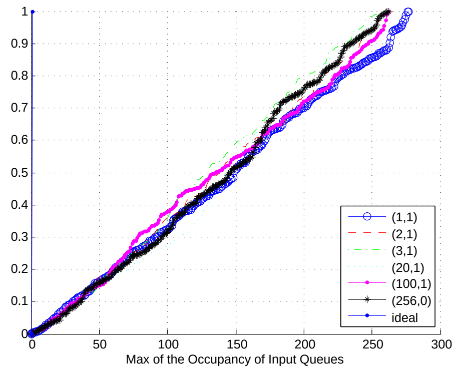
(a)



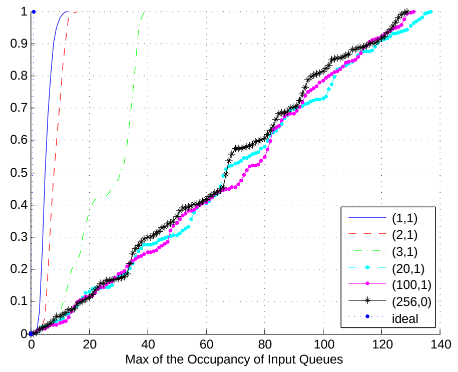
(b)



(b)

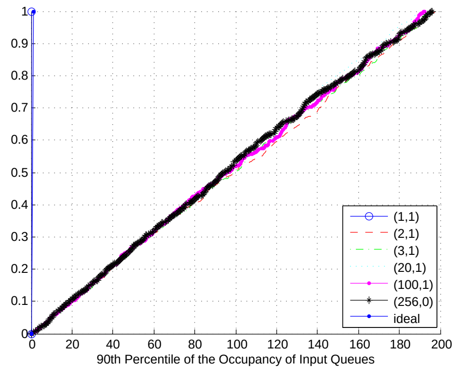


(c)

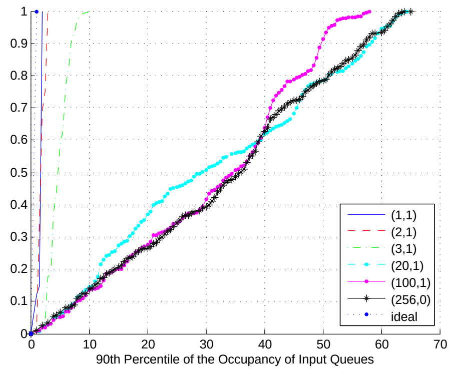


(c)

Fig. 17. CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Input Queues' Occupancy Over Time for Heavy Traffic and No Speedup for (d,m) Policies and *ideal* policy in a 256*256 Switch.

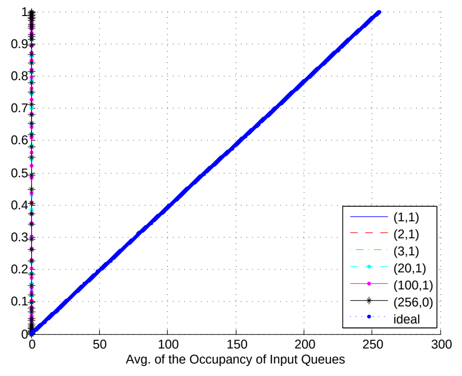


(d)

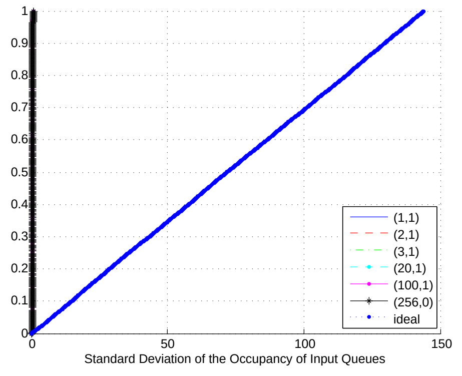


(d)

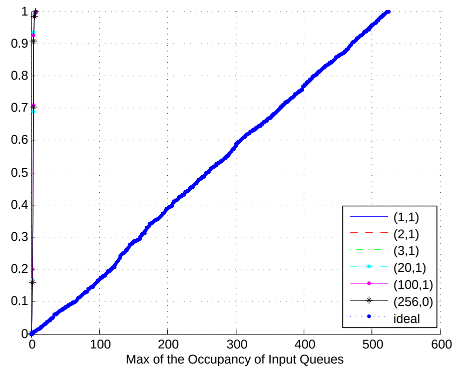
Fig. 18. CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Input Queues' Occupancy Over Time for Heavy Traffic and Speedup of 2 for (d,m) Policies and *ideal* policy in a 256*256 Switch.



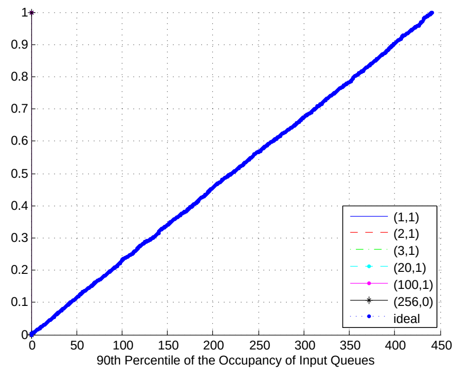
(a)



(b)

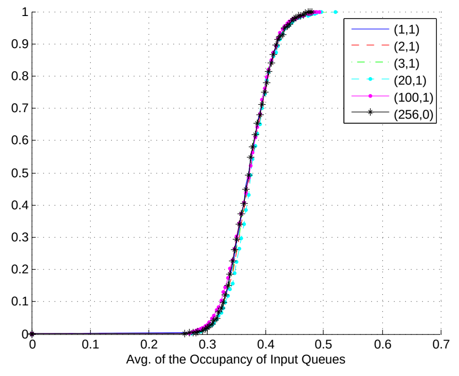


(c)

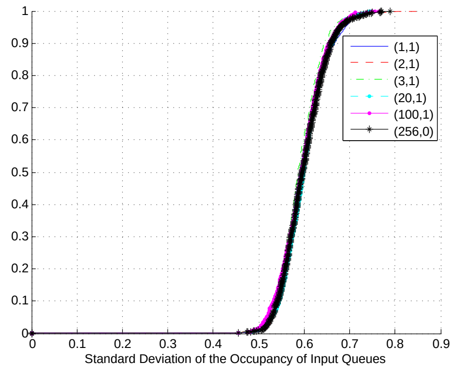


(d)

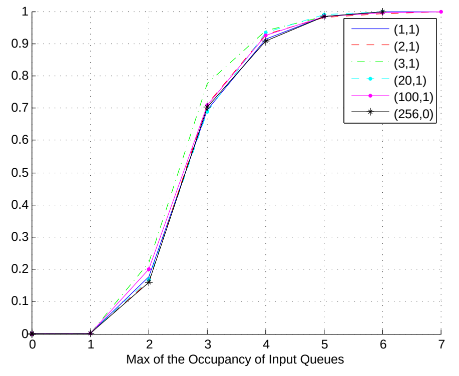
Fig. 19. CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Input Queues' Occupancy Over Time for Light Traffic and No Speedup for (d,m) Policies and *ideal* policy in a 256*256 Switch.



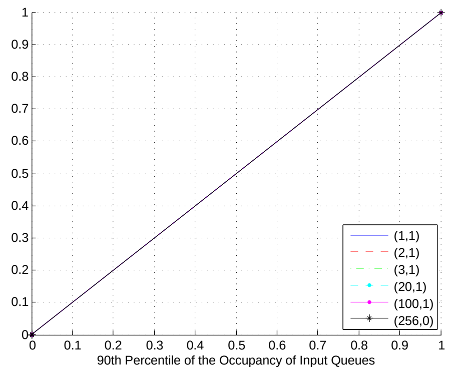
(a)



(b)

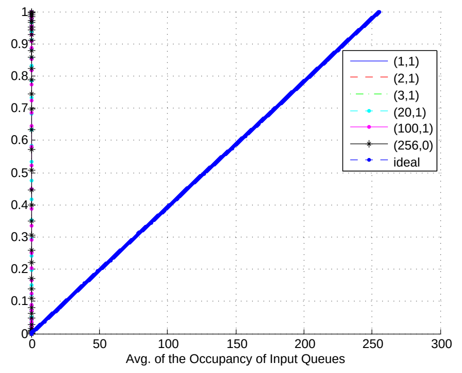


(c)

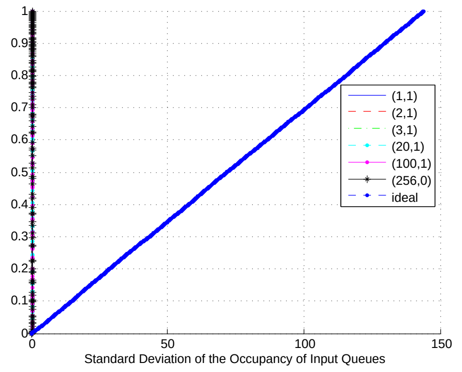


(d)

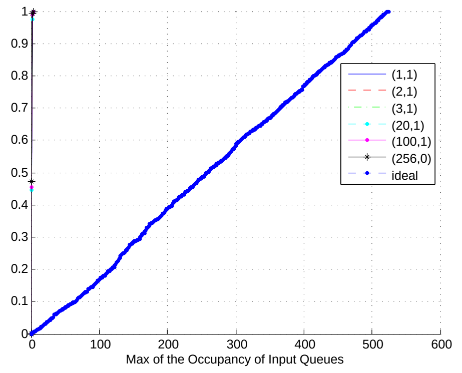
Fig. 20. Zoomed CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Input Queues' Occupancy Over Time for Light Traffic and No Speedup for (d,m) Policies and *ideal* policy in a 256*256 Switch.



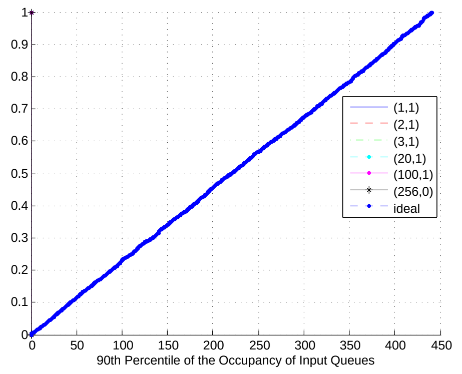
(a)



(b)

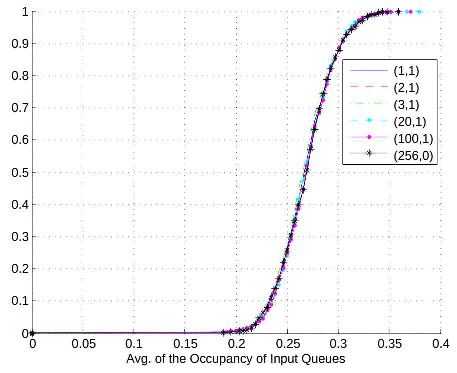


(c)

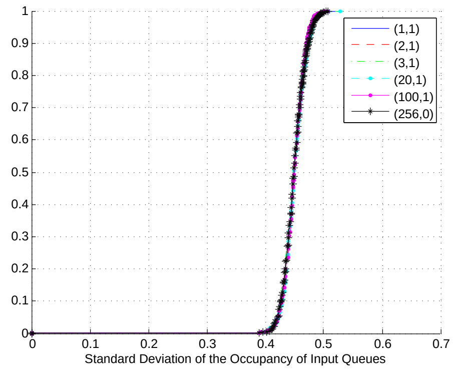


(d)

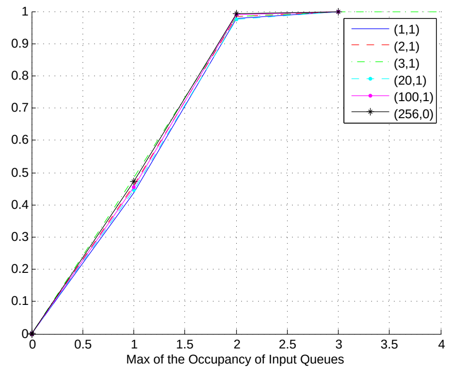
Fig. 21. CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Input Queues' Occupancy Over Time for Light Traffic and Speedup of 2 for (d,m) Policies and *ideal* policy in a 256*256 Switch.



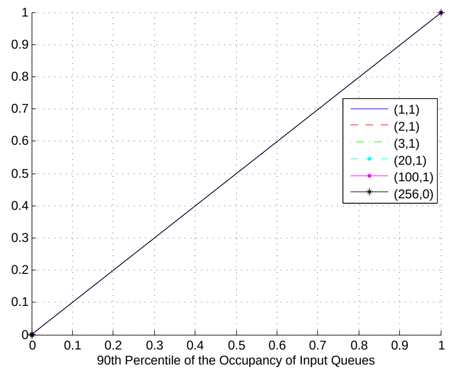
(a)



(b)

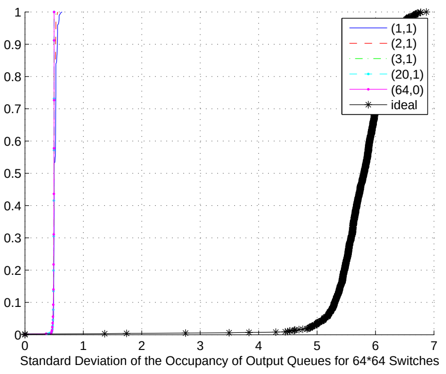


(c)

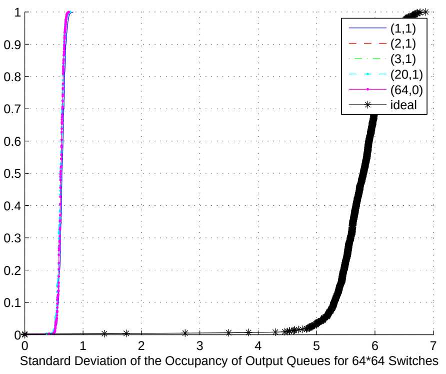


(d)

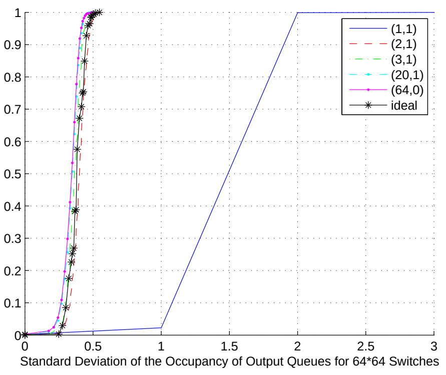
Fig. 22. Zoomed CDF of (a) Avg., (b) Standard Deviation, (c) Max and (d) 90th Percentile of Input Queues' Occupancy Over Time for Light Traffic and Speedup of 2 for (d,m) Policies and *ideal* policy in a 256*256 Switch.



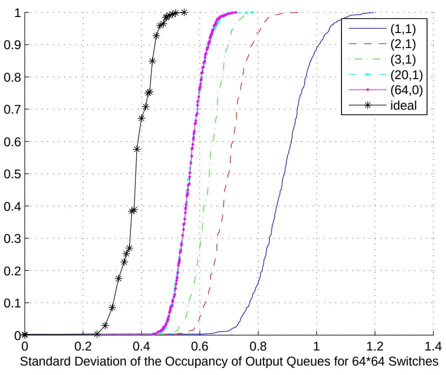
(a)



(b)



(c)



(d)

Fig. 23. CDF of Standard Deviation of Output Queues' Occupancy over Time for (a) Heavy Traffic and No Speedup, (b) Heavy Traffic and Speedup of 2, (c) Light Traffic and No Speedup, and (d) Light Traffic and Speedup of 2 for a 64*64 Switch.