# Using Traffic Models in Switch Scheduling

Hammad M. Saleem, Imran Q. Sayed
*{hsaleem, iqsayed}@stanford.edu*

## *I. Background*

Conventional scheduling algorithms use only the current virtual output queue (VOQ) occupancies to make scheduling decisions, i.e. which inputs to connect to which outputs at any time slot. The stability of such techniques such as Maximum Weight Matching (MWM) [McK99] and some randomized algorithms is proven for admissible traffic [Tass98][Giac01].

MWM also performs close to Output Queuing in terms of mean delay characteristics but has very high computational complexity and is therefore not a practical solution.

Longest Port First (LPF) [Mekk98] has been proposed as a hardware feasible way to overcome the complexity problem of MWM. It has a notion of port occupancy that is defined as all the cells occupying an input port or all the cells destined for an output port. The weight of VOQ*ij* is the sum of the port occupancies of input port *i* and output port *j*.

Randomization is also an interesting technique for reducing the computational complexity of algorithms. Using only randomization does not necessarily give stability but it can easily be achieved by the use of memory as shown by Tassiulas [Tass98]. The following simple procedure applied in each time slot of an NxN switch ensures its stability.

TASS:

- Store the matching used in the previous time slot in memory, call it $M_m$
- Randomly select a matching $M_r$ according to a distribution over all possible matchings
- Compare $M_r$ with $M_m$ and choose the one with higher weight as new $M_m$

Several variations and specific implementations of this simple randomized algorithm have been proposed. These techniques have the feature of giving close to MWM performance at a much lower cost.

One of three such algorithms proposed by Giaccone et al [Giac01] is SERENA. It uses input arrivals as the source of randomization while determining $M_r$. It creates a new match by merging $M_r$ and $M_m$, which is better than choosing either of the two.

## *II. Introduction*

We intend to use traffic models (future arrival predictions or arrival rate estimates) to improve switch performance. Modeling Internet traffic accurately can be fairly challenging [Pax97], especially when attempted at the time scale of interest in switch scheduling. Also, any such model has to fulfill the additional constraint of being able to run online at current traffic rates in Gbits/sec range. This means that the model must be simple in addition to being reasonably accurate.

Although it is difficult to model, there exist several reasons why we expect to see patterns in Internet traffic. Some of the processes contributing to these patterns are:

- Flow control mechanisms in connection oriented transmissions (such as TCP)
- Long duration and constant rate streaming applications
- Burstiness caused by queuing at switches
- Breaking up of large packets into fixed size cells inside a switch (as seen by the scheduler)

We present a few traffic models and identify conditions for their accuracy. These models have however not been tested due to unavailability of actual router traces. We also show the effect of potential inaccuracies of these models on switch performance.

Previous work by T. Brown [Brow01] shows the potential for improved performance by use of prior information about future arrivals, but does not propose any practical technique for using future information; neither does it specify how to make the predictions. We present a 1-slot prediction technique, and a modified LPF algorithm, which uses this information to reduce average cell latency at medium loads. Note that as found by [Brow01], using prediction does not give much benefit under very low or very high loads. This 1-slot technique can be directly extended to use multiple slot predictions.

An alternative to making future predictions is to use traffic history to estimate arrival rates. We propose such a model and a randomized technique which is a specialization of Tassiulas' scheme. It uses the estimated rates to generate the new matching. Interestingly, SERENA is shown to be a special case of this technique. A hardware implementable version of the technique is also presented.

## *III. Traffic Models*

We identify these two categories of traffic modeling:

1) Make explicit predictions about arrivals in future time slots
2) Gather statistical information about arrivals such as estimated arrival rate

The second of these can be considered to be giving implicit predictions about future but does not involve the issue of accuracy like the first one does. We introduce two models in this section, one for each category and then present two algorithms that employ these models in the next section.

### One Slot Prediction

Let A(n) be the arrival matrix at time n such that $A_{ij}(n) = 1$ if there is an arrival to $VOQ_{ij}$ and $A_{ij}(n) = 0$ otherwise. We propose an Adaptive Autoregressive model to estimate arrivals to a VOQ:

$$\hat{A}_{ij}(n) = [1- b_{ij}(n)]\ \hat{A}_{ij}(n-1) + b_{ij}(n)\ A_{ij}(n-1)$$

where the estimated arrival probability $\hat{A}$ in a time slot n is expressed as a weighted sum of the arrival estimate and the actual arrival A in the previous time slot. Arrival estimate $\hat{A}$ also serves the purpose of keeping a history of arrivals. $\hat{A}(n)$ is thresholded to give a 1/0 (or yes/no) as to whether a cell would arrive in nth time slot. The quantity b(n) in this equation is interpreted as 'burstiness' or 'correlation between consecutive arrivals' and is computed from actual arrivals as:

$$b_{ij}(n) = (1-\beta) * r_{ij}(n-1) + \beta * A_{ij}(n-1) \text{ XNOR } A_{ij}(n-2)$$

Here parameter $\beta$ reflects the adaptation of the correlation coefficient b(n) and its choice is considered to be of secondary importance; any non-zero value below 0.1 is good for a traffic with slowly changing burstiness.

### Rate Estimation

A simple weighted moving average model to keep track of instantaneous arrival rates at any queue is defined as:

$$\lambda_{ij}(n) = (1- a)\ \lambda_{ij}(n-1) + a\ A_{ij}(n-1)$$

Here, $\lambda_{ij}(n)$ is the estimated arrival rate (also interpreted as probability of arrival) at time slot n for VOQij. Again $A_{ij}(n) = 1$ if an arrival occurred at VOQij at time slot n, and 0 otherwise. *a* is called the arrival coefficient.

*a* is the only design parameter in this equation and has several interpretations and implications. It can be thought of as determining the following things:

- How much more significance is given to the latest arrival as compared to the ones in the past, in evaluating the arrival rate
- How much history is contained in the estimated rates
- How fast can the model track a rapidly changing arrival rate

The tradeoff is between choosing a high value of *a* in order to keep track of rapid rate fluctuations, such as short traffic bursts; and choosing a low value of *a* to get a longer term average, better suited for traffic evenly spread out in time.

## *IV. Algorithms and Results*

### Future Assisted LPF

*Algorithm*

We have modified LPF algorithm to utilize 1-slot predictions and call it LPF-F.

LPF-F:

- Predict VOQ arrivals for next slot
- Compute

  $L'_{ij} = L_{ij} + F_{ij}$ if $L_{ij} > 0$

  $L'_{ij} = 0$ otherwise

  where,

  $L_{ij}$ = current occupancy of $VOQ_{ij}$

  $F_{ij}$ = predicted future arrival in $VOQ_{ij}$
- Use $L'_{ij}$ to compute LPF

The rationale behind using LPF versus some other algorithm such as length based MWM to supplement with future information can be described as three reasons:

- LPF has lower complexity i.e. $O(N^{2.5})$ and is a more practical algorithm [Mekk98] compared to MWM which is $O(N^3)$
- LPF has better performance than MWM for most of the loadings
- LPF has a notion of contention avoidance which is a natural way to apply future information to. This contention avoidance in view of the present and future occupancies is expected to yield a lower average cell latency.

The last reason is demonstrated with an example in Appendix A. Intuitively, we wish to give more importance to a queue which has (or will have) packets and will face contention in the future from its competitor VOQs. The more packets it has (or will have) and the greater its competition in the future, the higher its weight.

*Stability*

Theorem 1: LPF-F is stable for all admissible independent arrival processes.

Proof:
It is shown in [Mekk98] that LPF is stable for weights that are k slots old. The proof resorts to the fact that at any time n, the difference between weights of the matchings obtained using correct weights and delayed weights is bounded by a constant. Viewing 'weights with arrival predictions' as weights with a bounded noise (k=1 for 1 slot predictions), we get the stability of LPF-F. ÿ

Note that this result holds regardless of the accuracy of predictions.

*Simulation Results*

LPF-F was simulated with uniform traffic for various degrees of prediction accuracy. The objective is to find whether we achieve the expected benefits and if we do, how much inaccuracy can be tolerated. Figure 1 shows the results for a 3x3 switch where average latency has been normalized with the delays in an OQ switch at the same loads. It can be seen that the delay is reduced to within 7% of OQ with the use of perfect predictions compared to over 20% with simple LPF. Also noticeable is that this reduction in delay holds as long as the accuracy of prediction is above approximately 50% (expressed as 0.5 on the graph). The LPF curve is interestingly almost coincident with the 0.5 LPF-F curve.

The benefit is gained at medium loads and not at very low or very high loads. This can be understood by observing that the IQ switch performance is already close to an OQ switch at these

loads and there is not much to be gained. [Brow01] showed similar results for a 3x3 switch and brought delays to within 2% of OQ using information about 6 future slots but it makes an exhaustive search for the best match, the technique having exponential complexity and also assumes perfect predictions. This result (i.e. 7% vs 2%) also implies that most of the benefit is obtained by 1-slot predictions and looking further into the future is not that useful. Also that an exhaustive search is not going to give much more than what a much simpler technique such as LPF-F achieves.

Although LPF-F can be directly extended to use multiple slot predictions, the complexity of making such predictions may be significantly higher and the degradation in performance due to inaccurate predictions would be more severe. Note however that if (1-slot or multiple-slot) LPF-F is to be used in a real switch, the predictor can observe the accuracy of its predictions by comparing predicted and actual arrivals and disconnect itself from the scheduler when accuracy is below a threshold, say 60%.

Figure 2 shows similar simulation results for an 8x8 switch, indicating that the performance improvements hold for more practical/larger switch sizes as well.

### Rate Randomized Scheduler (RRS)

This algorithm belongs to the family of randomized scheduling algorithms introduced by [Tass98] and is based on the algorithm described in the background section as 'TASS'.

RRS estimates the arrival rate at each VOQ using the Weighted Moving Average model described in the Traffic Models section. These arrival rates ($\lambda_{ij}$'s) are updated at each time slot and are used as probabilities for randomly selecting a match without explicit consideration of the occupancy of the VOQs. Since this is a probabilistic selection process, multiple iterations can be used to successively improve the weight of the randomly selected match for each time slot.

*Algorithm*

RRS*:*

For each Time Slot
Update VOQ arrival rates ($\lambda_{ij}$'s)
For each Iteration

For each InputPort *m*
Considering $\lambda_{mj}$ as probability of Output Port *j* being selected by Input Port *m*, choose an Output *n*

If more than one InputPorts select an OutputPort *n* (or a selected OutputPort was matched with another InputPort in a previous iteration): the InputPort with the longer VOQ*in* is kept in the match. The others are free to participate in next iterations.

Unmatched InputPorts are connected to unmatched OutputPorts in a round-robin manner to give the random matching $M_r$

Merge [Giac01] $M_r$ with $M_m$ (the matching stored in memory) to give $M_f$ (the final matching)

*Stability*

Theorem 2: RRS is stable for $0 <$ *arrival-coefficient* $\leq 1$ under all admissible Bernoulli i.i.d arrival processes.

Proof:
Restating Tassiulas' result [Tass98], a random algorithm is stable if the probability of selecting the maximum weight match as the new random match at any time n is lower bounded by a non-zero constant.

For Bernoulli i.i.d. arrivals, there is a definite non-zero probability of arrival to a VOQ at any time n unless the rate of arrival to that queue is 0. Therefore the probability of randomly selecting MW matching is lower bounded by a constant $\delta$ such that,

$$\delta \geq (a \lambda_{\min})^N$$

where $a$ is the arrival coefficient and

$$\lambda_{\min} = \min_{i,j,\lambda_{ij} \neq 0} \{\lambda_{ij}\}$$

This gives stability for $0 < a \leq 1$. ÿ

Note that 'merging' effectively alters the probability distribution on the space of matchings when selecting a new matching in TASS, putting 0 probability on those which have less weight than the match in memory. This as well as other dynamics of RRS algorithm such as multiple iterations result in a higher value of $\delta$ above.

*Simulation Results*

As mentioned earlier, the major design parameters in RRS technique are arrival coefficient $a$ used in rate estimation, and the number of iterations. We ran two sets of experiments to evaluate the effect of each of these parameters.

Simulation Set 1:

To evaluate the effect of arrival coefficients we ran simulations using 4 iterations. We took readings for different loadings of three different types of traffic:

- *Uniform Traffic:* Under this traffic if the Input Port $i$ has an arrival load of $\lambda_i$ then this loaded is distributed uniformly randomly across all output ports. i.e.
$$\lambda_{ij} = \lambda_i / N$$

- *Diagonal Traffic:* Under this traffic if the Input Port $i$ has an arrival load of $\lambda_i$ then this loaded is distributed randomly across only two output ports such that
$$\lambda_{ii} = (2/3) * \lambda_i$$
$$\lambda_{i(i+1 \bmod N)} = (1/3) * \lambda_i$$

- *Bursty Traffic:* Packets arrive in geometrically distributed bursts of at least one packet with a specified mean burst size. The bursts are distributed uniformly across all output ports.

Figure 3 shows the simulation result for a 16x16 switch using uniform traffic. It is clear that $a$=0.6 and $a$=0.1 both perform better than $a$=1.0 (SERENA). We also observe that $a$=0.6 gives the overall best performance.

Figure 4 shows the simulation result for a 32x32 switch using diagonal traffic. It can be seen that except for very small values of $a$ at very low loads, all other values of $a$ perform better than $a$=1 under all loadings. Again $a$ = 0.6 gives best results at port loadings below 0.6. At loads higher than that, lower vales of $a$ such as 0.3 and 0.1 give better results. We can get about 3 times better performance than Serena across all loads; however we will have to change the value of $a$ to achieve this. But $a$=0.6 performs at least twice as good as $a$=1.0 across all loads.

Simulation Set 2:

To evaluate the effect of number of iterations we ran another set of simulations. The results are shown in Figure 5 and Figure 6. Results show that the effect of going from 1 iteration to 3 iterations is quite significant. Further increase in the number of iterations gives diminishing returns. This is equally true for N=16 and N=32 for both uniform and diagonal traffics.

It was found in the simulations that 1 iteration (almost) always gives best results for $a$ = 1.0 (compared to smaller values). The reason is that the sooner an edge is considered for the new match after an arrival to the corresponding VOQ, the lower it will tend to make the delays. But when multiple iterations are used, $a$=1.0 takes away the chances of adding good edges to the match in the subsequent iterations. We recognize these two forces as the 'Serena Force' and the 'Iteration Force' which work against each other resulting in an optimal value of $a$ between 0 and 1 (depending on the type and rate of the traffic). Ideally, a scheduler would home into this value by observing actual delays and adjusting $a$ accordingly.

## *Issues in Implementation and their Solutions*

RRS has been shown to give better results than the simpler algorithm SERENA. It is clear however that it involves these two major difficulties when it comes to hardware implementation:

i) Real number arithmetic involved in rate estimation
ii) Generation of random numbers in the range [0,N-1] for output selection according to an arbitrary distribution (rate vector for each input)

Here we present an implementation architecture (see Figure 8) for a randomized algorithm, effectively equivalent to RRS. A circular buffer is kept for every input that records the destination output port number for every arrival. The pointer moves (to the left) as it writes this number and writes a blank (encoded as -1, for example) if there is no arrival at that input queue in the given slot. All the buffers - N in number - have pointer at the same location. A set of registers supply geometrically distributed random numbers which are used in every RRS iteration as offsets from the current pointer in the circular buffer; offset is added to the pointer modulo M where M is size of the buffer. Choosing M to be a power of 2 makes the modulo arithmetic simple. The output port number at the selected location in the buffer is actually the output port requested by the respective input. One can visualize a wrapped geometric distribution moving along the pointer, giving higher weights to output ports of recent arrivals. Unlike the original moving average model, this scheme has a finite size history window, size determined by the value of M. An

appropriate value for M can be determined from the arrival-coefficient (e.g. such that less than 5% of the cumulative geometric distribution is wrapped).

The random numbers can be supplied from a pool of numbers and shifted through the registers against the buffers such that at most *I* such numbers need to be read from the pool in each slot where *I* is the number of iterations of RRS. The pool of numbers can be refreshed at any feasible rate; if the scheduler decides to change the value of *a* at any point, the pool will be gradually updated to reflect this change.

Each element in the buffers is at most 1 byte for $N \leq 256$; this scheme is therefore not demanding on memory size and requires much lower memory bandwidth than already available for cell reads/writes in a switch of the same size. A further simplification results from storing the set of N output port numbers together and reading them as a single cache line (which in Figure 8 would imply using the same random offset for every input in a single iteration).

*Complexity of RRS*

Random selection of output ports based on their arrival rates at each input ports is an O(1) operation using the implementation scheme presented earlier. The task of selecting an Input from among multiple requesting Inputs can also be done in parallel by keeping separate selection units for each output port. So this too is a constant time operation. Round robin addition of edges is in the worst case O(n) operation, so is the merge procedure [Giac01]. We will use constant number of iterations and this does not grow with N. Therefore, the complexity of the overall algorithm is O(n).

## V. Conclusions

We proposed two schemes, LPF-F and RRS, for utilizing traffic patterns to improve switch performance. Both of them give improved performance as compared to their closest counterparts that do not use traffic modeling, i.e. LPF and SERENA. We proved their stability, and showed results for different types of traffic.

## VI. Future Directions

- Test the accuracy of 1-slot prediction model for real traffic
- Give a hardware friendly version of the 1-slot predictor
- Test RRS for various other types of traffic
- Compare performance of the hardware friendly RRS technique with the simulation results of this paper

## *VII. Appendices*

### Appendix A

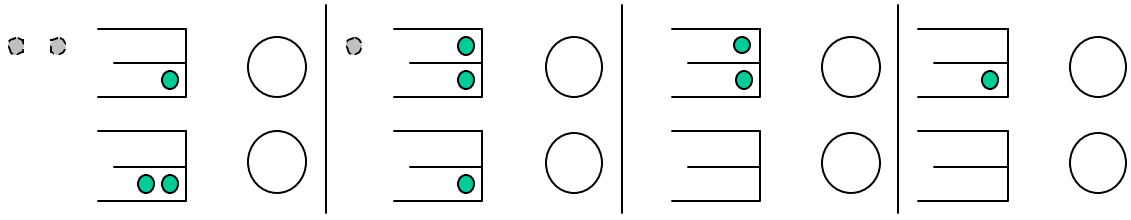
This section gives an example to demonstrate that LPF-F has the potential of reducing average queue sizes and average delay through its tendency to remove contention.

The figure shows evolution of queue states of a 2x2 switch for 4 consecutive time slots. The solid balls represent packets already occupying the queues whereas balls with broken outline are future arrivals. (Large circles represent output ports. Vertical lines separate time slots.)

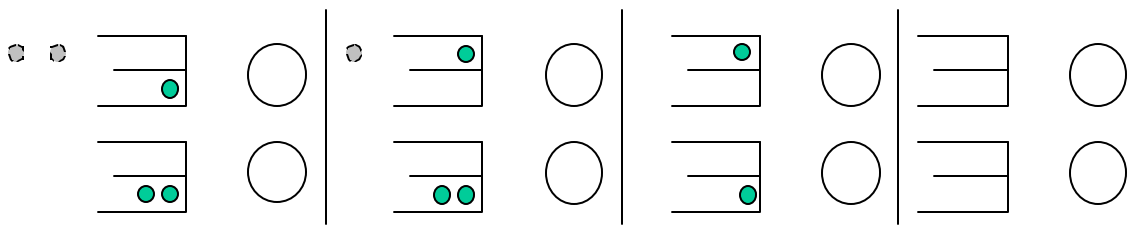
In an attempt to maximize instantaneous occupancy weight, MWM-L (current occupancy based MWM) schedules {(2,2)} in slot 1 and {(1,1), (2,2)} in slot 2 and ends up with two packets at input 1, only one of which can depart in slot 3.

In case of LPF-F however, future arrivals at input 1 cause (1,2) to be scheduled in the first time slot and two packets each can be departed in the subsequent two time slots. The total delay of these packets is reduced by 1 time slot, compared to MWM-L.

MWM - L



LPF - Future



## *VIII. Figures/Plots*

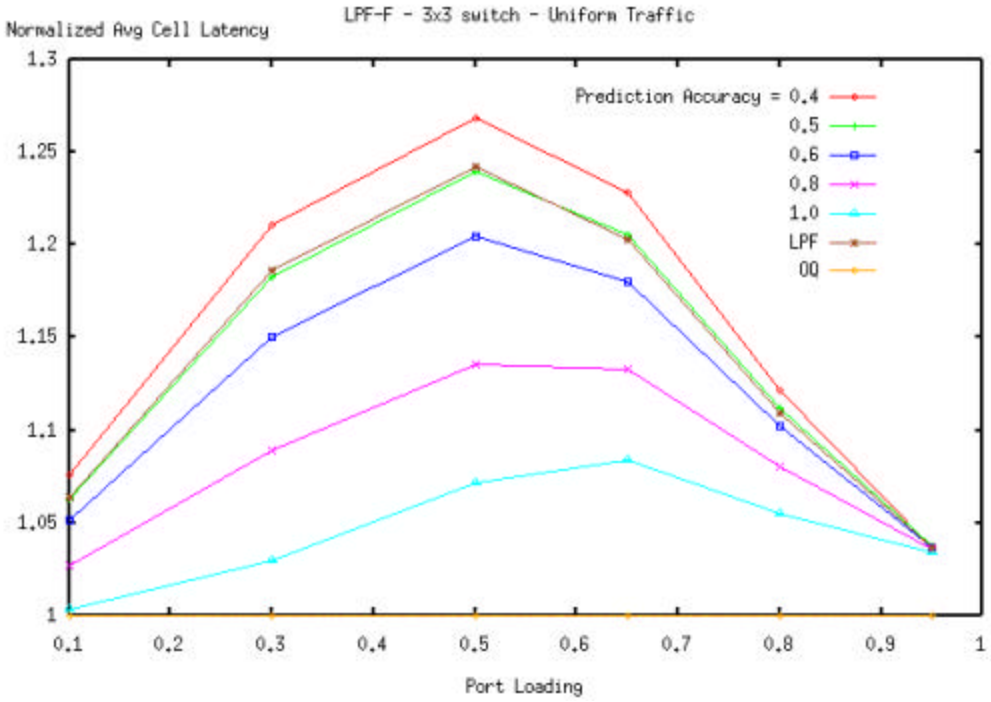



**Figure 1: LPF-F Simulation Results - 3x3 Switch**

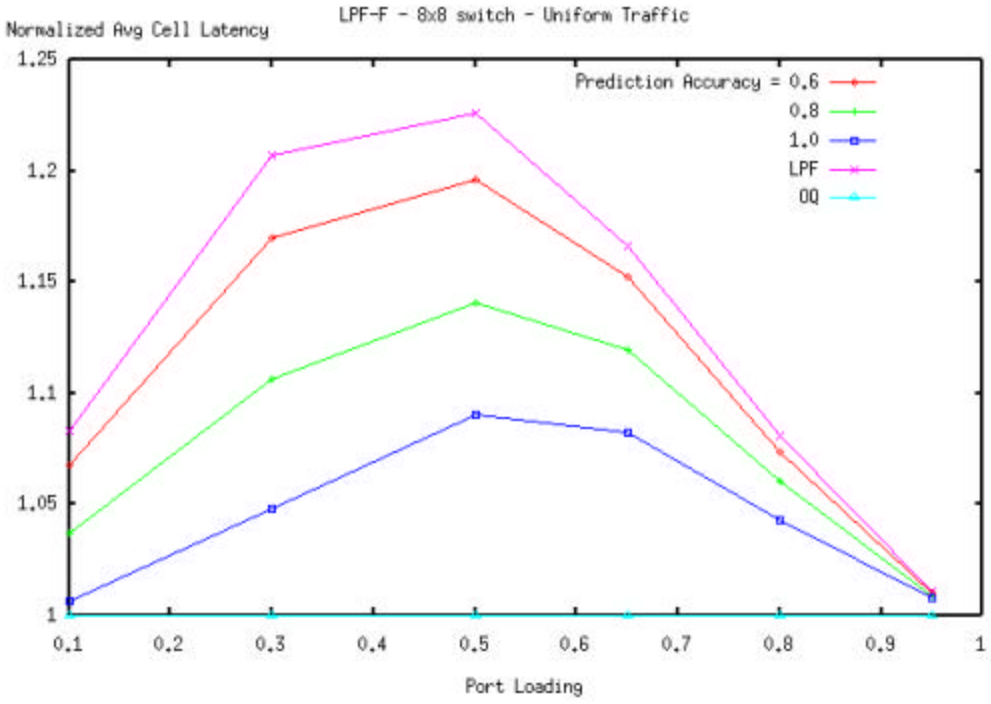



**Figure 2: LPF-F Simulation Results - 8x8 Switch**

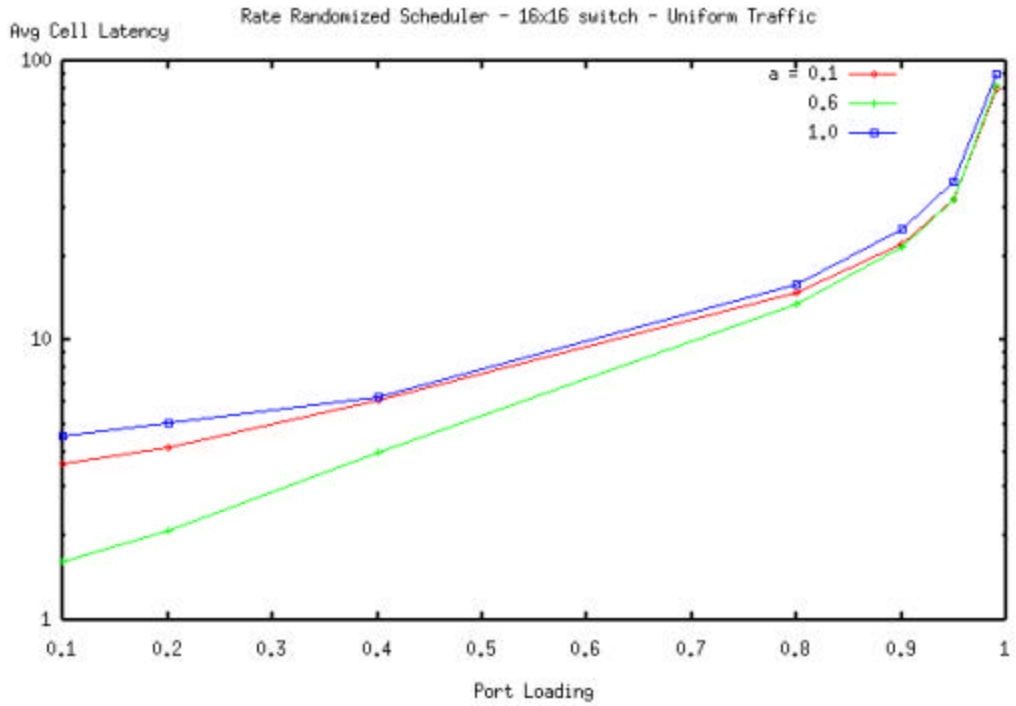



**Figure 3: RRS – 16x16 Switch. Uniform Traffic**

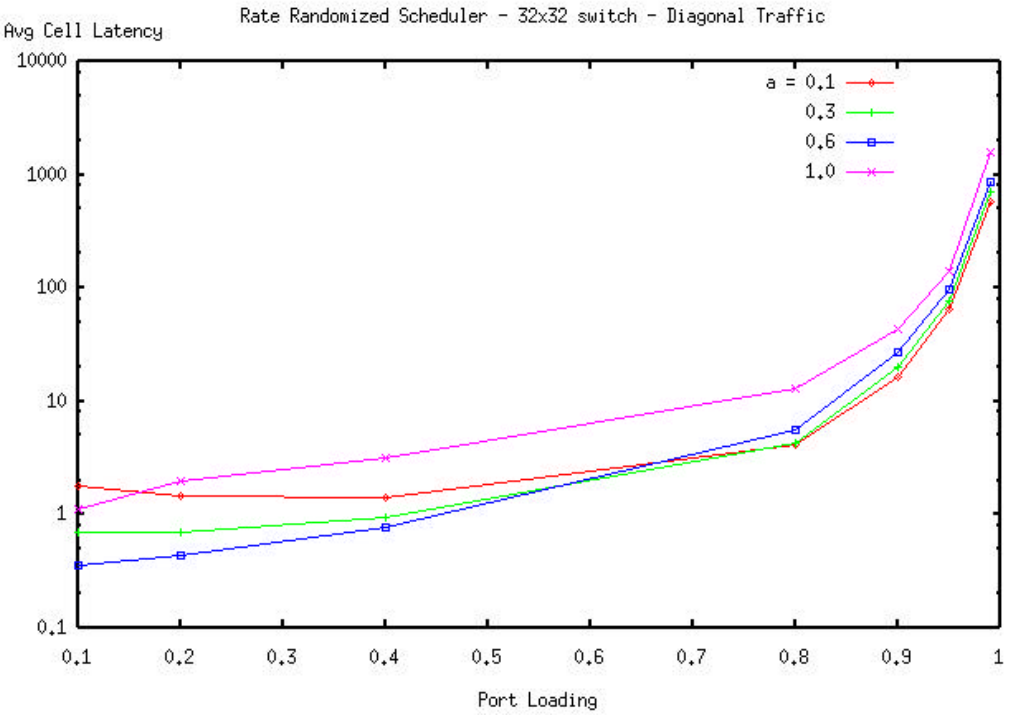



**Figure 4: RRS – 32x32 Switch. Diagonal Traffic**

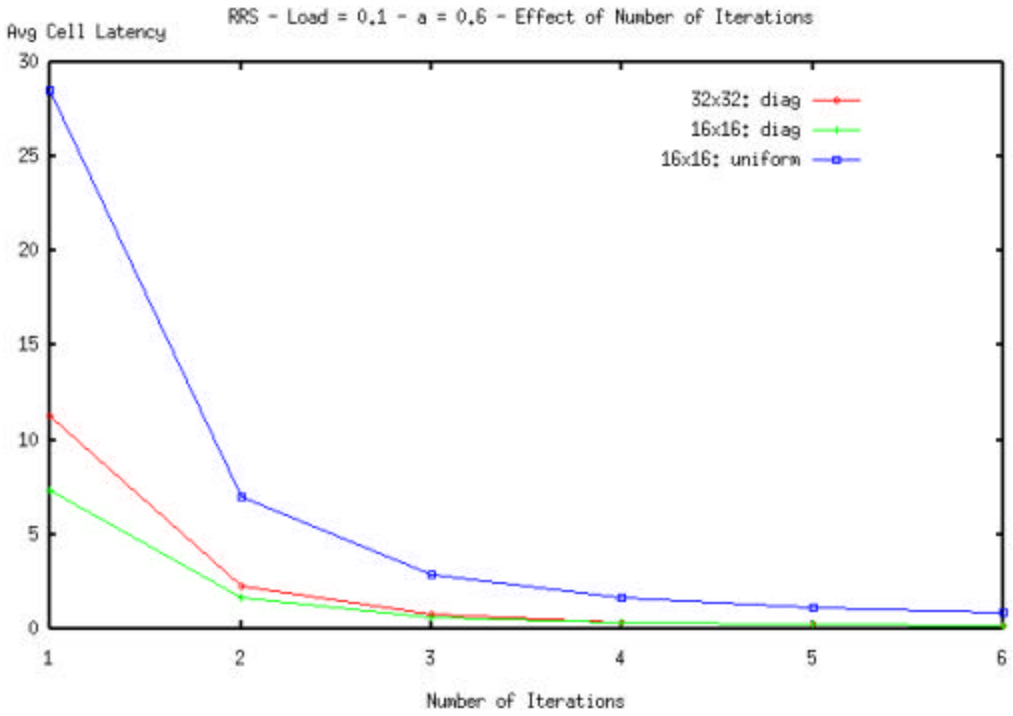



**Figure 5: RRS – Effect of Number of Iterations**
**Load = 0.1, a = 0.6**

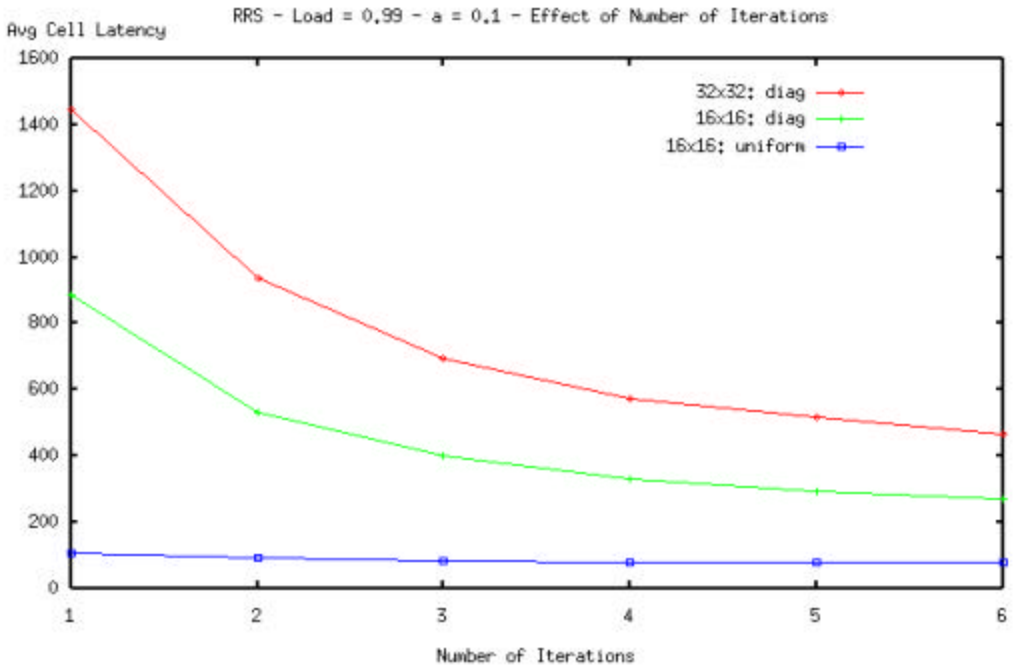



**Figure 6: RRS – Effect of Number of Iterations**
**Load = 0.99, a = 0.1**

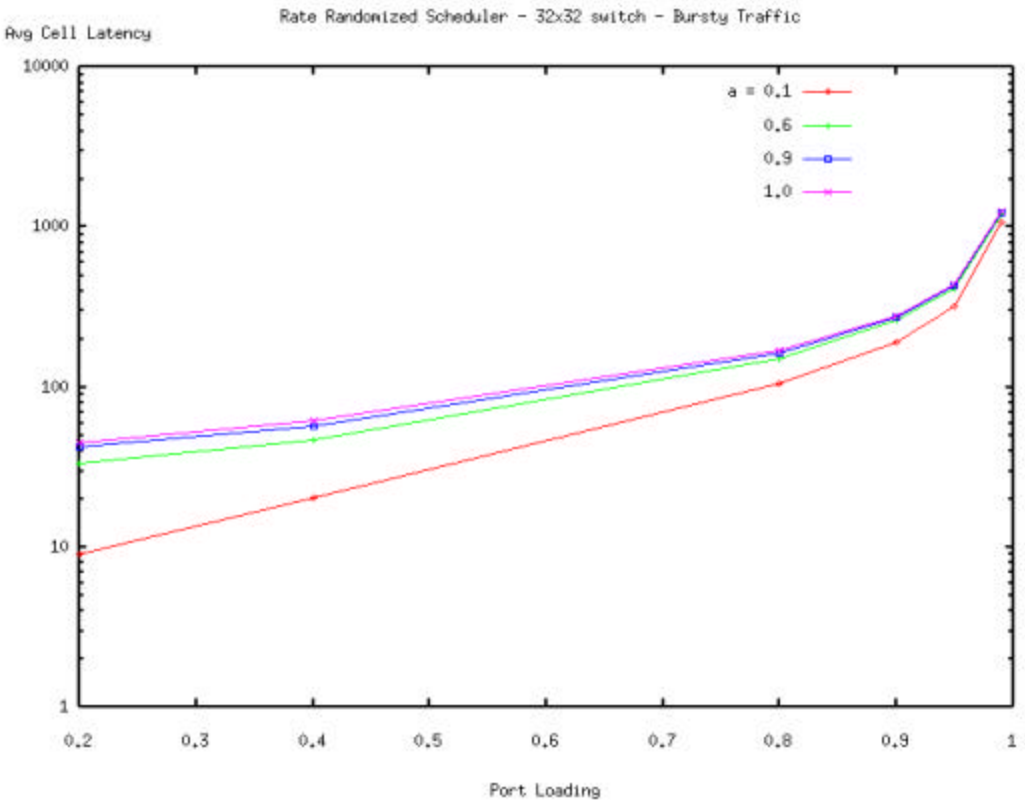



**Figure 7: RRS – 32x32 Switch. Bursty Traffic**

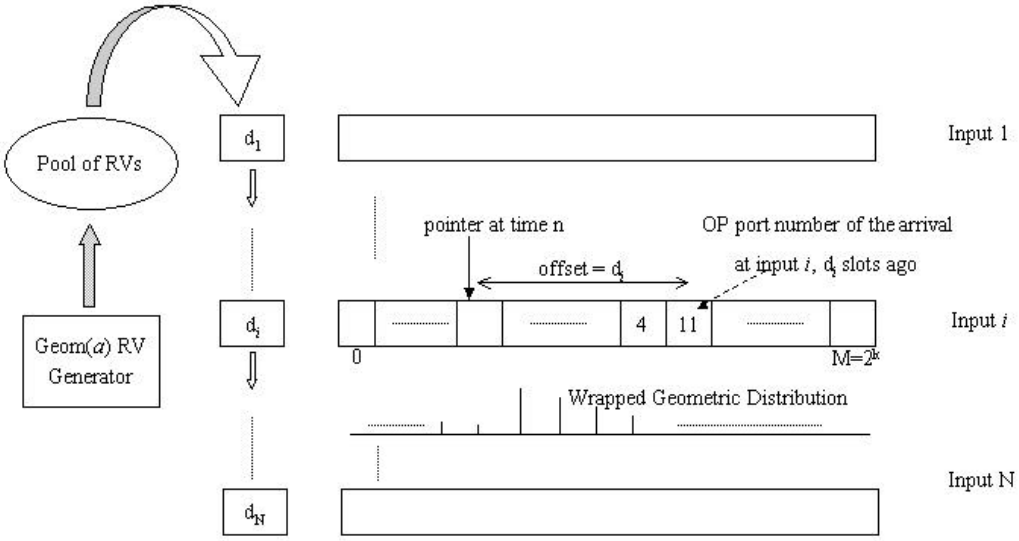



**Figure 8: Hardware Implementable Version of RRS**

## *IX. References*


[McK99] N. McKeown, et al., *Achieving 100% Throughput in an Input-Queued Switch*, 1999

[Brow01] T. Brown, H. Gabow, *Future Information in Input Queueing*, 2001

[Pax97] V. Paxson, S. Floyd, *Why We Don't Know How To Simulate The Internet*, 1997

[Tass98] L. Tassiulas, *Linear Complexity Algorithms for Maximum Throughput in Radio Networks and Input Queued Switches*, 1998

[Giac01] P. Giaccone, B. Prabhakar, D. Shah, *Towards Simple, High-performance Schedulers for High-aggregate Bandwidth Switches*, 2001

[Mekk98] A. Mekkittikul, N. McKeown, *A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches*, 1998