A new approach for scheduling in cell-based routers

GRIGORIS BAKLAVAS
American College of Thessaloniki
Department of Computer Science
Pylea, PO Box 21021, Thessaloniki
GREECE

MANOS ROUMELIOTIS
University of Macedonia
Department of Applied Informatics
156 Egnatia Str., Thessaloniki
GREECE

Abstract: A new algorithm for scheduling the transmission of packets in routers is discussed. The approach takes into account the separation of queues in the input ports of a router and the segmentation of incoming packets into cells. Its main advantage is that, while performing as good as other existing algorithms, its complexity is considerably smaller. The feature that differentiates the algorithm is that it relies on past information in order to avoid processing of all data at every given timeslot, which is a time-consuming process.

Key–Words: Input Queue Switches, Packet Scheduling

1 Introduction

The Internet was initially designed as a means to provide low cost global communication. As time progressed, it was obvious that the inherent features of the Internet were limiting its scalability in terms of dimension, services and business. Internet is a network providing only one service to its customers, named ”best-effort”; this means that the network will do its best to transfer the data, but it does not make any guarantees for the quality of this service over time. On the other hand, the transmission of multimedia and most of the other advanced applications require a minimum guaranteed level of service quality in order to operate. In general, providing Quality-of-Service (QoS) in the Internet has become one of the most significant tasks in the networking world.

For the Internet to improve its performance, changes have to be applied to one of its core devices, the router. The main area where bottlenecks appear in a router is the scheduler module. This is a procedure that selects the packet to be transferred through the switching fabric. This paper discusses the design of an efficient scheduling algorithm, which shows very good performance and can be implemented in hardware. The architecture of the scheduler will be discussed from the algorithmic point of view, and the paper will not focus on the technologies needed to implement it.

Several proposals for switching architectures exist in the literature. Some of them result from the innovations in the telephony world. A good survey of these architectures is included in [1]. All packet switching architectures are based on a switch fabric and some buffers. The role of the switch fabric is to forward the packets from the inputs to their destination ports. The two most common architectures of switch fabrics are the following:

- crossbar: this architecture comes from the old electro-mechanic telephony crossbars, with N ports and N^2 contact points. This is a strictly non-blocking switching fabric, meaning that whenever a free input port should be connected to a free output port, they can be connected always (”non-blocking”) and without reconfiguring all the other connections (”strictly non-blocking”). Despite the fact that the crossbar does not scale well for a large number of ports, it is considered as the preferred switching fabric in the design of high-speed routers. The crossbar can work in parallel and transport up to N packets coming from different inputs heading to different outputs. At times where the crossbar’s transfer rate is close to the aggregate arrival rate, the bottleneck is caused mainly by the procedure that coordinates the transfer of packets. This is extremely important especially when packets arriving at different input ports have to be transferred to the same output. In this case, there is a contention for the common destination that the scheduler has to solve: one of the contending packets is chosen and transferred to the desired output, while at the same time, all others are stored in queues.

- bus: the bus is also strictly non-blocking, with the difference that it allows only one packet to be transferred at the same time. As a consequence,

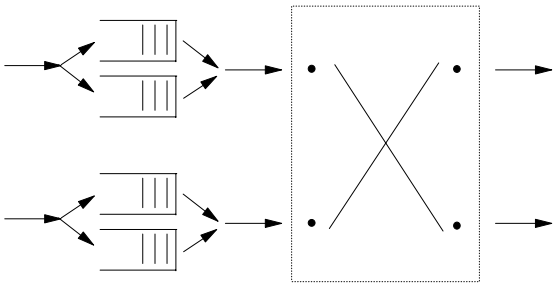

Figure 1: Input Queued (IQ) Switch

it requires the coordination of all ports. The performance of this architecture is limited by the bus capacity and the arbitration process.

As far as the buffers are concerned, they can be shared among different ports, at input or output level; in which case shared memory fabrics are used [2]. The main limiting factor of the performance of the buffer is the memory bandwidth.

In the design of the algorithm and the simulations made, we considered only the crossbar architecture. The switching fabric is integrated with the input-queue (IQ) buffering scheme presented below, which stores packets contending for the same switching resource.

At this point we have to make a clear distinction between a packet and a cell. A packet is defined as a data-unit with variable size, like an IP-datagram or a TCP/IP packet. Borrowing from the ATM terminology, we shall use the term cell to identify a fixed-size data unit. To make things clearer, the basic switching architectures will be referred as cell-switches, whereas the overall switching system of a router will be referred as packet-switch.

Each port of the router has line interfaces where the data-link and physical layer protocols are used to receive and transmit IP datagrams. The IP protocol is located on top of the data-link protocol at the input and at the output of the router. Inside the IP layer at the input, the operations that associate an output port with the destination IP address take place (we do not consider issues such as table look-up or others related to the implementation of these functions). The incoming IP datagrams have to be segmented into ATM cells, which will then be transferred to output ports by the ATM switching fabric.

When the cells arrive at an output port, they are reassembled into the original IP datagram, and then transmitted on the output line according to the line format.

In the literature, many queueing techniques for routers exist. Figure 1 shows the architecture of an

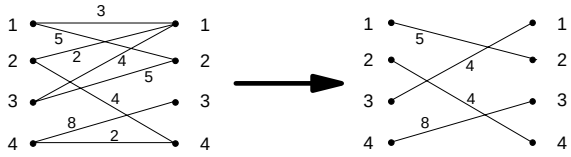

Figure 2: From the graph to the matching

input-queued (IQ) switch. We assume that the incoming data units are of fixed-size, for example ATM cells. In practice, our results do not strictly require that fixed-size cells are used, but refer to any switch that takes switching decisions at equally spaced time intervals. The time between two consecutive switching decisions is called a timeslot.

Cells are stored at the input interfaces. At each input, we have N queues, one for each output. Consequently, a total of N^2 queues are present. Each of these queues has a capacity of L cells and when a cell arrives finding its queue full, it is dropped. This method of separating the queues is well known in the literature and is called Virtual Output Queuing (VOQ) or Destination Queuing [3].

2 Role of the scheduling Algorithm

In most cases, scheduling problems in switches are modelled as matching problems in bipartite graphs. The switch can be described as a bipartite graph $G = [V, E]$ (see Figure 2) in which the N nodes to the left represent the N input interfaces of the device, whereas the N right-most nodes designate the N output ports. When an edge connects two ports, the need for cell transfers in this direction is indicated. These edges can be labelled with a weight (denoted by w_{ij}), representing the "urgency" to serve the queue VOQ_{ij}. The metrics used to assign weights to edges is a key part of the scheduling algorithm. In some cases, the weight is binary and simply indicates whether the queue is empty or not. Other possibilities are that it refers to the number of cells to be transferred, or to the time waited by the oldest cell in the queue.

The job of the scheduling algorithm is to select a matching M, i.e., a set of input-output pairs with no conflicts, such that each input is connected with at most one output, and each output is connected with at most one input. For every matching, if input i is connected to output j, a cell is removed from VOQ_{ij} and transferred to output j. It can never happen that two cells are extracted from the same input, or that two cells are transferred to the same output.

The weight of a matching is the sum of the metrics corresponding to the edges included in the matching. A matching has maximum size if the number of

edges is maximized; a matching has maximum weight if its weight is maximized. A matching is heavier than another matching if its weight is greater than the other one.

In [4] there is a description of the maximum weight matching (MWM) algorithm, that can be proven to yield the maximum achievable throughput using as metrics either the number of cells to be transferred or the time waited by the oldest cell. It is a variation of the Hungarian Algorithm for bipartite graphs. It is shown that the complexity of MWM is $O(N^3)$.

The main problem that makes the implementation of MWM unfeasible is its complexity. To address this issue, a number of practicable scheduling algorithms were proposed. Some of them are: iSLIP [5], iLQF [6], RPA [7], MUCS [8] and ALG [9]. However, these algorithms perform poorly compared to MWM when the input traffic is non-uniform: they induce very large delays and they experience severe packet loss rates.

The new algorithm for scheduling cell switches, proposed in the following section, is shown to perform very close to MWM not only in the uniform traffic scenario, but also under other traffic conditions. We will prove that its complexity is $O(Nlog_2^2 N)$ or $O(N^2)$ depending on the version.

3 The FSA Algorithm

The fundamental idea that the algorithm is based on is the correlation of the state of the system along the time. Note that packets arrive (depart) at most one per input (output) per time slot. This means that queue lengths, taken to be the weights by MWM, change very little during successive time slots. Thus, a heavy matching will continue to be heavy for a few more time slots, suggesting that carrying some information, or retaining memory, between iterations should help to simplify the implementation while maintaining a high level of performance. In [9], Tassiulas was the first to exploit this fact to design a scheduler for an IQ switch.

The algorithm presented in this paper is named Fast Scheduling Algorithm (FSA). Its main feature is that it exploits randomization. In a variety of situations where the scalability of deterministic algorithms is poor, randomized algorithms are easier to implement and provide a surprisingly good performance. The main idea is simply stated: Basing decisions upon a few random samples of a large state space is often a good surrogate for making decisions with complete knowledge of the state. In [9], Tassiulas proposed a pure randomized scheme for scheduling in IQ switches and its stability properties were stated. Some other hybrid schemes were also proposed to exploit

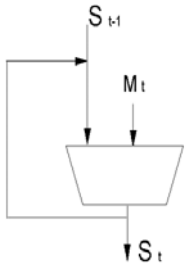

Figure 3: The FSA algorithm

randomization in scheduling.

The mathematical foundations and the stability properties of FSA are presented in detail in [10]. FSA takes as input the matching of the previous timeslot, named S_{t-1}. Then, it iteratively augments the weight of this matching by combining its heavy edges with the heavy edges of a (non-uniformly) randomly chosen matching M_t. Figure 3 shows a schematic representation of the architecture of FSA.

The algorithm consists of an initialization stage, and then three steps that are repeated for as long as it is necessary. The procedure is described as follows:

Initialization. Let S_t be the matching used by FSA at time t. Let $L_f(M)$ be the minimal set of edges in a matching M whose weight is at least f, where f a fraction of its weight $0 \leq f < 1$. Initially, all inputs and outputs are considered to be unmarked. At time t, FSA will calculate a new matching M_t in the following way:

Step 1. Let i be the current iteration number. Let $k \leq N$ be the number of unmarked input ports. Obviously, at the same time, we would have k unmarked outputs. Out of the $k!$ possible matchings between these unmatched input-output pairs, a matching $M_i(k)$ is chosen uniformly at random.

Step 2. If $i < I$, retain the edges corresponding to $L_f(M_i(k))$ and mark the nodes they cover. If $i = I$, then retain all edges of $M_i(k)$.

The above steps are repeated I times.

Step 3. The matching M_t produced by the above procedure is then merged with the matching of the previous timeslot S_{t-1} in the following way:

Let G be a bipartite graph and two matchings M_1 and M_2 for this graph. Let E_1, E_2 denote the edges of these matchings. We start from output node j_1 and follow the edge belonging to E_1 to an input node i_1. From input node i_1, we follow the (only) edge belonging to E_2 that leads to the output node j_2. If $j_2 = j_1$, stop. Else continue to follow the above procedure until we reach j_1 again. In the end, we will have a 'cycle' of visited nodes. When we finish with the first 'cycle', we repeat the same procedure starting from an unvis-

ited output. In the same fashion, we find all cycles in the graph. Suppose there are m cycles, $C_1, \ldots, C_m$ at the end. Each of these cycles will contain two matchings: E_{1i} containing only E_1 edges, and E_{2i}, which has only E_2 edges. The matching returned (S_t) is given by the following equation:

$$S_t = \bigcup_{i=1}^{m} \max_{S \in \{E_{1i}, E_{2i}\}} SX_t, \quad (1)$$

where X_t is the queue occupancy vector at time t.

3.1 Versions of FSA

The FSA algorithm was tested under a variety of different scenarios both in terms of the arrival distribution and the settings of its variables. According to the simulations performed, we have arrived at three different versions of the algorithm. These versions are:

- FSA-1, where we set the number of iterations of every step to be equal to 1.
- FSA-M, standing for FSA-Maximized, where we perform a maximization of the matching produced by the algorithm. The maximization is done in the following way:

  Assume that there are k unmatched input and k unmatched output ports in S_t. Out of these, the algorithm finds a matching where all edges have non-zero weight.

Simulation results comparing the three version of the algorithm are shown in section 4.1.

3.2 Complexity of FSA

To estimate the complexity of FSA, we first need to find the complexity of its three steps. The complexity of the first two steps is estimated as follows:

Given that the selection of the random samples is independent from the state of the queues, its complexity is considered to be equal to finding a random sample. We assume that there are I iterations in the random sample selection. In each of these iterations, a matching on the "remaining nodes" is chosen uniformly at random. Since this selection does not depend on the state of the weight matrix, which is $O(1)$ operation.

The edges that constitute the "heaviest" fraction of this random matching are kept and the rest of the nodes are used for the next iteration. In the last iteration, the selection of heavier edges is avoided.

The selection of heavy edges involves:

1. The ordering of the edges according to their weights, which is $O(Nlog_2N)$.
2. Choosing the heavier edges that constitute the f fraction of the net weight, which is $O(N)$.

Since there are I iterations, the first phase of the algorithm is $O(INlog_2N + IN)$.

The complexity of the third step is $O(N)$, since each one of the $2N$ edges is visited at most once.

As far as FSA-M is concerned, the complexity of the matching maximization is estimated as follows:

Assume that there are k unmatched input and k unmatched output ports in S_t. Out of these, the algorithm finds a matching where all edges have non-zero weight. The complexity of this procedure is $O(k^2)$, as in the worst case k^2 comparisons are needed.

The complexity of FSA is therefore $O(INlog_2N + IN)$ without the maximization and $O(N^2)$ if we decide to maximize the matching. In our simulation study, we set $I = log_2N$. Thus, the running time of the FSA algorithm is $O(Nlog_2^2N)$.

4 Simulations

To evaluate the performance of the scheduling algorithms, we conducted a large number of simulation experiments. Each input queue VOQ_{ij} was assumed to be able to store a finite number of cells Q_{LIM}: when a cell directed to output j arrives at input i, and queue VOQ_{ij} is full, the cell is dropped. No buffer sharing among queues is allowed.

Simulation runs were executed until the estimate of the average cell delay reaches with probability 0.95 a relative width of the confidence interval equal to 2%.

A router simulator has been developed to study the performance of the considered algorithms. The simulator has been written in the special simulation language MODSIM III and is described in detail in [11].

Main features of the simulator are the following:

- support of the majority of the router architectures
- flexible traffic generation
- support of a large number of network algorithms
- support of large switches

In our simulations, we considered traffic generated at cell level. Each cell is generated according to an i.i.d. Bernoulli process such that the normalized load ρ never exceeds 1.

The traffic scenarios used in our tests were the following:

Table 1: Simulation Parameters

Parameter	Symbol	Value
Switch Size	N	32
Iterations	I	5
Selection Fraction	f	0.5
Buffer Size	Q_{LIM}	10,000

- Uniform scenario. This scenario is the common test-bed in literature. All queues have equal probability of having an arrival.

- Diagonal scenario. Incoming arrivals at input i are headed either to output i or, with double probability, to output $i + 1$.

4.1 Simulation Results

In this section, we study the performance of different algorithms on different test cases, in terms of different performance measures. The traffic patterns used to obtain these results are stationary in distribution.

We study the performance of the following algorithms: MWM, iSLIP, iLQF, MUCS, RPA, ALG along with FSA and FSA-M.

The assumptions used in the simulations are presented in table 1.

First we consider uniform traffic, which is the most basic scenario. Figures 4, 5 and 6 present the performance of different algorithms with respect to Mean Delay, Mean Queue Length and Maximum Queue Length respectively.

With respect to all measures, all algorithms show the same qualitative behavior. The following are interesting results to note:

1. Both FSA and FSA-M experience zero loss rates, which is the basic property of efficient scheduling algorithms.

2. When the load is below 0.8 (not shown in the graphs), FSA performs worse than most of the other algorithms, but this misbehavior is practically negligible. This can be easily explained: FSA is a randomized algorithm that does not necessarily find a maximal matching at lower values of load. In FSA-M, we force it to be a maximal matching. Hence, FSA-M performs as well as any other algorithm, at lower load values. For high values of load, FSA and FSA-M behave identically.

3. ALG does not lose packets, but its delay is much worse than the other algorithms.

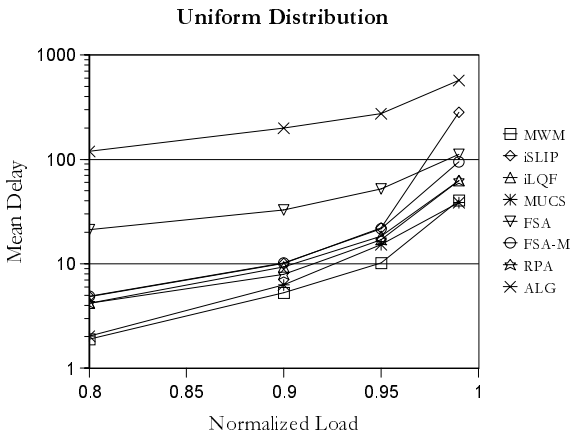

Figure 4: Load vs Delay: Uniform Scenario

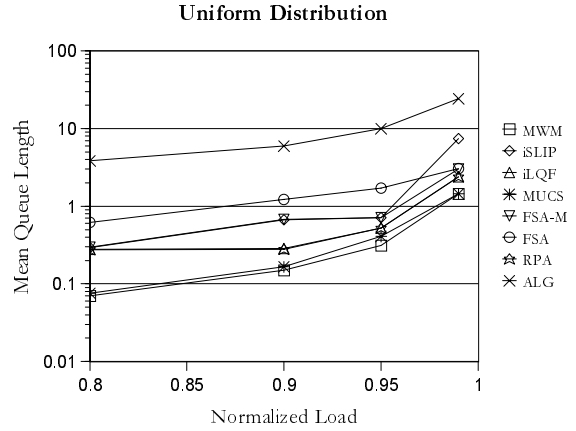

Figure 5: Load vs Mean Queue Length: Uniform Scenario

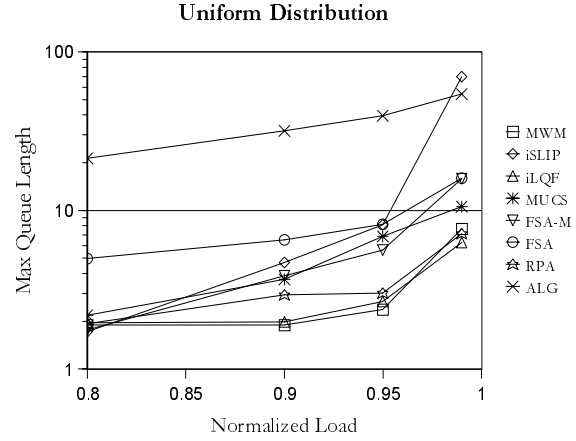

Figure 6: Load vs Maximum Queue Length: Uniform Scenario

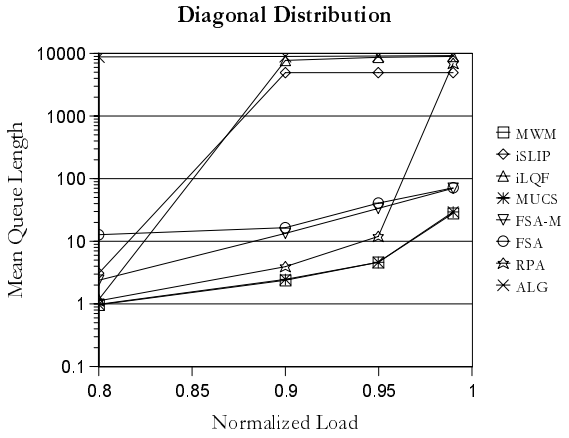

Figure 7: Load vs Mean Queue Length: Diagonal Scenario

We now consider diagonal traffic, as described above. With respect to all metrics, relative performances of the algorithms remain the same. Hence we will present only results with respect to Mean Queue Length. Figure 7 plots Mean Queue Length for different algorithms. For high load values, iSLIP, iLQF, RPA experience losses. MUCS and MWM perform identically. ALG starts to experience losses for loads around 0.4. Hence, ALG experiences unacceptable average delays from a practical point of view.

Both FSA and FSA-M experience no packet loss and perform almost as good as MWM. As before, FSA performs worse than FSA-M for lower load values, but they perform identically for higher values.

5 Conclusions & Further work

In this paper we have discussed the design of scheduling algorithms for high speed switching systems. We have focused our study on input-queued switches, since they are the most promising architectures in terms of scalability with speed and number of ports.

We have described the main issues involved in the design of high performance routers and have motivated the problem of scheduling for switches. After defining the problem of scheduling as a problem of finding a matching, we have introduced the FSA algorithm and the simulations that compared its performance with the other algorithms in the area.

So far, the scheduling algorithms known in literature have been either simple to implement but with poor performance or too complex but with good performance. We proposed a new scheduling algorithm, with low complexity and very good performance in terms of delays and throughput. The main ideas we exploited have been the memory of the state of the queues from the past and randomization. Further work on the algorithm will include its testing under different sets of assumptions (number of ports, buffer size, selection fraction) and different traffic conditions in terms of arrival distribution.

References:

[1] Pattavina A., ”Switching theory: architectures and performance in broadband ATM networks”, Wiley, John & Sons, Dec. 1997

[2] Awdeh Ra’ed Y., Mouftah H.T., ”Survey of ATM switch architectures”, Computer Networks & ISDN Systems, vol. 27, n. 12, Nov. 1995, pp. 1567-1613

[3] Anderson T., Owicki S., Saxe J., Thacker C., ”High speed switch scheduling for local area networks”, ACM Transactions on Computer Systems, vol. 11, n. 4, Nov. 1993, pp. 319-351

[4] Tarjan R.E., Data structures and network algorithms, Society for Industrial and Applied Mathematics, Pennsylvania, 1983

[5] McKeown N., ”iSLIP: a scheduling algorithm for input-queued switches”, IEEE Transactions on Networking, vol. 7, n. 2, Apr. 1999, pp. 188-201

[6] McKeown N., Mekkittikul A., ”A starvation free algorithm for achieving 100% throughput in an input queued switch ”, ICCCN’96, Rockville, MD, Oct. 1996, pp. 694-700

[7] Ajmone Marsan M., Bianco A., Leonardi E., Milia L., ”RPA: a flexible scheduling algorithm for input buffered switches”, IEEE Transactions on Communications, vol. 47, n. 12, Dec. 1999, pp. 1921-1933

[8] Duan H., Lockwood J.W., Kang S.M., Will J.D., ”A high performance OC12/OC48 queue design prototype for input buffered ATM switches”, IEEE INFOCOM’97, vol. 1, Kobe, Japan, 1997, pp. 20-28.

[9] Tassiulas L., ”Linear complexity algorithms for maximum throughput in radio networks and input queued switches”, IEEE INFOCOM’98, vol. 2, New York, NY, 1998, pp. 533-539

[10] Baklavas G., Roumeliotis M., ”Stability conditions of a recursive routing algorithm”, International Journal of Pure and Applied Mathematics, vol. 14, no. 1, Apr. 2004, pp. 75-89.

[11] Baklavas G., Souravlas S., Roumeliotis M., ”Modeling Queue Disciplines on a Multi-Port Router”, Proceedings of CSCC 2001, Rethymno, Crete