Int. J. Electron. Commun. (AEÜ) 69 (2015) 629–635

Contents lists available at ScienceDirect

International Journal of Electronics and Communications (AEÜ)

journal homepage: www.elsevier.com/locate/aeue

GENIUS – A genetic scheduling algorithm for high-performance switches

Emilio C.G. Wille, José R. Hoffmann

Graduate Program in Electrical and Computer Engineering, Federal University of Technology – Paraná, Av. Sete de Setembro 3165, 80230-901, Curitiba, PR, Brazil

ARTICLE INFO

Article history:
Received 7 October 2013
Accepted 1 December 2014

Keywords:
Routers
Input-queued switches
Scheduling algorithms
Genetic algorithms
Performance evaluation

ABSTRACT

The most important switching systems in the Internet are the routers. Current routers employ input-queued crossbar switches that require sophisticated scheduling techniques for packet transmission. The performance of the router then directly depends on the scheduling algorithm, considering its throughput and complexity. In this paper, a new genetic-based scheduling algorithm, called GENIUS, is developed and tested for high-performance Internet switches operation. The GENIUS approach presents low complexity and the results show a performance close to the optimal.

1. Introduction

Internet is a very large packet switched network, built around a large variety of transmission and switching systems. The most important switching systems in the Internet are the routers. The main tasks of a router are to receive the packets from input ports, to find their local destination port on the basis of the routing table and finally to transfer the packets to output ports. These two basic functions, *routing* and *switching*, are very difficult to implement when the aggregate bandwidth is very large, since complex algorithms should run in a very short time. The design of high performance routers is particularly important, given that today's Internet is composed by a relatively small number of very fast backbone networks, which connect a very large number of smaller networks. All the architectures for packet switching are constituted by a *switch fabric* and some buffers. The switch fabric forwards the packets at the inputs to their destination ports.

Fig. 1 shows the logical structure for an input-queued (IQ) switch [1]. The switch operates on fixed-size cells (or packets), and takes switching decision at equally-spaced time instants. The time between two switching decisions is called time slot. During each time slot a *scheduling algorithm* decides the configuration of the switch by finding a *matching* on a bipartite graph.

We consider a switch with N inputs and N outputs, and assume that all input and output lines run at the same speed. Packets are stored at input interfaces. Each input manages one queue for each output, hence a total of $N \times N$ queues are present. Each queue can store up to M cells and excess packets are dropped. This queue separation, called *virtual output queuing* (*VOQ*), permits to avoid performance degradations due to head-of-the-line (HoL) blocking [2]. The switch fabric is non-blocking and memoryless, and introduces no delay: at most one cell can be removed from each input and at most one cell can be transferred to each output in every time slot. The scheduling algorithm decides which cells are transferred from the inputs to the outputs of the switch in every time slot. Hence, the router performance is more influenced by the scheduling algorithm, considering its throughput and complexity. A number of scheduling algorithms can be found in literature presenting, however, various degrees of performance [1,3–6].

Due to the high complexity that matchings problems usually show, the use of intelligent methods is a mandatory requirement when facing them. In this sense, evolutionary computation emerge as efficient methods able to solve this kind of problems. Genetic algorithms (GAs) are examples of evolutionary computation methods used to search, optimization and machine learning. They are based on concepts of natural selection and natural genetics; and can handle arbitrary kinds of constraints and objectives [7]. GAs have been used to tackle a wide variety of problems in an extremely diverse variety of fields as can be seen, for example, in [8,9]. In this paper, the GA's innate features of search in changing environments [10] are explored in the construction of a scheduling algorithm, presenting good performance, called GENIUS (*Genetic Scheduling Algorithm for High-Performance Switches*). This intuition has been already proposed in the literature [11,12]. However, we assess the effectiveness of the proposed approach considering an in-depth analysis. Two versions are proposed and analyzed: the elitist version GENIUS-E, in order to achieve performance close to optimal, and the simplified version GENIUS-S with reduced complexity.

http://dx.doi.org/10.1016/j.aeue.2014.12.001
1434-8411/

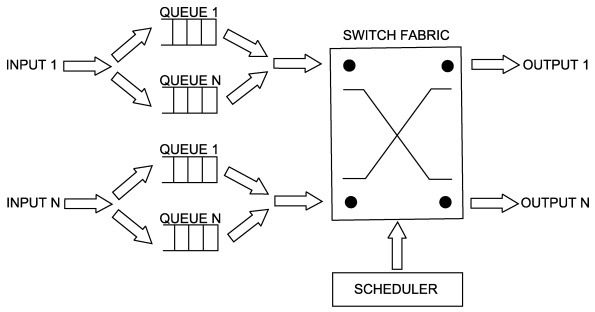

Fig. 1. Logical structure of an input-queued cell switch.

In summary, the main contributions of the paper are the following: (i) we propose a scheduling algorithm for IQ switches which is based on the GA's natural characteristics of search in changing environments; (ii) we study the system behavior as a function of various parameters, and evaluate several performance metrics of interest; (iii) we demonstrate that the genetic approach is quite robust to several input traffic patterns; and (iv) we found results at least as good as those from current scheduling algorithm found in the literature. The remainder of this paper is structured as follows. Section 2 presents the architecture and operation of the IQ switch as well as the mathematical notation adopted. Relevant scheduling algorithms related to our proposal and their basic functioning are presented in Section 3. Section 4 describes the proposed genetic-based scheduler and its variants. In Section 5 a set of numerical results is discussed as well as survival and transient analysis are performed. Related work is considered in Section 6. Finally, concluding remarks and future directions are presented in Section 7.

2. Switch model

We consider an $N \times N$ input-queued switch. The arrival process at input i, ($i = 1, \ldots, N$), is a discrete-time process of fixed sized packets, or cells. At the beginning of each time slot, either zero or one cell arrive at each input. Each cell contains an identifier that indicates which output j, ($j = 1, \ldots, N$), it is destined for. The buffer at input i is partitioned into N virtual output queues. When a cell destined for output j arrives at input i it is placed in the FIFO queue VOQ_{ij}. Denote $Q(t) = [q_{ij}(t)]$ the $N \times N$ matrix that records the lengths of all $VOQs$ at time t.

Let λ_{ij} be the average rate at which packets arrive at input i for output j, and let $\Lambda = [\lambda_{ij}]$ be the average arrival rate matrix, also called the *load matrix*. We require the load matrix to be admissible:, $\forall i$, and, $\forall j$. In words, this condition ensures that no input or output is oversubscribed [13]. The maximum input load is given by $\rho = \max_i(\sum_j \lambda_{ij})$.

A feasible connection configuration can be seen as a *matching* in a bipartite graph, where inputs and outputs correspond to nodes in the graph and an edge between input i and output j denotes that they are connected or matched. We use the binary variables $x_{ij}(t)$, to denote connections. Input i is connected with output j at time t if, and only if, $x_{ij}(t) = 1$. Without loss of generality, we consider only complete connections; that is, we allow a connection between input i and an output j even if $q_{ij}(t) = 0$.

The FIFO queues are served as follows. A scheduling algorithm selects a matching, X, between the inputs and outputs. At the end of the time slot, if input i is connected to output j, one cell is removed from VOQ_{ij} and sent to output j. Let $X(t) = [x_{ij}(t)]$ denote the matching matrix at time t. Denote the *weight* of a matching $X(t)$ as $W[X(t)] = \sum_{ij} q_{ij}(t)x_{ij}(t)$, taking the weight of the edge between input i and output j to be equal to the queue-length $q_{ij}(t)$.

Note that for an $N \times N$ switch the set of all possible matchings, denoted by S, has a cardinality of $N!$

3. Scheduling algorithms

3.1. MWM algorithm

A scheduling algorithm of particular interest is the *Maximum Weighted Matching* (MWM). The MWM algorithm chooses, at each time t, the matching with the highest weight. More precisely, if $X_{MWM}(t)$ denotes the matching determined by MWM at time t, then:

$$X_{MWM}(t) = \arg\max_{X \in S}\{W[X]\} \quad (1)$$

There are two main quantities for measuring the performances of a switch scheduling algorithm: throughput and delay. Past theoretical works on packet switches have been concerned with studying algorithms that achieve 100% throughput. Such algorithms are referred to as stable algorithms. The MWM scheduler delivers up to 100% throughput for all admissible independent and identically distributed Bernoulli input traffic patterns [14,15]. The literature shows that the MWM algorithm presents low delays (by maintaining queue sizes small). However, being the problem solved by applying the Hungarian algorithm [16], it presents temporal complexity $O(N^3)$.

3.2. APSARA algorithm

The APSARA (*A Parallel Scheduling Algorithm and its Random Approximation*) scheduling algorithm was proposed in [13]. APSARA presents good performance and, being based on a population of solutions, is taken as a basis for comparison in this work.

The following principles are fundamental for APSARA: (a) Queue lengths change very little during successive time slots, suggesting that a heavily weighted matching will continue to be heavily weighted for a few more time slots, and (b) Two randomly chosen matchings that differ by very few edges will quite likely be just as heavy [13].

Let $X(t)$ be the matching determined by APSARA at time t, and let $\mathcal{N}[X(t)]$ be the neighborhood of $X(t)$, i.e., the set of matchings that differ by few edges of $X(t)$. Let $Q(t+1)$ be the queue lengths at the beginning of time $t+1$. At time $t+1$, APSARA works as follows:

- Determine $\mathcal{N}[X(t)]$ and the matching $Z(t+1)$ that correspond to a Hamiltonian walk[1] at time $t+1$.
- Let $S(t+1) = \mathcal{N}[X(t)] \cup \{Z(t+1) \cup X(t)\}$. Compute the weight of every matching $Y \in S(t+1)$ as $W[Y] = \sum_{ij} y_{ij} q_{ij}(t+1)$.
- Determine the matching with the highest weight, denoted $X(t+1)$, from the set $S(t+1)$.

This basic version (APSARA-B) requires computing the weight of neighbor matchings. Such computation is easy because in APSARA each neighbor Y differs from matching $X(t)$ in exactly two edges. However, computing the weights of all $\binom{N}{2}$ neighbors, if done in parallel as shown in Fig. 2, will require a lot of space in hardware for large values of N.

If one wants to build a switch with a larger number of ports the number of parallel modules will make the system prohibitively

[1] A Hamiltonian walk on the set of all matchings is defined as follows [17]. Consider a graph with $N!$ nodes, each corresponding to a distinct matching, and all possible edges between these nodes. If $Z(t)$ is a Hamiltonian walk on this graph; then, $Z(t)$ visits each of the distinct nodes exactly once during times $t = 0, \ldots, N! - 1$. For $t \geq N!$ it is defined $Z(t) = Z(t \bmod N!)$.