Dynamic Weighted Round Robin in Crosspoint Queued Switch

Soko Divanovic, Milutin Radonjic, Member, IEEE, Gordana Gardasevic, Member, IEEE, Igor Radusinovic, Member, IEEE

Abstract — Achievement of desired performance levels (quality of service - QoS) in switches and routers is one of the most important tasks in switching systems. This implies providing guaranteed throughput, cell loss probability, average and maximal latency within the required bounds. To achieve this task in crosspoint queued switch, we implemented the weighted round robin (WRR) algorithm. We showed that WRR algorithm can achieve very good performance levels regarding the throughput and latency, but had major drawback because it require knowledge of arrival traffic. To overcome this problem, we implemented and presented in this paper the dynamic weighted round robin (DWRR) algorithm that can work with unknown arrival traffic. We showed that DWRR can achieve same performance as WRR, without the need for incoming traffic information, which makes it suitable for practical implementation.

Keywords — WRR, DWRR, throughput, average latency.

I. INTRODUCTION

One of the main tasks of modern telecommunication networks is to provide guaranteed performances, i.e. Quality of Service (QoS) for the end user [1]-[2]. This implies providing guaranteed throughput, cell loss probability, average and maximal latency within the required bounds. The development of modern Internet services, such as Internet Protocol Television (IPTV), cloud computing and social networking has led to exponential growth of Internet traffic. This significantly aggravated the task of providing desired network performance levels.

Switches and routers represent key elements of any network, and their performance levels determine the performance of entire network. Thus, in order to support and accommodate ever-growing traffic, development of high performance packet switches (and routers) capable of handling such traffic is very important.

Many switch architectures with different buffer position, have been proposed to achieve desired QoS: output queued (OQ) [1], [3], input-queued (IQ) [3], combined input output queued (CIOQ) [3], combined input crosspoint queued (CICQ) [3] and crosspoint queued (CQ) [4].

CQ switch is a crossbar switch with buffers only at crosspoints of crossbar fabric. Although it has simple architecture, the CQ switch does not require switching and/or memory speedup, avoids head of line (HOL) blocking (since there are no input buffers) and does not require control communication between linecards and switching fabric. This switch was not deeply analyzed until recently, because of difficulties with implementation of large enough crosspoint buffers. Due to the technological advancements, that have enabled implementation of large crosspoint buffers together with crossbar fabric on the single chip, the CQ switch has recently drawn attention of research community [5].

Development of scheduling algorithms used within switch architecture is very important. It is especially important when multimedia traffic is observed, because this traffic can support various types of traffic simultaneously. Therefore, it is necessary to develop and implement scheduling algorithms capable of providing different services for different types of traffic. For example, real-time audio traffic requires low delay but allows small loss of data. On the other hand, transfer of textual files, images, e-mails, etc. requires transfer without data loss, while achieving low delay is desired, but not required.

To achieve these requirements, the CQ switch with WRR scheduling algorithm [6] was implemented and analyzed in [7]. By applying this scheduling algorithm we showed in [8] that it is possible to achieve desired QoS regarding the switch throughput and latency simply by adjusting the frame size. These results were compared with other scheduling algorithms and OQ switch, to give better perspective of performances achieved with WRR scheduling algorithm.

However, this algorithm has major implementation drawback because it requires knowledge of incoming traffic in order to calculate weights of traffic flows. To overcome this problem, in this paper we propose implementation of dynamic weighted round robin (DWRR) algorithm that can work with unknown arrival traffic. Several variations of DWRR algorithm will be presented and analyzed, to find the best solution. We will show that DWRR algorithm can provide the same performance as WRR, but without the need for incoming traffic information. This makes DWRR algorithm suitable for implementation, where the arrival traffic is unknown.

This paper is organized as follows. The CQ switch,
WRR and DWRR scheduling algorithms are presented in Section 2. Simulation model is explained in Section 3. Performance analysis is presented in Section 4. Finally, conclusions are drawn in Section 5.

II. CROSSPOINT QUEUED CROSSBAR SWITCH

A. CQ switch architecture

CQ switch [5], [9] architecture is shown in Fig. 1.

![CQ switch architecture](image)

Fig. 1. CQ switch architecture

Cell arriving from input \( i \) and addressed for output \( j \) is stored in crosspoint buffer \( B_{ij} \). The cell is queued at the crosspoint buffer if that buffer is not full; otherwise the cell is discarded. Due to absence of input buffers, HOL blocking is eliminated and need for control communication between linecards and switching fabric is avoided, which reduce switch implementation complexity. In each time slot, the output scheduler chooses one of the non-empty crosspoint buffers from the common output line and forwards its head-of-line cell to an output linecard.

B. WRR and DWRR scheduling algorithm

WRR scheduler services buffers on the particular output line \( j \) according to the weights of corresponding flows. Higher weights are assigned to the flows with higher priority. The weight of a flow \( (i, j) \) represents the maximal number of cells that can depart from buffer \( B_{ij} \) before moving to the next buffer in a round robin order. Flow weights can be assigned in two different ways: manually (by user or admin) and automatically, based on some parameter. In our implementation, weights are assigned automatically based on incoming traffic [7].

The automatic weight assignment can be dynamic or offline. In [7], we used offline weight assignment where the incoming traffic is known. To achieve this, we implemented the traffic generator which generates incoming traffic in form of a matrix. According to this approach, weights are calculated in following manner:

- Based on the traffic matrix, cell request (CR) matrix whose elements \( x_{ij} \) represent the number of cells assigned for crosspoint buffer \( B_{ij} \) is obtained.
- By dividing each element of CR matrix with sum of elements from corresponding column, matrix of weight factors (WF) is obtained \((0 < y_{ij} < 1)\).
- Finally, by dividing all elements in column \( j \) of WF matrix with minimal element of that column, minimal weights for each flow are derived. This procedure is performed for each column and weight matrix \( W \) is obtained, given as such:

\[
W = \begin{bmatrix}
    w_{11} & w_{12} & \cdots & w_{1N} \\
    w_{21} & w_{22} & \cdots & w_{2N} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{N1} & w_{N2} & \cdots & w_{NN}
\end{bmatrix}, \quad w_{i,j} \in N
\]

where \( w_{ij} \) represent weight of buffer \( B_{ij} \). The sum of flow weights belonging to the same column is called frame. The frame is defined as a time needed for scheduler to forward cells from all buffers in one output line based on their weights. After that time, the new frame starts. The frame size can be increased by multiplying matrix \( W \) with integer value \( \alpha \). It is shown in [8] that frame size has direct impact on performances achieved with the WRR scheduling algorithm.

As one can see from this explanation, the WRR scheduling algorithm requires knowledge of incoming traffic. But in realistic scenarios, incoming traffic is unknown, and WRR as such is not practical. Thus, it is necessary to implement the modification of this algorithm that can work when incoming traffic is unknown. To achieve this, we implemented several DWRR based algorithms which analyze incoming traffic, and based on that analysis, calculate and assign weights to crosspoint buffers.

Considering that WRR works with known traffic, performances obtained with it are best that can be achieved with such algorithm. Therefore, results obtained for WRR will be used as a reference level of performances for DWRR. To obtain best solution for DWRR algorithm, four variations are implemented and analyzed:

- DWRR1 – In this implementation the incoming traffic is saved in the form of a CR matrix, and after a certain period of time (regarded as refresh period \( r \)) weights are calculated by method used in WRR. After that, matrix is reset (every element is set to 0) and the whole process is repeated. It should be noted that due to the refreshing of CR matrix, matrix \( W \) can contain elements equal to 0 (after weight calculation). Considering that weights cannot be lower than 1, every element of weight matrix \( W \) is incremented. This is done (even for non-zero elements) to keep relation between weights.
- DWRR2 – In this implementation, like with DWRR1, the matrix is deleted after the refresh period \( r \), but weights are calculated differently. Here, before obtaining matrix of weight factors \( WF \), each element of \( CR \) is increased by one. This is done so that occurrence of zero elements in weight matrix \( W \) is avoided. Although this method for calculating weights seems the same as in DWRR1, here the matrix \( CR \) is increased by one, and then matrices \((WF \) and \( W \) are calculated. We will show that this modification, as simple as it is, provides significant improvement in performances.
- DWRR3 – In this implementation, after the refresh period \( r \), weights are calculated like at DWRR1, but the \( CR \) matrix is not reset.
- DWRR₄ – In this implementation, after the refresh period r, weights are calculated as like at DWRR₂, but the CR matrix is not reset.

III. SIMULATION MODEL

In our simulations, we assume that incoming packets have fixed size length (referred to as cells) [10]. Time is divided into equal time slots, corresponding to the time needed for transferring one cell. Arrival traffic is bursty and modeled with IBP traffic model [9]. Each input port is described by the two-state ON-OFF traffic model, as shown in Fig. 2, where both active and idle periods are geometrically distributed.

As shown in Fig. 2, if the input port is in the ON state, it will stay in that state with probability 1-a, or switch to the OFF state with probability a. Similarly, if the input port is in the OFF state, it will stay in that state with probability 1-b, or switch to the ON state with probability b. These probabilities are given by the following equations:

\[
a = \frac{1}{Bs} \\
b = \frac{pa}{1 + pa - p}
\]

where Bs is the average burst size (the average duration of ON state) and p is input load.

To properly evaluate WRR algorithm performance, the incoming cells are distributed non-uniformly to all output ports, according to the unbalanced probability (w) parameter. This is performed because uniform distribution produces weights of flows that are mutually almost equal and WRR algorithm behaves like ordinary RR algorithm. Traffic load from input i to output j is derived from the equation (4) [8].

\[
p_{i,j} = \begin{cases} 
  p \left( w + \frac{1 - w}{N} \right), & i = j \\
  p \frac{1 - w}{N}, & \text{otherwise}
\end{cases}
\]

The parameter p is the input load at every input port of a switch, while N is the switch size. The aggregate offered load for output port j, considering all input ports, is P. The offered traffic is uniform (equally distributed between output ports) when w=0, and is completely directional from port i to port j when w=1.

To analyze performances of DWRR scheduling algorithm extensive simulations for different values of input load p (0.5, 0.6, 0.7, 0.8, 0.9), burst length Bs (1, 8, 16) and unbalanced probability w (0.0, 0.3, 0.5, 0.7, 0.9) are conducted. Simulations are also performed for different values of frame size α (3, 10, 100).

All simulations are performed for a million time slots on a 32x32 CQ switch with crosspoint buffers’ size of L=16, 32, 64, 128, 256, 512. CQ buffer size is represented by the number of cells that can be accommodated in it.

IV. PERFORMANCE ANALYSIS

A. Throughput

To analyze the impact of refresh period r on DWRR scheduling algorithm performance, the throughput of a CQ switch with DWRR is observed for different values of refresh period r, and compared with throughput obtained with WRR algorithm.

Our analyses showed that correlations between analyzed algorithms do not change significantly for different simulation parameters. Therefore, to avoid presentation of diagrams that carry the same information, throughput will be presented only for chosen set of simulation parameters. Thus, in Fig. 3 throughput is given as a function of refresh period r, for input load p=0.9, average burst Bs=8, unbalanced probability w=0.5, buffer length of L=32, and frame size increased by multiple of α=3.

Simulation results show that, under same conditions, DWRR₁ and DWRR₂ algorithms achieve lowest throughput. This happens because CR matrix is reset after each refresh period. Consequently, sudden changes in calculated weights can occur due to the changing nature of incoming traffic. Therefore, such calculated weights do not match incoming traffic well, and cell loss occurs.

In addition, from Fig. 3 can be noticed that for smaller r DWRR₂ achieves better throughput than DWRR₁. This justifies implementation of new method for weight calculation proposed in DWRR₂. With further increase of refresh period r, the difference between throughput of DWRR₁ and DWRR₂ is less prominent. This happens because, for higher values of r, CR matrix has enough time to be filled. Therefore, there are no zero elements in CR matrix and methods for weight calculation in DWRR₁ and DWRR₂ give more similar results.

A significant improvement in throughput is achieved when CR matrix is not reset after each refresh period. It can be noticed from Fig. 3 that DWRR₃ and DWRR₄
significantly outperform DWRR1 and DWRR2 algorithms in terms of throughput. In addition, the throughput of DWRR3 and DWRR4 is not dependent on refresh period \( r \). This is an important feature of these algorithms, because need for explicit and precise choice of \( r \) is avoided.

Refresh period can be chosen randomly, but it is suggested to choose higher values of \( r \) due to the fact that flow weights are recalculated after each refresh period. This process takes a time, but for high values of \( r \) it is done rarely, and therefore it does not decrease the speed of algorithm significantly.

By comparing DWRR3 and DWRR4 in Fig. 3, it can be seen that DWRR3 gives lower throughput. This happens because in DWRR3 weights are calculated following method explained in DWRR4. When this method is used, the cell loss occurs when \( CR \) matrix is at the initial phase of padding and contains zero elements. By using the second method for weight calculation in DWRR4 (explained in DWRR3), this problem is resolved, and the throughput same as for WRR is achieved.

Same results (in terms of relations between analyzed algorithms) were obtained for every other set of simulation parameters. With this, we showed that DWRR can emulate WRR in terms of throughput when proper modifications are made.

### B. Average latency

In Fig. 4, average latency as a function of refresh period is given, for input load \( p=0.9 \), average burst \( Bs=8 \), unbalanced probability \( w=0.5 \), buffer length of \( L=32 \), and frame size increased by multiple of \( \alpha=3 \).

From Fig. 4 one can see that the lowest average cell latency is obtained with DWRR1 and DWRR2 scheduling algorithms. However, this is due to the fact that these algorithms have lowest throughput. DWRR3 and DWRR4 have higher average latency compared to DWRR1 and DWRR2 which does not depend on refresh period. Finally, it can be noticed that DWRR3 algorithm invokes highest latency, which is same as with WRR, regardless of the refresh period. These results give us further confirmation of excellent feature of DWRR4 to emulate performance of WRR algorithm.

![Fig. 4. Average latency as a function of refresh period, for \( p=0.9, Bs=8, w=0.5, L=32, \alpha=3 \)](image)

### V. Conclusion

In this paper we continued our analyses of CQ switch performance with WRR based scheduling algorithm. Considering that WRR scheduling algorithm requires the knowledge of incoming traffic, it is impractical and has only the theoretical value. Therefore, in this paper we proposed modification of incoming traffic, named dynamic weighted round robin algorithm - DWRR, which does not require the knowledge of incoming traffic. DWRR scheduling algorithm analyzes incoming traffic, and based on that analysis recalculates weights of each flow after the refresh period. We showed, through our simulation results, that one of the proposed variants can provide the same performance as WRR algorithm. The fact that DWRR can obtain these results without the need for knowledge of incoming traffic makes it an ideal solution for practical implementation.

### ACKNOWLEDGMENT

This work is partly supported by the Ministry of Education and Science of Montenegro and Ministry of Education and Science of Bosnia and Herzegovina under the bilateral Project “Performance analysis of CQ packet switch from the aspect of QoS guarantees”, and by the Ministry of Education and Science of Montenegro under grant 01-451/2012 (FIRMONT).

### REFERENCES


