Achieving Flow Level Constant Performance Guarantees for CICQ Switches without Speedup

Hao Jin, Deng Pan, Niki Pissinou
Florida International University
Miami, FL

Kia Makki
TUA
Miami, FL

***Abstract*—Performance guarantees provided by switches can be at different granularity: port level and flow level. As a trade-off, it is usually more expensive to provide performance guarantees at finer granularity. Existing solutions for switches to provide flow level performance guarantees require either expensive hardware support or centralized scheduling algorithms with multiple iterations. In this paper, we present the Flow-level Fair Scheduling (FFS) algorithm to provide flow level performance guarantees for combined-input-crosspoint-queued (CICQ) switches, which are special crossbar switches with a small exclusive buffer at each crosspoint of the crossbar. FFS uses hierarchical and multidimensional fair queueing to emulate the ideal Generalized Processing Sharing (GPS) model. The main features of FFS include: constant performance guarantees, bounded crosspoint buffer sizes, no speedup requirement, and distributed operation. We theoretically analyze the performance of FFS, and conduct simulations to verify the analytical results.**

***Keywords*-performance guarantees; CICQ switches; speedup;**

I. Introduction

It is important for switches and routers to provide performance guarantees, to support applications with QoS requirements [1] and even help defend against DoS attacks [2]. Performance guarantees provided by switches can be at different granularity: port level and flow level [3]. With port level performance guarantees, a switch can differentiate packets from different input ports. For example, if each input port of such a switch connects to a different user, the switch can ensure that each user has its own share of bandwidth at every output port, and further provide delay and jitter guarantees. However, such a switch has no way to differentiate packets from the same input port but different flows. With the above example, if a specific user has a file downloading flow as well as a video streaming flow, the former may aggressively consume all available bandwidth, resulting in service degradation for the latter. On the other hand, if a switch provides flow level performance guarantees, resources are allocated on a per flow basis, and each flow can have its guaranteed bandwidth and delay performance.

It is usually more expensive to provide flow level performance guarantees, since there are more states to maintain and more information to process. Existing solutions [3], [4], [5] for switches to provide flow level performance guarantees need either expensive hardware support or centralized scheduling algorithms with multiple iterations. To be specific, the scheme in [4] needs speedup of three for the crossbar switching fabric, i.e. the crossbar bandwidth having three times bandwidth as that of the input port and output port, or at least N^3 on-chip buffers for the crossbar, where N is the switch size. The scheme in [5] needs speedup of two, and runs a centralized scheduling algorithm with N iterations. The scheme in [3] achieves a tradeoff between those in [4] and [5], but still needs speedup of two and N iterations in the worst case. Furthermore, all the above three schemes can only process fixed length cells, and need to apply segmentation-and-reassembly (SAR) [6] for variable length packets.

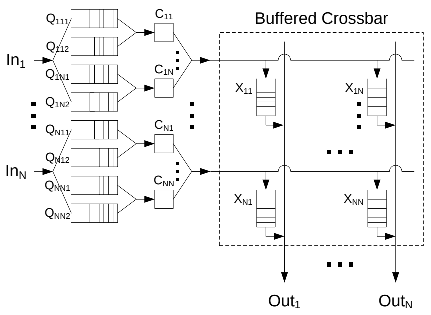


Figure 1. Switch Structure

Combined input-crosspoint queued (CICQ) switches have recently attracted considerable attentions [1], [4], [7] as promising high speed interconnects. They are special crossbar switches with a small exclusive buffer at each crosspoint of the crossbar, as shown in Figure 1. The crosspoint buffers decouple input ports and output ports, and simplify the scheduling process. CICQ switches can directly handle variable length packets without SAR, and enable distributed packet scheduling.

In this paper, we present the Flow-level Fair Scheduling (FFS) algorithm to provide flow level performance guarantees for CICQ switches without speedup. The basic idea is to use hierarchical and two-dimensional fair queueing to emulate the ideal Generalized Processor Sharing (GPS) model [9], which creates a logical transmission channel with dedicated bandwidth for each flow. The main features of FFS can be summarized as follows. First, FFS provides constant performance guarantees, which means that the service difference of any flow under FFS and GPS is bounded by constants. Second, FFS has a bounded crosspoint buffer size, independent of the flow number and switch size. Third, FFS needs no speedup for the crossbar, reducing the hardware cost. Finally, FFS works in a distributed manner, i.e. different inputs port and output ports independently making scheduling decisions, and thus processes variable length packets without SAR. We theoretically analyze the performance of FFS, and present simulation data to evaluate our design.

II. Flow-level Fair Scheduling

In this section, we formulate the flow level fair scheduling problem and present our FFS algorithm.

A. Problem Formulation

We consider an $N \times N$ CICQ switch. For easy representation, denote the i^{th} input port as In_i and the j^{th} output port as Out_j. Each input port and output port have bandwidth of R. Multiple flows may exist from In_i to Out_j, and each flow is allocated a certain amount of bandwidth. Denote the k^{th} flow from In_i to Out_j as F_{ijk}, and its allocated bandwidth as R_{ijk}. The bandwidth allocation should be feasible, i.e.

$$\forall i, \sum_{j,k} R_{ijk} \leq R, \text{ and } \forall j, \sum_{i,k} R_{ijk} \leq R \tag{1}$$

In the ideal GPS fairness model [9], each flow has a logical decidecated transmission channel with bandwidth R_{ijk}, and therefore predictable and guaranteed performance.

The objective of flow level fair scheduling is to ensure that a flow receives the same amount of service as in GPS at any time. Use $toO_{ijk}(0,t)$ and $toO_{ijk}^{GPS}(0,t)$ to represent the numbers of bits transmitted by F_{ijk} to the output line during interval $[0,t]$ in FFS and GPS, respectively. The difference $toO_{ijk}(0,t) - toO_{ijk}^{GPS}(0,t)$ is expected to be tightly bounded independent of R_{ijk} and t.

B. Switch Structure

The CICQ switch structure is shown in Figure 1. N input ports and N output ports are connected by a buffered crossbar without speedup, or in other words the bandwidth of the crossbar is also R. Apparently, to provide performance guarantees at the flow level, it is necessary to store packets on a per flow basis in the input buffer to achieve traffic isolation among flows. At the input port In_i, there is a queue for each flow F_{ijk}, denoted as Q_{ijk}. In addition, In_i has a special candidate buffer for each output port Out_j, denoted as C_{ij}, to store scheduling candidate packets to be sent to the crossbar. Each crosspoint of the crossbar has a small exclusive buffer. Denote the crosspoint buffer connecting In_i and Out_j as X_{ij}. Output ports have no buffers.

C. Algorithm Description

FFS consists of three phases: flow scheduling, input scheduling, and output scheduling. All the three scheduling phases are based on the WF^{2}Q fair queueing algorithm [10].

In *flow scheduling*, In_i selects a packet from one of the flow queues Q_{ijk} destined for Out_j, and sends the packet to the corresponding candidate buffer C_{ij}. For easy description, denote the n^{th} packet of F_{ijk} as P_{ijk}^n. Flow scheduling calculates two time stamps for each packet p: virtual flow start time $VFS(p)$ and finish time $VFF(p)$. They are the departure time of the first bit and last bit of p in GPS, and are calculated as

$$VFS(P_{ijk}^n) = \max(IA(P_{ijk}^n), VFF(P_{ijk}^{n-1})) \tag{2}$$

$$VFF(P_{ijk}^n) = VFS(P_{ijk}^n) + \frac{L(P_{ijk}^n)}{R_{ijk}} \tag{3}$$

where $IA(p)$ is the arrival time of p at the flow queue F_{ijk}, and $L(p)$ is its packet length. Because we do not consider reallocating the leftover bandwidth of empty flows to backlogged flows, the virtual time in GPS progresses at the same pace as the real time, and the time stamp calculation is simpler than that in [10]. The first step of flow scheduling is to identify eligible packets. To conduct flow scheduling at time t, a packet p is eligible if its virtual flow start time is less than or equal to t, i.e. $VFS(p) \leq t$. In other words, a packet that has started transmission in GPS is eligible for flow scheduling. The second step is to select among eligible packets the one p with the smallest virtual flow finish time, i.e. $\forall p' VFS(p') \leq t \rightarrow VFF(p') \geq VFF(p)$. The selected packet will be sent to the corresponding candidate buffer C_{ij}. If there are no eligible packets, In_i will wait until the next earliest virtual flow start time. Accordingly, we define two additional time stamps for p: actual flow start time $AFS(p)$ and finish time $AFF(p)$, which are the time that the first bit and last bit of p leave Q_{ijk} in flow scheduling. If p is selected in flow scheduling at t to be sent to C_{ij}, then $AFS(p) = t \geq VFS(p)$. Flow scheduling multiplexes all flows F_{ijk} from In_i to Out_j as a logical flow F_{ij} to simplify the remaining scheduling. In the logical flow F_{ij}, we have $AFF(p) = AFS(p) + L(p)/R_{ij}$, where $R_{ij} = \sum_k R_{ijk}$ is the total bandwidth of all the flows from In_i to Out_j.

In *input scheduling*, In_i selects a packet from one of its N candidate buffers C_{ij}, and sends it to the corresponding crosspoint buffer X_{ij}. Input scheduling also uses two time stamps for each packet p: virtual input start time $VIS(p)$ and finish time $VIF(p)$, which are equal to the actual flow start and finish time, respectively, i.e. $VIS(p) = AFS(p)$ and $VIF(p) = AFF(p)$. Similarly, the first step of input scheduling is to identify eligible packets whose virtual input start time is no later than the current scheduling time. The second step is to find among eligible packets the one with the smallest virtual input finish time. The selected packet is then sent from the candidate buffer to the crosspoint buffer. Accordingly, we define the actual input start time $AIS(p)$ and finish time $AIF(p)$ to be the time that the first bit and last bit of p leave C_{ij} in input scheduling. We have $AIS(p) \geq VIS(p)$ and $AIF(p) = AIS(p) + L(p)/R$, since the bandwidth of the crossbar is R.

In *output scheduling*, Out_j selects a packet from one of its N crosspoint buffers X_{ij}, and sends it to the output line. Denote the n^{th} packet from In_i to Out_j as P_{ij}^n. Output scheduling calculates two time stamps for each packet p: virtual output start time $VOS(p)$ and finish time $VOF(p)$, as follows

$$VOS(P_{ij}^n) = \max(XA(P_{ij}^n), VOF(P_{ij}^{n-1})) \quad (4)$$

$$VOF(P_{ij}^n) = VFS(P_{ij}^n) + \frac{L(P_{ij}^n)}{R_{ij}} \quad (5)$$

where $XA(p)$ is the arrival time of p at the crosspoint buffer X_{ij}. Similarly, the first step of output scheduling identifies eligible packets based on the virtual output start time, and the second step selects the packet to be transmitted based on the virtual output finish time. Define the actual output start time $AOS(p)$ and finish time $AOF(p)$ to be the time that the first bit and the last bit of p leave X_{ij} in output scheduling, respectively. Since the bandwidth of the crossbar is R, we have $AOF(p) = AOS(p) + L(p)/R$

Regarding the time complexity of FFS, because all the three scheduling phases use WF^{2}Q, they have logarithmic time complexity [11]. In order to transfer a packet to the output line, flow scheduling, input scheduling, and output scheduling each will be conducted once. Note that the scheduling at each input port and output port is independent without a centralized controller, and hence FFS is suitable for distributed implementation. Finally, FFS can directly process variable length packets without SAR. On the other hand, due to fine granularity traffic isolation, FFS has a sophisticated input buffer structure with flow queues and candidate buffers.

III. Performance Analysis

In this section, we theoretically analyze the performance of FFS. We will show that it has a bounded crosspoint buffer size and provides constant service guarantees.

A. Crosspoint Buffer Size Bound

Crosspoint buffers are expensive on-chip memories, and we would like to find the maximum number of bits buffered at a crosspoint to avoid overflow.

As have be seen, all the three scheduling phases use WF^{2}Q. WF^{2}Q schedules packets of a number of flows, each with its allocated bandwidth, to emulate GPS, in which each flow has a logical dedicated channel with the allocated bandwidth. Theorem 1 in [10] gives the relationship between the amount of service of a flow in WF^{2}Q and GPS, and we have similar properties for the three scheduling phases of FFS. Define $toX_{ij}(t_1, t_2)$ and $toO_{ij}(t_1, t_2)$ to represent the numbers of bits transmitted by flow F_{ij} during interval $[t_1, t_2]$ in input scheduling and output scheduling of FFS, respectively. Additionally, define $toX'_{ij}(t_1, t_2)$ and $toO'_{ij}(t_1, t_2)$ to represent the numbers of bits transmitted by F_{ij} during $[t_1, t_2]$ in the input scheduling and output scheduling logical dedicated channels, respectively. By Theorem 1 in [10], we have

$$-L_m \le toX_{ij}(0,t) - toX'_{ij}(0,t) \le L_m(1 - \frac{R_{ij}}{R}) \quad (6)$$

$$-L_m \le toO_{ij}(0,t) - toO'_{ij}(0,t) \le L_m(1 - \frac{R_{ij}}{R}) \quad (7)$$

Lemma 1: During interval $[0, t]$, the number of bits transmitted by flow F_{ij} in the input scheduling logical dedicated channel is less than or equal to that in the output scheduling logical dedicated channel plus $2L_m$, i.e.

$$toX'_{ij}(0,t) \le toO'_{ij}(0,t) + 2L_m \quad (8)$$

Proof: Assume that X_{ij} is empty immediately before time s and is continuously backlogged during $[s, t]$ in the output scheduling logical dedicated channel. If X_{ij} is not backlogged at t, then $s = t$.

By (6), we have $toX_{ij}(0,s) \ge toX'_{ij}(0,s) - L_m$. Because X_{ij} is empty before s and backlogged after s in the output scheduling logical dedicated channel, all packets arriving at B_{ij} before s have been transmitted to Out_j, and a new packet arrives at B_{ij} at s. Thus

$$toO'_{ij}(0,s) \ge toX_{ij}(0,s) - L_m \ge toX'_{ij}(0,s) - 2L_m \quad (9)$$

On the other hand, because X_{ij} is continuously backlogged during $[s, t]$, we have

$$toO'_{ij}(s,t) = R_{ij}(t - s) \ge toX'_{ij}(s,t) \quad (10)$$

Adding (9) and (10), we have proved the lemma. ■

The following theorem gives the bound for the crosspoint buffer size.

Theorem 1: In FFS, the maximum number of bits buffered at crosspoint buffer X_{ij} is bounded by $4L_m - L_m R_{ij}/R$, i.e.

$$toX_{ij}(0,t) - toO_{ij}(0,t) \le 4L_m - L_m\frac{R_{ij}}{R} \quad (11)$$

Proof: Combining Lemma 1, (6), and (7), we have proved the theorem. ■

B. Service Guarantees

We show below that FFS achieves constant service guarantees, i.e. the difference between the service amount of a flow in FFS and GPS bounded by constants. Use $toO_{ijk}(t_1, t_2)$, $toX_{ijk}(t_1, t_2)$, and $toC_{ijk}(t_1, t_2)$ to represent the numbers of bits transmitted by F_{ijk} during interval $[t_1, t_2]$ to Out_j, X_{ij}, and C_{ij} in FFS, respectively. Correspondingly, use $toC'_{ijk}(t_1, t_2)$ to represent the numbers of bits transmitted by F_{ijk} during $[t_1, t_2]$ to C_{ij} in GPS. By neglecting transmission delay, we have $toC'_{ijk}(t_1, t_2) = toO_{ijk}^{GPS}(t_1, t_2)$.

Lemma 2: When a packet P_{ijk}^n starts to be transmitted to the crosspoint buffer X_{ij}, the difference between the number of bits transmitted by its flow F_{ijk} to the crosspoint buffer X_{ij} in input scheduling of FFS and that to the candidate buffer C_{ij} in the flow scheduling logical dedicated channel is greater than or equal to $-L_m(1 + R_{ijk}/R_{ij} + R_{ijk}/R)$, i.e.

$$toX_{ijk}(0, AIS(P^n_{ijk})) - toC'_{ijk}(0, AIS(P^n_{ijk}))$$
$$\geq \; -L_m \left(1 + \frac{R_{ijk}}{R_{ij}} + \frac{R_{ijk}}{R}\right) \quad (12)$$

Proof: Since flow scheduling and input scheduling are based on WF^{2}Q, by Theorem 1 in [10] we know $AFF(p) \leq VFF(p) + \frac{L_m}{R_{ij}}$ and

$$AIF(p) \leq \; VIF(p) + \frac{L_m}{R} = AFF(p) + \frac{L_m}{R}$$
$$\leq \; VFF(p) + \frac{L_m}{R_{ij}} + \frac{L_m}{R} \quad (13)$$

Therefore

$$\begin{aligned} & toC'_{ijk}(0, AIS(P^n_{ijk})) \\ \leq \; & toC'_{ijk}(0, AIF(P^n_{ijk})) \\ \leq \; & toC'_{ijk}\left(0, VFF(P^n_{ijk}) + \frac{L_m}{R_{ij}} + \frac{L_m}{R}\right) \\ = \; & toC'_{ijk}\left(0, VFF(P^n_{ijk})\right) + \\ & toC'_{ijk}\left(VFF(P^n_{ijk}), VFF(P^n_{ijk}) + \frac{L_m}{R_{ij}} + \frac{L_m}{R}\right) \\ \leq \; & \sum_{x=1}^{n} L(P^x_{ijk}) + R_{ijk}\left(\frac{L_m}{R_{ij}} + \frac{L_m}{R}\right) \quad (14) \end{aligned}$$

Note that $toX_{ijk}(0, AIS(P^n_{ijk})) = \sum_{x=1}^{n-1} L(P^x_{ijk})$ and thus the lemma is proved. ■

The following theorem shows that FFS achieves constant service guarantees.

Theorem 2: At any time, the difference between the numbers of bits transmitted by a flow to the output port in FFS and GPS is greater than or equal to $-6L_m$ and less than or equal to L_m, i.e.

$$-6L_m \leq toO_{ijk}(0,t) - toC'_{ijk}(0,t) \leq L_m \quad (15)$$

Proof: We first prove $toO_{ijk}(0,t) - toC'_{ijk}(0,t) \geq -6L_m$. By Lemma 2, it is easy to show that, for any t

$$toX_{ijk}(0,t) - toC'_{ijk}(0,t)$$
$$\geq \; -L_m \left(1 + \frac{R_{ijk}}{R_{ij}} + \frac{R_{ijk}}{R}\right) \quad (16)$$

Note that $toX_{ijk}(0,t) - toO_{ijk}(0,t)$ is the number of bits of F_{ijk} in the crosspoint buffer X_{ij}. By Theorem 1, we know

$$toX_{ijk}(0,t) - toO_{ijk}(0,t) \leq 4L_m - L_m\frac{R_{ij}}{R} \quad (17)$$

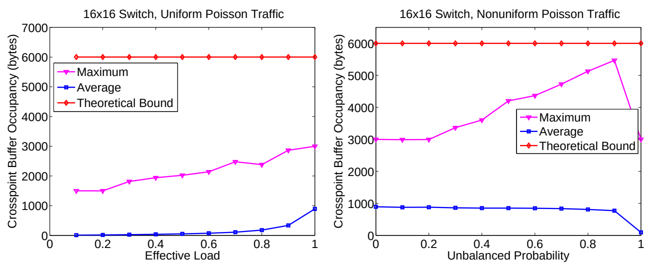
Combining (16) and (17) and noting $R_{ij} \geq R_{ijk}$, we have

$$toO_{ijk}(0,t) - toC'_{ijk}(0,t) \geq -6L_m \quad (18)$$

Next, we prove $toO_{ijk}(0,t) - toC'_{ijk}(0,t) \leq L_m$. By Theorem 1 in [10]

$$toC'_{ijk}(0,t) \geq \; toC_{ijk}(0,t) - L_m \geq toX_{ijk}(0,t) - L_m$$
$$\geq \; toO_{ijk}(0,t) - L_m \quad (19)$$

■



(a) (b)

Figure 2. Crosspoint Buffer Occupancy (a) Uniform Traffic (b) Nonuniform Traffic

IV. Simulation Results

In this section, we present simulation data to verify the obtained analytical results and evaluate the effectiveness of FFS. We consider a 16×16 CICQ switch without speedup. Each input port or output port has bandwidth of 1 Gbps. There are two flows from In_i to Out_j with $R_{ij2} = 2R_{ij1}$. The packet length is uniformly distributed between 40 and 1500 bytes [12], and packets arrive based on a Markov modulated Poisson process [8].

We use two traffic patterns. For traffic pattern one, or uniform traffic, we set $R_{ij} = R/N$, and change the effective load l of the incoming traffic from 0.1 to 1 by step 0.1. For traffic pattern two, or nonuniform traffic, we fix the effective load l to 1, and define R_{ij} by i, j and an unbalanced probability w as follows

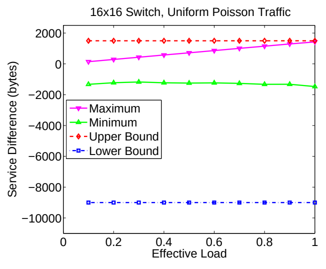
$$R_{ij}(t) = \begin{cases} R(w + \frac{1-w}{N}), & \text{if } i = j \\ R\frac{1-w}{N}, & \text{if } i \neq j \end{cases} \quad (20)$$

where w is increased from 0 to 1 by step 0.1.

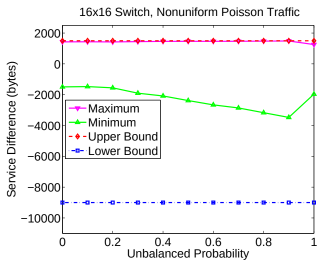
A. Crosspoint Buffer Occupancy

In this subsection, we measure the crosspoint buffer occupancy. Recall that Theorem 1 gives the size bound $4L_m - L_mR_{ij}/R$ for the crosspoint buffer X_{ij}. For easy plotting, we enlarge the bound slightly to $4L_m$. Figure 2(a) shows the maximum and average crosspoint occupancies of all crosspoint buffers under uniform traffic. As can be seen, both the maximum and average occupancies increase with the load. The maximum occupancy value does not exceed the theoretical bound, and is much higher than the average value.

Figure 2(b) presents the data under nonuniform traffic. We can see that the theoretical bound is tight. Specifically, the maximum occupancy rises gradually while the average one keeps relatively constant. The reason is that, as the unbalanced probability increases, the crosspoint buffers X_{ii} receive more packets. On the other hand, the steady total traffic load results in the constant average occupancy. When the unbalanced probability becomes one, all traffic of In_i goes to Out_i and only the crosspoint buffer X_{ii} is used. As a result, both the maximum and average occupancies drop suddenly.

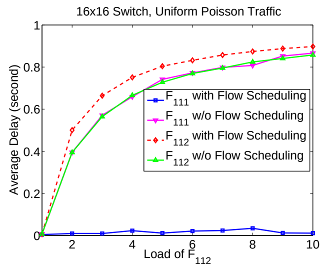


(a)

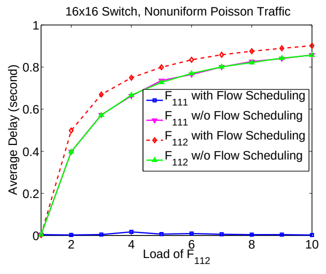


(b)

Figure 3. Service Difference (a) Uniform Traffic (b) Nonuniform Traffic



(a)



(b)

Figure 4. Flow Bandwidth Guarantees (a) Uniform Traffic (b) Nonuniform Traffic

B. Service Guarantees

In this subsection, we study the service difference between FFS and GPS. It is proved in Theorem 2 that the service difference of any flow in FFS and GPS has a lower bound of $-6L_m$ and upper bound of L_m.

Figure 3(a) shows the maximum and minimum service differences among all the flows during the entire simulation run under uniform traffic. As shown in the figure, the maximum service difference increases with the traffic load, and is always lower than the upper bound. The minimum service difference is relatively constant and always greater than the lower bound. Figure 3(b) plots the data under nonuniform traffic. We can find that the maximum service difference is almost identical with the upper bound. One notable feature is that the maximum service difference drops when the unbalanced probability becomes one. This is because all packets of In_i only go to Out_i in this case, and no switching is necessary. On the other hand, the minimum service difference is always greater than the lower bound. It drops gradually with the increasing of the unbalanced probability, and rises as the unbalanced probability becomes one for the same reason as above.

C. Bandwidth Guarantees

In this subsection, we show the effectiveness of FFS by comparing it with port level fair scheduling, i.e. without the flow scheduling phase. We adjust the load of a particular flow F_{112} from 1 to 10, and fix the load of all other flows, including F_{111}, at 1.

Figure 4(a) shows the average delay of F_{111} and F_{112} under uniform traffic. We can see that, with flow scheduling, the average delay of F_{111} remains constant no matter what the load of F_{112} is, while the delay of F_{112} grows steadily with its load. Without flow scheduling, the delay of F_{111} is approximately the same as that of F_{112}, which increases with the load of F_{112}. The results fully demonstrate that FFS is effective in achieving traffic isolation among flows and providing flow level performance guarantees. Figure 4(b) plots the data under nonuniform traffic with the unbalanced probability equal to 0.5, and similar conclusions can be obtained.

V. Conclusions

In this paper, we have presented the Flow-level Fair Scheduling (FFS) algorithm to provide flow level performance guarantees for CICQ switches, which are crossbar switches with each crosspoint of the crossbar equipped with a small buffer. FFS uses hierarchical and multidimensional fair queueing to emulate the ideal GPS model. FFS requires no speedup for the crossbar and is suitable for distributed implementation. By theoretical analysis, we show that FFS achieves constant performance guarantees, and has bounded crosspoint buffer sizes. We also present simulation data, which demonstrate consistency with the analytical results.

References

[1] J. Turner, "Strong performance guarantees for asynchronous crossbar schedulers," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1017-1028, Aug. 2009.

[2] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: network-layer DoS defense against multimillion-node botnets," *ACM SIGCOMM 2008*, Seattle, WA, Aug. 2008.

[3] D. Pan and Y. Yang, "Providing flow based performance guarantees for buffered crossbar switches," *IEEE IPDPS 2008*, Miami, FL, Apr. 2008.

[4] S. Chuang, S. Iyer, and N. McKeown, "Practical algorithms for performance guarantees in buffered crossbars," *Proc. of IEEE INFOCOM 2005*, Miami, FL, Mar. 2005.

[5] S. Chuang, A. Goel, N. McKeown and B. Prabhkar, "Matching output queueing with a combined input output queued switch," *IEEE INFOCOM'99*, pp. 1169-1178, New York, 1999.

[6] M. Katevenis and G. Passas, "Variable-size multipacket segments in buffered crossbar (CICQ) architectures," *IEEE ICC 2005*, Seoul, Korea, May 2005.

[7] S. He et al., "On Guaranteed Smooth Switching for Buffered Crossbar Switches," *IEEE/ACM Transactions on Networking*, Jun. 2008.

[8] D. Pan and Y. Yang, "Localized independent packet scheduling for buffered crossbar switches," *IEEE Transactions on Computers*, vol. 58, no. 2, pp. 260-274, Feb. 2009.

[9] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single node case," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 344-357, Jun. 1993.

[10] J. Bennett and H. Zhang, "WF2Q: worst-case fair weighted fair queueing," *IEEE INFOCOM 1996*, San Francisco, CA, Mar. 1996.

[11] P. Valente, "Exact GPS simulation with logarithmic complexity, and its application to an optimally fair scheduler," *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1454-1466, Dec. 2007.

[12] C. Farleigh et al., "Packet-level traffic measurements from the Sprint IP backbone," *IEEE Network*, vol. 17, no. 6, pp. 6-16, Nov. 2003.