Port Partitioning and Dynamic Queueing for IP Forwarding[*]

Chae Y. Lee** and Hee K. Cho**

**Department of Industrial Engineering, KAIST,

373-1, Kusung Dong, Taejon, 305-701, Korea

Scope and Purpose: To cope with the increasing internet traffic, it is necessary to improve the performance of routers. To accelerate the switching from input ports to output in the router partitioning of ports and dynamic queueing are proposed. Input and output ports are partitioned into two groups A/B and a/b respectively. The matching for the packet switching is performed between group pairs (A, a) and (B, b) in parallel at one time slot and (A, b) and (B, a) at the next time slot. Dynamic queueing is proposed at each input port to reduce the packet delay and packet loss probability by employing the popup decision rule and applying it to each delay critical packet.

The partitioning of ports is illustrated to be highly effective in view of delay, required buffer size and throughput. The dynamic queueing also demonstrates good performance when the traffic volume is high.

[*] This research was supported by the fund from Electronics and Telecommunications Research Institute.

Abstract: With the increase of internet protocol (IP) packets the performance of routers became an important issue in internetworking. In this paper we examine the matching algorithm in gigabit router which has input queue with virtual output queueing. Dynamic queue scheduling is also proposed to reduce the packet delay and packet loss probability.

Port partitioning is employed to reduce the computational burden of the scheduler in a switch which matches the input and output ports for fast packet switching. Each port is divided into two groups such that the matching algorithm is implemented within each pair of groups in parallel. The matching is performed by exchanging the pair of groups at every time slot. Two algorithms, maximal weight matching by port partitioning (MPP) and modified maximal weight matching by port partitioning (MMPP) are presented. In dynamic queue scheduling, a popup decision rule for each delay critical packet is made to reduce both the delay of the delay critical packet and the loss probability of loss critical packet.

Computational results show that MMPP has the lowest delay and requires the least buffer size. The throughput is illustrated to be linear to the packet arrival rate, which can be achieved under highly efficient matching algorithm. The dynamic queue scheduling is illustrated to be highly effective when the occupancy of the input buffer is relatively high.

Key words: IP-forwarding, scheduling, switch, dynamic-queueing

1. Introduction

With the development of communication technology, the number of telecommunication users is growing rapidly especially in the web interface. As the number of Internet users grows exponentially these years, the so called 80/20 rule in LAN which means 80% internal traffic and 20% external traffic is not appropriate any more. Thus the bottleneck problem in the router becomes severe with the increase of external traffic.

Two approaches to solve bottleneck problem in the router have been proposed. One is "route once and switch many" and the other is "gigabit router"[9]. "Route once and switch many" is the way to minimize the frequency of routing. This approach needs new protocols and network components with high cost. IP switching of Ipsilon and Tag switching of Cisco [11] are the examples of this method.

An alternative approach to achieve routing at gigabit per second is to implement high speed layer-3 packet header processing with an internal switch fabric at a router. The processor's internal cache is employed as a least recently used cache of IP destination addresses, and longest prefix matching algorithm is used to look up the routing table. The multi gigabit router is an example of this approach [2].

It is known that the cost of "gigabit router" is less than that of "route once and switch many" due to the use of the existing network system. Thus we focus our attention in this paper to the "gigabit router" which has input queue with virtual output queueing at each port.

To reduce the computational burden required in the matching process of input and output ports, port partitioning is suggested. Each port is partitioned into two groups and matching is accomplished within paired input and output port groups.

Also, to improve delay and packet loss at the input buffers, dynamic queue scheduling is proposed by classifying the packets into two classes: delay critical and loss critical packets. The dynamic queue scheduling decides whether to pop up or not the delay critical packet to reduce the packet delay and loss.

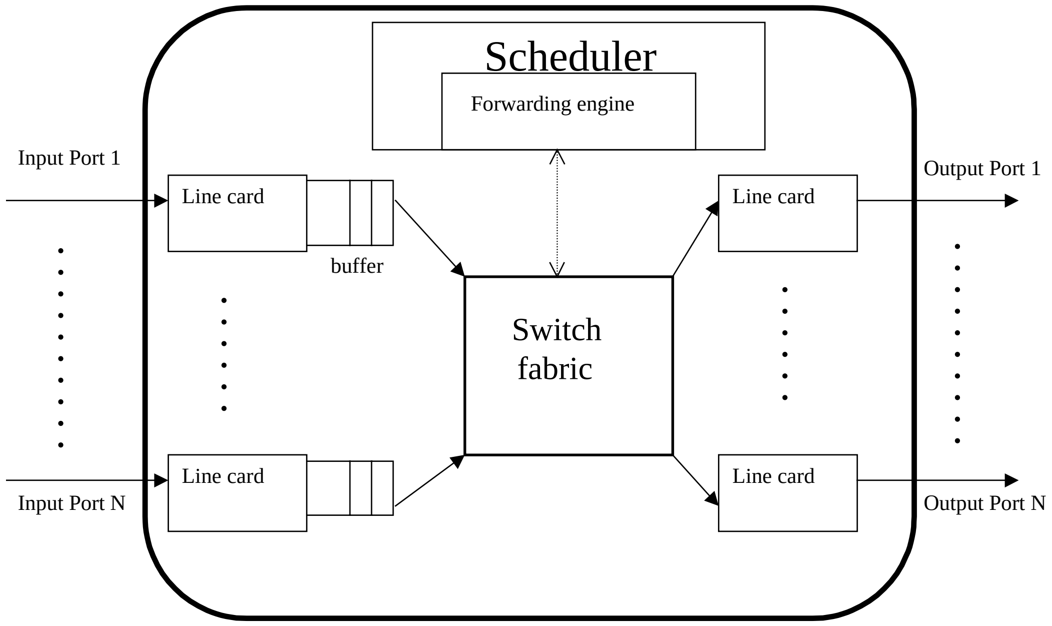


Figure 1. Basic Structure of a Switch

This paper is organized as follows. Switching algorithm between input and output ports are examined in Section 2. Section 3 suggests an improved switching algorithm by port partitioning. In Section 4, dynamic queue scheduling is considered for mixed traffic streams of voice and data packets. The effectiveness of the proposed algorithm is demonstrated with computational experiments in Section 5. Finally, Section 6 concludes this paper.

2. Matching Algorithms for Fast Packet Switching

A basic structure of switch is shown in Figure 1. The switch fabric interconnects input and output ports at each time slot. The matching of each pair of input and output ports is scheduled by the scheduler and implemented by the switch fabric.

At each input port, virtual output queueing (VOQ) [8] is assumed to overcome limitations of head of line (HOL) blocking as shown in Figure 2.

The HOL blocking can be entirely eliminated by using a simple buffering strategy at each input port as in Figure 2 (b). In VOQ, rather than maintaining a single first in first out (FIFO) queue for all packets (see Figure 2 (a)), each input maintains a separate queue for each output. It

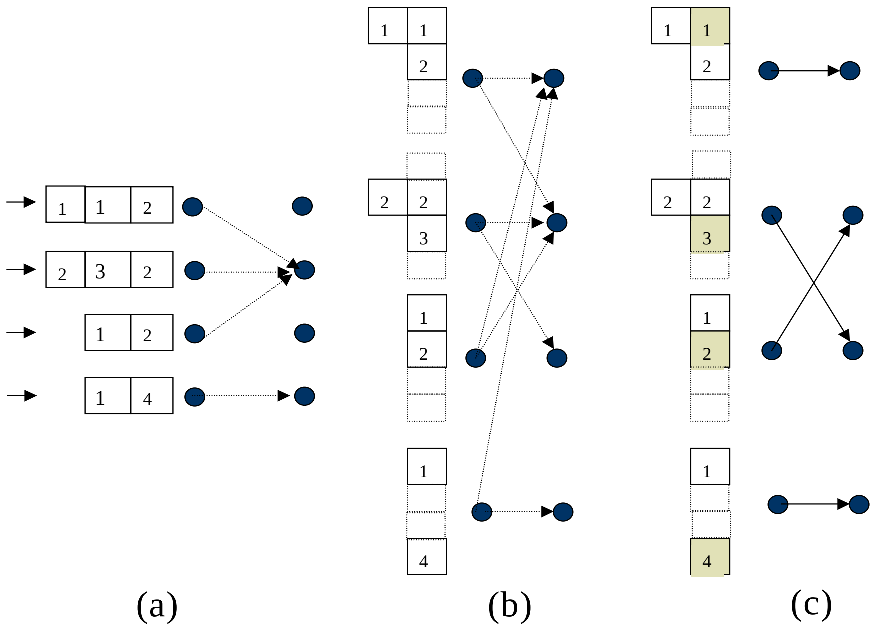


Figure 2. HOL Blocking (a), VOQ (b) and Switching (c)

is known that the throughput by VOQ is improved to 100% [3] compared the 58% [1] at an input queue switch with HOL blocking.

Algorithms are developed to solve the matching problems. Maximum size matching algorithm [6] is proposed to find the match that maximizes the number of edges, where each edge represents a packet to be switched. Maximum weight matching algorithm [3] maximizes the sum of edge weights, where each packet has priority in delivery. Clearly, maximum size matching is a special case of the maximum weight matching.

It was demonstrated using simulation that the maximum size matching algorithm is stable for independent and identically distributed arrivals up to offered load of 100% when the traffic is uniform [3]. However, the algorithm does not take into consideration the condition of each port, since each edge has same weight.

On the other hand, in the maximum weight matching algorithm, the matching process is solved by considering the queue status of each port via the weight of each link. The most efficient algorithm for solving this maximum weight matching problem is known to converge in $O(N^3 \log N)$ running time [7] by assuming N input and N output ports.

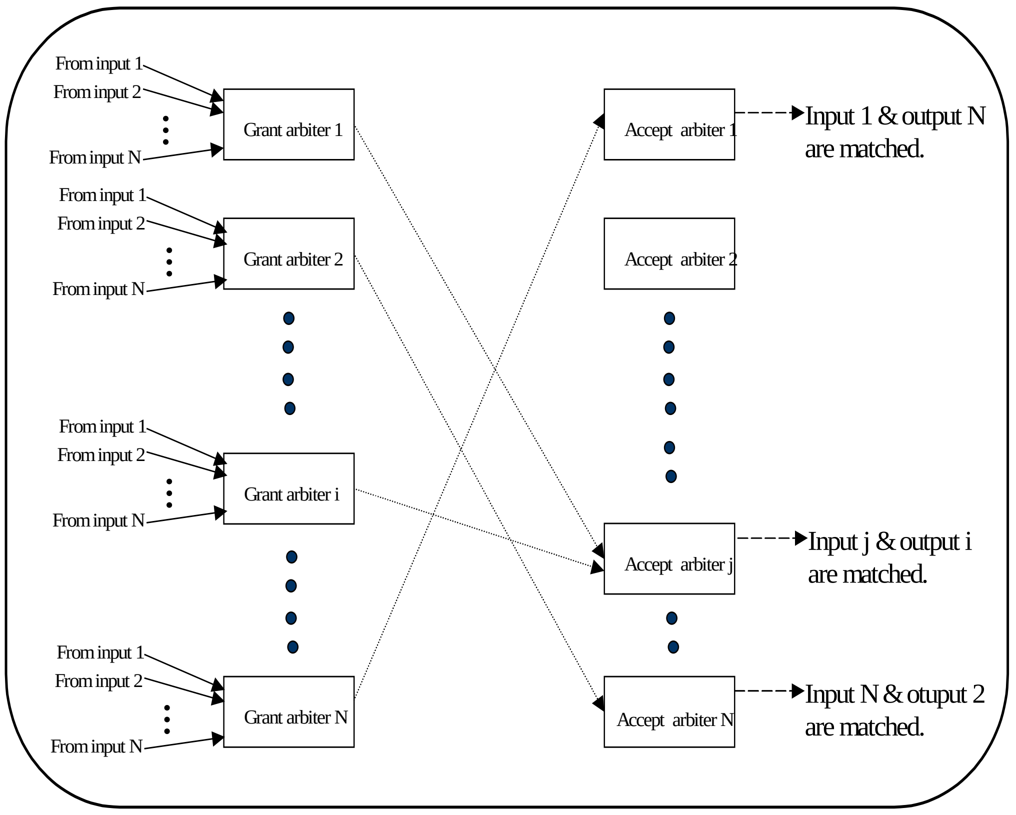


Figure 3. Implementation of Arbiters in Scheduler

Since the maximum weight matching algorithm is very complex to implement in a hardware, we are interested in an iterative approximation of maximum weight matching, i.e., the iterative Maximal Weight Matching (i-MWM) algorithm [2]. N input and N output arbiters operate in parallel as in parallel iterative matching, which was developed by DEC Systems Research Center for the 16-port, 1 Gbps AN2 switch [8]. At each time slot, the matching for the next time slot is scheduled as follows. Note that each iteration of i-MWM consists of three steps: request, grant and accept. All inputs and ouputs are initially unmatched. At the end of each iteration, only those inputs and outputs not matched are eligible for matching in the following iterations. Connections made in one iteration are never removed by a later iteration, even if a larger weight match would result. The three steps of each iteration are as follows:

Step 1. Request: each unmatched input sends a request word to each output for which it has weight.

Step 2. Grant: if an unmatched output receives any requests, it chooses the request with largest weight. Ties are broken arbitrarily.

Step 3. Accept: if an unmatched input receives one or more grants, it accepts the grant with largest weight. Ties are broken arbitrarily.

The implementation of i-MWM in scheduler is shown in Figure 3.

3. Iterative Maximal Weight Matching by Port Partitioning

In i-MWM, the number of comparisons required at each arbiter becomes N-1 in one iteration in the worst case. This is true at both grant and accept arbiters. However, by dividing the ports into two groups, the number of operations at each arbiter becomes half. Thus, we consider partitioning the input and output ports into two groups such that the computational burden required in the process of request, grant and accept is reduced and the matching process is accomplished before transmission. An example of port partitioning is shown in Figure 4. As shown in the figure, the paired input and output port groups considered at one time slot is exchanged at the succeeding time slot. The matching is performed within the two paired groups in parallel. As an example in Figure 4, input ports 1, 2, 3 and 4 and output ports 1, 2, 3 and 4 are paired at one time slot. At the next time slot, input ports 1, 2, 3 and 4 and output ports 5, 6, 7 and 8 are paired. Thus traffic only between each pair of groups are considered for matching as indicated in the figure. The dashed lines after one time slot in Figure 4 represents the traffics not matched at the previous time slot. The matching process of port partitioning is explained in the following Algorithm MPP.

Algorithm MPP

```
Step 1.   Let G_0 = {grant arbiter i | $1 \le i \le N/2$ }, *N: total number of input or output ports
              G_1 = { grant arbiter i | $N/2 + 1 \le i \le N$ },
```

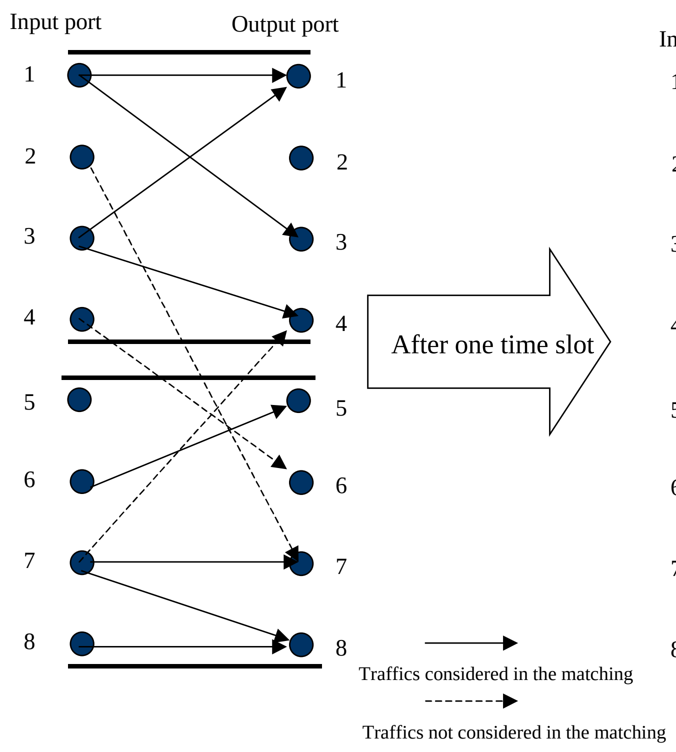

Input port
Output port
1
2
3
4
5
6
7
8
After one time slot
Traffics considered in the matching
Traffics not considered in the matching

Figure 4. An Example of Port Partitioning

I_0 = { input port i | 1≤ i ≤ N/2 },

and I_1 = { input port i | N/2 + 1 ≤ i ≤ N }.

Step 2. time = current timeslot;

Step 3. Each unmatched input sends its weight information to grant arbiters in parallel.

Step 4. Each grant arbiter in G_0 selects one input port to have max weight in I_(time%2), and each grant arbiter in G_1 selects one input port to have max weight in I_(1-(time%2)).

*If time is even number, time%2 is 0 and if time is odd number, time%2 is 1.

Step 5. Each grant arbiter sends the selected input index to the respective accept arbiter.

Step 6. Each accept arbiter selects one output to have max weight.

Step 7. If no matching exists, goto Step 9.

Step 8. goto Step 3.

Step 9. time ++; each input transmits the matched traffic.

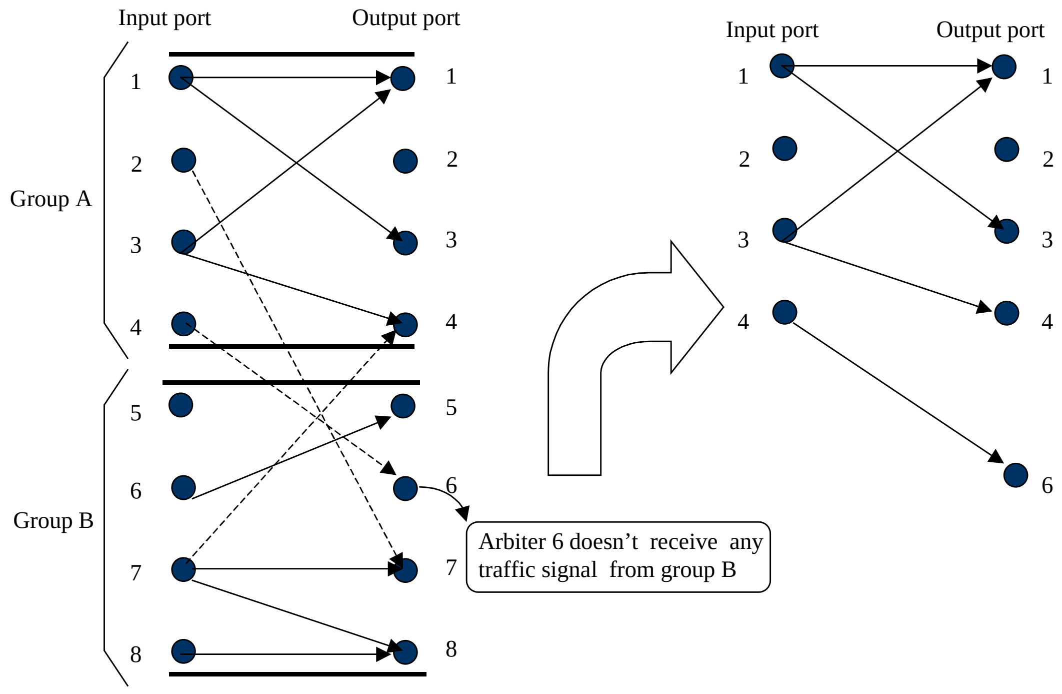


Figure 5. An Example of Modified Matching

In Figure 4, notice that the packet from input port 4 to output port 6 is delayed one time slot due to the unpaired grouping. In this case, by including such a packet into the matching as in Figure 5, we can improve the efficiency of the switch. It will not only reduce the delay of packets but also increase the throughput by giving the chance to accept packets at the corresponding two ports (input port 4 and output port 6 in this example) in the very next time slot.

By considering such a case, we present a Modified Matching by Port Partitioning (MMPP) as follows.

Algorithm MMPP

```
Step 1.   time = current time slot;
Step 2.   Let G_0 = {grant arbiter i | 1≤ i ≤ N/2 }, *N: total number of ports
              G_1 = { grant arbiter i | N/2 + 1 ≤ i ≤ N },
        I_0 = { input port i | 1≤ i ≤ N/2 },
        and I_1 = { input port i | N/2 + 1 ≤ i ≤ N }.
Step 3.  Each input sends its weight information to grant arbiters in parallel.
Step 4.  For all G_0, if grant arbiter k didn't receive any signal from I_(time%2),
             G_0 = G_0 \ {k} and G_1 = G_1 ∪ {k}.
         Also, for all G_1, if grant arbiter k didn't receive any signal from I_(1-(time%2)),
             G_1 = G_1 \ {k} and G_0 = G_0 ∪ {k}.
Step 5. Each grant arbiter in G_0 selects one input port to have max weight in I_(time%2), and
        each grant arbiter in G_1 selects one input port to have max weight in I_(1-(time%2)).
         *If time is even number, time%2 is 0 and if time is odd number, time%2 is 1.
Step 6.  Each grant arbiter sends the selected input index to the respective accept arbiter.
Step 7.  Each accept arbiter selects one output to have max weight.
Step 8.  If no matching exists, goto Step 10
Step 9.  goto Step 2.
Step 10.  time ++; each input transmits the matched traffic.
```

Finally in this section, we compare the computational complexity of the two algorithms: i-MWM and MPP. The complexity of the MMPP is expected to follow MPP in the worst case. Note that the number of matched ports at each time slot is dependent on the number of iterations of the two algorithms. Also, at each iteration the number of operations is determined by the worst case port since the request, grant and accept process is performed in parallel at each port. If we assume that there is packet flow from every input to every output port, then a grant arbiter needs to select one of the N requests. In this case, the number of comparison operation required is N-1, to select the maximum weight among N candidates. In the worst case, all grant arbiters may select the same input request. As a result, the corresponding accept arbiter performs N-1 comparisons. This is the first iteration of the algorithm. At the first iteration, the number of

Iteration	Algorithm i-MWM		Algorithm MPP	
	# of operations at each iteration	Cumulative # of operations	# of operations at each iteration	Cumulative # of operations
1	2(N-1)	2N-2	N-2	N-2
2	2(N-2)	4N-6	N-4	2N-6
3	2(N-3)	6N-12	N-6	3N-12
4	2(N-4)	8N-20	N-8	4N-20
N/2-1	N+2	3N(N-2)/4	2	N(N-2)/4
N/2	N	N(3N-2)/4	0	N(N-2)/4
N-1	2	N(N-1)		
N	0	N(N-1)		

Table 1. Worst Case Computational Complexity of the Two Algorithms

operations becomes 2(N-1). However, when the number of ports is N/2 as in the port partitioning algorithm, the number of operations is reduced to N-2.

At the second iteration, by assuming one pair of input and output ports is matched, the number of operations in i-MWM becomes 2(N-2). In case of MPP, the number of operations is N-4. Thus the total number of operations during the two iterations becomes 4N-6 in i-MWM, and 2N-6 in MPP. Table 1 shows the worst case computational complexity of the two algorithms. When we compare the cumulative number of operations, the proposed Algorithm MPP requires approximately between 1/4 and 1/2 as many computations as the Algorithm i-MWM depending on the required number of iterations in the matching.

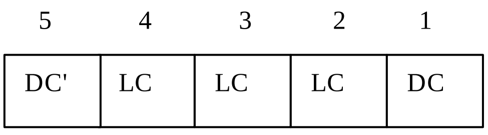
4. Dynamic Queue Scheduling

In this section, we consider queue scheduling in each input buffer to provide the flexibility for integration of mixed traffic streams, such as voice, data, and video. Since the primary benefit of fast packet switching lies in its flexibility to serve different traffic streams, scheduling in queues is necessary for reliable transmission of packets. Scheduling algorithms for a single priority class may not perform well when applied to two priority classes scheduling. The first class encompasses packets requiring low loss probability and the other class requiring very low delay. An example of the latter is voice and video traffic, while an example of the first is data traffic. We denote the traffic requiring low loss probability as loss critical (LC) traffic and that requiring very low delay as delay critical (DC) traffic.

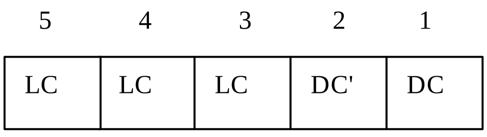
Several scheduling schemes [4,5,10] are proposed to support two priority classes of packets. Pao and Lam [4] propose a cell scheduling scheme for an input and output queued ATM switch. The input buffer is physically divided into two parts, one for DC cells and the other for LC cells. In transmission from input buffer to output buffer, DC has the priority over LC. LC can be transmitted only when there is no DC to the same output destination. A scheduling scheme for output queued switch is presented by Alnuweiri et. al. [5]. In the method an incoming DC or LC packet is immediately forwarded to its output buffer provided that the buffer is not full. If the buffer is full, DC is discarded and the LC is forwarded to a logical memory in the scheduler associated with the output buffer. At each succeeding time slot the memory attempts to forward the LC to its associated output buffer.

We consider in this study scheduling of an input queued switch with VOQ. We propose dynamic queue scheduling to take into account the changing ratio of different traffic streams, in which the popup of DC packet is decided by comparing the delay and the loss probability.

In an input buffer, LC packets get storage priority over DC packets and DC packets get delay priority over LC packets. When the input buffer is saturated, new LC packets are allowed to replace stored DC packets, but they cannot replace other LC packets. In this method, the blocking probability of LC packets is decreased. To schedule packets in a queue we define the following two terms:

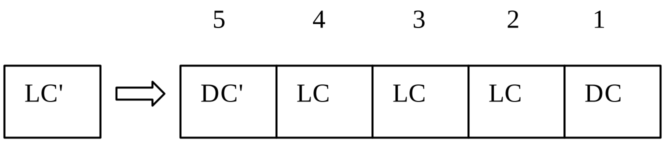


(a) Input queue state before DC-popup

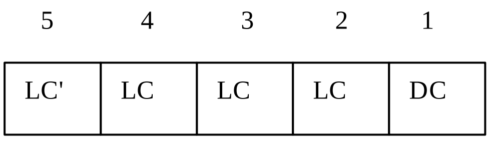


(b) Input queue state after DC-popup

Figure 6. The process of DC-popup



(a) New LC' packet arrives when the input queue is full



(b) LC-replace replaces DC' with LC'

Figure 7. The process of LC-replace

LC-replace: When a LC packet just arrived finds the input buffer is full and the last packet in its VOQ is a DC packet, then the LC replaces the last DC packet.

DC-popup: When a DC packet arrives at its VOQ, the packet is popped up to the earliest LC packet in the queue that does not precede a DC packet. We call the earliest LC packet LC_HOL.

Figure 6 and 7 show the process of DC-popup and LC-replace respectively.

When a DC or LC packet arrives at an input buffer, the input buffer decides how to store the packet at each VOQ. At this point, the following two queue scheduling methods can be considered. The first is FIFO (First In First Out) with LC-replace in which packets are queued in the order of arrival and the LC-replace is applied. The second is DC-priority queue scheduling in which both LC-replace and DC-popup are applied.

It is clear that the FIFO with LC-replace decreases the blocking probability of LC packets, and the DC-priority queue scheduling decreases the delay of DC packets. Thus the effectiveness of the two schemes depends on the portion of DC and LC packets. Clearly, FIFO with LC-replace is effective as the portion of LC packet increases, while the DC-priority queue scheduling is as the portion of DC packet increases. We are thus interested in a dynamic queue scheduling which adopts either FIFO with LC-replace or the DC-priority queue scheduling depending on occupancy of the buffer. Note that the DC-priority queue is different from FIFO in that it applies DC-popup. Therefore, in the dynamic queue scheduling we need to decide when to apply or not to apply the DC-popup.

When a DC packet is popped up, the gain is the decreased delay of the DC packet when a VOQ is not full. However, when a VOQ is full with the DC packet located at the last, the popup results in the gain as well as the loss. The loss is the lost chance of LC-replace for the following LC packet. This occurs when the VOQ has the packets as in Figure 6 (a). In Figure 6 (a) when the last DC packet is popped up, the delay of the DC packet is decreased while the chance of LC-replace by the newly arriving LC packet becomes zero.

To compare the gain and loss and to decide whether to apply DC-popup or not when a VOQ is full with the DC packet located at the last, we compare the ratio of the reduced time delay of the DC packet and that of the reduced probability of LC-replace when the DC-popup is applied.

Let n be the number of packets in the VOQ and m be the number of packets that precede the LC_HOL in the VOQ. We assume the expected service time of each head of line packet is E(s). Suppose that a DC packet is arrived, then the expected delay of the DC packet is $(n+1)$E(s). When popup is applied to the DC packet, the expected delay becomes $(m+1)$E(s). Thus the popup gain is the decreased delay of $(n-m)$E(s). In other words, when DC packet is popped up, the delay is decreased by $(n-m)$E(s) and the ratio of time delay of the DC packet by applying the popup becomes $(n-m)/(n+1)$.

Now, we consider the reduced probability of LC-replace after the DC-popup. Clearly, the chance of LC-replace is never lost, if a VOQ is not full. This is because the LC-replace occurs only when the input buffer is full. Thus, DC-popup results in reduced time delay when the VOQ is not full.

However, when the VOQ is full with the DC packet located at the last, two cases follow:

1. The second last packet is DC.
2. The second last packet is LC.

In the first case, the DC-popup cannot occur, since a DC cannot pass another DC in the queue. In the second case, the probability of LC-replace is equal to one before applying the popup (see Figure 6 (a)) and that of LC-replace is zero after the popup (see Figure 6 (b)). Thus the ratio of the reduced probability of LC-replace becomes one. Here, note that the popup of a DC packet can be applied when the gain is greater than the loss, i.e.,

$(n - m) / (n + 1) > 1$, if the last packet in the VOQ is a LC packet after popup.

However, the above case is not possible since $(n\text{-}m) / (n+1) < 1$. Therefore, applying popup of a DC packet when the VOQ is full is not recommended.

Based on the above analysis we propose the following Dynamic queue scheduling:

1. Apply FIFO with LC-replace.
2. Apply DC-popup at any VOQ which is not full.
3. Do not apply DC-popup when the VOQ is full with DC packet located at the last.

The performance of the proposed dynamic queue scheduling is examined by comparing with the FIFO with LC replace and the DC-priority queue scheduling in Section 5.

5. Computational Results

In order to simulate scheduling algorithm, we assume Bernoulli traffic; the probability of traffic occurrence is p, $0 \le p \le 1$, at each input at every time slot. Each packet is assumed to have uniform switching to all output ports. We also assume that one time slot is one packet transmission time. The packet size is assumed fixed. The input queue switch is assumed to have 32 input and 32 output ports. The input queue switch adopts VOQ. Each algorithm is implemented for 500,000 time slots. The weight is determined base on the occupancy of the queue such that the port with longest queue has the priority to be matched.

Packet delay, buffer size and throughput are examined at various packet arrival rates, which are measured by the number of packets arrived per time slot. Delay is measured by the number of time slots during which a packet is in the queue and the unit of buffer size is the number of packets. Throughput represents the average number of transmitted packets over 32 input ports at one time slot.

Figure 8 and 9 show the average delay of three algorithms: i-MWM, MPP, MMPP. Note that the number of computational operations by Algorithm MPP and MMPP is approximately half of that by Algorithm i-MWM when the required number of iterations in the matching is relatively small. Thus, for fair comparison, the number of iterations by the two proposed algorithms is made twice of that by the i-MWM. At low packet arrival rate, i-MWM and MMPP have slightly lower delay than MPP. However, at high packet arrival rates, MMPP has the lowest delay compared to two other methods. Clearly, the packet delay is reduced by an increased number of iterations.

In Figure 10 and 11, the required maximum input buffer size is presented. From Figure 11 approximately 20-60 packets are buffered by Algorithm MMPP when the packet arrival rate exceeds 0.9. However, the buffer size is exponentially increased with i-MWM at high packet arrival rate. From Figure 12 and 13 it is clear that the throughput is almost linear to the packet arrival rate with the two proposed methods, which can be obtained under highly efficient matching algorithms.

Now, to test the performance of dynamic queue scheduling proposed in Section 4, delay of packets and loss probability of LC packets are examined. Note that the LC packet loss occurs when the buffer is saturated. Since the VOQs are virtual and they are operated as one queue in reality, we implement the dynamic queue scheduling with three versions: Dynamic-90, Dynamic-95 and Dynamic-97. In the three versions, the popup of DC packet is applied according to the rule in Section 4. The VOQs are considered full when 90%, 95% and 97% of the buffer is filled respectively in the three versions. Algorithm MMPP is implemented with six iterations per time slot. Buffer size and packet arrival rate are fixed to 50 and 0.98 (heavy traffic) respectively. Three input queue scheduling methods are compared: FIFO, DC-priority queue scheduling and dynamic queue scheduling.

Figure 14 shows the average delay of DC packets. At high packet arrival rate, the delay by Dynamic-95 and Dynamic-97 slightly exceeds the DC-priority queue in which every DC packet is popped up to reduce the delay. The loss probability of LC packets is shown in Figure 15. Clearly, the proposed methods are competitive to the FIFO which shows the least loss probability.

The performance of Figure 14 and 15 is summarized with the normalized utility to compare three queue scheduling methods in Figure 16. Note that the utility of loss probability by FIFO is set to one and that by DC-priority is set to zero. Also, the utility of delay by FIFO is set to zero and that by DC-priority is set to one. The two utilities are then normalized with the proportion of DC and LC packets. Figure 16 shows the normalized utility of three queue scheduling methods. As shown in the figure, the normalized utility by FIFO is decreased as the ratio of DC packet increases. The utility by DC-priority is decreased with the increase of LC packets. However, the utility of the dynamic queue scheduling is the highest and not sensitive to the ratio of the two packets. Above 84% utility is demonstrated by the Dynamic-95 and Dynamic-97 without regard to the ratio of DC and LC packets.

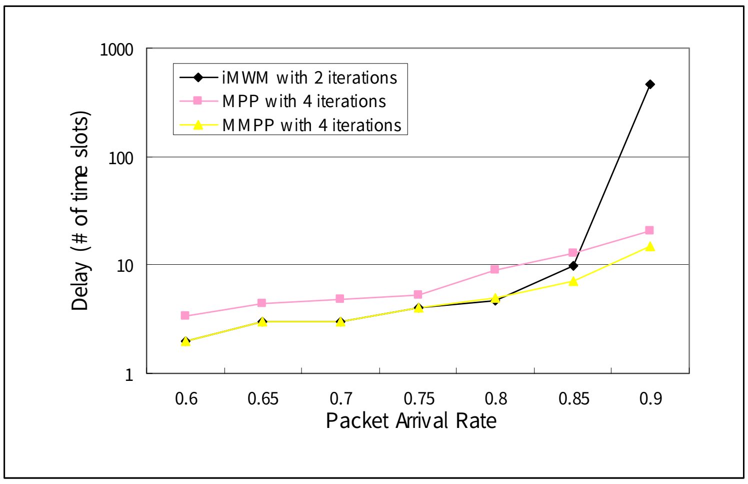


Figure 8. Average Packet Delay with 4 iterations of MMPP

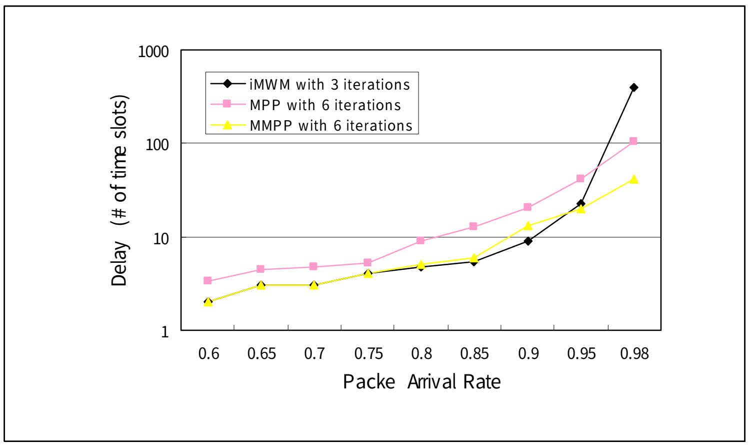


Figure 9. Average Packet Delay with 6 iterations of MMPP

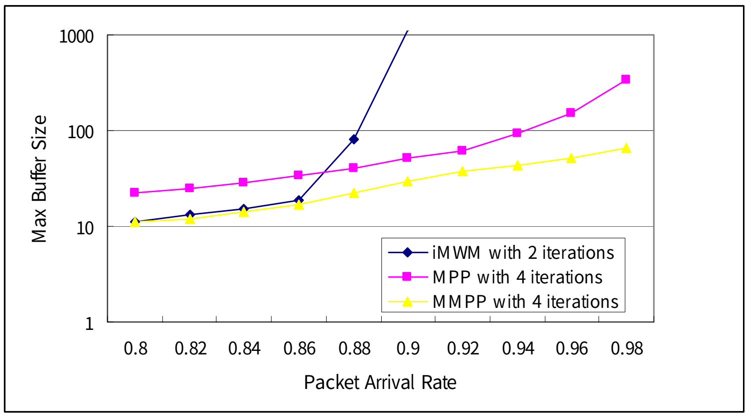


Figure 10. Input Buffer Size with 4 iterations of MMPP

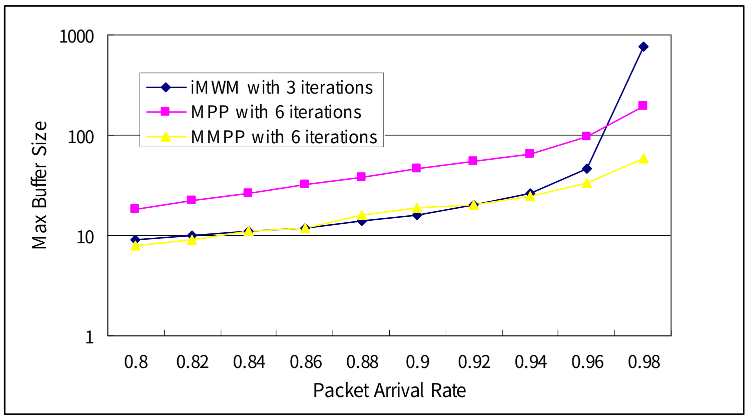


Figure 11. Input Buffer Size with 6 iterations of MMPP

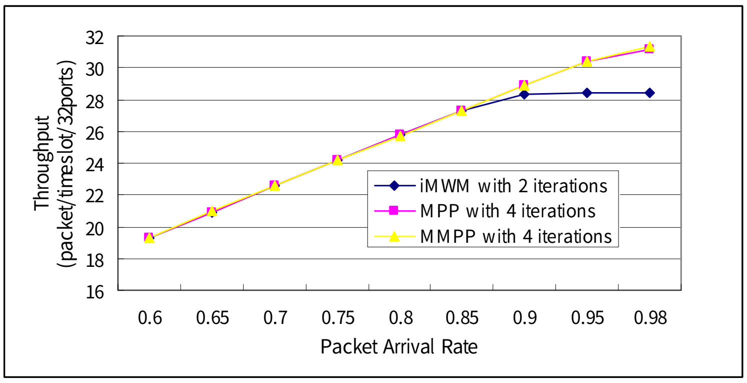


Figure 12. Throughput with 4 iterations of MMPP

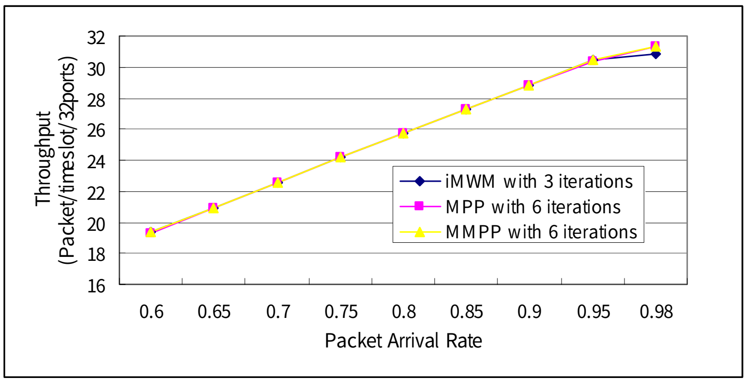


Figure 13. Throughput with 6 iterations of MMPP

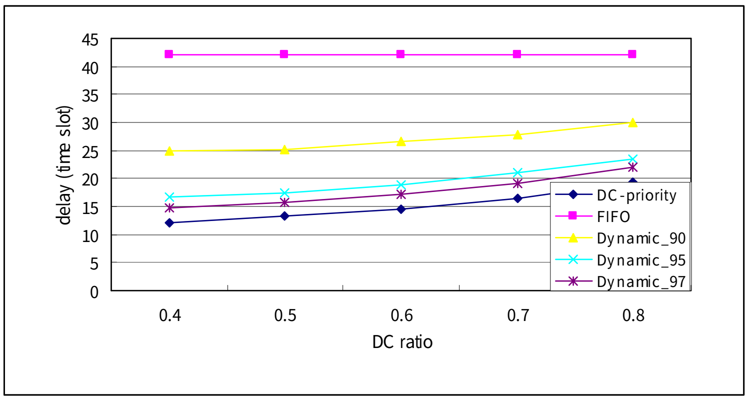


Figure 14. Delay of DC Packets

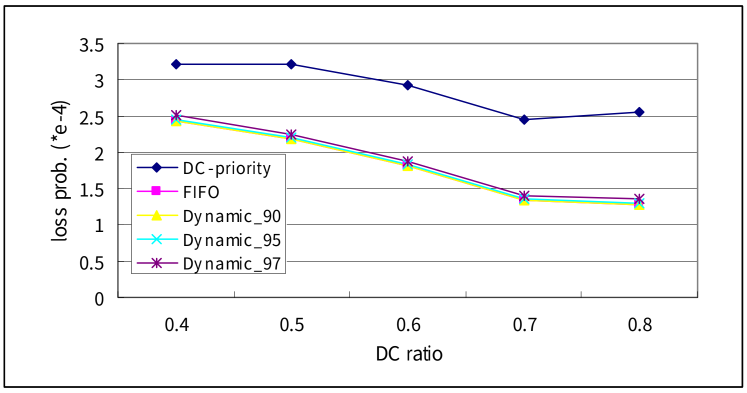


Figure 15. Loss Probability of LC Packets

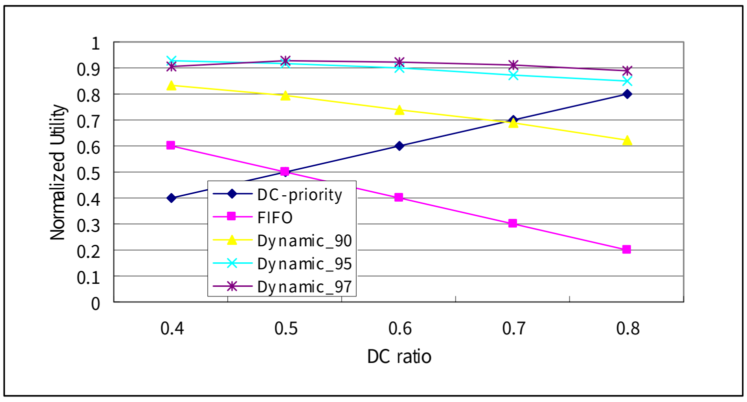


Figure 16. Normalized Utility of Three Queueing Methods

6. Conclusion

In this paper, we have proposed input and output matching algorithms by port partitioning in a gigabit router which has input queue with virtual output queueing.

Two iterative maximal weight matching MPP and MMPP are proposed by partitioning input and output ports into two groups. The matching process is accomplished within each pair of input-output groups in parallel. The input-output pair is exchanged at each time slot. In MMPP, matching input-output ports that are not in the same pair of groups is allowed, when each of the ports is idle in its current pair. The effectiveness of port partitioning is illustrated by computational results. A better performance is obtained when the packet arrival rate is relatively high. MMPP demonstrated the best performance in delay, required buffer size and throughput.

In dynamic queue scheduling packets are divided into two classes: loss critical packet and delay critical packet. A popup decision rule of the delay critical packet is suggested to improve the delay of DC packets and to reduce the loss probability of LC packets under heavy traffic environment. The suggested dynamic queueing outperforms the FIFO and DC-priority queue scheduling in view of a combined utility of delay and loss.

Reference

[1] M. Karol, M. Hluchyj, and S. Morgan, “Input versus Output Queueing on A Space Division Switch,” IEEE Trans. Communications, 35(12), 1987, pp. 1347-1356.

[2] N. McKeown, “Scheduling Algorithms for Input-Queued Cell Switches,” phD Thesis. University of California at Berkeley, 1995.

[3] N. McKeown, V. Anatharam and J. Walrand, “Achieving 100% Throughput in an Input-Queued Switch,” Proc. INFOCOMM ’96.

[4] D. C. W. Pao and S. P. Lam, “Cell Scheduling for ATM Switch with Two Priority Classes,” IEEE ATM Workshop Proceeding, 1998, pp. 86-90.

[5] H. M. Alnuweiri, Y. He and M. Ito, “Multipriority Packet Switchin on the HYPER Switch,” IEEE ATM Workshop Proceeding, 1998, pp. 34-42.

[6] J. E. Hopcroft and R. M. Karp, “An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs,” Society for Industrial and Applied Mathematics I. Comput., 2, 1973, pp. 225-231.

[7] R.E. Tarjan, “Data Structures and Network Algorithms,” Society for Industrial and Applied Mathematics, Pennsyvania, Nov 1983.

[8] T. Anderson and S. Owicki, J. Saxe, “High Speed Switch Scheduling for Local Area Networks,” ACM Trans. on Computer Systems, Nov. 1993, pp. 319-352.

[9] ETRI, “A study on gigabit ethernet interface technology,” Dec. 1997.

[10] H. Ohnishi, T. Okada and K. H. Noguchi, “Flow Control Schemes and Delay/Loss Tradeoff in ATM Networks,” IEEE J. Select. Areas Commun. , vol 6, Dec. 1988, pp. 1609-1616.

[11] B. Davie, P. Doolan and Y. Rekhter, “Switching in IP Network,” Morgan Kaufman Publisher Inc.