Journal of the Korean Institute of Industrial Engineers
Vol. 25, No. 2, pp. 233-239, June 1999.

A Scheduling of Switch Ports for IP Forwarding†

Chae Y. Lee[1] · Wang Hwan. Lee[2] · Hee K. Cho[1]

[1]Department of Industrial Engineering, Korea Advanced Institute of Science and Technology /

[2]Broadband Communications Dept., ETRI

IP 포워딩을 위한 스위치 포트 스케쥴링

이채영[1] · 이왕환[2] · 조희권[1]

With the increase of internet protocol (IP) packets the performance of routers became an important issue in internetworking. In this paper we examined the matching algorithm in gigabit router which has input queue with virtual output queueing. Port partitioning concept is employed to reduce the computational burden of the scheduler within a switch. The input and output ports are divided into two groups such that the matching algorithm is implemented within each input-output pair group in parallel. The matching is performed by exchanging input and output port groups at every time slot to handle all incoming traffics. Two algorithms, maximal weight matching by port partitioning (MPP) and modified maximal weight matching by port partitioning (MMPP) are presented. MMPP has the lowest delay for every packet arrival rate. The buffer size on a port is approximately 20-60 packets depending on the packet arrival rates. The throughput is illustrated to be linear to the packet arrival rate, which can be achieved under highly efficient matching algorithm.

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 the bottleneck problem in routers have been proposed. One is route once and switch many and the other is gigabit router (ETRI, 1997). 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 (Davie *et al.*, 1998) and Tag switching of Cisco (Davie *et al.*, 1998) 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 uses longest prefix matching algorithm to look up the routing table. The multi gigabit router (MGR) is an example of this approach (Mckeown, 1995).

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, in this paper, focus our attention to the gigabit router which has input queue with virtual output queueing at each port (Anderson *et al.*, 1993). Virtual ouput queueing (VOQ) is suggested to overcome limitations limitations of head of line (HOL) blocking in input queue system.

The HOL blocking can be entirely eliminated by using a simple buffering strategy at each input port. In VOQ, rather than maintaining a single first in

† This research was supported by the fund from Electronics and Telecommunications Research Institute.

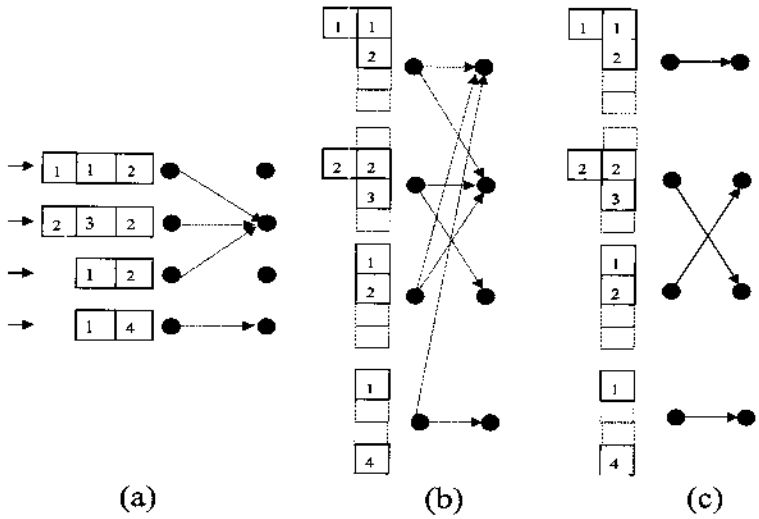

Figure 1. HOL Blocking (a), VOQ (b) and Switching (c).

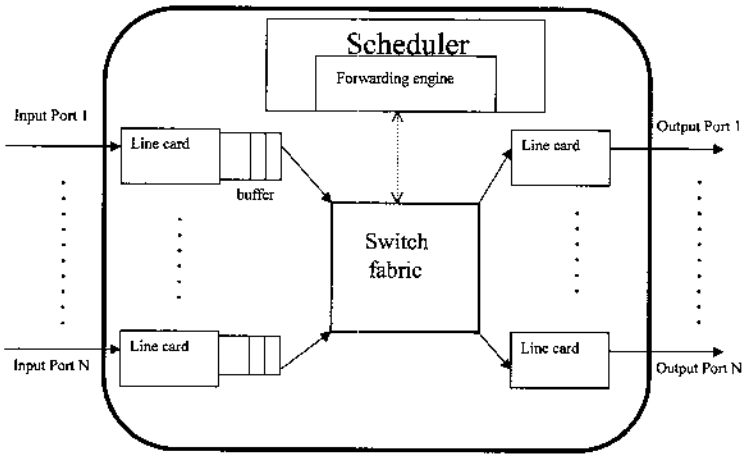

Figure 2. Basic Structure of a Switch.

first out (FIFO) queue for all packets, each input maintains a separate queue for each output(Anderson *et al.*, 1993), as shown in <Figure 1>. It is known that the throughput by VOQ is improved to 100% (Mckeown, 1996) compared to the 58% (Karol *et al.*, 1987) at an input queue switch with HOL blocking.

2. Input and Output Ports for Switching

A basic structure of switch is shown in <Figure 2>. 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 impelmented by the switch fabric.

Algorithms are developed to solve the matching problems. Maximum cardinality matching algorithm (Hopcroft and Karp, 1973) is proposed to find the match that maximizes the number of edges, while maximum weight matching algorithm (Tarjan, 1983) maximizes the sum of edge weights. Clearly, maximum size matching is a special case of the maximum weight matching.

It was demonstrated using simulation that the maximum cardinality matching algorithm is stable for independent and identically distributed arrivals up to offered load of 100% when the traffic is uniform (Mckeown, 1996). 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 (Tarjan, 1983).

Since the maximum weight matching algorithm is very complex to implement in hardware, we are interested in iterative approximation to maximum weight matching in the iterative Maximal Weight Matching (i-MWM) algorithm (Mckeown, 1995). N input and N output arbiters operate in parallel as in Parallel Iterative Matching (PIM) which was developed by DEC Systems Research Center for the 16-port, 1 Gbps AN2 switch (Anderson et al., 1993). At each time slot, the matching for the next time slot is scheduled as follows. 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 one to which it made the largest weight grant. Ties are broken arbitrarily.

<Figure 3> shows the above matching process at an iteration.

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

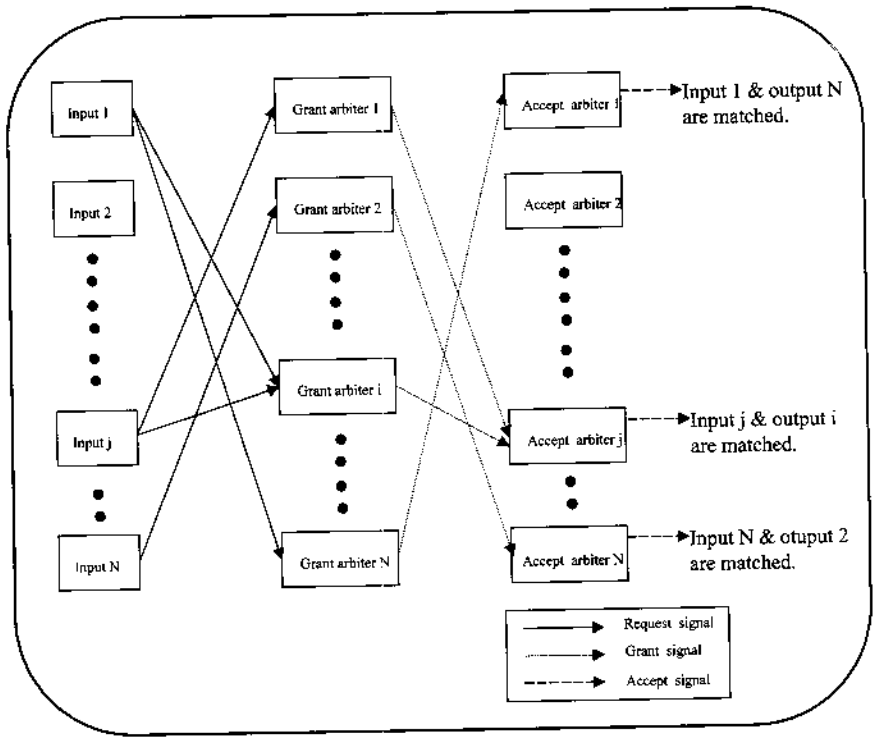

Figure 3. Implementation of Arbiters in Scheduler.

becomes a 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 in one time slot before transmission. An example of port partitioning is shown in <Figure 4>. As shown in the figure, the input and output port group considered in one time slot is exchanged in the succeeding time slot. The matching is performed within the group in parallel. In other words in <Figure 4>, input ports 1, 2, 3 and 4 and output ports 1, 2, 3 and 4 are in one group in one time slot. In the next time slot, input ports 1, 2, 3 and 4 and output ports 5, 6, 7 and 8 are in one group. Thus only within group traffics are considered for matching as indicated in Figure. The matching process of port partitioning is explained in the follow- ing Algorithm MPP.

Algorithm MPP

Step 1. Let G_0 = {grant arbiter i | $1 \le i \le N/2$,
*N: total number of ports
G_1 = {grant arbiter i | $N/2 + 1 \le i \le N$},
I_0 = { input port i | $1 \le i \le N/2$ },
and I_1 = { input port i | $N/2 + 1 \le i \le N$}.

Step 2. time = current timeslot; iteration = 1;

Step 3. Each 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(mod 2)).

Step 5. Each grant arbiter in G_1 selects one input port to have max weight in I_(1-(time(mod2))).

Step 6. Each grant arbiter sends the selected input index to that accept arbiter.

Step 7. Each unmatched accept arbiter selects one output to have max weight. If an accept arbiter selects an output port, then the corresponding input and output port is matched. Thus, at next iteration, the corresponding input and output port is not considered to find match.

Step 8. If no matching exisits, goto Step 10.

Step 9. iteration ++; goto Step 3.

Step 10. Input port having matched traffic, transmits the matched traffic.

In <Figure 4>, notice that the packet from input port 4 to output port 6 is delayed due to the grouping. In this case, by including such a packet into the matching as shown in <Figure 5>, we can improve the switch efficiency. It will not only reduce the delay of packets but also increase the throughput by giving the chance to accept packets at the two ports in the following 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; iteration = 1.

Step 2. Let G_0 = {grant arbiter i| $1 \le i \le N/2$},
*N: total number of ports
G_1 = {grant arbiter i | $N/2 + 1 \le i \le N$},
I_0 = {input port i | $1 \le i \le N/2$},
and I_1 = {input port i | $N/2 + 1 \le i \le N$}.

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(mod 2)),
$G\_0 = G\_0 \setminus \{k\}$ and $G\_1 = G\_1 \cup \{k\}$.

Step 5. For all G_1, if grant arbiter k didn't receive any signal from I_(1-(time(mod 2))),
$G\_1 = G\_1 \setminus \{k\}$ and $G\_0 = G\_0 \cup \{k\}$.

Step 6. Each grant arbiter in G_0 selects one input port to have max weight in I_(time (mod 2)).

Step 7. Each grant arbiter in G_1 selects one input port to have max weight in I_(1-(time(mod 2))).

Step 8. Each grant arbiter sends the selected input index to that accept arbiter.

Step 9. Each unmatched accept arbiter selects one output to have max weight. If an accept arbiter selects an output port, then the corresponding input and output port is matched. Thus, at next iteration, the corresponding input and output port is not considered to find match.

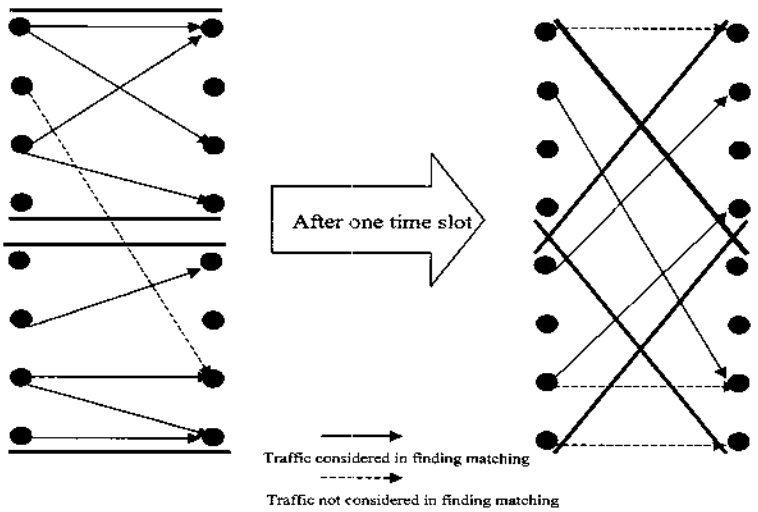

Figure 4. An Example of Port Partitioning.

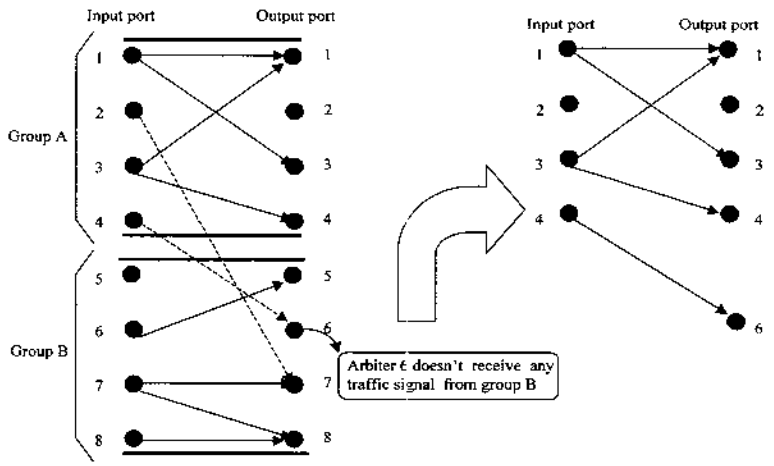

Figure 5. An Example of Modified Matching.

Step 10. If no matching exists, goto Step 12.

Step 11. iteration ++; goto Step 2.

Step 12. Input port having matched traffic, transmits the matched traffic.

4. Computational Results

Algorithms presented in the previous section were implemented in Visual C++ (Version 5.0), and ran on a 90 MHz Intel Pentium based personal computer with 16 Mb of memory under Window 95. In order to simulate the scheduling algorithm, we assume the Bernoulli traffic; the probability of traffic occurrence is p, $0 \leq p \leq 1$, at each input in every time slot. 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.

Delay, buffer size and throughput are employed to measure the performance. The unit of delay is time

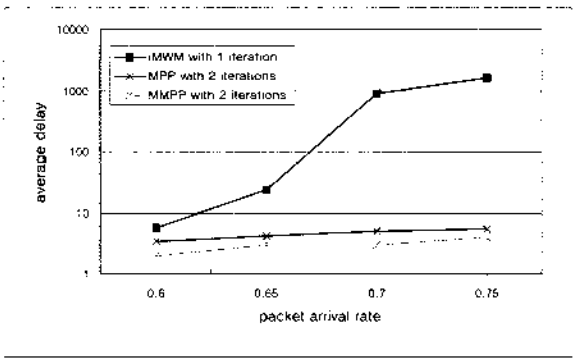

Figure 6. Average Delay with 2 Iterations in MMPP.

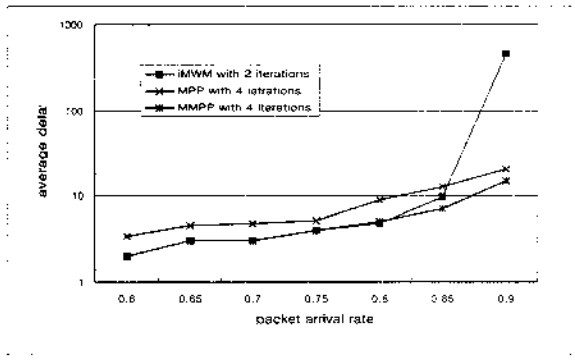

Figure 7. Average Delay with 4 Iterations in MMPP.

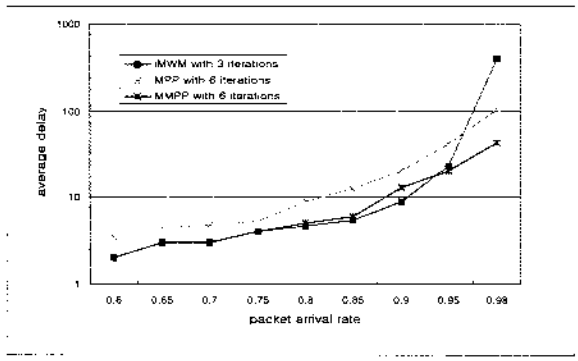

Figure 8. Average Delay with 6 Iterations in MMPP.

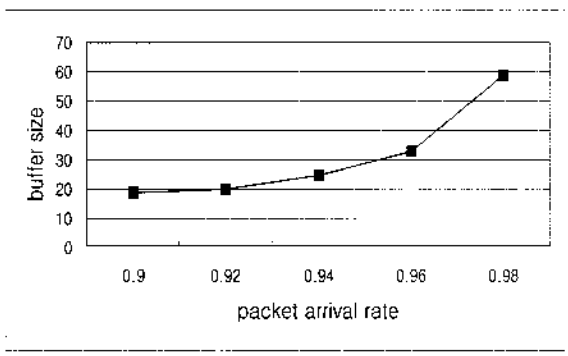

Figure 9. Buffer Size by MMPP with 6 iterations.

slots and the unit of buffer size is the number of packets. Throughput is the average number of transmitted packets over 32 input ports in one time slot.

<Figure 6>, <Figure 7> and <Figure 8> show the average delay of three algorithms (i-MWM, MPP, MMPP). At low packet arrival rate, i-MWM and MMPP have slightly lower delay than MPP. However, at high packet arrival rate, MMPP has the lowest delay compared to two other methods. In the three figures the number of iterations represents the number of request-grant-accept processes for the matching at each time slot. Since the input and output ports are divided into two groups, two iterations in the proposed MPP and MMPP corresponding to one iteration of the algorithm i-MWM.

In <Figure 9> it is shown that the required buffer size by the 32 ports is approximately 20-60 depending on the packet arrival rates. In <Figure 10>, it is clear that the throughput is almost linear to the packet arrival rate, which can be obtained under highly efficient matching algorithm. Notice that

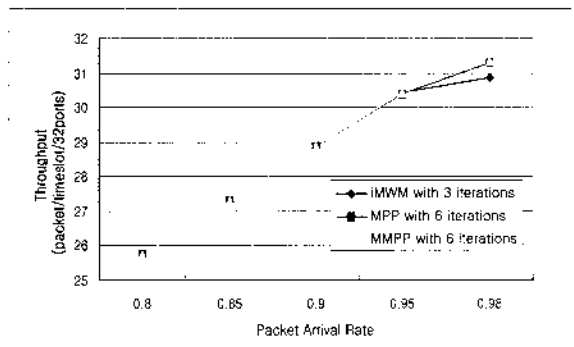

Figure 10. Throughput of MMPP with 6 iterations.

higher throughput is obtained by MMPP than the existing method when the packet arrival rate is extremely high. For lower arrival rates almost same result is obtained by the three methods, which is

mainly due to the fixed number of input and output ports.

5. Conclusions

Input and output matching algorithms by port partitioning are proposed for IP forwarding in gigabit router. The matching process is accelerated by partitioning input and output into two groups respectively. Matching is accomplished within each pair of input-output groups in parallel. In MMPP, matching input-output ports that are not in the same pair of groups is allowed, when each of the port is idle in its current pair. The effectiveness of port partitioning is illustrated with computational results. Better performance is obtained when the packet arrival rate is relatively high. MMPP demonstrated the best performance in delay, required buffer size and throughput.

References

Anderson, T., Owicki, S. and Saxe, J. (1993), High speed switch scheduling for local area networks, *ACM Trans. on Computer Systems*. Nov., 319-352.

Davie, B., Doolan, P. and Rekhter, Y. (1988), *Switching in IP Networks*, Morgan Kaufman Publisher Inc.

ETRI (1997), *A study on gigabit ethernet interface technology* Dec.

Hopcroft, J. E. and Karp, R. M. (1973), An n5/2 algorithm for maximum matching in bipartite graphs, Society for Industrial and Applied Mathematics I. Comput., 2, 225-231.

Karol, M., Hluchyj, M. and Morgan, S. (1987), Input versus output queueing on a space division switch, *IEEE Trans. Communications*, 35(12), 1347-1356.

McKeown, N. (1995), Scheduling algorithms for input-queued cell switches, *phD Thesis*. University of California at Berkeley.

McKeown, N., Anatharam, V. and Walrand, J. (1996), Achieving 100% throughput in an input-queued switch, *Proc. INFOCOMM '96*, San Francisco, 296-302.

Tarjan, R. E. (1983), Data structures and network algorithms, *Socirty for Industrial and Applied Mathematics*, Pennsylvania, Nov.