Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

Design Issues of the Hybrid Buffer Management Schemes for QoS Fairness

Abstract

There are many link-scheduling algorithms for network switches. FIFO is easy to implement but can not provide rate guarantees. WFQ can provide rate and delay guarantees but is hard to be implemented in a backbone environment with thousands of sessions, as their complexity increases with the number of sessions. The *hybrid* algorithm is a scalable architecture for QoS provisioning that integrates the proposed buffer management scheme with WFQ scheduling that uses a small number of queues. Various design issues of the *hybrid* algorithm are investigated in this paper. Several *queue selection algorithms* are proposed, simulated and analyzed. The *adaptive* algorithm is found to be the best queue selection algorithm. Two important design principals of the buffer management schemes are proposed and testified.

1 Introduction

In order to provide QoS (Quality of Service) on networks, network switches or routers need to assure rate and delay bounds for individual links. The traditional FIFO scheduling algorithm is good for the *best-effort* networks, but unsuitable to support QoS because it requires network switches to control the amount of resources that each flow or set of flows is allowed to consume. The network resources, whose consumption is to be controlled, consist primarily of buffers and link bandwidth, with buffer management and scheduling being the associated mechanisms.

The mechanisms used to provide such control have two main overheads: a) the storage and processing of the state information associated with service guarantees; b) the cost of making the per packet admission and transmission decisions required to enforce service guarantees. As link speeds keep increasing, the overhead of packet scheduling is often higher than that of the buffer management.

In the FIFO algorithm, there is only one queue. Incoming packets from all flows are put in this queue and will be outputted in a first-in-first-out order. The overhead of this scheduling algorithm is very low. However, although the FIFO algorithm is simple and can achieve high link utilization, there is no limitation on the buffer size that can be occupied by one flow. Therefore, an aggressive flow can easily eat up too many buffer spaces and impact the rate guarantees of all the other flows. Guérin, et al. propose the *Buffer Thresholds* and *Buffer Sharing* schemes for FIFO algorithm [2]. The buffer threshold scheme helps protect flows from aggressive non-conformant flows. However, it does not allow the sharing of the excess bandwidth among flows. The buffer sharing scheme can improve link utilization greatly without lead to worse protection for flows. Moreover, it can distribute excess bandwidth among all flows fairly and efficiently.

The *Fair Queuing* (FQ) [1] algorithm performs a logical bit-by-bit round robin on all flows. *The Weighted Fair Queuing* (WFQ) algorithm generalizes FQ to relax the requirement that each of the N sources gets 1/N of the bandwidth. It associates a "weight" with each source, which determines the relative number of service time units it gets in each full cycle. In this paper, I use the Virtual Clock [5] to implement the idea of the WFQ.

In WFQ, there is one queue per flow. Each queue is assigned a *weight* based on the arrival rate, packet size or other parameters. A *virtual tick* is computed from the weight. A *virtual clock* is assigned to the first packet of each queue. Whenever a packet of a queue departs, the virtual clock of that queue is advanced by a virtual tick. Only the packet with the smallest virtual clock value is the next candidate for output. Therefore, the packet scheduling cost is the sorting of the virtual clocks. If there are N flows in the switch, the packet scheduling cost could be *logN*. In a backbone environment with thousands of flows and high link speed, WFQ is too expensive to implement.

Similar to FIFO, the WFQ algorithm without buffer management scheme can not protect non-aggressive flows if there are aggressive non-conformant flows. These aggressive flows could occupy many buffer spaces and impact the performance of other flows. The buffer thresholds and buffer sharing schemes can also be applied to WFQ. WFQ with buffer management schemes can not only protect conformant flows from aggressive flows, but fairly distribute excess bandwidth among flows as well.

Paper [2] proposes a Hybrid algorithm which can overcome the shortage of both FIFO and WFQ. However, many design issues of this algorithm are unspecified. These design issues are studied in this paper. Section 2 discusses the Hybrid algorithm and some possible queue selection algorithms. Section 3 describes the experiment design and the simulation setup. The input source generation is also investigated. Section 4 presents and analyzes the simulation results of the queue selection algorithms which are described in section 2. Section 5 briefly summarizes the work of the paper, its limitation, and some future directions.

2 The *Hybrid* Algorithm

As discussed in [2], the FIFO algorithm with proper buffer management schemes shows some potential ability to provide rate guarantees in addition to high link utilization. On the other hand, WFQ can archive better performance with much higher overhead. A *hybrid* of FIFO and WFQ tries to combine the virtues of both— scalability and rate guarantee — while avoiding their problems. It replaces the single FIFO queue with multiple FIFO queues, with a WFQ scheduler providing each queue its own rate guarantees. Multiple flows are grouped into one queue. A buffer threshold and a rate guarantee are assigned to each queue. The buffer threshold is used by the buffer sharing scheme to manage the buffer space occupied by each queue. The rate is used as the weight in the WFQ scheduler. By keeping the number of such queues fixed and reasonably small, the overall architecture still remains scalablility in terms of the total number of flows.

2.1 Buffer sharing scheme

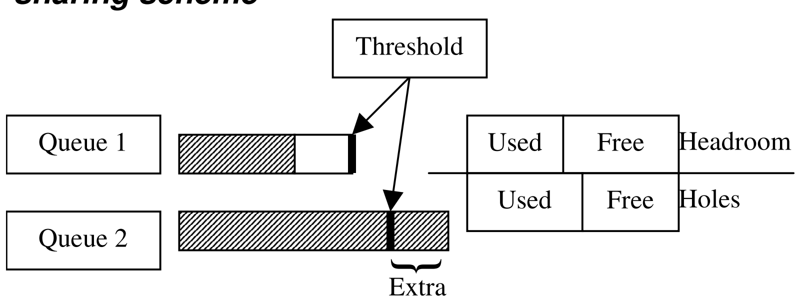


Figure 1 Buffer sharing scheme

The buffer-sharing scheme is an improvement to the fixed threshold scheme. It is intended to increase link utilization and promote the sharing of excess bandwidth. For every queue *i*, there is a threshold B_i. The whole buffer is partitioned into two regions: *headroom* and *holes*. Headroom is reserved for all queues whose length is below their thresholds. Holes are unused buffer that can be shared by all queues. As in Figure 1, queue 1 can accept a new packet if there is space in either holes or headroom. However, because queue 2 has exceeded its threshold, it can accept a new packet only when there are holes left and the extra packets in queue 2 is less than the free holes.

Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

Whenever some buffer space is freed, it is always added into the headroom instead of the holes, unless the headroom has reached its maximal value. Larger headroom can give more protection to non-aggressive flows. Larger holes can let aggressive flows utilize more unused buffer. When the headroom is equal to the buffer size, the buffer sharing scheme degrades to the buffer threshold scheme.

2.2 Determination of thresholds and queue rates

In order to employ WFQ scheduler in Hybrid algorithm, a rate should be assigned to each queue. The idea is to assign each queue the minimum requested rate plus a fraction of the excess link capacity. From [2], both the buffer thresholds and queue rates are determined by the following equations.

It is useful to first introduce some notations:

- A leaky bucket profile (i.e. describing the flow through a token rate and burst size) is used to describe a flow. Therefore, flow i has a traffic profile (ρ_i, σ_i). If a flow is conformant, this flow is regulated by a leaky bucket with parameters corresponding to their traffic profile.
- R is the output link rate.
- ρ and σ denote the sum of rates and bursts of all flows.
- k is the number of FIFO queues in which flows are classified.
- R_i denotes the rate at which the *i*th FIFO queue is served by a WFQ-like scheduler.
- $\hat{\rho}_i$ is the sum of the rate requirements of all flows in *i*th queue.
- $\hat{\sigma}_i$ is the sum of the burst requirements of all flows in the *i*th queue.
- B_i denotes the minimum amount of buffer space required for the *i*th queue.

The rate assigned to the *i*th queue is:

$$R_i = \hat{\rho}_i + \alpha_i (R - \rho) \tag{1}$$

where α_i is chosen as

$$\alpha_i = \frac{\sqrt{\hat{\sigma}_i \hat{\rho}_i}}{\sum_{i=1}^{k} \sqrt{\hat{\sigma}_i \hat{\rho}_i}} \tag{2}$$

The buffer space assigned to the *i*th queue is:

$$B_i = \frac{R_i \hat{\sigma}_i}{R_i - \hat{\rho}_i} \tag{3}$$

2.3 Design Parameters of the Hybrid Algorithm

There are many parameters in the Hybrid algorithm. For example, how many queues should be used to group the flows, and how to assign flows to each queue dynamically when a new flow comes. In [2], a benefit equation is defined to describe the difference of the buffer requirement between a single FIFO queue and the hybrid system with *k* queues:

$$Benefit = B_{FIFO} - B_{Hybrid} = \frac{\sum_{i,j=1}^{k} (\sqrt{\hat{\sigma}_i \hat{\rho}_j} - \sqrt{\hat{\sigma}_j \hat{\rho}_i})^2}{(R - \rho)} \tag{4}$$

Properly assigning flows to Hybrid queues could generate a maximum benefit of the Hybrid algorithm. However, it is not feasible to keep using the best grouping of flows because reshuffle

Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

flows among queues may be too expensive to implement. Therefore, some heuristic queue selection strategies need to be used. These algorithms are discussed in the following subsections.

2.4 Queue Selection Algorithms

In a network switch, flows come and go. For the Hybrid algorithm, when a flow comes, it must be assigned to a queue. Various kinds of possible queue selection algorithms are discussed below and the simulation results are presented and analyzed in section 4.

The following are some *simple* methods:

- **Random Selection.** When a flow comes, select a random queue from all available queues.
- **Round-robin Selection.** When a flow comes, select a queue in a round-robin order.
- **Rate Balance Selection.** This algorithm tries to distribute the rate of flows evenly on all queues. Select a queue *i* with the smallest rate $\hat{\rho}_i$, then assign the new flow to queue *i.*
- **Burst Balance Selection.** This algorithm tries to distribute the burst of flows evenly on all queues. Select a queue *i* with the smallest rate $\hat{\sigma}_i$, then assign the new flow to queue *i.*

The above queue selection algorithms are simple, but they may group flows in an inefficient way, because they do not consider the properties of the hybrid queue. In order to assign a proper queue to the new flow, the value of the equation (4) should be maximized. The following algorithms utilize the result of this equation. Suppose the traffic profile of the new flow is (ρ_{new}, σ_{new}), we have the following *benefit-based* algorithms:

- **Similar Queue Selection.**

  When flows with similar properties are grouped together, the benefit from equation (4) can reach a higher value. Therefore, this algorithm tries to group similar flows together. If queue i has n_i flows, then the average rate $\overline{\rho}_i$ and $\overline{\sigma}_i$ of this queue is defined as:

$$\overline{\rho}_i = \frac{\hat{\rho}_i}{n_i} \tag{5}$$

$$\overline{\sigma}_i = \frac{\hat{\sigma}_i}{n_i} \tag{6}$$

  Then difference $Diff_i$ between the new flow and each queue i is computed as:

$$Diff_i = \frac{|\rho_{new} - \overline{\rho}_i|}{\overline{\rho}_i} + \frac{|\sigma_{new} - \overline{\sigma}_i|}{\overline{\sigma}_i} \tag{7}$$

  This definition is used because the value $\overline{\rho}_i$ and $\overline{\sigma}_i$ may have a big difference, and should be normalized before summing them together. Then, the queue i with the minimum $Diff_i$ is selected as the candidate, and the new flow will be assigned to the queue i.

- **Near Optimal Selection.**

This algorithm tries to get the largest benefit based on equation (4). $Benefit_i$, which equals to the benefit if the new flow is assigned to queue i, is computed for each queue. This is done by using

Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

$\hat{\rho}_i + \rho_{new}$ and $\hat{\sigma}_i + \sigma_{new}$ for queue i in equation (4). Then the new flow is assigned to the queue with the maximum $Benefit_i$.

- **Adaptive Selection.**

This algorithm not only considers the equation (4), but also considers the aggressiveness of the flows. There are three groups of queues which serve for three kinds of flows: *conformant flows* which conforms to their traffic profile, *moderate non-conformant flows* which temporarily exceed their traffic profile, and *aggressive flows* which often exceed their traffic profile a lot. If a flow is non-conformant to its profile, the network switches should detect this and make corresponding actions. First, all flows are assigned to the conformant group. If a flow is detected not conforming to its profile, it will be moved to the moderate non-conformant group. Further, if it is found to be aggressive, it will be then moved to the aggressive group. When assigning flows to each group of queues, the **Similar Queue Selection** algorithm is used.

The Similar Queue Selection is a heuristic implementation of the Near Optimal Selection, and has a lower overhead. The Adaptive Selection protects non-aggressive flows from the aggressive flows. Therefore, it may result in a lower packet loss when there are aggressive flows in the switch. However, it needs to shuffle flows between queues, which results in higher overhead.

2.5 Potential Problems of the Hybrid Algorithm

In the Hybrid algorithm, it uses fixed number of queues. This makes the algorithm scalable, but also brings some problems. Because many flows are on one queue, they might interfere with each other, just like what happens in FIFO algorithm. The worse thing is when one of the flows is aggressive, it may impact the rate guarantee of all the other flows on the same queue greatly. Because the Adaptive Selection algorithm takes this into consideration, it should overcome all other algorithms if there are many aggressive flows.

3 Experiment Design

3.1 Simulation Setup

A network switch simulator is developed to simulate the switch activities. This simulator is developed from scratch and has about 3400 C++ source code. Various parameters like flow profiles, packet size, link speed, headroom size, buffer size, and switch algorithms can be set in a configuration file. The data presented in section 4 are generated by this simulator.

The output link is operating at 48Mb/s, which is a little over T3 capacity, over which a number of flows with various traffic patterns and rate guarantees are multiplexed. The buffer sharing scheme discussed above is always used in the simulation. The headroom is always a half of the whole buffer size in this paper.

3.2 Input Source Generation

Each flow behaves as a Markov-modulated ON-OFF source and specifies a traffic profile (peak rate, token rate, and token bucket size). When the source is in ON state, the flow continuously transmits maximum size (500 bytes) packets at its peak rate. There are three kinds of flows: *conformant flows* which conform to their traffic profile, *moderate non-conformant flows* which temporarily exceed their traffic profile, and *aggressive flows* which often exceed their traffic profile a lot. The conformant flows are regulated by a leaky bucket with parameters corresponding to their traffic profile. Both the static flows and dynamic flows are tested in the simulator. The static flows start at the beginning of the simulation, and continue until the simulation finishes. In the dynamic flows, hundreds of flows come and go. This is used to test the queue selection algorithms described in section 2.

3.2.1 Static Flows

Static flows are used to test the basic properties of the FIFO, WFQ, and Hybrid algorithm. A 9-flow traffic and a 30-flow traffic are used in the simulation. The 9-flow traffic profiles are listed in Table 1.

Flow	Peak rate (Mbits/s)	Avg rate (Mbits/s)	Token bkt (Kbytes)	Token rate (Mbits/s)	Type	Average On Duration (sec)
0-2	16.0	2.0	50.0	2.0	Mod non-Conf	0.028
3-5	40.0	8.0	100.0	8.0	Mod non-Conf	0.025
6-7	40.0	4.0	50.0	0.4	Aggressive	0.051
8	40.0	16.0	50.0	2.0	Aggressive	0.052

Table 1 9-flow traffic

Flows 0 through 5 are moderate non-conformant flows. Their average rates are equal to their token rates. Their average burst sizes are conformant to their token bucket. By summing the token rate values of Table 1, it can be seen that the aggregate reserved rate is 32.8 Mb/s, or about 68% of the link capacity. On the other hand, because non-conformant flows generate substantial traffic in excess of their profile, the mean offered load is a little over 100% of the output link's capacity.
To test the ability of Hybrid algorithm under many flows, a 30-flow traffic is used (see Table 2). The average burst size of flow 0-19 is nearly equal to their token bucket size. However, for aggressive flows 20-29, the average burst size is 500Kbytes, which is much larger than their token bucket.

Flow	Peak rate (Mbits/s)	Avg rate (Mbits/s)	Token bkt (Kbytes)	Token rate (Mbits/s)	Type	Average On Duration (sec)
0-9	8.0	0.6	15.0	0.6	Mod non-Conf	0.028
10-19	24.0	2.4	30.0	2.4	Mod non-Conf	0.025
20-29	8.0	2.4	35.0	0.3	Aggressive	0.051

Table 2 30-flow traffic

3.2.2 Effect of Leaky Bucket

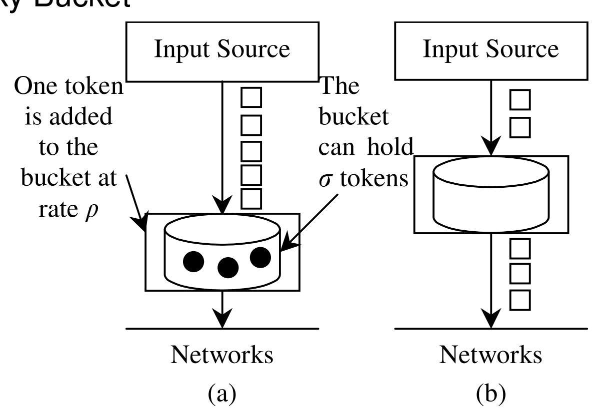


Figure 2 The leaky bucket algorithm. (a) Before (b) After

A leaky bucket, which has an infinite packet buffer and a finite token bucket, is used to regulate the conformant flows. The size of token bucket and the token rate is from the flow traffic profile. In the leaky bucket algorithm [4][3], the leaky bucket (size σ) holds tokens, generated by a clock at the rate of ρ. It allows idle input to save up permission to send large bursts later (see Figure 2).

If the burst of a flow enters the leaky bucket at peak rate κ, the flow will becomes to a burst with maximum $\frac{\sigma}{\kappa - \rho}$ time then followed by a constant flow with rate ρ (see Figure 3).

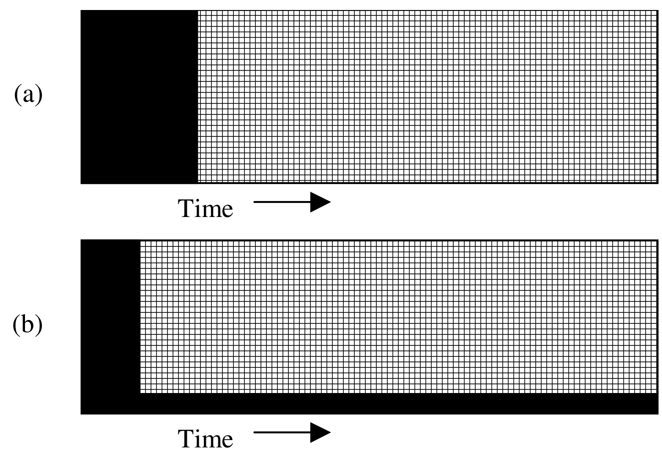


Figure 3 Effect of leaky bucket (a) Before regulation (b) After regulation

If two bursts of the input flow are too close, i.e., the second burst comes when the leaky bucket is still draining the first burst, the second burst will disappear (see Figure 4). Because the token is generated at rate ρ and tokens may loss when the token bucket is full, the maximal possible average rate of a leaky bucket (ρ, σ) is ρ. When the average rate of input source equals to the token rate ρ, the flow after regulation may be very close to CBR. If the leaky bucket algorithm is applied to the flow 0 in Table 1, about 90% packets of the regulated flow arrives in a constant rate of 2 Mbits/s. This is very different compared with the input source which is a Markov modulated ON-OFF flow. Because most packet losses in the switch buffer are resulted from bursts, the flow after regulation only has a very low packet loss even in a very small buffer. Because of this property of the conformant flows, data of the conformant flows are not presented in section 4.

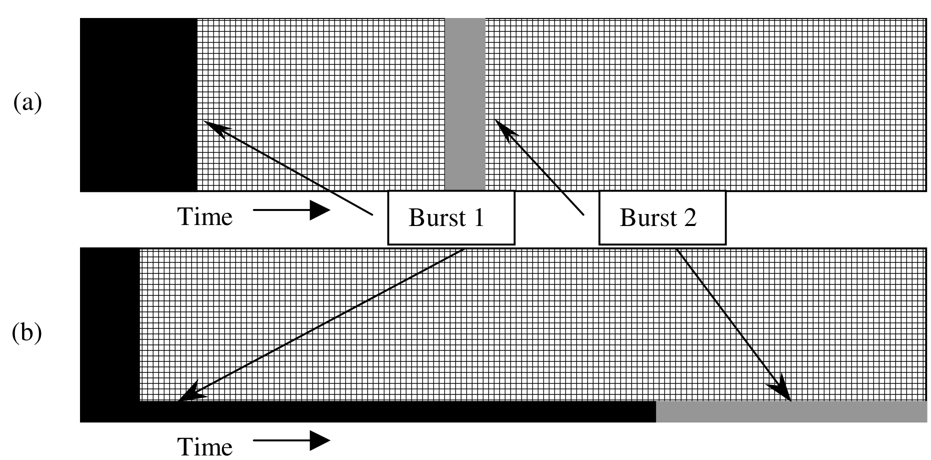


Figure 4 Burst 2 disappears after regulation

Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

3.2.3 Dynamic Flows

Flow	Type	Average On Duration (sec)	Average Off Duration (sec)
0-9	Mod non-Conf	12	0.1
10-19	Mod non-Conf	10	0.1
20-29	Aggressive	7	0.1

Table 3 Dynamic flows

In order to test the queue selection algorithms, dynamic flows are generated. The traffic of each flow is generated by a Markov modulated ON-OFF process. Every flow uses the traffic profile of the 30-flow traffic. For each dynamic flow, two more parameters are listed in Table 3.

A flow starts at the beginning of the ON duration and stops at the end of this ON duration. The ON duration and the OFF duration are exponential distributed, and the parameter of the distribution is listed in Table 3. The dynamic flows in the system are illustrated in Figure 5. Therefore, there are always about 30 flows in the switch, and the offered workload always maintains at a constant.

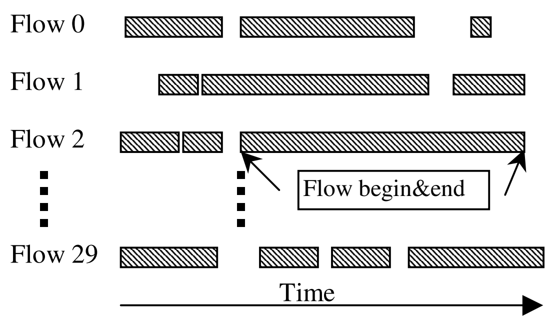


Figure 5 Dynamic flows illustration

3.3 Performance Metrics

To compare the results of different buffer management algorithms, the following performance metrics are used:

- **Throughput** includes the non-aggressive throughput and the aggregate throughput. This metric represents the link utilization.
- **Packet loss ratio** indicates the packet loss ratio of all non-aggressive flows. Because aggressive flows exceed their profiles too much, their packet losses are unavoidable. Therefore, only the loss ratio of non-aggressive flows is counted.

4 Simulation Results and Analysis

This section illustrates the simulation results of the Hybrid algorithm and their comparison to FIFO and WFQ. In section 4.1, the static flows are used to test the basic behavior of the Hybrid algorithm. Section 4.2 shows the simulation results on the dynamic flows. Section 4.3 is the analysis of all simulation results. The number of queues in all Hybrid algorithms is 3 if not specified. The traffic used is the 9-flow traffic if not specified.

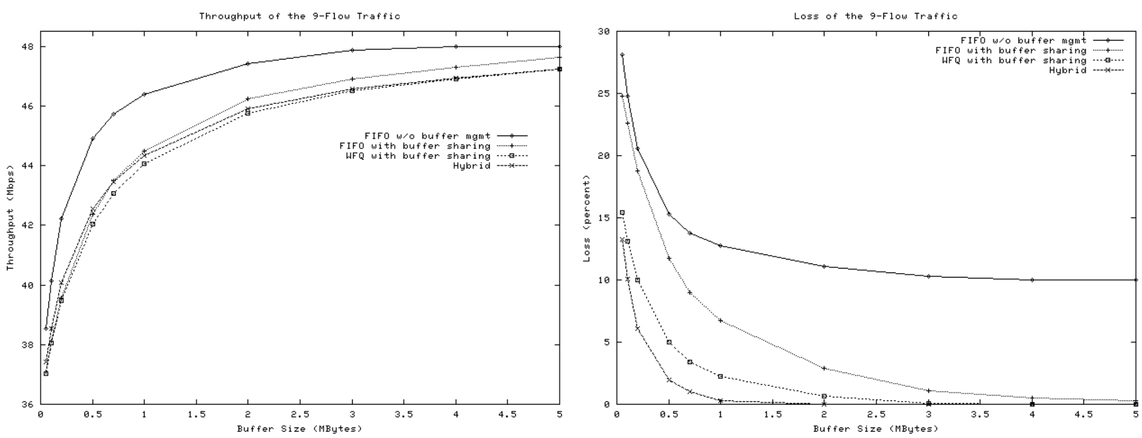
Before doing the simulation, a hypothesis is proposed here first: the following two principles are important to the performance of the throughput and packet loss ratio in the buffer management of network switches.

- **Separating aggressive flows and non-aggressive flows.** Flows can be protected by different queues. In the Hybrid algorithm, there is more than one flow on one queue, and flows inside the same queue could not be protected if one of them misbehaved. Therefore, if aggressive flows are on same the queue with non-aggressive flows, the aggressive flows will occupy too much buffer space and impact the rate guarantee of non-aggressive flows on this queue. We can only get good performance if we put aggressive flows and non-aggressive flows on different queues.
- **Putting non-aggressive flows together.** When the flows are non-aggressive, if they can share more buffer space, the packet loss will be lower. Because when a burst of one flow comes, it can utilize more buffer that belongs to the other flows share.

4.1 *Static Flows*

In this subsection, flows are assigned to different queues of the Hybrid algorithm manually for the testing of the different tradeoffs in this algorithm.

4.1.1 Performance of the Hybrid algorithm



(a) Throughput (b) Loss Ratio

Figure 6 Throughput and loss of the 9-flow traffic

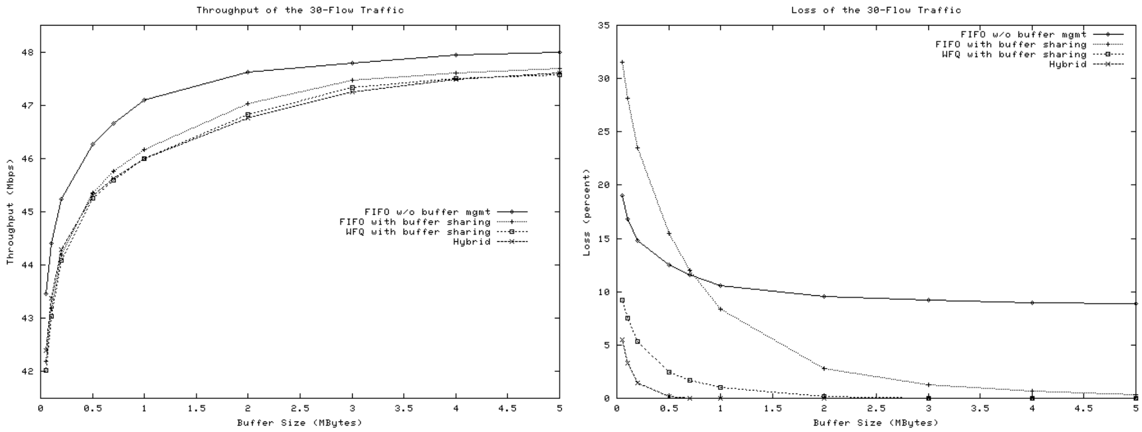
Figure 6 presents the throughputs and loss ratio of the *Hybrid* algorithm, as well as the *FIFO without buffer management*, *FIFO with buffer sharing*, and *WFQ with buffer sharing*. The 9-flow traffic is used here. There are three queues in the Hybrid algorithm: flow 0-2 is assigned to queue 0, flow 3-5 is assigned to queue 1, and the aggressive flow 6-8 is assigned to queue 2. Figure 6 shows that the Hybrid algorithm has similar throughput with WFQ and FIFO with buffer sharing. Their throughput is lower than the pure FIFO. The Hybrid algorithm has the lowest loss ratio. The pure FIFO has a much higher loss ratio.

The loss of Hybrid, WFQ and FIFO with buffer sharing is better than FIFO without buffer sharing, because they protect non-aggressive flows by putting aggressive flows and non-aggressive flows on different queues. The loss of Hybrid is better than WFQ because the non-aggressive flows in the Hybrid can share more buffer (3 flows are on one queue). Section 4.1.2 also testifies that sharing more buffer among non-aggressive flows is beneficial.

Figure 7 shows the results of the 30-flow traffic. The 30 flows are grouped into 3 queues in the Hybrid algorithm: (0-9), (10-19), and (20-29). They have similar properties as in Figure 6. The only difference is that when the buffer is very small, the loss ratio of FIFO with buffer sharing is much higher than the pure FIFO. This is because that when buffer is very small, the thresholds of

Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

all flows are very small (especially in the 30-flow traffic) compared to the bursts of all flows. Therefore, its loss ratio is very high in this situation.

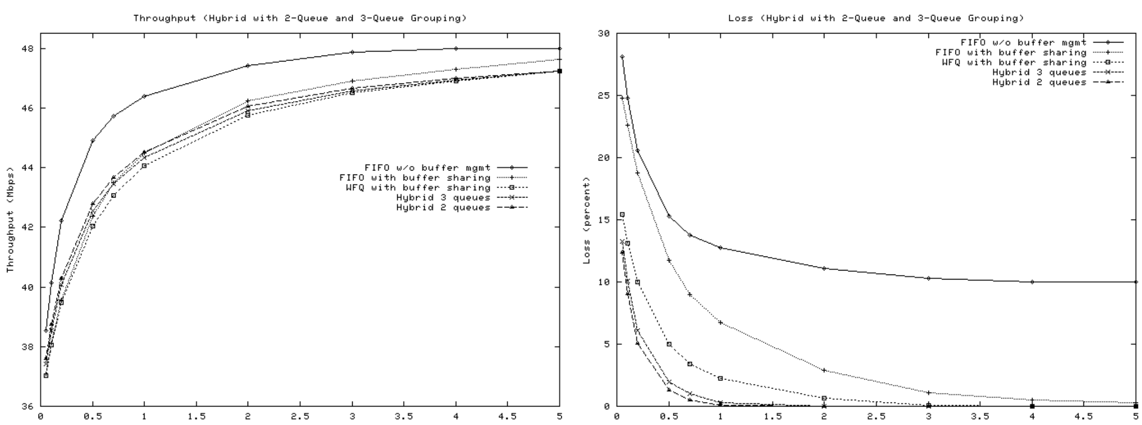


(a) Throughput (b) Loss Ratio

Figure 7 Throughput and loss of the 30-flow traffic

4.1.2 Using only two queues, one for aggressive flows, and one for non-aggressive ones.

Figure 8 shows the Hybrid algorithm with a different grouping of flows. There are two queues here. All non-aggressive flows (0-5) are on one queue, and all aggressive flows (6-8) are on another queue. The throughput does not change much. The loss ratio of the new grouping policy is lower than the original grouping policy. This is because all non-aggressive flows can share more spaces in this case.



(a) Throughput (b) Loss Ratio

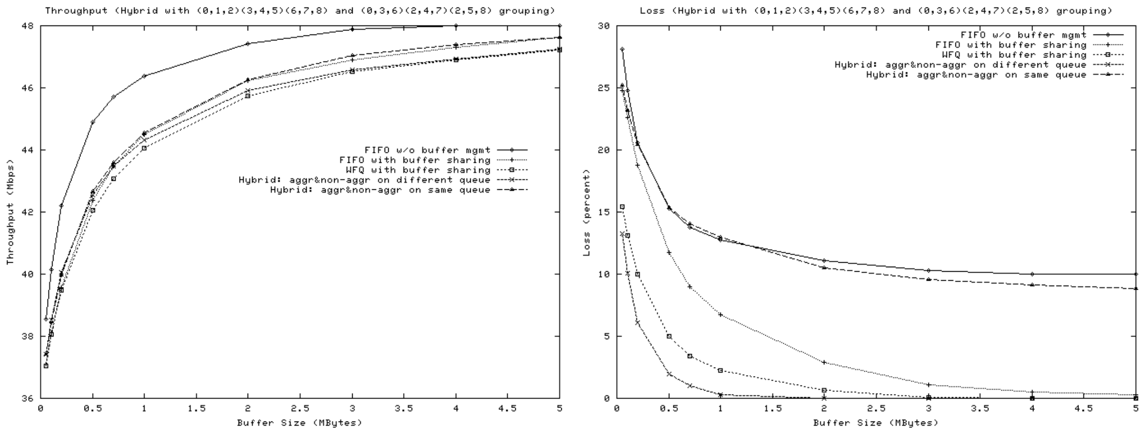
Figure 8 Throughput and loss of the 2-queue grouping

4.1.3 Putting aggressive flows and non-aggressive flows together

Now we try another kind of grouping of flows for the Hybrid algorithm. There are still three queues, and the three groups of flows are (0,3,6), (1,4,7), and (2,5,8). Because flow 6-8 is aggressive flows, aggressive flows and non-aggressive flows are on same queues.

Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

Figure 9 shows the results of this test. The new grouping policy yields a litter higher throughput, but a much higher loss ratio. This is because the non-aggressive flows are not protected in this case. They are hurt by the aggressive flows on the same queue.



(a) Throughput (b) Loss Ratio

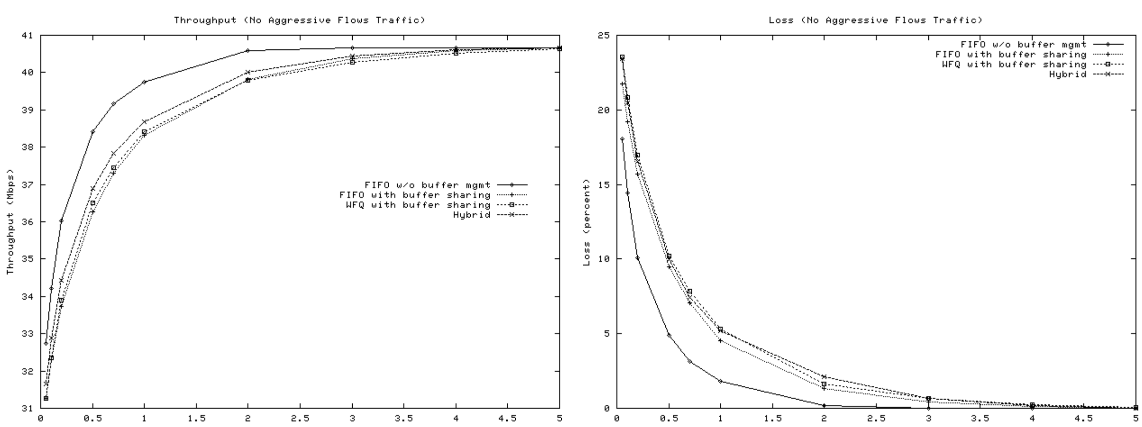
Figure 9 Aggr. And Non-Aggr. are on same queue

4.1.4 No aggressive flows

Because the aggressive flows have a big impact on performance, we change all flows to non-aggressive flows and see what will happen. This test uses the 9-flow traffic but the aggressive flows 6-8 are changed to the profile in Table 4. The flows are grouped to three queues: (0,1,2), (3,4,5), and (6,7,8).

Flow	Peak rate (Mbits/s)	Avg rate (Mbits/s)	Token bkt (Kbytes)	Token rate (Mbits/s)	Type	Average On Duration (sec)
6-7	40.0	3.0	50.0	3.0	Mod non-Conf	0.01
8	40.0	4.0	250.0	4.0	Mod non-Conf	0.05

Table 4 Non-aggressive traffic



(a) Throughput (b) Loss Ratio

Figure 10 Non-aggressive flow traffic

Figure 10 presents the results of this test. Contrary to previous data, the algorithm FIFO without buffer management now has the best throughput and the lowest loss ratio. All other three

Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

algorithms do not have big differences among their performance. Why the loss ratio of FIFO without buffer management is best in this case? In the pure FIFO, all flows share the whole buffer. In other algorithms, the buffer space can be shared by each flow is limited by their thresholds. Therefore, when there is no aggressive flows, sharing scheme is better than the partition scheme. This testifies the second principle proposed at the beginning of Section 4.

4.1.5 Share of excess throughput

Figure 11 illustrates how the link bandwidth is shared by two aggressive flows, flows 6 and 8, that differ in the amount of excess traffic they generate, with flow 8 generating substantially more excess traffic (see Table 1). For WFQ algorithm, two flows roughly share the excess bandwidth in the ratio of their reserved rates. However, the other algorithms can not achieve such sharing. Therefore, the WFQ algorithm has the best ability to share the excess bandwidth in aggressive flows.

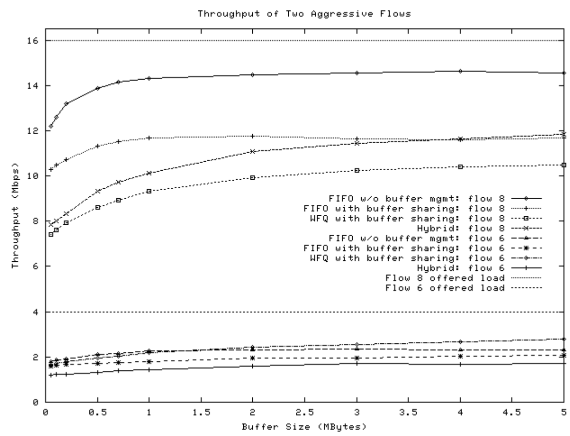
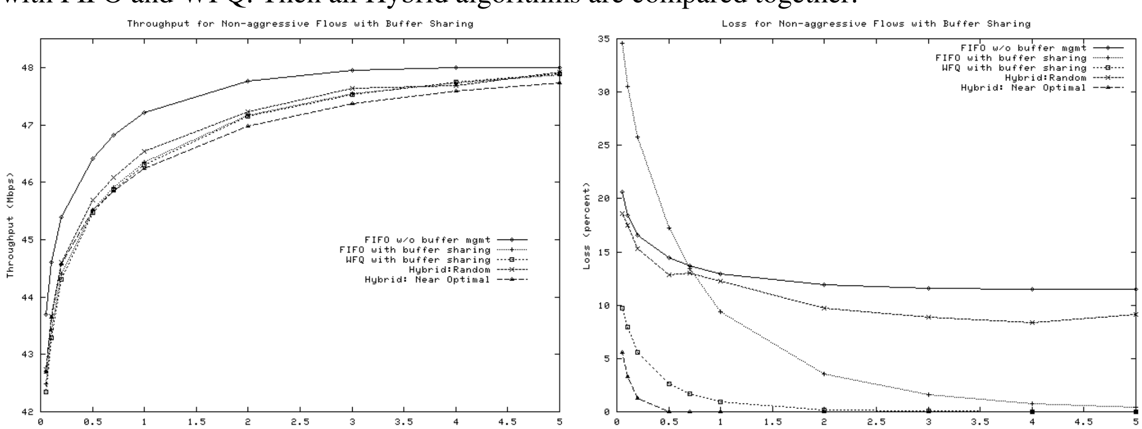


Figure 11 Throughput of two aggressive flows

4.2 Dynamic Flows

Dynamic flows are used to test the different queue selection algorithms of the Hybrid algorithm. There are two categories of queue selection algorithms: The Random and Near-Optimal are selected to represent the simple and benefit-based algorithm, respectively. They are compared with FIFO and WFQ. Then all Hybrid algorithms are compared together.



(a) Throughput (b) Loss Ratio

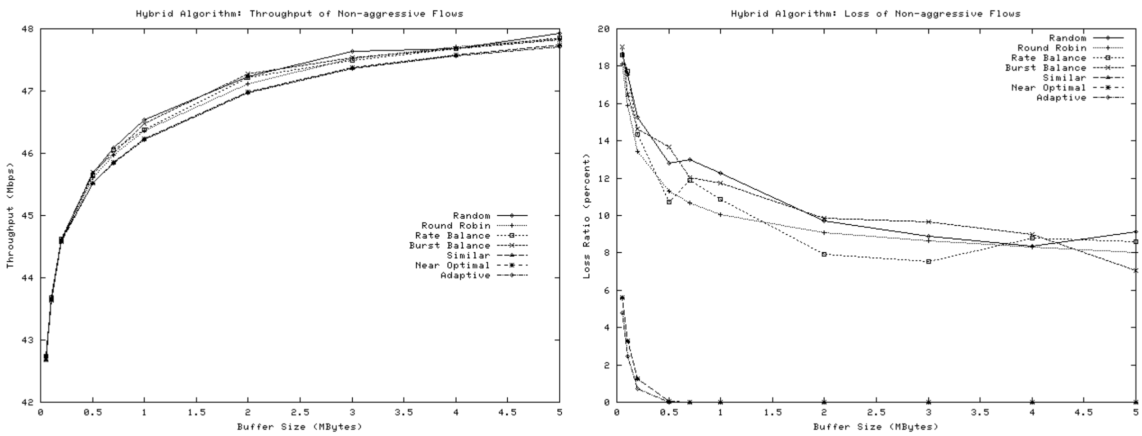
Figure 12 Performance of Hybrid, FIFO, and WFQ

Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

Figure 12 presents the throughput and loss ratio of the FIFO, WFQ, Hybrid (Random) and Hybrid (Near Optimal) algorithms. The throughput of FIFO and WFQ is similar as before. Just like the 30-flow traffic (see Figure 7), when the buffer is very small, the loss ratio of FIFO with buffer sharing is much higher than FIFO without buffer management. This figure also shows that the Hybrid (near-optimal) has a very low loss ratio, but the Hybrid (random) has a very high loss ratio. This is reasonable because the near-optimal algorithm always try to get the maximum benefit by assigning flows to a proper queue. These results also show that the benefit equation (see equation (4)) discovers some insights of the Hybrid algorithm.

4.2.1 Different queue selection algorithms

Figure 13 shows the throughput and loss ratio of all queue selection algorithms discussed in section 2. The throughput of them is similar. However, the loss ratio has a big difference between the simple and benefit-based queue selection algorithms. All simple queue selection algorithms have a very high loss ratio which is close to the FIFO without buffer management. All benefit-based algorithms have a very low loss ratio which is better than WFQ with buffer sharing.



(a) Throughput (b) Loss Ratio

Figure 13 Performance of Hybrid algorithms

4.2.2 Another dynamic flows

In the above test, all aggressive flows have same traffic profiles which are different from all non-aggressive flows. This is not a realistic situation. The Hybrid (similar) and Hybrid (near-optimal) only consider the traffic profile, not include the aggressiveness of flows. Therefore, the reason that they get low loss ratio may be that the input source is so special for them. To test this, new dynamic flows are defined in Table 5. Similar to the original dynamic traffic, there are 30 flows. The difference is that 15 of them are aggressive and have same traffic profile parameters as the other 15 non-aggressive flows. Because of the nature of Hybrid (similar) and Hybrid (near-optimal), they can not distinguish aggressive flows and non-aggressive flows. Therefore, they should perform badly in this case. The Hybrid (adaptive) should manage this situation well.

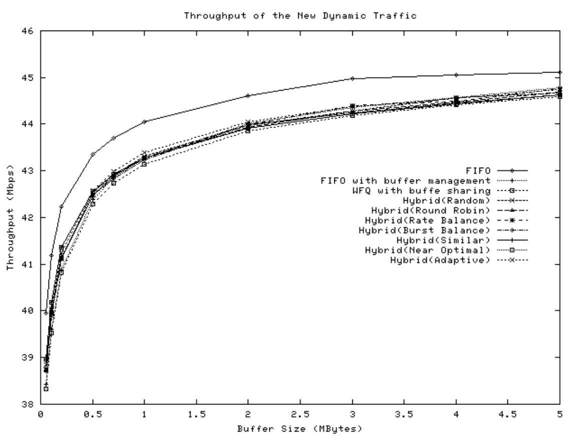
Figure 14 shows the results of this new dynamic traffic. The aggregate throughput does not change much in this new traffic. However, Figure 14 (b) shows that the loss of the Hybrid (random) becomes worse than the pure FIFO. The loss of the Hybrid (adaptive) becomes much better than before. Figure 14 (e) illustrates loss ratio of all hybrid queue selection algorithms. Although the Hybrid (similar) is identical to the Hybrid (near-optimal) which is much better than the simple algorithms, the Hybrid (adaptive) gets a much lower loss here. This is because that in the adaptive algorithm, aggressive flows and non-aggressive flows are put on different queues.

Wayne (Wenguang) Wang
Email: wew036@cs.usask.ca

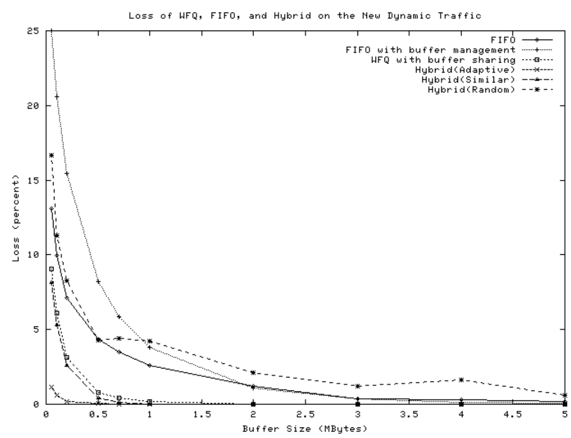
Therefore, not like the Hybrid (similar) or Hybrid (near-optimal), possible interference between them is avoided.

Flow	Peak rate (Mbits/s)	Avg rate (Mbits/s)	Tkn bkt (Kbytes)	Token rate (Mbits/s)	Type	Average On (sec)
0-4	8.0	0.6	15.0	0.6	Conformant	0.015
5-9	8.0	1.2	15.0	0.6	Aggressive	0.015
10-14	24.0	2.4	30.0	2.4	Mod non-Conf	0.011
15-19	24.0	3.6	30.0	2.4	Aggressive	0.011
20-24	8.0	0.3	35.0	0.3	Conformant	0.052
25-29	8.0	2.4	35.0	0.3	Aggressive	0.052

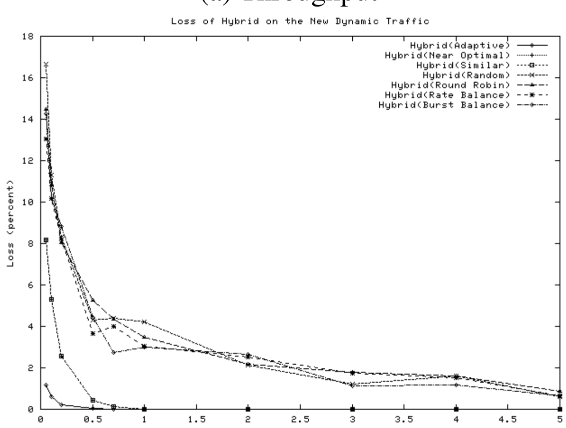
Table 5 New Dynamic Traffic



(a) Throughput



(b) Loss Ratio of WFQ, FIFO and Hybrid



(c) Loss Ratio of the Hybrid Algorithm

Figure 14 Performance of Hybrid algorithms on the new dynamic traffic

4.3 Behavior Analysis

All experiments testify the hypothesis proposed at the beginning of section 4. Because the simple queue selection algorithms do not consider the aggressiveness, nor the benefit of the flow assignment, it has high loss ratio. Therefore, they are not practical. The near-optimal and similar algorithm works well when they happen to put aggressive flows and non-aggressive flows on different queues on some special workloads. However, they can not distinguish the aggressive

flows with the non-aggressive flows only by traffic profiles. Therefore, they can not perform well in some situations.
The best algorithm is the adaptive queue selection algorithm. It considers both the aggressiveness and the benefit equation. By obeying the two principals discussed above, it will perform well in more general workloads.

5 Conclusion and Future Work

In this paper, the Hybrid algorithm which is a hybrid of the FIFO and WFQ algorithm is discussed. This algorithm is a scalable architecture for QoS provisioning that integrates the buffer sharing scheme with WFQ scheduling that uses a small number of queues.

Various queue selection algorithms of the Hybrid algorithm are proposed. The simulation results show that the *adaptive* algorithm is the best one under various workloads.

A hypothesis about two important design factors of the network switch buffer management schemes are proposed and testified: aggressive flows and non-aggressive flows should be put on different queues; and non-aggressive flows should be put together to maximize the buffer space available for sharing.

The limitation of this paper is that the input source is not the real traces. The Markov Modulated ON-OFF sources used in the paper are quite different from the real traces because the workload of network normally shows self-similarity. However, the simulator used for this paper can be very easily changed to use the real trace.

Another limitation is that the metrics used in this paper is very simple. They only include the total throughput and the total packet loss of all non-aggressive flows. To achieve the fair access to the link bandwidth, other metrics should be used. Moreover, when more metrics are applied, some conclusion presented in this paper may change.

The future works include: testing the conclusions of this paper by trace driven simulation; using more metrics to test the fairness of the Hybrid algorithm.

References

[1] A. Demers, S. Keshav, and S. Shenker. *Analysis and Simulation of a Fair Queueing Algorithm*. In *Proceedings of Symposium on Communication Architectures and Protocols.* ACM SIGCOMM, September 1989.
[2] R. Guérin, S. Kamat, V. Peris, and R. Rajan. *Scalable QoS Provision Through Buffer Management*. SIGCOMM'98, Vancouver, B.C.
[3] Raif O. Onvural. *Asynchronous Transfer Mode Networks*. Second Edition. Artech House. 1995.
[4] Andrew S. Tanenbaum, *Computer Networks*. Third edition. Prentice hall. 1996.
[5] Lixia Zhang. VirtualClock: A New Traffic Control Algorithm for packet Switching Networking. In Proceedings of Symposium on Communication Architectures and Protocols. ACM SIGCOMM, September 1990.