# On Fair Scheduling in Heterogeneous Link Aggregated Services

Satya R. Mohanty and Laxmi N. Bhuyan
Department of Computer Science and Engineering
University of California, Riverside, CA 92521
satya,bhuyan@cs.ucr.edu

***Abstract*— Provisioning Quality of Service (QoS) across an aggregate of transmission entities (e.g. link aggregation) or processing elements (e.g. network processors) is a challenging problem. The difficulty lies in simultaneously satisfying fairness to flows (with different bandwidth requirements) and ensuring minimized intra-flow reordering. This problem is crucial to many applications (that utilize parallel communication or processing paths) like multi-path load distribution, multi-path storage I/O, web service, data processing by network processors in the data-path of routers, trans-coding multimedia flow traffic content over the Internet to name a few. We present two algorithms for multi-link systems that aim at reducing undesired reordering, ensure fair sharing of flows and optimal utilization of the links. Our algorithms are based on a new approach of dynamically partitioning flows among links. We perform simulations on real Internet traces to validate our algorithmic approach.**

## I. Introduction

Link aggregation increases transmission bandwidth by simultaneously introducing multiple physical links to the same destination that together carry the aggregate traffic. The advantages of this method in addition to the linear scaling of bandwidth between the two end devices are increased reliability and load balancing. Implementations in the market place include the Cisco Etherchannel in the Cisco ONS 1500 Series based on the proprietary Inter-Switch trunking (ISL), Adaptec's Duralink port aggregation, 3COM, Bay Networks, Extreme Networks, Hewlett Packard and Sun, to name a few.

Implementing Quality of Service ("Fairness") requires that communication traffic be serviced selectively so that customers who pay more, or customers with critical requirements receive better service. This is usually done by dividing the traffic into disjoint *flows* (a flow can be thought of as a unique source-destination "conversation"; its packets are always stored in a logically distinct per-conversation queue at a router), and treating them in accordance with their reserved rates of service. Fairness ensures immunity from misbehaving flows, thereby resulting in effective congestion control and rate-adaptive applications. Stringent Quality of Service (QoS) requirements also lead to strict delay bounds and better throughput by selective admission control.

This paper addresses the quality of service issue for multi-link systems. Nevertheless an identical situation resides in the Network Processor NP context. Several microengines process packets simultaneously with a main core performing the scheduling or allocation policy to individual microengines. Thus, although our algorithms are in the link context, they are analogous to the NP scheduling area. In fact if the time for packet transmission on a link is replaced with the estimated time of processing in an NP, the two problems are *one and the same*. References [13], [8] relate to the problem of scheduling the workload across multiple processing engines of a network processor. While [8] assumes that the traffic is always backlogged, [13] assumes that the workload is known. A recent work [11] addresses the same problem but obviates the need for a known workload by using an on-line estimator with less complexity.

Recent research in Fair Queuing [1], [5], [7], [10], [12] has addressed the QoS issue in the *single* link context i.e. scheduling competing flows over a *single* link in accordance with their reserved rates of service. QoS scheduling over a multi-link system does not follow trivially from the single link scheduling case. An issue that was non-existent in the single link case manifests prominently in a multi-link setting.

Reordering of packets is likely whenever a flow is concurrently serviced and may actually increase when link rates vary. However, enforcing strict order may cause the multi-link system utilization and thus the throughput to drop substantially. On the other hand, it is also well known that, reordering of packets has direct bearing on the transmission control protocol (TCP). TCP is designed to handle reordering, however this involves additional processing at the TCP end points. Reordering has also been proved to be detrimental to TCP performance. Blanton and Allman [3] state that reordering causes performance problems for TCP's fast retransmission algorithm, which uses the arrival of duplicate acknowledgments to detect segment loss. TCP for Persistent Reordering [4] (TCP-PR) is a variant of TCP that attempts to improve TCP performance in the presence of persistent reordering phenomena. But this is a software solution, feasible only at the end-hosts. Hence imposition of QoS over a heterogeneous multi-link system deserves consideration of an additional factor over that of the single link case.

The problem of provisioning QoS among competing flows over a system of links was addressed by Blanquer and Özden [2]. Their algorithm, Multi Server Fair Queueing (MSFQ) is based on the Generalized Processor Sharing (GPS) [10] system, an idealized service discipline, that is also representative of a perfectly fair system. MSFQ simply selects the next packet that would leave in the associated GPS

system and sends it over the next available link (algorithm PGPS [10]). Although MSFQ is defined for the case when all links are homogeneous (all links operate at the same rate), the algorithm itself may be applied to a system of heterogeneous links (individual links operate at different rates).

Cobb and Lin [6] propose a general technique of constructing a multi-link scheduling algorithm from a single link one. They also consider several sorting techniques to avoid packet reordering that require access to upper layer protocol headers and thus potentially incur significant overhead.

*Our Contributions*

We take a completely different approach from the work cited above. We present two algorithms, each of which reduce the reordering problem while still providing a good approximation of GPS by assigning the output links to flows. Whenever a link is idle and packets are outstanding to be scheduled, only head of line (HOL)packets of flows that are assigned on that link are eligible to be scheduled. Once this assignment is done any Fair Queuing algorithm for a *single* link system can work in conjunction to determine the final order in which the packets should be sent.

We perform various simulation tests and evaluate packet reordering using several different metrics. The experiments show that our algorithms outperform MSFQ no matter which metric is used to quantify packet reordering. In some cases, even for homogeneous systems, our algorithms outperform MSFQ drastically.

## II. Background

### A. Generalized Processor Sharing

If $\phi_i$ is a reservation assigned to flow $i$ and $B(t)$ denotes the set of backlogged flows at time $t$, then $\phi_i(t) = \phi_i / \sum_{j \in B(t)} \phi_j$, is the fraction of the total available bandwidth that flow $i$ requires at time $t$. GPS is a work conserving service discipline that can be described as follows: Let $W_i(t, t+\tau)$ denote the number of bits transmitted by the GPS system from flow $i$ in time interval $[t, t+\tau]$. Then the GPS always ensures that for any two flows $i, j$

$$\frac{W_i(t, t+\tau)}{W_j(t, t+\tau)} \geq \frac{\phi_i(t)}{\phi_j(t)} \tag{1}$$

holds true for a time interval $[t, t+\tau]$ during which the set of backlogged flows does not change. This is possible only if we assume that packets are infinitely divisible. In reality it is not possible to keep the inequality (1) satisfied at all times, so an effort is made to approximate the GPS system. In the single link case, this was achieved by simulating the GPS system, and transmitting the packets in the order they would depart in the GPS system.

In the multi-link case, any scheduling algorithm not only needs to make the decision of which packet to transmit next, but also which link should be used to transmit the packet. The following approach was taken by Blanquer and Özden [2]. Let $N$ be the number of links and let $R$ be the individual link rate.

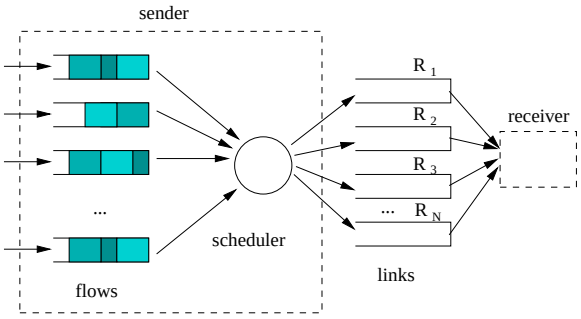


Fig. 1. System model

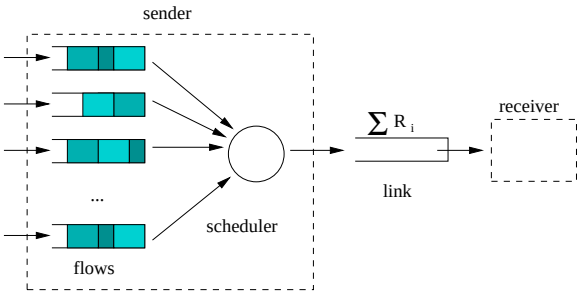


Fig. 2. Aggregated system

(In [2], it is assumed that $R_j = R$, a constant, for every link $j$.) See Figure 1. Then MSFQ schedules the next packet that would depart in the reference GPS system operating at rate $NR$ (see Figure 2) on the next available link. Such a reference system is often referred to as (GPS, 1, $NR$), denoting a single link, operating at a rate $NR$ serviced by the GPS discipline. The MSFQ algorithm is denoted hereby as (MSFQ, $N$, $R$).

Two things should be noted at this point. First, the order of departure times of packets in MSFQ and in the reference GPS system is not always the same, since packets may arrive while all MSFQ links are busy. These packets may start receiving service in GPS immediately and may depart even before MSFQ begins their transmission. This is a well understood phenomenon and exists even in the single link case. Second, in the multi-link case packets that start receiving service later in time can depart *before* packets that started receiving service earlier. This happens because of concurrent service of different length packets of the same flow on different links. This problem is prominent in the scenario when the rates of the links vary.

### B. Notation

We provide some notation here. Let $N$ be the number of output links in the system. Link $i$ operates at rate $R_i$. By $R_{max}$ and $R_{min}$ we denote the rates of the fastest and slowest links respectively. The aggregate rate of the multi-link system is therefore $R_{ag} = \sum_{i=1}^{N} R_i$. This is the rate at which the reference GPS link operates for MSFQ. When all the links have the same rate $R$, then $R_{ag}$ is simply $NR$. By $L_k$ we denote the length of packet $k$. By $L_{max}$ and $L_{min}$ we denote

the maximum and the minimum packet length respectively. By $n$ we denote the number of backlogged flows at a given time.

### C. Packet reordering

Quantifying packet reordering is not easy, as different protocols may be able to tolerate sequences with quite different reorder degree and type. Consider for example sequences (2, 3, 4, 5, 6, 7, 8, 1) and (2, 1, 4, 3, 6, 5, 8, 7). In the first one, only one packet is "out-of-order". However, if an application needs to read the packets in the original order to process the whole sequence, the second sequence seems to have a lower degree of reordering (only two consecutive numbers need to be read to restore the original sequence.)

We therefore need a proper definition of reordering and relevant performance metrics. In this paper, we adopt the definitions of packet reordering from the IETF draft by Marton *et al.* [9]. The reordering measures stated there, apart from defining reorder and quantifying its degree also have relevance to receiver design that can impact TCP and Real-time application performance. In the following, without loss of generality, we assume that only a single flow is incident. Let $s_i$ be the original sequence number of the $i$'th packet received by the destination. The following code determines if a packet is received in order. It uses a variable $NextExp$ which is initially set to the index number of the first received packet of that flow.

**Algorithm II.1:** ON PACKET ARRIVAL($s$)

**if** $(s_i \geq NextExp)$ **then**
  //packet with seq. num $s_i$ is in-order
  $NextExp$++;
**else**
  //packet with seq. num $s_i$ is reordered

We now briefly present the metrics.

*Reordered ratio.* Reordered ratio is simply the ratio of packets that were received out of order to the total number of packets.

*Reordering extent.* Reordering extent of packet $s_i$ is defined to be $i - j$ for the smallest value of $j$ such that $s_j > s_i$. The reordering extent is a rough estimate on the size of the buffer needed to restore the original order.

*Reorder free run.* Reorder free run is the count of packets that arrived in order, since the last reordered packet.

## III. BANDWIDTH PARTITIONING ALGORITHM

MSFQ chooses the packet with smallest virtual time to transmit on the link. This decision needs to be done more carefully in the heterogeneous system, as transmitting packets with high reservations on slow links (or, conversely, transmitting packets with low reservations on fast links) may result in local unfairness and high packet reordering rate. Consider the following example:

**Example:** Fix a flow $i$ and let $p_k$ be a maximum length packet of this flow. Suppose that the next packet to leave the system is $p_k$ and the only available link is the one with the lowest rate. If packet $p_k$ is scheduled on this link, then after time $\delta, (\delta \approx 0)$, all the other links may become available and a large number of packets of flow $i$ with minimum length may arrive. If the GPS finishing times of all these packets are smaller than the GPS finishing time of all other packets in the system then potentially $(N-1)\left\lfloor \frac{R_{max}L_{max}}{R_{min}L_{min}} \right\rfloor$ packets of flow $i$, with sequence numbers greater than $p_k$ can leave the system completely before $p_k$ does.

It is obvious that MSFQ does not take into account a flow's individual reservation or how many links it is spanned across at any given moment in time. We build on this intuition and aim at a corresponding match between flows and links such that an individual flow gets fair service and is minimally spread across links within the limits of discrete packet transmission. The match or partition is the key feature of our algorithm.

### A. Partitions and Partitioning

We discuss *partitions* in this section. A *partition* of a flow on a link is an assignment of some fraction of the link's bandwidth to that flow. The idea is to have the partitions proportional to the backlogged reservations.

*Partitions:* For a fixed time step $t$, normalize the reservations, by setting

$$\overline{\phi}_i(t) = \phi_i(t) R_{ag}$$

where $R_{ag}$ is defined as the aggregate bandwidth of all the links. A *partition* is denoted by $\left(\psi_i^j\right)_{1..n}^{1..N}$, where $\psi_i^j$ is simply the fraction of flow $i$ that is assigned to link $j$.

We say, that a partition is *canonical*, if it satisfies the following conditions:

(C1) $\sum_{j=1}^{N} \psi_i^j = \overline{\phi}_i(t)$, (total bandwidth assigned to flow $i$ is equal to its normalized reservation)
(C2) $\sum_{i=1}^{n} \psi_i^j = R_j$ (total bandwidth of flows assigned to link $j$ does not exceed its rate)

Let $\mathcal{P}_j = \{i : \psi_i^j > 0\}$ be the set of flows assigned to link $j$ and let $\mathcal{Q}_i = \{j : i \in \mathcal{P}_j\}$ be the set of links on which flow $i$ can be scheduled. We leave the question of how to compute appropriate partitions to the subsequent section. Before that we describe some desirable properties of any partition.

The algorithms use two main components: (1) a component to compute *canonical* partitions and sets $(\mathcal{P}_j)_{j=1,\ldots,N}$, (2) a Fair Queuing FQ algorithm for *single link* system.

The algorithm FQ is used to determine *the order* in which the packets would leave the system $(\mathsf{FQ}, 1, R_{ag})$. This order is used to schedule the packets in the main algorithm in the way described below.

When an output link $j$ becomes idle, the set $\mathcal{P}_j$ is computed. The packet that is transmitted over link $j$ is the HOL packet of a flow in $\mathcal{P}_j$ that would depart the $(\mathsf{FQ}, 1, R_{ag})$ system first. In other words the algorithm FQ is used to assign a departure sequence number to packets for the pending flows, and the main algorithm chooses the one with the minimum such sequence number among packets belonging to flows $\mathcal{P}_j$. Note that once the sets $\mathcal{P}_j$ are computed, the main algorithm does not base its decision of which packet to send next on

the values $\psi_i^j$ but, rather, on the departure sequence number obtained from FQ only.

We would like to show that MSFQ is also canonical. Indeed, it is enough to consider partition $\psi_i^j = \overline{\phi}_i(t) R_j / R_{ag}$ and set FQ = PGPS. We obtain $\mathcal{P}_j = \{1, 2, \ldots, N\}$ for any $j$, so the next packet to leave the system GPS, $1, R_{ag}$ is always send over the next available link. It is worth noticing that the partition defined in this way is canonical. Indeed,

$$\begin{aligned} \sum_{j=1}^{N} \psi_i^j &= \overline{\phi}_i(t) \sum_{j=1}^{N} R_j / R_{ag} \\ &= \overline{\phi}_i(t) \\ \sum_{i=1}^{n} \psi_i^j &= R_j / R_{ag} \sum_{i=1}^{n} \overline{\phi}_i(t) \\ &= R_j \end{aligned}$$

The time complexity of the algorithms depends on the complexity of the algorithms used to compute the partition and the algorithm FQ. It, however, is at least $O(\log n)$ per packet, as at each step a packet with minimum sequence number needs to be found.

### *B. Partition Detection Algorithm*

Partitioning is invoked at certain epochs. These epochs coincide with the scheduling time of some packet that just starts to receive service in the system. The condition for partitioning is simple: Whenever a link is idle, a check is done to determine if the set of flows has changed since the last partitioning instance. If the flow set changes, then the partitioning needs to be computed. If $t_1$ and $t_2$ are two consecutive packet departure times in the aggregated link system and $n_{t_1}$ and $n_{t_2}$ are the corresponding backlogged flow sets, a naive algorithm that performs the partitioning check through direct sort and compare is of complexity $O(n' \log n')$, where $n' = max(n_{t_1}, n_{t_2})$. This is clearly prohibitive at high speed links since this check needs to be done at every packet scheduling time. We propose a simple yet efficient algorithm to solve this problem. The algorithm makes use of two queues, a *Vanish and* Arrival Queue, denoted by $VQ$ and $AQ$, and whose sizes are $N$ and 1 respectively. When a new flow arrives, the algorithm first checks if the flow is already present in $VQ$. If the flow is present, its entry is removed from $VQ$ (this means the flow only disappeared temporarily). If the flow did not appear in $VQ$, and $AQ$ is empty, an entry of the flow is made in $AQ$. Otherwise no bookkeeping is done in the detection algorithm. The Pseudo-code is presented in Figure 3. Let the *migration degree* of flow $i$ be $|\mathcal{Q}_i|$. We will say that a flow *migrates* if its migration degree is greater than 1. Reordering is caused by migrations. The larger migration degree, the more likely it is for the reordering to take place. The second factor that influences the degree of reordering is the difference in link rates between which a flow migrates.

Upon arrival of a packet $P_k$ of flow $f$ s.t no packets of flow $f$ except those being transmitted are backlogged:
$ToPartition = FALSE$

- If $((ToPartition == FALSE) \wedge (f \in VQ))$
  Enqueue $P_k$ onto $FlowQ[f]$
  Delete flow $f$ from $VQ$
- If $FlowQ[f] == \Phi$
  Enqueue $P_k$ onto $FlowQ[f]$
  If $(ToPartition == FALSE)$
  $ToPartition := TRUE$
  Enqueue $f$ onto $AQ$

When a link is free

- If $ToPartition == TRUE$
  Do Partition
- If scheduled packet causes flow $f$ to disappear
  Enqueue flow $f$ onto $VQ$.

Fig. 3. Algorithm 1. Partition Detection Algorithm

### *C. Algorithms for computing partitions*

Algorithm BestFit:

The BestFit algorithm tries to localize backlogged (here backlogged means flows which have packets pending to be transmitted in real time, not in GPS) flows with high reservations to the faster links (with high bandwidth), as these flows are most likely to migrate. BestFit proceeds by recursively mapping the flow with the largest remaining reservation to the link with the largest remaining bandwidth. At initiation of the mapping process, each link's share is its nominal rate and each backlogged flow's share is its instantaneous reservation.

In the following $F_i$ is an unmapped portions of flow $i$ and $T_j$ is the remaining bandwidth of link $j$. The algorithm simply takes the flow with the largest unmapped portion and assigns it to the link with the largest unmapped bandwidth. The excess bandwidth of either the link or the flow is their available bandwidth for the next iteration. The canonical property ensures that that once the BestFit algorithm terminates, no excess bandwidth is unmapped.

**Algorithm III.1:** BESTFIT($\Phi, R$)

**for** $i \leftarrow 1$ **to** $n$, $j \leftarrow 1$ **to** $N$ **do**
  $F_i \leftarrow \Phi_i(t)$; $T_j \leftarrow R_j$; $\psi_i^j \leftarrow 0$

**while** $(\sum_i F_i > 0)$ **do**
  $i \leftarrow \text{argmax}(F_i)$
  $j \leftarrow \text{argmax}(T_j)$
  $\psi_i^j = \psi_i^j + \min(F_i, T_j)$
  $F_i = \max(F_i - T_j, 0)$
  $T_j = \max(T_j - F_i, 0)$

The last two steps are performed in parallel.

Algorithm FirstFit:

The second algorithm, FirstFit uses similar approach, but is faster since it does not sort the flows according to the reservations. At initiation both the links and flows are arranged in descending order of bandwidth rates and reservations. The

flow with the highest reservation is mapped to the link with the highest bandwidth. Any surplus of either is mapped to the corresponding next of the other entity. As can be seen the canonical property assures that no excess bandwidth remains unmapped after the mapping process.

**Algorithm III.2:** FIRSTFIT($\Phi, R$)

**for** $i \leftarrow 1$ **to** $n$, $j \leftarrow 1$ **to** $N$ **do**
$F_i \leftarrow \Phi_i(t)$; $T_j \leftarrow R_j$; $\psi_i^j \leftarrow 0$

**for** $i \leftarrow 1$ **to** $n$ **do**
**while** $(\sum_i F_i > 0)$ **do**
$j \leftarrow \text{argmax}(T_j)$
$\psi_i^j = \psi_i^j + \min(F_i, T_j)$
$F_i = \max(F_i - T_j, 0)$
$T_j = \max(T_j - F_i, 0)$

As in case of BestFit, the last two steps are performed in parallel.

Time complexity:
In our case FQ= PGPS and the partition is calculated in time $O(N \log N + n \log n)$, so the time complexity of our algorithms is $O(N \log N + n \log n)$.

## IV. SIMULATIONS

### A. Simulation and Results

*Simulation settings:* We simulated a $4$ link system with individual rates equal to $R_1 = 0.025, R_2 = 0.075, R_3 = 0.15$ and $R_4 = 0.25MB$ respectively. Flows were grouped into four classes; the assigned reservations of these classes were in the ratio $10 : 20 : 40 : 80$.

We used traces from the NLANR Auckland-II link. To simulate buffering we merged several (actually 6) such traces, into one monolithic dense trace and decreased the throughput of the link by half. After filtering out flows with a low number of packets we obtained approximately 5000 flows. Since the traces do not carry any QoS information we generated additional information in two different ways. In the first case, flows were assigned to classes based on their size (this approach is referred to as "by size" later in the text and in the figures). This classification was motivated by the fact that QoS streams with high priority (i.e. video or audio stream) usually carry a large amount of traffic. To perform the classification we ordered the flows by increasing size and classified the first $\frac{2}{3}$ into the $1st$ class. The remaining flows were classified into classes $2, 3$ recursively; class $4$ contained the remaining flows after the assignment to classes $1, 2, 3$ was done. In the second case, flows were classified to classes randomly with the probability of a flow belonging to the lowest or highest classes low (the exact probabilities were $\frac{1}{7}$, $\frac{3}{7}$, $\frac{2}{7}$, $\frac{1}{7}$ for classes $1, 2, 3, 4$ respectively). This approach takes into account the "economical" aspects, with the assumptions being that only few users will be able to afford the high priority class. We think that the problem of classifying the flows into QoS classes is very important, and deserves much consideration on its own. We initially tried to assign QoS values (reservations) to flows based on the the source and destination port numbers (i.e. the original protocol used in the Internet trace). Accordingly, we took into account flows using port numbers between $1$ and $1024$, and classified the flows into $4$ different categories. Class $1$, consisted of low priority flows (like SMTP), Class $2$ contained file transfer protocols like FTP, SAMBA, MS data sharing, rsync etc. Class $3$ included protocols like DNS, HTTP, IMAP etc. and Class $4$ consisted of the highest priority protocols that require hard real-time response like SSH, TELNET, and RTSP for streaming media. After performing this classification we filtered out flows which contained less than $100$ packets. We did so, because these flows greatly perturb the reorder ratio metric (for example, if in a flow with two packets only one packet gets reordered, the reorder ratio reaches $50\%$, which isn't very meaningful in the scope of the full simulation.) For each of the above classification schemes, we divided the simulation further into two parts. In the first one packets maintained their original sizes. In the second one, the size of each packet was generated randomly from a uniform distribution over $[1, 1500]$. It can be observed that in the real Internet traces packets belonging to a single flow very frequently have the same size. The set of experiments with random packet lengths was to show how important this observation is.

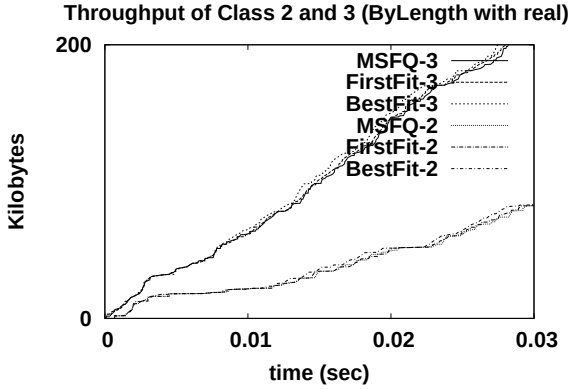


Fig. 4. Service Curve of Flows 2 and 3 with Classification "by size real"

*Simulation results:* The simulation results are presented in terms of the service per reservation ("throughput") and the reordering metrics. Since the simulations were performed for four different flow classifications and four scheduling algorithms (GPS, MSFQ, BestFit and FirstFit), only the results pertaining to certain flows and reordering information is shown. Figure 4 shows the service curve for classes $2$ and $3$ when the trace is generated according to the "by size with real packet length" criterion (In Fig. 4 MSFQ-3 denotes the service curve of class $3$ under MSFQ scheduling policy and similarly). Similarly, Figures 5 and 6 show the service curves for classes $3$ and $4$ respectively by each of the three (for clarity the GPS

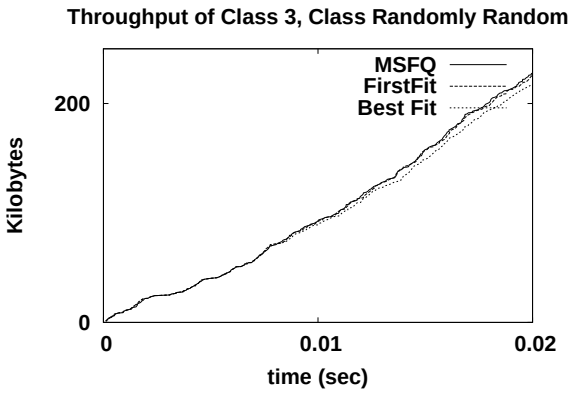


Fig. 5. Service Curve of class 3 with Classification "Randomly random"

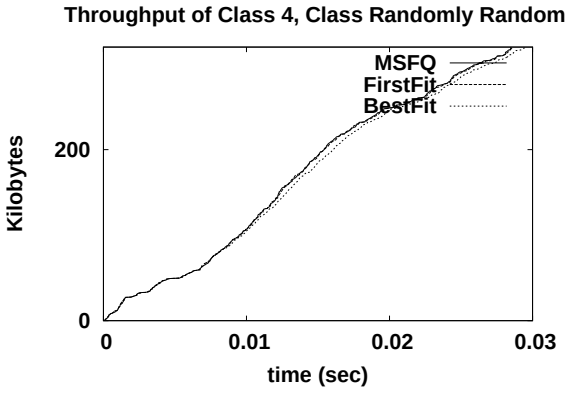


Fig. 6. Service Curve for Class 4 with Classification "Randomly random"

is excluded) algorithms with traffic classification "randomly with random packet length". Note how closely the partitioning algorithms FirstFit and BestFit approximate MSFQ in terms of service. Simulation results show that the packet delay and work per flow of BestFitand FirstFit are similar to MSFQ. However, it is important to state, and we do not show for purposes of brevity, that a tight bound does not exist for the partitioning algorithms unlike that as in PGPS. This will happen when a particular flow will frequently be mapped to different links and is a rarity and the proof is outside the scope of this paper.

Figure 7-10 and Table I summarizes the reordering information. Figure 7-10 shows the top 100 flows (and thus the class) with the maximum reorder ratio for the different types of experiments (two types of QoS classification and two types of packet lengths.) The results show that the packets from the highest class experience the highest reordering. The simulations also show that both BestFit and FirstFit give superior performance as far as reordering is concerned. In fact, in all the scenarios, the number of flows that undergo reordering is significantly higher in case of MSFQ than in case of BestFit or FirstFit, often, an order of magnitude more. The experiments also show that introducing randomness (either though a random QoS classification, or random packet lengths) increases the reordering greatly. The reordering free run data sets are not presented because of brevity reasons, but from the simulations it is found that they are higher for the partitioning algorithms as compared to corresponding values for MSFQ.

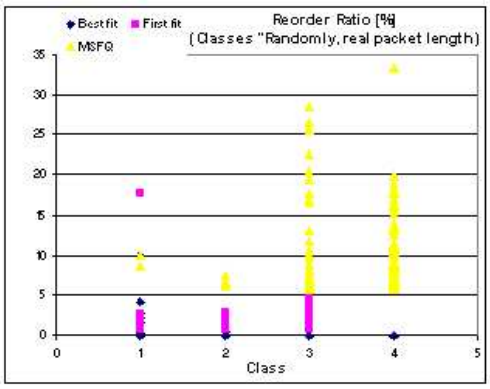


Fig. 7. Reorder ratio with "randomly real" traffic

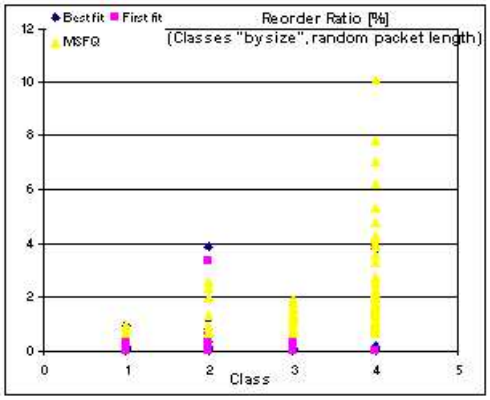


Fig. 8. Reorder ratio with "by size random" traffic

The simulation results reported in Table I (BFit and FFit there refer to BestFit and FirstFit respectively) show the number of flows with a particular reorder extent. (For example an entry 88 in row 1 means that there were 88 flows with reordering extent equal to 1.) We see that this metric for MSFQ (for any class) is substantially higher than the same for BestFit and FirstFit algorithms. Recall that the reordering extent is an approximation of the size of a buffer which is required at the client side to restore the original order. This observation shows that the dynamic partitioning algorithms imply a reduction in the total size of reordering buffers. Our simulations are performed for a large number of flows. As the number of flows increase, the reordering is expected to decrease for all the algorithms because the probability that a given flow is concurrently serviced decreases. It is also very likely that for links with a high throughput and low load the performance of all algorithms is similar. This is because all the algorithms are work conserving and tolerate reordering to some extent.

TABLE I

THE NUMBER OF FLOWS WITH A GIVEN MAXIMAL REORDER EXTENT.

| R.Ext. | by size, real | | | by size, random | | | random, real | | | random, random | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MSFQ | BFit | FFit | MSFQ | BFit | FFit | MSFQ | BFit | FFit | MSFQ | BFit | FFit |
| 1 | 88 | 56 | 98 | 82 | 50 | 44 | 48 | 33 | 27 | 55 | 42 | 44 |
| 2 | 10 | 2 | 1 | 17 | 0 | 1 | 51 | 1 | 3 | 43 | 2 | 3 |
| 3 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 2 | 1 | 1 |

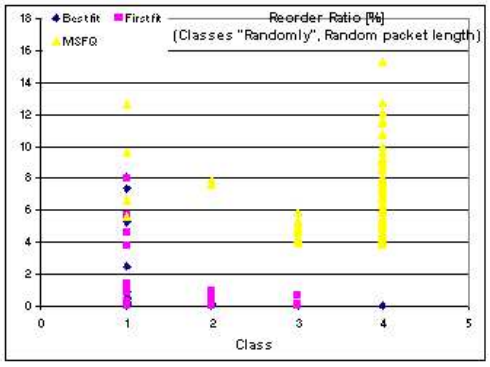


Fig. 9. Reorder ratio with "randomly random" traffic

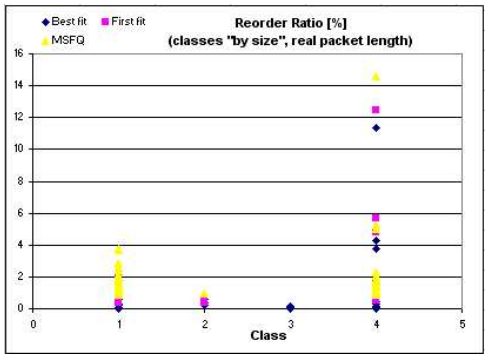


Fig. 10. Reorder ratio with "by size real" traffic

## V. CONCLUSIONS AND FUTURE WORK

We present dynamic partitioning algorithm for the QoS provisioning problem for a multi-link system. We also present an algorithm of low complexity to detect if partitioning needs to be done. This is done at the line rate. In relation to the algorithms themselves, these aims at fair sharing of bandwidth and minimize the intra-flow reordering. We furnish simulation results and evaluate reordering quantitatively through relevant metrics. We also note that both algorithms, FirstFit and BestFit behave similarly and it is difficult to say which is to be preferred when selective service is required. However, it appears that BestFit performs better when reordering is taken into account. The overall improvement suggests that FirstFit and BestFit are useful in the context of a busy LAN traffic with a relatively low number of persistent flows where trunking is employed to scalably increase bandwidth. Similarly although our algorithms were presented in the link context the same scenario is applicable to the network processor scheduling realm, where packets with different urgency and bandwidth criteria have to be scheduled onto micro engines that operate in parallel. Last but not least, they are relevant to QoS aware backup strategies in the Storage Area Network (SAN) space, where traffic of varying importance needs to be selectively stored to remote repositories.

Future work can address the issue of hard real-time guarantees in the dynamic partitioning framework. It appears from the literature that MSFQ is the only algorithm in the multi-link system that provides hard real time guarantees. An area worth exploring is whether is it possible to find efficient partitioning based work-conserving algorithms that provide provably real-time guarantees.

*Acknowledgement*

The authors wish to thank Wojciech Jawor for the idea of traffic classification and helpful discussions.

## REFERENCES

[1] J.C.R Bennet and H. Zhang. $WF^2Q$: Worst-case fair weighted fair queueing. *Proceedings of the IEEE INFOCOM,*, March 1996.

[2] Josep M. Blanquer and Banu Özden. Fair queueing for aggregated multiple links. *Proceedings of the ACM SIGCOMM*, October 2001.

[3] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM Computer Communication Review*, 32(1), January 2002.

[4] Stephan Bohacek, Joao P. Hespanha, Junsoo lee, chansook Lim, and Katia Obraczka. TCP-PR: TCP for persistent reordering. *Proceedings of the IEEE 23rd Int. Conf. on Distributed Computing Systems, pages 222-231*, May 2003.

[5] Guo Chuanxiong. SRR: an O(1) time complexity packet scheduler for flows in multi-service packet networks. *Proceedings of the ACM SIGCOMM,*, August 2001.

[6] Jorge A. Cobb and Miaohua Lin. A theory of multi-channel schedulers for quality of service. *J. High Speed Netw.*, 12(1,2):61–86, 2003.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Internetworking: Research and Experience*, 1(1), 1990.

[8] Jiani Guo, Jingnan Yao, and Laxmi Bhuyan. An efficient packet scheduling algorithm in network processors. In *IEEE INFOCOM '05*, 2005.

[9] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser. Packet reordering metric for IPPM. Technical report, IETF, August 2004.

[10] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, June 1993.

[11] Fariza Sabrina and Sanjay Jha. Scheduling resources in programmable and active networks based on adaptive estimations. *28th Annual IEEE International Conference on Local Computer Networks*, October 2003.

[12] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. *Proceedings of the ACM SIGCOMM,*, August 1995.

[13] Tilman Wolf, Prashanth Pappu, and Mark A. Franklin. Predictive scheduling of network processors. *Computer Networks,*, 41(5):601–621, April 2003.