Vertical Dimensioning: A Novel DRR Implementation for Efficient Fair Queueing

Spiridon Bakiras[1] Feng Wang[2] Dimitris Papadias[2] Mounir Hamdi[2]

[1]Department of Mathematics and Computer Science

John Jay College of Criminal Justice, City University of New York (CUNY)

[2]Department of Computer Science

Hong Kong University of Science and Technology

Email: `sbakiras@jjay.cuny.edu`

`{fwang, dimitris, hamdi}@cs.ust.hk`

Abstract

Fair bandwidth allocation is an important mechanism for traffic management in the Internet. Round robin schedulers, such as Deficit Round Robin (DRR), are well-suited for implementing fair queueing in multi-Gbps routers, as they schedule packets in constant time regardless of the total number of active flows. The main drawback of these schemes, however, lies in the maintenance of per flow queues, which complicates the buffer management module and limits the sharing of the buffer space among the competing flows. In this paper we introduce a novel packet scheduling mechanism, called *Vertical Dimensioning* (VD), that modifies the original DRR algorithm to operate without per flow queueing. In particular, VD is based on an array of FIFO buffers, whose size is constant and independent of the total number of active flows. Our results, both analytical and experimental, demonstrate that VD exhibits very good fairness and delay properties that are comparable to the ideal Weighted Fair Queueing (WFQ) scheduler. Furthermore, our scheduling algorithm is shown to outperform significantly existing round robin schedulers when the amount of buffering at the router is small.

Index Terms

Packet Scheduling, Fair Queueing, Deficit Round Robin.

I. Introduction

The fair sharing of bandwidth among competing flows inside the network is of paramount importance for efficient congestion control. The design philosophy of the Internet relies on the end-hosts to detect congestion (mainly through packet losses) and reduce their sending rates. Consequently, flows that do not respond to congestion (e.g., UDP) may end up consuming most of the available bandwidth at the expense of TCP-friendly flows, while at the same time increase the level of congestion. Fair bandwidth allocation is becoming even more important lately, due to the increasing popularity of streaming applications, such as Internet radio/TV. These applications require a stable throughput for a relatively long period of time, in order for the end-user to perceive an acceptable level of service quality.

Ideally, a router should be able to approximate a max-min fair allocation of the available bandwidth, i.e., each flow should be allocated as much bandwidth as possible, given that this allocation does not affect the throughput of any other flow. Fair queueing schedulers, such as Weighted Fair Queueing (WFQ) [1], [2], are extremely effective in providing tight fairness guarantees. In fact, fair queueing is a very well-studied problem, and many variations have been proposed throughout the years that offer different levels of complexity and fairness (a detailed overview is given in Section II).

Among all the packet schedulers reported in the literature, round robin algorithms (e.g., Deficit Round Robin (DRR) [3] and its variants) are probably the best candidates for incorporating fair queueing in multi-Gbps routers, as they schedule packets in constant $O(1)$ time regardless of the total number of active flows. The main drawback of these schemes, however, lies in the maintenance of per flow queues that raises two important issues with respect to their performance. First, the complexity of the buffer management module increases with the number of active flows, since the longest queue needs to be identified for dropping packets in the presence of congestion. Second, and most important, the sharing of the buffer space among the competing flows becomes less effective with decreasing buffer size (a fact that is demonstrated in our simulation experiments), and thus the flow isolation property of fair queueing is not strictly enforced.

To further illustrate the importance of achieving efficient statistical multiplexing with small buffer space, we should briefly discuss the current design practice of commercial routers. The rule-of-thumb (based on the dynamics of TCP's congestion control mechanism) is that the amount of buffering at a router should be equal to the bandwidth-delay product (BDP), i.e., the product of the average round-trip time (RTT) times the link capacity. Consequently, today's core routers (with multi-Gbps links) contain buffers that can hold millions of packets. However, a recent study [4] suggests that the buffering capacity at the backbone routers could be reduced by up to two orders of magnitude without significantly reducing the link utilization. If this theory holds, it will have a positive impact on future communication networks. For instance, the end-to-end delay and delay jitter will be reduced, while the cost and complexity of backbone routers will be decreased dramatically.

To this end, we introduce a novel packet scheduling mechanism, called Vertical Dimensioning (VD), that modifies the original DRR algorithm so that it can operate without per flow queueing. In particular, we introduce a simple data structure for storing the incoming packets, based on an array of FIFO buffers. We illustrate that this structure has two very attractive properties compared to previous approaches: (i) it simplifies considerably the buffer management module at the router, and (ii) it enables efficient statistical multiplexing, even with very small buffer sizes. Our results, both analytical and experimental, indicate that VD exhibits very good fairness and delay properties that are comparable to the ideal WFQ scheduler. Furthermore, our scheduling algorithm is shown to significantly outperform existing round robin schedulers when the amount of buffering at the router is much smaller than the bandwidth-delay product.

In summary, the main contributions of our work are the following:

- We introduce a novel packet scheduling algorithm for fair bandwidth allocation that does not need to maintain per flow queues.
- We provide analytical results on the delay and fairness bounds of our algorithm, and investigate its performance (with simulation experiments) under various network conditions.
- We present initial results from a prototype implementation on a software router, and demonstrate VD's effectiveness in a real network environment.

The remainder of the paper is organized as follows. Section II overviews the related work in the area of fair queueing algorithms. Section III describes in detail the VD scheduling mechanism, and analyzes its performance and implementation complexity. Section IV presents the results from the simulation experiments, while Section V provides some initial experimental results from a prototype implementation. Finally, Section VI concludes our work.

II. Related Work

Fair queueing schedulers may be generally classified into two categories, namely timestamp-based schedulers and round robin schedulers. Timestamp-based schedulers emulate as closely as possible the ideal Generalized Processor Sharing (GPS) [2] model, by computing a timestamp at each packet arrival that corresponds to the departure time of the packet under the reference GPS system. Packets are then transmitted based on their timestamp values, using a priority queue implementation. WFQ [1], [2], Worst-case Fair Weighted Fair Queueing (WF^{2}Q) [5], and WF^{2}Q+ [6] fall into this category. In particular, WF^{2}Q achieves–what is called–"worst-case fairness", by only scheduling packets that would have started service under the reference GPS system. Although all the above algorithms exhibit excellent fairness and delay properties, the time complexity of both maintaining the GPS clock and selecting the next packet for transmission is $O(\log N)$, where N is the number of active flows [7].

The high complexity of GPS-based schedulers has led to a significant number of implementations that approximate fair queueing without maintaining exact GPS clock. Start-Time Fair Queueing (STFQ) [8], Self-Clocked Fair Queueing (SCFQ) [9], and Virtual Clock (VC) [10] are typical examples of schedulers that calculate timestamps in constant $O(1)$ time. However, since they need to maintain a sorted order of packets based on their timestamp values, the overall complexity is still $O(\log N)$ using a standard heap-based priority queue.

Leap Forward Virtual Clock (LFVC) [11] and Bin Sort Fair Queueing (BSFQ) [12] further reduce the complexity of the dequeue operation, by using an approximate sorting of the packets. Specifically, LFVC reduces the timestamp space to a set of integers, in order to make use of the Van Emde Boas priority queue that runs at $O(\log \log N)$ complexity. However, the Van Emde Boas tree is a very complex data structure, and its hardware implementation is not straightforward. BSFQ, on the other hand, achieves an $O(1)$ dequeue complexity, by grouping packets with similar deadlines into the same bin. Inside a bin, packets are transmitted in a FIFO order. This is a very efficient method for implementing fair queueing, but the number and the width of the bins must be properly set, in order to avoid empty bins (which will compromise the $O(1)$ dequeue complexity).

Round robin schedulers do not assign a deadline to each arriving packet, but rather schedule packets from individual queues in a round robin manner. As a result, most round robin schedulers are able to process packets with an $O(1)$ complexity, at the expense of weaker fairness and delay bounds. Deficit Round Robin [3] is probably the most well-known scheduler in this category. It improves on the round robin scheme proposed by Nagle [13], by taking into account the exact size of individual packets. Specifically, during each round, a flow is assigned a quantum size that is proportional to its weight. Since the size of the transmitted packet may be smaller than the quantum size, a deficit counter is maintained that indicates the amount of unused resources. Consequently, a flow may transmit (at each round) an amount of data which is equal to the deficit counter plus the quantum size. It is easy to notice that DRR has certain undesirable properties. First, it has poor delay guarantees, since each flow must wait for $N-1$ other flows before it gains access to the output link. Second, it increases the burstiness of the flows, since packets from the same flow may be transmitted back-to-back.

The above shortcomings of DRR have been addressed by many researchers, and several variations of DRR have been proposed. Smoothed Round Robin (SRR) [14], for instance, employs a Weight Spread Sequence to spread the quantum of each flow over the entire DRR round, thus reducing the output burstiness. Aliquem [15] introduces an Active List Management method that allows for the quantum size to be scaled down without compromising complexity. As a result, it exhibits better fairness and delay properties compared to the original DRR implementation. Finally, Stratified Round Robin (STRR) [16] and Fair Round Robin (FRR) [17] group flows with similar weights into classes, and use a combination of timestamp and round robin scheduling to improve the delay bound. In particular, they employ a deadline-based scheme for inter-class scheduling, and a variation of DRR for scheduling packets within a certain class. Both algorithms improve over the performance of DRR, with FRR providing better short-term fairness.

III. Vertical Dimensioning

We first present in detail the VD scheduling algorithm, and then derive analytical results on its fairness and delay properties. In particular, Section III-A discusses the technical aspects of the algorithm, while Section III-B presents its performance bounds from a worst-case analysis. Finally, Section III-C outlines the space and time complexity of VD.

A. *The Algorithm*

We consider a single link with capacity C that provides service to N backlogged flows. Each flow i has an associated weight $w_i \geq 1$, which corresponds to the relative service that flow i should receive compared to the rest of the backlogged flows. In a best-effort architecture, $w_i = 1$, for all i. Ideally, the amount of bandwidth that flow i receives during any time interval should be equal to

$$r_i = \frac{w_i}{\sum_{k=1}^{N} w_k} \cdot C \tag{1}$$

Notice that we do not assume any admission control mechanism, i.e., the value of r_i will constantly change depending on the total number of backlogged flows.

Our motivation in developing the Vertical Dimensioning mechanism is to avoid the maintenance of per flow queues. To this end, we propose the use of an array of M FIFO queues, where each queue may contain packets from any active flow. The whole structure is based on the DRR mechanism, i.e., packet transmissions are organized into a number of distinct rounds. Within each round, a flow may transmit a certain amount of data that is proportional to its weight. More specifically, during each round, we assign to every flow i a quantum equal to $w_i L_M$, where L_M is the maximum packet size (i.e., the MTU size inside the network that the router belongs to). Unlike DRR, though, we do not maintain per flow queues, but rather assign one queue to each round. In other words, each packet in the VD scheduler is placed in a queue that corresponds to a complete round of transmissions under the DRR scheduler.

Figure 1 illustrates the basic functionality of the VD scheduler with $M = 10$ queues, and four flows with weights 2, 1, 1, and 1, respectively. A total of 10 packets arrive while the link is idle (for ease of presentation, however, no packet leaves the queue). The number on each packet corresponds to its order of arrival. Assuming that the size of each packet is equal to L_M, the first three packets will join $q[0]$, as they correspond to flows with an individual backlog of L_M bytes. When the fourth packet arrives, it increases the backlog of flow 4 to $2L_M$, and thus joins $q[1]$ (since $w_4 = 1$). Because the fifth packet is the first one to arrive from flow 3, it is placed in the first round (i.e., $q[0]$). This is also the case for the sixth packet, since the weight value of flow 1 allows it to transmit both packets in the same round. The rest of the packets follow in a similar fashion. In summary, VD distributes the packets from a single "flow queue" into multiple vertical "round queues", hence the name Vertical Dimensioning.

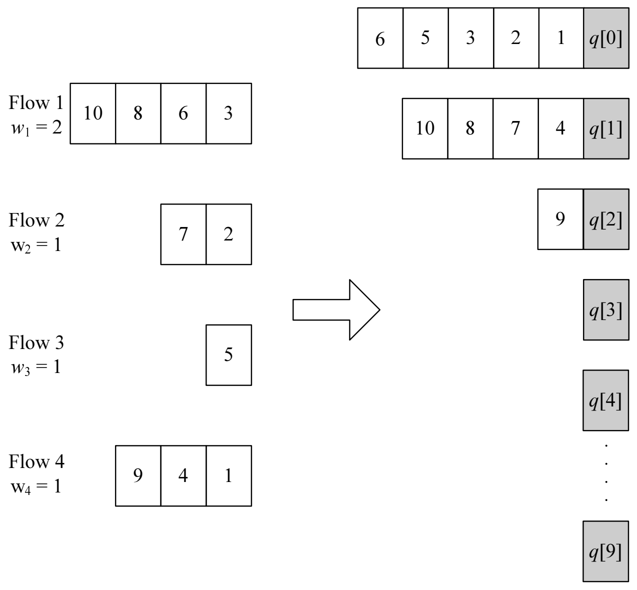


Fig. 1. An example showing how VD inserts the arriving packets from four flows into the FIFO buffers. The numbers on the packets correspond to their order of arrival.

The value of M should be set to account for the worst case scenario, i.e., when a single flow with weight value equal to 1 occupies the whole buffer space. Therefore, to avoid wrap-around, M should be set to $\lceil \frac{B}{L_M} \rceil$, where B is the buffer size of the router. Notice that, even if the value of M is fixed for the worst case, this fact has no effect on the performance of the VD scheduling algorithm. The FIFO queues do not waste any buffer space when they are idle, and are merely represented by two pointers at the head and tail of the corresponding queues.

The actual packet transmission in the VD scheduler is performed as follows. A counter $current$ is maintained, indicating the queue that is currently feeding the output link. Once this queue is empty, the counter is increased, all the packets from the following queue are transmitted, and the same process is repeated until all queues are empty. In addition, a counter $last$ identifies the queue containing packets to be dropped in the case of overflow. Both counters take values between 0 and $M-1$.

The most important function of the scheduler is to correctly identify the queue (i.e., round number) where an incoming packet should be placed at. In order to achieve that we need to maintain some per flow information. Specifically, the following variables must be kept for every active flow i:

- $bytes_i$: the total number of bytes currently in the queue for flow i.

- $deficit_i$: this value corresponds to the amount of unused resources that are carried over from one round to the next (i.e., the deficit counter in the DRR terminology). It is also utilized for counting the number of bytes transmitted in the $current$ round for flow i.
- $round_i$: the round number during which flow i transmitted its last packet.

The variable $deficit_i$ deserves some further attention, since its purpose is twofold. First, due to the variable size of IP packets, a flow i may not be able to consume its entire quantum (i.e., $w_i L_M$) during one round. Therefore, the amount of unused resources (let it be D_i) should be carried over to the next round, in order to ensure fair bandwidth allocation. It is easy to see, and has been proven in [3], that D_i may take the following values

$$0 \leq D_i < L_M \tag{2}$$

Initially, when a flow becomes active (i.e., when its first packet is enqueued) its deficit variable is initialized to zero. Then, the value of $deficit_i$ is adjusted at the beginning of each new round (when a packet from that flow is processed), in order to reflect the new value of D_i. Consider, for instance, two consecutive rounds, namely k and $k+1$. The deficit counters at the beginning of round k (D_i^k) and at the beginning of round $k+1$ (D_i^{k+1}) are connected through the following equation

$$D_i^{k+1} = (D_i^k - b_i^k) + w_i L_M$$

where b_i^k is the number of bytes transmitted in the kth round for flow i, and $w_i L_M$ is the quantum assigned to flow i in the kth round. Therefore, we choose the variable $deficit_i$ to represent $(D_i^k - b_i^k)$, i.e., during each dequeue operation its value is reduced according to the size of the transmitted packet. Consequently, the variable $deficit_i$ for flow i is bounded as follows

$$-w_i L_M \leq deficit_i < L_M$$

and is maintained through the following procedure:

- When flow i becomes active, set $deficit_i = 0$.
- When a packet of flow i is dequeued, set $deficit_i = deficit_i - size_i$, where the variable $size_i$ corresponds to the size of the transmitted packet.
- At the beginning of each new round (i.e., when processing the first packet of flow i in the

new round), set $deficit_i = deficit_i + w_i L_M$.

Given the above information, the queue number for a random packet of flow i is computed from the following formula

$$pos = \left(current + \left\lceil \frac{bytes_i - deficit_i + size_i}{w_i L_M} \right\rceil - 1 \right) \bmod M \tag{3}$$

where the variable $size_i$ corresponds to the size of the incoming packet. It is easy to verify that this formula places each packet in the exact round that it would have been transmitted under the DRR scheduler.

The detailed pseudo-code of the enqueue, dequeue and drop operations is shown in Figure 2. The only points requiring some further clarification are lines 6-8 in the enqueue operation, and lines 5-6 in the dequeue operation. Both pieces of code perform the exact same function, i.e., they update the variable $deficit_i$ to reflect the new value of the deficit counter. However, within each round this initialization is performed only once, inside the function that is invoked first.

The Vertical Dimensioning mechanism borrows the basic concepts from the DRR algorithm, but it has several advantages over the original DRR technique:

- Packets from the same flow that are scheduled in the same round are not necessarily transmitted back-to-back.
- The delay properties of VD are significantly better, since a packet does not need to wait for its turn in the round robin schedule before it can be transmitted. Instead, packets in the same round are transmitted in the order of their arrival.
- It enables efficient statistical multiplexing, since the entire buffer space is shared by all competing flows.

This last property of VD distinguishes it from all other round robin schedulers in the literature. When per flow queues are employed, the sharing of the buffer space becomes a burden, and may significantly increase the overall complexity. For instance, the buffer stealing scheme of DRR (originally proposed by McKenney [18]) suggests that, in the event of buffer overflow, a packet from the longest queue should be dropped[1]. However, maintaining a sorted order of queue lengths has a complexity of $O(\log N)$. In fact, McKenney's implementation is based on a linked list of all possible queue length values, where each entry consists of a list of queues

[1]Actually, when the flows have different weights, the length of flow i's queue should be weighted by a factor of $1/w_i$.

```
enqueue (packet $p$)
(1)  $i = p.flowid$;
(2)  if (new flow) /* initialize variables */
(3)      $f[i].bytes = 0$;
(4)      $f[i].deficit = 0$;
(5)      $f[i].round = current$;
(6)  else /* if new round, initialize $deficit_i$ if needed */
(7)      if ($f[i].round$ != $current$ and $f[i].deficit < 0$)
(8)          $f[i].deficit = f[i].deficit + w_i L_M$;
(9)  Calculate $pos$ from Equation (3);
(10) if ($q[pos]$ is empty)
(11)     $last = pos$;
(12) Insert packet $p$ at $q[pos]$;
(13) $f[i].bytes = f[i].bytes + p.size$;
(14) $bytes = bytes + p.size$;
(15) while ($bytes >$ buffer size)
(16)     drop();

packet dequeue()
(1)  $p = q[current].head$;
(2)  $i = p.flowid$;
(3)  $f[i].bytes = f[i].bytes - p.size$;
(4)  $bytes = bytes - p.size$;
(5)  if ($f[i].round$ != $current$ and $f[i].deficit < 0$)
(6)      $f[i].deficit = f[i].deficit + w_i L_M$;
(7)  $f[i].deficit = f[i].deficit - p.size$;
(8)  $f[i].round = current$;
(9)  if ($q[current]$ is empty)
(10)     $current = (current + 1)$ mod $M$;
(11) return $p$;

drop()
(1)  $p = q[last].tail$;
(2)  $i = p.flowid$;
(3)  $f[i].bytes = f[i].bytes - p.size$;
(4)  $bytes = bytes - p.size$;
(5)  if ($q[last]$ is empty)
(6)      $last = (last - 1 + M)$ mod $M$;
(7)  delete $p$;
```

Fig. 2. The pseudo-code of the enqueue, dequeue, and drop operations.

that currently have that exact length. The cost of this approach can be high if a large number of queues have approximately the same length. In VD, we always drop the packet at the tail of the last non-empty queue (e.g., $q[2]$ in Figure 1), which has a constant cost. An alternative technique with $O(1)$ complexity, called Approximated Longest Queue Drop, is also proposed by Suter et al. [19]. Instead of searching for the longest queue, the authors only store the length and *id* of the longest queue from the previous queueing operation (i.e., enqueue, dequeue or drop). However, as the authors state, this scheme does not lead to optimal behavior and may occasionally fail to provide the flow isolation property of fair queueing.

Besides the complexity of identifying the longest queue, per flow queueing has another undesirable property. When the buffer size at the router is smaller than the typical bandwidth-delay product, the sharing of the buffer space among the competing flows becomes very ineffective. Specifically, our simulation results (Section IV) indicate that flows with large weight values end up consuming most of the available bandwidth, leading flows with smaller weights to bandwidth starvation. This is due to the weighting of the queue lengths (by a factor of $1/w_i$) that essentially favors flows with large weights (notice that using the same weight value for all queues has the exact opposite effect, i.e., the high bandwidth flows cannot reach their fair share). VD, on the other hand, results in very good fairness even with small buffer sizes. Although current routers provide ample buffer space, the results in Ref. [4] suggest that the buffering capacity at the backbone routers could be reduced by up to two orders of magnitude without significantly affecting the performance. In that case, VD presents itself as an excellent candidate for implementing fair queueing in future multi-Gbps routers.

B. Performance Bounds

In this section we derive some analytical results on the fairness and delay properties of Vertical Dimensioning. For the sake of simplicity, we assume that the number of backlogged flows is constant and equal to N.

We begin by calculating the upper and lower bounds on the service that a flow i receives during X consecutive rounds.

Lemma 1: Consider a flow i that is continuously backlogged during X successive rounds. Then, the amount of service $S_i(X)$ received by that flow is bounded by

$$Xw_iL_M - L_M < S_i(X) < Xw_iL_M + L_M$$

Proof: Let D_i^{start} and D_i^{end} be the deficit values prior to the beginning and after the completion of the X rounds, respectively. The amount of service that flow i receives during the X rounds is equal to

$$S_i(X) = Xw_iL_M + D_i^{start} - D_i^{end}$$

Therefore, according to Equation (2)

$$S_i(X) > Xw_iL_M - D_i^{end} > Xw_iL_M - L_M \tag{4}$$

and

$$S_i(X) < Xw_iL_M + D_i^{start} < Xw_iL_M + L_M \tag{5}$$

Combining (4) and (5) proves the Lemma. ■

Next, we derive the corresponding bounds on the service that a flow i receives during any time interval (t_1, t_2).

Lemma 2: Consider a flow i that is continuously backlogged in the interval (t_1, t_2). The amount of service $S_i(t_1, t_2)$ received by flow i within this time interval is given by

$$(X-2)w_iL_M - L_M < S_i(t_1, t_2) < Xw_iL_M + L_M$$

where X is the number of rounds that completely enclose (t_1, t_2).

Proof: If X is the number of rounds that completely enclose (t_1, t_2), then flow i will be served at most X times. Thus, according to Lemma 1

$$S_i(t_1, t_2) < Xw_iL_M + L_M \tag{6}$$

Similarly, flow i will be served at least $(X-2)$ times, and the amount of bytes transmitted will be

$$S_i(t_1, t_2) > (X-2)w_iL_M - L_M \tag{7}$$

Combining (6) and (7) proves the Lemma. ■

In the next theorem we calculate the Golestani [9] fairness index, which measures the difference between the normalized service received by any two flows.

Theorem 1: Consider two flows i and j that are continuously backlogged in the interval (t_1, t_2). Then, the following inequality holds

$$\left|\frac{S_i(t_1, t_2)}{r_i} - \frac{S_j(t_1, t_2)}{r_j}\right| < \frac{2L_M}{r_{min}} + L_M\left(\frac{1}{r_i} + \frac{1}{r_j}\right)$$

where r_{min} is the guaranteed service rate for any flow with weight equal to 1.

Proof: Applying Lemma 2 to flow i,

$$(X-2)w_iL_M - L_M < S_i(t_1,t_2) < Xw_iL_M + L_M \Rightarrow$$

$$\frac{(X-2)w_iL_M}{r_i} - \frac{L_M}{r_i} < \frac{S_i(t_1,t_2)}{r_i} < \frac{Xw_iL_M}{r_i} + \frac{L_M}{r_i}$$

From Equation (1) it follows that

$$\frac{w_i}{r_i} = \frac{\sum_{k=1}^N w_k}{C} = \frac{1}{r_{min}}$$

where $r_{min} = C/\sum_{k=1}^N w_k$ is the guaranteed rate for any flow with weight equal to 1 (i.e., the minimum possible weight value). Therefore,

$$\frac{(X-2)L_M}{r_{min}} - \frac{L_M}{r_i} < \frac{S_i(t_1,t_2)}{r_i} < \frac{XL_M}{r_{min}} + \frac{L_M}{r_i} \tag{8}$$

Similarly, for flow j

$$\frac{(X-2)L_M}{r_{min}} - \frac{L_M}{r_j} < \frac{S_j(t_1,t_2)}{r_j} < \frac{XL_M}{r_{min}} + \frac{L_M}{r_j} \tag{9}$$

Combining (8) and (9) yields the desired result

$$\left|\frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j}\right| < \frac{2L_M}{r_{min}} + L_M\left(\frac{1}{r_i} + \frac{1}{r_j}\right)$$

■

The next theorem gives a measure of the worst-case fairness index (WFI) for VD. This index was first introduced by Bennet and Zhang [5], and gives an upper bound on the difference between the service that flow i receives and the service it should receive in the ideal case.

Theorem 2: Suppose a packet from flow i arrives at time t_1, increasing the backlog of flow i to q_i bytes. Let t_2 be the time that the last bit of q_i is transmitted. Then, the total time $\tau = t_2 - t_1$ that elapses is bounded by

$$\tau < \frac{q_i}{r_i} + \frac{2L_M}{r_{min}} + (N-1)\frac{L_M}{C} + \frac{L_M}{r_i}$$

Proof: During the time interval (t_1, t_2), the amount of bytes transmitted over the output link is equal to

$$S = q_i + \sum_{j\neq i} S_j(t_1,t_2)$$

and, therefore, τ is given by

$$\tau = \frac{S}{C} = \frac{q_i}{C} + \frac{1}{C}\sum_{j\neq i} S_j(t_1,t_2)$$

According to Theorem 1

$$\frac{S_j(t_1,t_2)}{r_j} - \frac{q_i}{r_i} < \frac{2L_M}{r_{min}} + L_M\left(\frac{1}{r_i}+\frac{1}{r_j}\right) \Rightarrow$$

$$S_j(t_1,t_2) < r_j\frac{q_i}{r_i} + r_j\frac{2L_M}{r_{min}} + r_j\frac{L_M}{r_i} + L_M$$

Hence,

$$\tau < \frac{q_i}{C} + \frac{q_i}{r_i}\sum_{j\neq i}\frac{r_j}{C} + \frac{2L_M}{r_{min}}\sum_{j\neq i}\frac{r_j}{C} + \frac{L_M}{r_i}\sum_{j\neq i}\frac{r_j}{C} + \sum_{j\neq i}\frac{L_M}{C}$$

Also, $q_i \ll C$ and, therefore

$$\tau < \left(\frac{q_i}{r_i} + \frac{2L_M}{r_{min}} + \frac{L_M}{r_i}\right)\sum_{j\neq i}\frac{r_j}{C} + (N-1)\frac{L_M}{C}$$

Since $\sum_{j\neq i} r_j < C$, it follows that

$$\tau < \frac{q_i}{r_i} + \frac{2L_M}{r_{min}} + (N-1)\frac{L_M}{C} + \frac{L_M}{r_i}$$

■

Finally, the next theorem gives a bound on the delay that a single packet experiences at the head of its “flow queue”. In other words, it gives a measure of the maximum inter-departure time between two consecutive packets of the same flow.

Theorem 3: The maximum inter-departure time between two consecutive packets of flow i is given by

$$d < \frac{2L_M}{r_{min}} + (N-1)\frac{L_M}{C}$$

Proof: We will prove this theorem, by considering the following scenario where a packet from flow i experiences the worst-case delay: all the packets from flow i are transmitted at the beginning of round k, while in round $k+1$ all of flow i's packets are transmitted last. In this case, the maximum inter-departure time d is equal to the time needed for all other $N-1$ flows to transmit the maximum amount of data possible (within the two rounds). Applying Lemma 1,

$$d < \frac{1}{C}\sum_{j\neq i}(2w_j L_M + L_M) = \frac{1}{C}\sum_{j\neq i}2w_j L_M + \frac{1}{C}\sum_{j\neq i}L_M \Rightarrow$$

$$d < 2L_M\sum_{j=1}^{N}\frac{w_j}{C} + (N-1)\frac{L_M}{C} \Rightarrow$$

$$d < \frac{2L_M}{r_{min}} + (N-1)\frac{L_M}{C}$$

■

To summarize, Vertical Dimensioning provides very good fairness and delay bounds that are comparable to previous round robin schedulers. The main limitation, though, is that these bounds are not strictly rate proportional. In other words, all the flows will experience similar average delay inside the network, regardless of their relative weights. In particular, the expression for the Golestani fairness is dominated by the term $\frac{2L_M}{r_{min}}$, while the expression for the WFI is dominated by the term $\frac{2L_M}{r_{min}} + (N-1)\frac{L_M}{C}$. However, this is an expected result, since the transmission order of the packets inside each round is based on their arrival time and not on their weight value.

C. Time and Space Complexity

The implementation complexity of a packet scheduler is probably of equal importance to its fairness properties. With link speeds reaching 40 Gbps, each packet must be processed within a time frame of a few ns. Therefore, it is imperative that the scheduler has a constant $O(1)$ time complexity that is independent of the total number of active flows. Furthermore, the per packet processing functions should be simple enough, in order to facilitate a fast hardware implementation.

Vertical Dimensioning has all the above properties, and it is extremely simple to implement in hardware. Looking back at Figure 2, all the pseudo-code lines can be implemented with simple arithmetic operations, while the most expensive procedure is the round number calculation for the incoming packet (Equation (3)). Compared to the simple DropTail queueing discipline, VD (as well as all the other fair schedulers) needs to perform some additional memory accesses during the queueing operations. Specifically, it requires two accesses for reading and writing back the per flow variables (for enqueue, dequeue, and drop). Nevertheless, these memory accesses take place on fast SRAM chips, and to not affect significantly the speed of the implementation.

Regarding space complexity, VD needs to maintain three variables for each active flow (as explained in Section III-A), and two pointers for each of the M FIFO queues. Therefore, the overall space complexity is $O(N + \lceil \frac{B}{L_M} \rceil)$. This may result in larger memory consumption, compared to per flow queueing schedulers, if $\lceil \frac{B}{L_M} \rceil > N$.

IV. Simulation Experiments

In this section we experimentally evaluate the fairness and delay properties of VD, and compare its performance to other well-known fair queueing schedulers. The experiments are performed with the ns-2 [20] network simulator. The simulation topology is shown in Figure 3, where all the links have a propagation delay equal to 1 ms. The packet size is uniformly distributed between 200 and 1000 bytes, while the simulation time is set to 60 seconds. The basic quantum size for all the schedulers (e.g., the value of L_M for VD) is set to 1000 bytes. Finally, the results presented in the following paragraphs, correspond to the average value from 10 independent simulation runs.

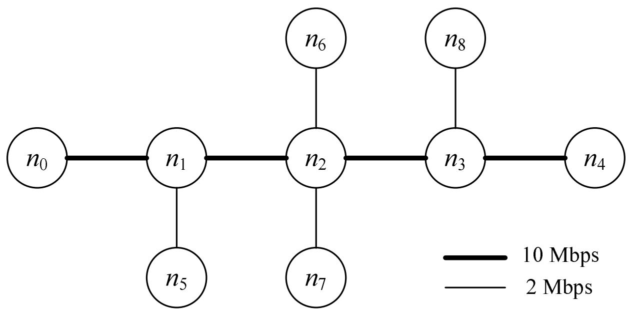
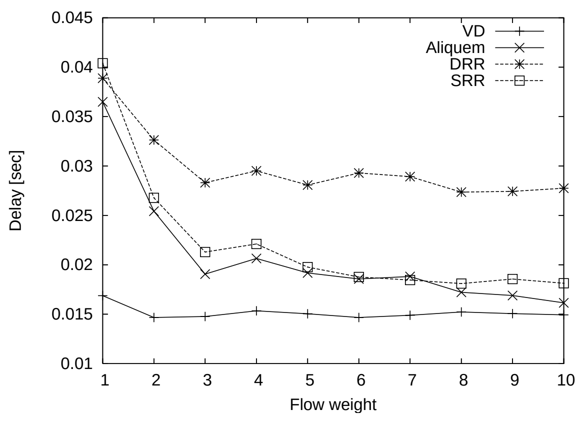


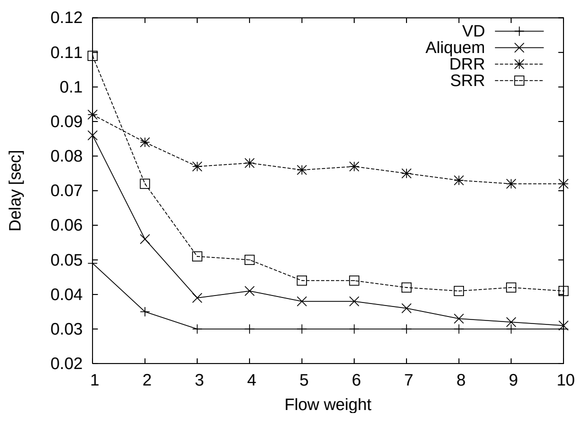
Fig. 3. Simulation topology.

There are 15 flows from n_0 to n_4, 10 flows from n_5 to n_6, and 10 flows from n_7 to n_8. These flows constitute the background traffic and we do not collect their individual statistics. They utilize the UDP transport protocol, and transmit Pareto on/off traffic with a shape parameter of 1.5. The on and off times are exponentially distributed with mean 500 ms, while the sending rate during the on periods is 250 Kbps.

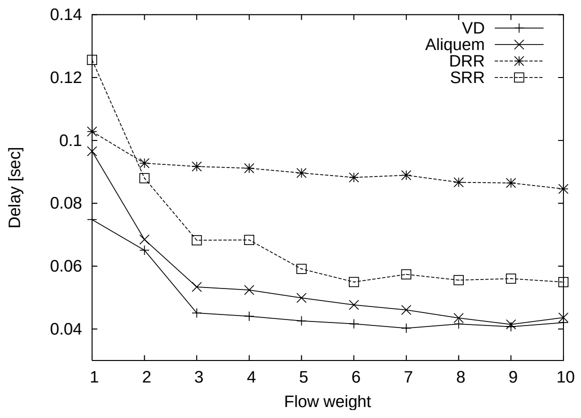
Furthermore, there are 10 reference flows from n_0 to n_4, for which we measure their performance in terms of throughput and end-to-end delay. Each reference flow i $(1 \leq i \leq 10)$ is



(a) Average.



(b) 99th percentile.

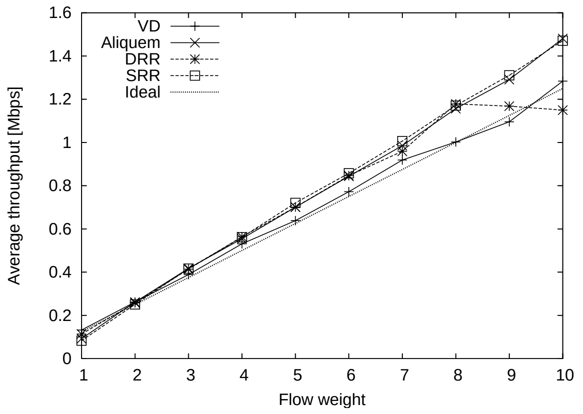


(c) Maximum.

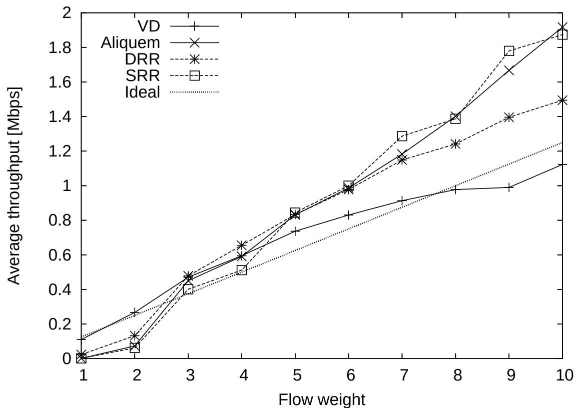
Fig. 4. Delay properties of the different scheduling algorithms (UDP sources).

assigned a weight value $w_i = i$. In order to get a complete picture of the relative performance of VD, we compare it to three representative round robin schedulers, namely DRR, Smoothed Round Robin (SRR), and Aliquem (with a scaling factor of $q = 5$). To ensure a fair comparison, we modified the DRR, SRR, and Aliquem implementations, by incorporating McKenney's [18] buffer stealing scheme (where each queue has a weight that is inversely proportional to the weight of its corresponding flow).

In the first experiment, we investigate the delay properties of the four schedulers. We configure the reference flows to transmit Constant Bit Rate (CBR) traffic over the UDP transport protocol, where flow i is transmitting at a rate of $125 \cdot i$ Kbps. Figure 4 shows the average, 99th percentile, and maximum delay achieved by the different schedulers. VD clearly outperforms the other three schedulers in all cases. For instance, VD's average delay is 8%-54% lower compared to Aliquem,



(a) $B = 50000$ bytes.



(b) $B = 12500$ bytes.

Fig. 5. Fairness properties of the different scheduling algorithms (TCP sources).

18%-58% lower compared to SRR, and 46%-57% lower compared to DRR. Also notice that the delay under the VD technique does not vary considerably for different weight values, verifying the analytical results in Section III-B.

In the above scenario each source is transmitting at a constant rate without responding to congestion. In its current state, though, the Internet relies on the end-hosts to adjust their sending rates according to the congestion level inside the network. Consequently, a scheduling algorithm should be able to provide fair bandwidth allocation regardless of the transport layer mechanism. In the next experiment we investigate the effectiveness of the four schedulers in the presence of an end-to-end congestion control protocol. In particular, we set the reference flows to be FTP applications running on top of the TCP congestion control protocol, and allow them to compete for the available bandwidth with the rest of the background flows.

Figure 5 illustrates the throughput for each of the reference flows, under different buffer sizes. SRR, DRR, and Aliquem perform poorly for a buffer size of 12500 bytes (approximately equal to 25% of the BDP). Specifically, flows with large weight values (> 4) are allocated an excessive amount of bandwidth, at the expense of flows with small weight values (1 and 2). Also notice that DRR seems to perform better for weight values greater than 7. This is due to its strict round-robin schedule, that allows each flow to access the output link once during each round. Therefore, if a packet arrives after its flow has accessed the link, it has to wait until the next round (even if it could be transmitted in the current round). Consequently, flows with large weight values cannot sustain a high throughput, because their queues are not emptied fast enough in order to avoid frequent packet losses.

Unlike the per flow queueing schedulers, VD performs very well, and its allocation of the available bandwidth is comparable to the ideal case, regardless of the buffer size. This is due to the novel placement of packets into actual DRR rounds that essentially pushes the packets that need to be dropped (in the event of congestion) to the rear of the buffer. DRR, SRR, and Aliquem are based on per flow queues, and during the congestion period the scheduler has no information regarding the importance of each packet. Dropping the packet from the longest weighted queue is obviously not the best method, since it favors flows with large weights. As a result, these schedulers require a large amount of buffer space, so that each flow can store enough packets in the queue to sustain its throughput.

V. Experiments with a Real Implementation

We implemented Vertical Dimensioning in the Linux kernel, and in this section we show some representative experiments conducted in the network topology of Figure 6. We use a star topology with a single router in the center ($S5$) where we implemented the VD scheduling algorithm. $S1$ and $S2$ are connected to $S5$ through Gigabit Ethernet interface cards, while $S3$ and $S4$ are connected through 100 Mbps Ethernet cards. The hardware configuration of $S5$ consists of two Intel Pentium III 1.4 GHz processors and 1 GB of RAM. For the traffic generation we used the publicly available tool Iperf [21]. Finally, the packet size is set to 1500 bytes.

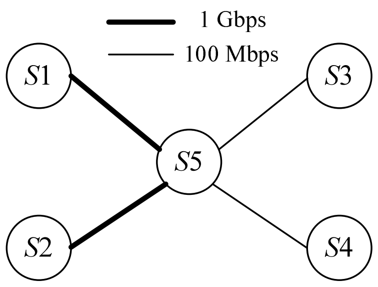


Fig. 6. Experimental testbed.

In the first experiment we test the fairness of VD when three TCP flows compete for an available bandwidth of 100 Mbps (all the flows have weight equal to 1). Specifically, nodes $S1$, $S2$, and $S3$ send traffic towards $S4$ through the VD scheduler. The starting times of the three flows are at 0, 4, and 8 sec, respectively. The throughput achieved by each flow evolves as shown in Figure 7. Clearly, VD provides excellent fairness and all three flows receive exactly their fair share.

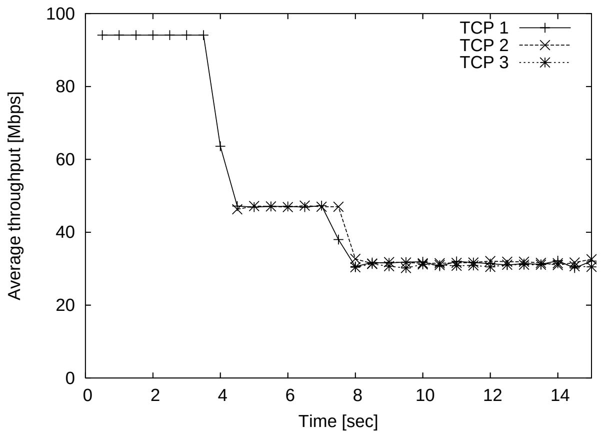


Fig. 7. Average throughput for three TCP flows.

Next, we investigate the flow isolation property of Vertical Dimensioning in the presence of unresponsive traffic. In this setting we configure both Gigabit interfaces (i.e., at $S1$ and $S2$) to send UDP traffic (towards $S4$) at their maximum rate. Their starting times are set to 0 and 4 sec, respectively. At time $t = 8$ sec we start a TCP connection from $S3$ to $S4$. Figure 8 illustrates the throughput of each flow as a function of time. Notice that in this scenario the TCP flow does not reach its maximum share, but maintains a throughput that is slightly lower. This is due to the large sending rate of the UDP sources that overwhelms the buffer, causing frequent packet losses. Consequently, the TCP source is often forced to halve its congestion window, thus reducing its sending rate. Nevertheless, even in this extreme case, the TCP flow is allocated an amount of bandwidth that is only 15% lower than its fair share.

VI. Conclusions

In this paper we introduce a new fair queueing scheduler, called Vertical Dimensioning, that modifies the original DRR algorithm to operate without per flow queueing. Similar to other round robin schedulers in the literature, VD organizes packet transmissions into a number of distinct rounds. Unlike previous approaches, though, where packets are placed in per flow queues, each packet in the VD scheduler is placed in a queue that corresponds to a complete round of

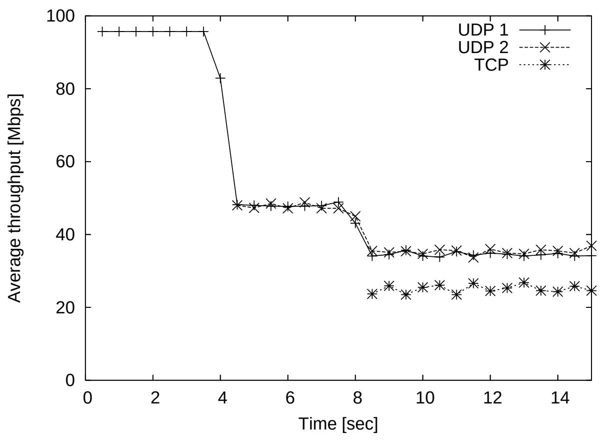


Fig. 8. Average throughput for one TCP and two UDP flows. The UDP flows are sending at their maximum rates (1 Gbps).

transmissions under the DRR scheduler. The result is a simple data structure consisting of an array of FIFO buffers that is independent of the total number of active flows. We analyze the performance of VD, both analytically and experimentally, and show that it exhibits very good fairness and delay properties that are comparable to the ideal WFQ scheduler.

In summary, VD has several attractive features that make it an ideal candidate for incorporating fair queueing in the current Internet architecture: (i) it simplifies considerably the buffer management module, (ii) it enables efficient statistical multiplexing with small buffer space, (iii) it provides fair bandwidth allocation regardless of the underlying congestion control mechanism, and (iv) it is very easy to implement in hardware.

In the future we plan to investigate the applicability of VD in a router-assisted congestion control protocol (i.e., similar to XCP [22]). In particular, the novel grouping of packets into rounds that can be translated directly into "number of bytes per unit time", may provide some useful feedback to the end-hosts that will allow them to calculate their fair share in a more efficient manner.

ACKNOWLEDGMENTS

We would like to thank Giovanni Stea for providing us with the implementation of Aliquem. This work was supported in part by PSC-CUNY award 60097-37-38.

REFERENCES

[1] A. J. Demers, S. Keshav, and S. Shenker, “Analysis and simulation of a fair queueing algorithm.” in *Proc. ACM SIGCOMM*, 1989, pp. 1–12.

[2] A. K. Parekh and R. G. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: The single-node case.” *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, 1993.

[3] M. Shreedhar and G. Varghese, “Efficient fair queueing using deficit round robin.” in *Proc. ACM SIGCOMM*, 1995, pp. 231–242.

[4] G. Appenzeller, I. Keslassy, and N. McKeown, “Sizing router buffers.” in *Proc. ACM SIGCOMM*, 2004, pp. 281–292.

[5] J. C. R. Bennett and H. Zhang, “WF^{2}Q: Worst-case fair weighted fair queueing.” in *Proc. IEEE INFOCOM*, 1996, pp. 120–128.

[6] ——, “Hierarchical packet fair queueing algorithms.” in *Proc. ACM SIGCOMM*, 1996, pp. 143–156.

[7] Q. Zhao and J. Xu, “On the computational complexity of maintaining GPS clock in packet scheduling,” in *Proc. IEEE INFOCOM*, 2004.

[8] P. Goyal, H. M. Vin, and H. Chen, “Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks.” in *Proc. ACM SIGCOMM*, 1996, pp. 157–168.

[9] S. J. Golestani, “A self-clocked fair queueing scheme for broadband applications.” in *Proc. IEEE INFOCOM*, 1994, pp. 636–646.

[10] L. Zhang, “Virtual clock: A new traffic control algorithm for packet switching networks.” in *Proc. ACM SIGCOMM*, 1990, pp. 19–29.

[11] S. Suri, G. Varghese, and G. P. Chandranmenon, “Leap forward virtual clock: A new fair queueing scheme with guaranteed delays and throughput fairness.” in *Proc. IEEE INFOCOM*, 1997, pp. 557–565.

[12] S. Y. Cheung and C. S. Pencea, “BSFQ: Bin sort fair queueing.” in *Proc. IEEE INFOCOM*, 2002.

[13] J. Nagle, “On packet switches with infinite storage.” *IEEE Transactions on Communications*, vol. 35, no. 4, pp. 435–438, 1987.

[14] G. Chuanxiong, “SRR: An $O(1)$ time complexity packet scheduler for flows in multi-service packet networks.” in *Proc. ACM SIGCOMM*, 2001, pp. 211–222.

[15] L. Lenzini, E. Mingozzi, and G. Stea, “Tradeoffs between low complexity, low latency, and fairness with deficit round-robin schedulers.” *IEEE/ACM Transactions on Networking*, vol. 12, no. 4, pp. 681–693, 2004.

[16] S. Ramabhadran and J. Pasquale, “Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay.” in *Proc. ACM SIGCOMM*, 2003, pp. 239–250.

[17] X. Yuan and Z. Duan, “FRR: A proportional and worst-case fair round robin scheduler.” in *Proc. IEEE INFOCOM*, 2005.

[18] P. E. McKenney, “Stochastic fairness queueing.” in *Proc. IEEE INFOCOM*, 1990, pp. 733–740.

[19] B. Suter, T. V. Lakshman, D. Stiliadis, and A. K. Choudhury, “Design considerations for supporting TCP with per-flow queueing.” in *Proc. IEEE INFOCOM*, 1998, pp. 299–306.

[20] “The network simulator – ns-2,” Available at http://www.isi.edu/nsnam/ns/.

[21] “Iperf,” Available at http://dast.nlanr.net/projects/Iperf/.

[22] D. Katabi, M. Handley, and C. E. Rohrs, “Congestion control for high bandwidth-delay product networks.” in *Proc. ACM SIGCOMM*, 2002, pp. 89–102.