Heart-like fair queuing algorithms (HLFQA)

Radu Dragos and Martin Collier
e_mail: {dragosr, collierm}@eeng.dcu.ie
School of Electronic Engineering, Dublin City University, Ireland

Abstract— **We propose a new family of fair, work conserving, traffic scheduling mechanisms that imitate the behavior of the human heart in the cardiovascular system. The algorithms have MAX (where MAX is the maximum packet size) fairness and $O(\log N)$ complexity and thus compare favorably with existing algorithms. The algorithms are simple enough to be implemented in hardware.**

I. Introduction

One important feature in *packet-switching* (*store-and-forward*) networks is the mechanism that determines which packet will be transmitted next on the output link. This mechanism is referred to as the traffic scheduling algorithm [16].

The role of traffic scheduling in the Internet QoS scheme is to guarantee the requirements specified in SLAs (Service Level Agreements). Hence, traffic schedulers must assure predictable delays as well as a fair share of the link bandwidth for concurrent traffic classes[1]. Such mechanisms must be able to guarantee the reserved traffic rate without packet loss independent of the behavior of other classes.

Traffic scheduling is principally required in the following situations:

- When multiple organizations share bandwidth over the same link;
- When different communication protocols share the same link;
- When different traffic types such as ftp, e-mail or real-time traffic share bandwidth on the same link.

Packet-switch architectures are in general classified into two main categories: *input-buffered* and *output-buffered* [8]. In this paper we refer only to switching in output-buffers.

A. Traffic classes

In [2], Demers *et al.* apply the term *"user"* to identify individual traffic classes that compete for the same resource (e.g. output interface). *User* could refer to the source address of a packet, the destination address, the pair source-destination, a TCP conversation, etc. What defines a *user*, is irrelevant for a traffic scheduler. The behavior of a traffic scheduler remains the same whatever the interpretation of *user*.

However, the effectiveness and complexity of a scheduler, depends on the number of *users*. The execution time of a scheduling algorithm increases with the number of concurrent *users*. Reducing the number of users will consequently increase the performance of a traffic scheduler.

[1]The concept of "traffic class" in this context will be explained in subsection I-A.

QoS technologies such as *diffserv* [1] solve the above mentioned scalability issue by grouping *users* into classes and at any router/switch along the path, each user inside a class receiving the same behavior. Therefore, a whole class of *users* becomes a single *user*. In Multi Protocol Label Switching (MPLS) [13], a class of users forwarded in the same manner and carrying the same label is called a Forwarding Equivalence Class (FEC). We will refer to competing classes of *users* as FECs by analogy with MPLS.

B. Best-effort traffic scheduling

In best-effort Internet service, packets that need to exit a router (or switch) through an interface share the same output queue. They are processed in a FCFS (first come first served) manner. This is the least complex and easiest to implement queuing discipline. However, it cannot offer fair or preferential services for traffic flows. Moreover, a single bursty FEC will have a negative impact on all competing FECs.

Although there are proposals to alleviate this issue whilst maintaining FCFS service (such as RED [4] and FRED [10]), fair bandwidth allocation can only be provided using multiple output queues.

C. Fair traffic scheduling

In order to prevent malicious FECs from affecting the well behaved ones, some level of isolation must be provided. This can be performed using a separate FCFS queue for each FEC.

The simplest approach to provide fair queuing is round robin processing of queues (RR) [11]. The main advantage of this method is its simplicity. A packet from each queue is processed in a round-robin fashion (empty queues lose their turn). However, if a queue consistently has larger packets than the others, that particular FEC will get a larger portion of the bandwidth. Improvements to the basic RR scheme include Deficit Round Robin (DRR) [15] and Hierarchical-Round-Robin [7].

Several other fair queuing mechanisms have also been proposed, all of which use a separate FCFS queue for each FEC. They are classified as work-conserving and non-work-conserving:

- **Work-conserving** schedulers are never idle when a packet is buffered in the system. Such algorithms include Generalized Processor Sharing (GPS) [12], Weighted Fair Queueing (WFQ) [2], VirtualClock [17], Delay-Earliest-Due-Date (Delay-EDD) [3] and Deficit Round-Robin (DRR) [15].

Proceedings of the Seventh IEEE International Symposium on Computer Networks (ISCN'06)
1-4244-0491-6/06/$20.00 © 2006 **IEEE**

- **Non-work-conserving** schedulers may remain idle even if there are available packets to transmit if higher priority packets are expected to arrive. Non-work-conserving schedulers include Hierarchical-Round-Robin [7] and Stop-and-go queueing [5].

D. Fairness of a scheduling algorithm

The fairness of a scheduling algorithms is measured by comparing it with the fairness of an ideal scheme called Generalized Processor Sharing (GPS). In GPS packets are considered infinitely divisible and during one cycle, an equal amount of data is processed from each queue. While this is an ideally fair algorithm, it is not suitable for packet switched networks where packets have various sizes and they are not divisible.

Therefore, the perfect fairness of GPS can not be achieved in a packet based network. However, the best approximation to the GPS algorithm is achieved when the difference in throughput at any time in any queue for any arrival pattern between the algorithm and the GPS discipline will never exceed MAX (MAX is the maximum packet size) [14]. For example, the fairness of WFQ is MAX, of DRR is 3MAX and of FQRR (Fair Queuing with Round Robin [14]) is 2MAX. Based on this metric, we will measure the fairness of our algorithm in section II-B.

II. The heart-like scheduling algorithm

In this section we present a work conserving traffic scheduling algorithm inspired by the principles of the human heart. First we explain the main concepts of our scheduling algorithm and the similarity with the atrium-ventricle model in the human heart. Then, we evaluate the fairness and complexity of the algorithm using an analytical model of our algorithm and computer simulations. An extended algorithm for weighted fair queuing is presented at the end of this section.

Then we derive a lighter version of this algorithm, easier to implement and having better storage complexity. This simplified version is also suitable for weighted scheduling.

A. The atrium-ventricle model

Our scheduling algorithm is based on the atrium-ventricle model of the heart in the cardiovascular system. The output queue of an interface is divided into an atrium section where packets are buffered and a ventricle section where packets are sent out by applying pressure to the ventricle.

Atrioventricular valves allow packets to move from the atrium to the ventricle during the atrial systole and prevent packets from running back from the ventricle to the atrium during the ventricular systole.

Whereas the human heart is quadric-cameral, in this model we have two chambers for each FEC: one atrial and one ventricular, as depicted in Fig. 1. The atrial and ventricular chambers for each FEC will be referred to as the holding queue and submit queue respectively.

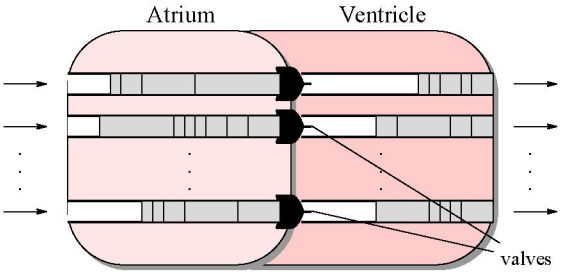

Fig. 1. Atrium-ventricle model

When the ventricle is contracted, packets are sent out through the output interface (aorta). The first packet to emerge is the one from the submit queue with the highest *pressure*[2]. After one packet (or more) is released from that submit queue, the pressure decreases sufficiently so that another submit queue will have the highest pressure and the next packet will be sent from this queue.

When one or more submit queues are empty, the packets are moved from holding queues into submit queues through atrioventricular valves.

1) Ventricular systole: Pressure in submit queues is a positive rational value. Before the first ventricular systole, the pressure is equalized by being set to unity for all submit queues so that each FEC starts with an equal chance of transmission. That is:

$$P_0^k \leftarrow 1; \text{ for } 0 < k \leq N$$

$$\begin{array}{ll} where & \\ P_0^k & \text{is the initial pressure for FEC k} \\ N & \text{is the number of FECs;} \\ Q & \text{is the maximum submit queue size.} \end{array} \quad (1)$$

At step i, a packet is selected from the queue with the highest pressure ($\max(P_i^k)$). When the i^{th} packet of size S_i^k is released from queue k, the pressure becomes:

$$P_i^k \leftarrow P_{i-1}^k - \frac{S_i^k}{Q}. \quad (2)$$

2) Ventricular diastole and atrial systole: These two phases are simultaneous. This happens when one or more submit queues are empty and the ventricle needs to relax so the packets from the atrium can be pushed into the ventricle through the atrioventricular valves.

The counter is reset to 0 and the pressure in all submit queues is reset to $P_0^k \leftarrow 1 + P_i^k$, where P_i^k is the pressure for FEC k before the ventricular diastole.

3) Atrial diastole: The atrium must be able to receive packets continuously. Therefore, the atrium will be in a permanent diastole. The short systolic contractions will take place during the atrial diastole phase.

The hold and submit queues have limited buffer capacity similar to the human heart. In the cardiovascular system if the

[2]The interpretation of the term *pressure* in this context will be described later.

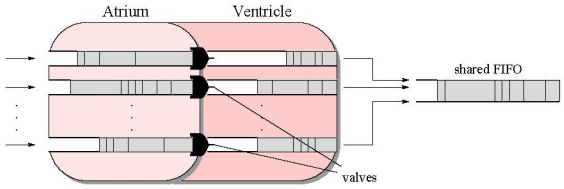

Fig. 2. Using a shared output FIFO as the aorta.

rate of blood from the veins increases, so will the heart rate and the amount of blood entering the atrium equals the amount that leaves the ventricle. In a similar way in a network switch, the input traffic rate almost equals the output rate (small variations may be accepted, depending on the size of the holding queue). Consequently, HLFQA will not accept packets if the holding queue is full and must be able to decide which packets to drop before entering the atrium.

4) Aorta: The output interface resembles the aorta in the cardiovascular system. However, the packets that leave the submit queues could be pre-buffered before sending them out through the interface. This is to avoid the idle times when the ventricle is in diastole and does not push out packets. Therefore, the output interface will always have packets to process in the buffer. In this model the shared output buffer is now the aorta as seen in Fig 2.

B. Evaluating the algorithm

The fairness of our scheduling mechanism comes from the fluid model of our approach. Compressing the ventricle equalizes the pressure in all the submit queues (although the equalization is never exact, given that the packet is the smallest unit we can transmit).

Let k and l be two FECs. T_i^k and T_i^l are the total amount of data sent for FECs k and l up to (and including) step i.

$$T_i^k = \sum_{j=1,i} S_j^k; \quad T_i^l = \sum_{j=1,i} S_j^l \tag{3}$$

At the beginning of each ventricular systole we have $T_0^k = T_0^l = 0$. At each step a single packet is sent out through the aorta. Therefore:

$$|T_1^k - T_1^l| \leq MAX \tag{4}$$

where MAX is the maximum packet size. Now, assuming that:

$$|T_{i-1}^k - T_{i-1}^l| \leq MAX \tag{5}$$

we want to prove that $|T_i^k - T_i^l| \leq MAX$.

If at step i no packet is sent out either from queue k or l, we have $T_{i-1}^k = T_i^k$ and $T_{i-1}^l = T_i^l$ and therefore $|T_i^k - T_i^l| = |T_{i-1}^k - T_{i-1}^l| \leq MAX$.

If at step i a packet is sent from one of the two queues, for instance queue l, it means that the pressure in queue l is greater than in queue k. That is:

$P_{i-1}^k \leq P_{i-1}^l$ and therefore,

$T_{i-1}^k \geq T_{i-1}^l$ and $|T_{i-1}^k - T_{i-1}^l| = T_{i-1}^k - T_{i-1}^l$.

Because a packet is sent from queue l and no packet is sent from queue k we have: $T_i^k = T_{i-1}^k$ and $T_i^l \leftarrow T_{i-1}^l + S_i^l$.

	FCFS	DRR	WFQ	HLFQA
fairness	-	3MAX	MAX	MAX
complexity	O(1)	$O(1)$	$O(\log N)$	$O(\log N)$

TABLE I
COMPARISON OF SCHEDULING ALGORITHMS

Hence:

$$|T_i^k - T_i^l| = |T_{i-1}^k - T_{i-1}^l - S_i^l| \leq |MAX - S_i^l| \leq MAX. \tag{6}$$

We have proved (4) and from assumption (5) we derived (6) to be true. Hence, using mathematical induction, we have proven that for any two queues, at any step i,

$$|T_i^k - T_i^l| \leq MAX. \tag{7}$$

Now, if we consider the total service provided until the moment i to be T then, the service of the ideal GPS discipline for each FEC will be $\frac{T}{N}$. But the total amount of service is also the sum of service of all FECs:

$$T = \sum_{j=1,N} T_i^j \tag{8}$$

In the worst case (and using (7)) we have a FEC k so that

$$\sum_{j=1,N} T_i^j = N \times T_i^k \pm N \times MAX \tag{9}$$

From 8 and 9 we have:

$$T = N \times T_i^k \pm N \times MAX \tag{10}$$

Hence, in the worst case, the difference between the service of GPS and the service of FEC k is:

$$\begin{aligned} &\tfrac{T}{N} - T_i^k = \tfrac{N \times T_i^k \pm N \times MAX}{N} - T_i^k = \\ &= T_i^k \pm MAX - T_i^k = \pm MAX \end{aligned} \tag{11}$$

Therefore, the fairness of our algorithm is MAX.

An additional feature to the fairness of this algorithm is that if one (or more) FECs are idle, the unused bandwidth is evenly (proportional) distributed among the remaining FECs. And since the algorithm does not keep state of previous events, the FECs are not penalized for using excess bandwidth when other FECs were idle unlike VirtualClock [17].

1) Complexity of the algorithm: The complexity of scheduling is given by the number of operations required to send one packet. In our approach the pressure for each submit queue is stored in a sorted array. The head of the array is the highest pressure value. A packet is sent from the submit queue with the highest pressure then, the pressure is recalculated for that queue. This requires (based on (2)) only two basic operations having constant complexity. The complexity of the algorithm derives from the operation of inserting the new pressure value in a sorted array which is of $O(\log n)$ complexity. Table II-B.1 shows the relation between fairness and complexity of our algorithm and other popular scheduling algorithms.

Although there are scheduling algorithms such as Emulated Weighted Fair Queueing (EWFQ) [9] or Self-Clocked Fair Queuing (SCFQ) [6] having lower complexity ($O(1)$), the price paid is the reduced level of isolation among the sessions, causing the end-to-end delay bounds to grow linearly with the number of FECs [16].

C. Weighted scheduling

The algorithm can be modified to provide weighted fair scheduling. If we have N concurrent FECs requesting p^k percent of available bandwidth, where $\sum_{k=1,N} p^k = 100$ then the initial pressures in submit queues can be set to:

$P_0^k \leftarrow \frac{\frac{p^k}{100} \times Q}{\frac{p^k}{100} \times Q}$ and $P_i^k \leftarrow P_{i-1}^k - \frac{S_i^k}{\frac{p^k}{100} \times Q}$.

This will provide service for FECs proportional with their requested p^k percent of the available bandwidth.

III. Implementing the algorithm

A. Storing packets

For each FEC we need 2 queues (hold and submit). Single linked lists can be used to implement the FIFO queues. Although, in the human heart, blood cells in the atrium are separated from those in the ventricle, in our implementation the linked lists of packets from the atrium are linked with those in the ventricle. Therefore, moving packets through the atrioventricular valve is seamless. Pointers are used to identify the first and the last packet in both hold and submit queues.

A doubly linked list stores the values of pressure in each submit queue. This is an ordered list; the values are stored in descending order.

B. Operations

1) Atrial diastole (receiving packets): The algorithm is in a permanent atrial diastole phase because the system must be able to receive packets continuously.

We make the assumption that packets are already classified into FECs. Therefore, buffering packets means linking every new packet at the end of its corresponding linked list.

2) Atrial systole and ventricular diastole (moving packets from atrium to ventricle): When one or more submit queues are empty, packets from the atrium will flow into the ventricle. In the actual implementation, only a few pointers are changed.

The pressure must be recomputed for each queue. The complexity of this operation is $O(N)$. However this operation is performed only when a ventricular queue is empty and it is not required for the simplified version of the algorithm.

3) Ventricular systole: The head of queue of the sorted doubly linked list of pressure values represents the submit queue with the highest pressure. The head of queue packet is selected from that submit queue and sent out via the network interface.

The pressure is recomputed only for that particular queue and the value inserted in the sorted list of pressure values. The complexity of this operation is $O(\log N)$.

IV. Simulation results

We performed two simulations. In the first, three FECs share a link equally. Their average rate will stabilize at one third (33.3%) of the bandwidth as shown in Fig. 3.

In the second test we weighted the flows with weights 1, 2, and 3, so they take 16.6%, 33.3% and respectively 50% of the bandwidth. After a while the third flow stops sending packets, then the second. In Fig.4 it can be seen that after FEC3 stops sending packets, the remaining flows share the bandwidth with weights 1 and 2 representing now 33.3 and respectively 66.6 percent of the bandwidth. When FEC1 remains alone it will use the entire bandwidth. The second simulation showed that if one (or more) FEC is idle, the unused bandwidth is evenly (proportional by) distributed among the remaining FECs.

V. Simplified HLFQA (s-HLFQA)

The analogy of the HLFQA algorithm with the operation of the human heart is attractive, but brings the disadvantage that two queues must be maintained per FEC. It is possible to reduce this to a single queue by appropriately modifying HLFQA to obtain a simplified (s-HLFQA) algorithm.

In the simplified version, there is only one queue per FEC, a unified hold and submit queue. In this context, since packets enter arbitrarily into the queue (the heart is always open) we cannot use queue pressures in making the scheduling decision. Hence, we use another measure to decide which packet will be sent next and from which queue. While in HLFQA we send a packet from the queue with the highest pressure, in s-HLFQA we send a packet from the queue which has received the least amount of service.

Therefore at step i, a packet will be sent from queue k if and only if:

$$T_i^k = min(T_i^j), \quad j = 1, 2...N; \tag{12}$$

where T_i^k is the total amount of data sent for FEC k until step i as described in 3.

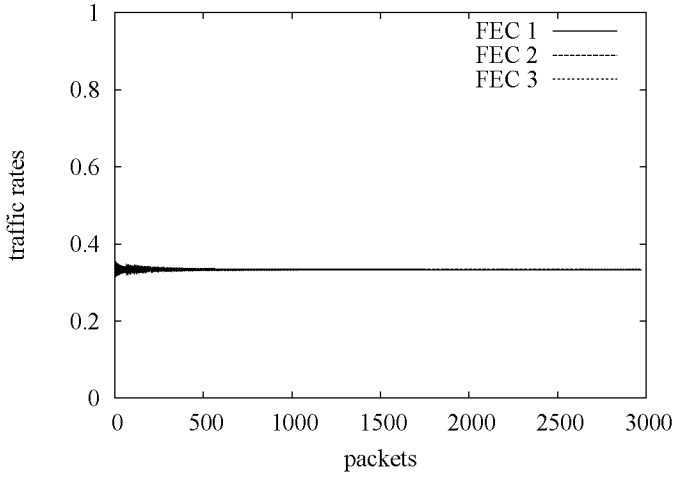

Fig. 3. 3 FECs sharing equally 33.3% of the link

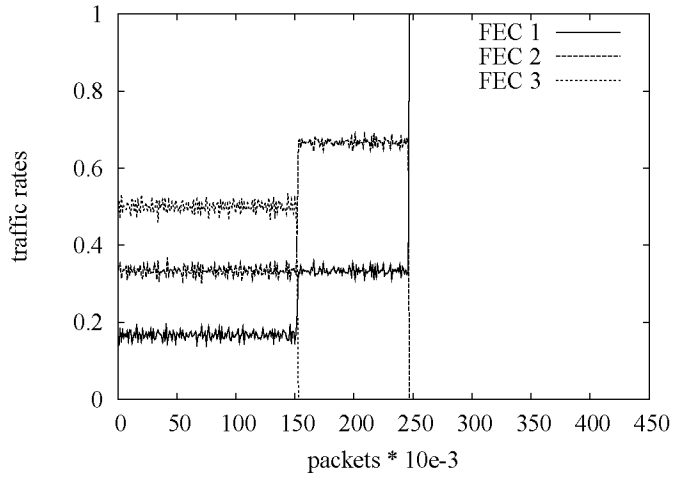

Fig. 4. 3 weighted FECs sharing respectively 16.6/33.3/50 % of the link then 2 flows 33.3/66.6 then 1 flow 100%

However, the T_i^k are continuously increasing values and therefore they can be normalized or reseted to lower values when they reach an upper bound and each time the value of N changes. The procedure is explained below:

- **Resetting T to lower values:** When the values T are considered too big, they can be reset to 0. However, to maintain the perfect fairness of the algorithm we suggest resetting them based on the following function:
  For each $k = 1, 2...N$;

$$T_i^k \leftarrow T_i^k - min(T_i^j), \quad j = 1, 2...N; \tag{13}$$

- **Normalizing** T**:** Another option is to use a normalized value for T representing actually the traffic rate:

$$T_i^k \leftarrow \frac{T_i^k}{i}, \quad k = 1, 2...N; \tag{14}$$

A. Fairness of s-HLFQA

Let k and l be two FECs. T_i^k and T_i^l are the total amount of data sent for FECs k and l until step i.

$$T_i^k = \sum_{j=1,i} S_j^k; \quad T_i^l = \sum_{j=1,i} S_j^l \tag{15}$$

Initially we have $T_0^k = T_0^l = 0$. At each step a single packet is sent out. Therefore:

$$|T_1^k - T_1^l| \leq MAX \tag{16}$$

where MAX is the maximum packet size. Now, assuming that:

$$|T_{i-1}^k - T_{i-1}^l| \leq MAX \tag{17}$$

we want to prove that $|T_i^k - T_i^l| \leq MAX$.

If at step i no packet is sent out either from queue k or l, we have $T_{i-1}^k = T_i^k$ and $T_{i-1}^l = T_i^l$ and therefore $|T_i^k - T_i^l| = |T_{i-1}^k - T_{i-1}^l| \leq MAX$.

If at step i a packet is sent from one of the two queues, for instance queue l, it means that $T_{i-1}^k \geq T_{i-1}^l$ and $|T_{i-1}^k - T_{i-1}^l| = T_{i-1}^k - T_{i-1}^l$

Because a packet is sent from queue l and no packet is sent from queue k we have: $T_i^k = T_{i-1}^k$ and $T_i^l \leftarrow T_{i-1}^l + S_i^l$

Hence:

$$|T_i^k - T_i^l| = |T_{i-1}^k - T_{i-1}^l - S_i^l| \leq |MAX - S_i^l| \leq MAX \tag{18}$$

We have proved (16) and from assumption (17) we derived (18) to be true. Hence, using mathematical induction, we have proven that for any two queues, at any step i,

$$|T_i^k - T_i^l| \leq MAX \tag{19}$$

This is the same result as in (7) for HLFQA. Therefore, we can again deduce (11) so the fairness of s-HLFQA is MAX.

B. Complexity of s-HLFQA

The time complexity of s-HLFQA is given by the number of operations performed in order to send one packet from N queues.

The values T are stored in a sorted array (or list). Selecting the min from that array requires one basic operation. Another basic operation is required to increase T: $T_{i+1}^k \leftarrow T_i^k + S_i^k$. The new value of T must be inserted in the sorted array. This operation has $O(\log N)$ complexity. Therefore s-HLFQA belongs to the $O(\log N)$ class of complexity.

C. Weighted s-HLFQA

s-HLFQA can be used to provide weighted fair scheduling as well. If we have N concurrent FECs requesting p^k percent of available bandwidth, where $\sum_{k=1,N} p^k = 100$, the total service for FEC k will be recorded in this way:

$$T_i^k = T_{i-1}^k + \frac{S_i^k}{p^k} = \frac{\sum_{j=1,i} S_j^k}{p^k}. \tag{20}$$

From (19) we have $|T_i^k - T_i^l| \leq MAX$ for any $l, k \in 1, 2, ..., N$. Since $T_i^k \gg MAX$ we may consider

$$T_i^k \approx T_i^l. \tag{21}$$

Let S_i be the total amount of packets processed for all the queues until step i. That is:

$$S_i = \sum_{j=1,N} S_i^j \tag{22}$$

From (21) we have: $T_i^1 \approx T_i^2 \approx ...T_i^N$ and using (20) we obtain: $\frac{\sum_{j=1,i} S_j^1}{p^1} \approx \frac{\sum_{j=1,i} S_j^2}{p^2} \approx ... \frac{\sum_{j=1,i} S_j^N}{p^N} \approx \frac{S_i}{\sum_{j=1,N} p^j} = \frac{S_i}{100}$.

Consequently, for any k we have: $\frac{\sum_{j=1,i} S_j^k}{p^k} \approx \frac{S_i}{100}$ and therefore:

$$\sum_{j=1,i} S_j^k \approx \frac{p^k}{100} \cdot S_i \tag{23}$$

Hence, the percentage of packets sent for FEC k is approximately equal with p^k.

VI. Conclusions

A new scheduling algorithm, suitable for deployment in MPLS networks, has been proposed, based on an analogy with the workings of the human heart. It has been shown in the previous sections that the HLFQA class of algorithms achieve MAX fairness and $O(\log N)$ complexity. This is the optimal fairness that can be achieved with packet based schedulers. Scheduling algorithms such as EWFQ and SCFQ have lower computational complexity ($O(1)$). However, they do not achieve the same optimal fairness and the price paid is the reduced level of isolation among the sessions, causing the end-to-end delay bounds to grow linearly with the number of FECs. WFQ has similar properties to HLFQA in term of fairness and complexity. However, the calculations to be performed are simpler for HLFQA. A simplified implementation (having the same fairness and complexity) called s-HLFQA has also been proposed. Both algorithms are simple enough to be implemented in hardware so that wire-speed operation is possible at high bit rates.

We are currently looking at ways to parallelise the algorithm. A parallel implementation should enable line rates of 40 Gb/s to be accommodated. At such rates, the scheduler will typically interface to a high-speed optical network core, where GMPLS is used to manage the combined MPLS/optical network. We are looking at how to combine the pre-buffering in HLFQAs holding queues to allow packets of the same FEC to be aggregated in larger frames in order to increase the average frame size in the Internet core. This will result in less stringent switching requirements in the Internet core. However, packet aggregation increases the value of MAX (the maximum packet size) and thus adversely affects scheduler fairness. Selective aggregation (where packets are merged only when it is fair to do so) can address this difficulty and is a topic for future research.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Service*, RFC 2475, IETF, December 1998. Status: INFORMATIONAL.

[2] A. Demers, S. Keshav, and S. Shenker, *Analysis and simulation of a fair queueing algorithm.*, SIGCOMM Symposium on Communications Architectures and Protocols, (1989), pp. 1–12.

[3] D. Ferrari and D. C. Verma, *A Scheme for Real-Time Channel Establishment in Wide-Area Networks*, IEEE Journal on Selected Areas in Communications, 8 (1990), pp. 368–379.

[4] S. Floyd and V. Jacobson, *Random early detection gateways for congestion avoidance*, IEEE/ACM Transactions on Networking, 1 (1993), pp. 397–413.

[5] S. J. Golestani, *A Framing Strategy for Congestion Management*, in INFOCOM, vol. 9, 1991, pp. 1064–1077.

[6] ———, *A Self-Clocked Fair Queueing Scheme for Broadband Applications*, in IEEE Journal on Selected Areas in Communications, 1994, pp. 636–646.

[7] C. R. Kalmanek and H. Kanakia, *Rate Controlled Servers for Very High-Speed Networks*, Proceedings of the Conference on Global Communications (GLOBECOM), (1990), pp. 12–20.

[8] M. Karol, M. Hluchyj, and S. Morgan, *Input versus Output Queueing on a Space-Division Packet Switch*, IEEE Trans. on Communications, COM-35 (1997), pp. 1347–1356.

[9] N.-S. Ko and H.-S. Park, *Emulated Weighted Fair Queuing Algorithm for High-Speed Packet-Switched Networks.*, in ICOIN, 2001, pp. 52–60.

[10] D. Lin and R. Morris, *Dynamics of Random Early Detection*, in SIGCOMM '97, Cannes, France, september 1997, pp. 127–137.

[11] J. Nagle, *On Packet Switches With Infinite Storage*, RFC 0970, IETF, December 1985.

[12] A. Parekh and R. Gallager, *A generalized procesor sharing approach to flow control - the single node case*, Proceedings of INFOCOM'92, 2 (1992), pp. 915–924.

[13] E. Rosen, A. Viswanathan, and R. Callon, *Multiprotocol Label Switching Architecture*, RFC 3031, IETF, January 2001. Status: STANDARDS TRACK.

[14] A. Sen, I. Mohammed, R. Samprathi, and S. Bandyopadhyay, *Fair Queuing with Round Robin: A New Packet Scheduling Algorithm for Routers*, in "Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)", 2002.

[15] M. Shreedhar and G. Varghese, *Efficient Fair Queueing Using Deficit Round Robin*, in SIGCOMM, 1995, pp. 231–242.

[16] D. Stiliadis, *Traffic Scheduling in Packet-Switched Networks: Analysis Design and Implementation*, PhD Thesis, Department of Computer Science and Engineering, University of California at Santa Cruz, 1996.

[17] L. Zhang, *VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks*, ACM Transactions on Computer Networks, 9 (1991), pp. 101–124.