Analysis of Distributed DDQ for QoS Router

Kicheon Kim

In a packet switching network, congestion is unavoidable and affects the quality of real-time traffic with such problems as delay and packet loss. Packet fair queuing (PFQ) algorithms are well-known solutions for quality-of-service (QoS) guarantee by packet scheduling. Our approach is different from previous algorithms in that it uses hardware time achieved by sampling a counter triggered by a periodic clock signal. This clock signal can be provided to all the modules of a routing system to get synchronization. In this architecture, a variant of the PFQ algorithm, called digitized delay queuing (DDQ), can be distributed on many line interface modules. We derive the delay bounds in a single processor system and in a distributed architecture. The definition of traffic contribution improves the simplicity of the mathematical models. The effect of different time between modules in a distributed architecture is the key idea for understanding the delay behavior of a routing system. The number of bins required for the DDQ algorithm is also derived to make the system configuration clear. The analytical models developed in this paper form the basis of improvement and application to a combined input and output queuing (CIOQ) router architecture for a higher speed QoS network.

Keywords: Quality-of-service, queuing, packet scheduler, real-time, traffic control, router.

Manuscript received Dec. 18, 2004; revised Jan. 12, 2006.

Kicheon Kim (phone: + 82 42 481 8374, email: kicheonkim@paran.com) was with Ajou University, Suwon and is currently with Korean Intellectual Property Office (KIPO), Daejeon, Korea.

I. Introduction

Since fair queuing [1] was first introduced to provide delay bounds and reserved rates to real-time traffic, a series of packet fair queuing (PFQ) algorithms were proposed to improve fairness and efficiency in an integrated packet network. A PFQ algorithm, weighted fair queuing (WFQ), emulating the ideal fluid fair queuing was proposed for the integrated services (IntServ) quality-of-service (QoS) mechanism in the Internet standards [2], [3]. The scalability problem of PFQ algorithms is well known, and the deployment is limited. The speed of the communication link increases rapidly, but the PFQ algorithms do not scale well enough.

Traditionally, researchers have been trying to sort out the scalability problem by improving the efficiency of the algorithms. The complexity of PFQ algorithms is based on the complexity of calculating the system virtual time to compute the timestamps of the packets, and on the complexity involved in sorting the timestamps in order to select the packet with minimum timestamp for transmission. Researchers have improved the performance by changing the system virtual time function and priority queue management methods. However, it seems that improving the efficiency alone is not enough to catch up with the ever increasing link speed.

Our approach is a little bit different from the conventional algorithms in that we are pursuing the benefits of a distributed system: the entire algorithm is divided into smaller pieces, and each piece of work is performed by separate processors so that we can lower the work load of each processor, increasing the speed of the entire algorithm. For example, the computation of a timestamp and management of priority queues of the PFQ algorithm can be performed by different processors at different positions in a routing system.

We devise a variant of the PFQ algorithm, called digitized delay queuing (DDQ), which is suitable for distributed

processing and is applied to an output queuing switch as a QoS router architecture. The main idea is use of hardware time obtained by sampling a counter triggered by a periodic clock signal. The clock signal is provided to all the modules to get synchronization in a distributed system, but there is a little difference in time between modules. An analytical investigation on delay bounds in a single processor system and distributed architecture gives a result in which the effect of the different time between modules is ignorable. This investigation is the main contribution of this paper and the traffic contribution model makes the delay bound model easier to understand. The study on the maximum number of bins shows that the routing system can be economically implemented without queuing loss. We also made a comparison of DDQ to other PFQ algorithms to show the applicability to a combined input and output queuing (CIOQ) architecture for a large-scale and high-speed router.

In section II, we discuss PFQ algorithms. In section III, we provide the details of the DDQ algorithm and simple analytical models for the delay bounds and number of bins. In section IV, we present the effect of time difference between line interface modules on the delay behavior, and a comparison of DDQ to other PFQ algorithms to show the applicability of DDQ to a large scale QoS router. We conclude in section V.

II. Fair Queuing and Router

When fair queuing was first introduced [1], it was expected to provide firewalls between real-time traffic and guarantee a reserved rate to each real-time packet flow. Soon, a more advanced scheduler, VirtualClock (VC) [4], was introduced. The main contribution of VC was that it provides delay bounds to real-time packet flows by sorting packets according to their timestamps. The timestamp is now widely understood as the service time by a fluid server at the reserved rate. It takes $L.f.i/R.f$ to serve the packet from the service start time, where *L.f.i* is the length of packet *i* of flow *f*, and *R.f* is the reserved rate of flow *f*. The service start time of the first packet of each flow is the arrival time of the packet, and the service start time of the successive packets are then obtained by taking the larger value between the timestamp of the previous packet and the arrival time. The packet queue is sorted in increasing order of the timestamp. The delay bound of the VC scheduler is equivalent to the timestamp of each packet [5].

A theoretical analysis of the fair queuing scheduler was first presented in generalized processor sharing (GPS) [3]. This scheduler is similar to VC in many ways, but it is different in timestamp calculation. The timestamp is obtained by simulating a fluid server. The scheduler serves each queued flow by the weighted share. The virtual time of the system should be recalculated at every change in the queued flows. When the flows are serviced at a rate faster than the reserved rate, the system virtual time gets faster than the real time. When the queue is empty, the system virtual time becomes zero. A GPS scheduler is an ideal fair scheduler, but the computing complexity of the algorithm is higher. The packet approximation of a GPS server was presented as WFQ and has been considered as an ideal PFQ model.

Self-clocked fair queuing (SCFQ) [6] is also a PFQ algorithm and suggests obtaining the system virtual time from the timestamp of the packet receiving service. This scheduler is more efficient than WFQ. Thus, it is suitable for a high-speed link but provides larger delay bounds than VC or WFQ.

Worst-case fair weighted fair queuing (WF^2Q) [7] is also a PFQ algorithm and additionally performs rate control by inhibiting transmission of a packet until the timestamp of the previous packet. The first packet of a flow does not need rate control. This rate control reduces the demand on the buffer in the downstream router. This can eliminate packet loss due to buffer shortage in the downstream router.

Bin sort fair queuing [8] approximates the timestamp. Thus, there could be many packets with the same approximated timestamp. These packets are queued to a single data structure called a *bin*. Therefore, the sorting overhead can be lowered. If we can accept additional approximation error of the delay bounds, this algorithm scales very well.

On the other hand, instead of sorting timestamps, a round-robin service discipline, called deficit round robin (DDR), was studied [9]. DRR is much more efficient than sorting-based PFQ algorithms, but it requires per-flow queuing, and the scheduler should be aware of the reserved rate of each flow. We found that these features of DRR made it difficult to modify for a distributed system. The performance of the round-robin scheduler is better than sorting-based PFQ algorithms, but in a distributed queuing architecture the round-robin scheduler is not the best fit. We will discuss this more after discussing our router architecture.

Another important research on fair queuing is the hardware-based packet scheduler [10], [11]. This method incorporates special hardware utilizing massive parallelism for sorting packets. The special hardware can be provided as a co-processor of the conventional processor and can help to lower the sorting burden of PFQ algorithms.

Most PFQ algorithms have similar delay bounds and similar computing complexity. However, they do not consider the architecture of the routing system. Thus, they are supposed to sit on a general-purpose computer architecture.

We consider a routing system with many links, and an output queuing architecture where a variant of the PFQ algorithm runs. Generally, a routing system consists of line interface modules and a switching module. The input link to the routing system is

connected to a line interface module, and packets arrive at the router through the input link. Then, the processor in the line interface module parses the packet header and determines the output link. The packet is segmented into smaller fixed size cells similar to an ATM cell, and the output link information is recorded onto the cell header. The segmented cells are sent to the switching module to be routed to the destination line interface modules. The cells are reassembled there and transmitted onto the output link, which is also connected to the line interface module. A line interface module has input and output links for outer communication links and ingress and egress connections to and from the switching module. Output queue management of the switching fabric is the focus of attention, as shown in [12]. At the end of the paper, we will examine another router architecture more recently developed, but we mainly concentrate on the output queuing architecture.

In the output queuing architecture, the latency through the switch fabric is fixed and very small. Thus, when many cells from different input links are switched to the same output link, contention for the egress connection between the switching module and the destination line interface module takes place. The speed of the connection should be equivalent to the aggregation of the input link rates. This applies to any egress connection between the switching module and any line interface module. Therefore, we can suggest a common egress connection between the switching module and the line interface modules. The line interface module should perform filtering of unnecessary cells before reassembly. Thus, the line interface module has ingress and egress parts with little interaction as shown in Fig. 1. In this architecture, the output queue is maintained in the egress part so that the switching module can be made simpler.

Sorting the timestamps should be performed in the egress part of the destination line interface module, and the queuing processor there should keep up with the peak rate. In order to lower the load of the queuing processor, some portion of the work of the queuing processor at the egress line interface module should be distributed to the ingress line interface modules. The timestamps can be calculated in the ingress line interface modules. However, some PFQ algorithms cannot be used because the system virtual time obtained from the state of the output queue is not available in the ingress line interface modules.

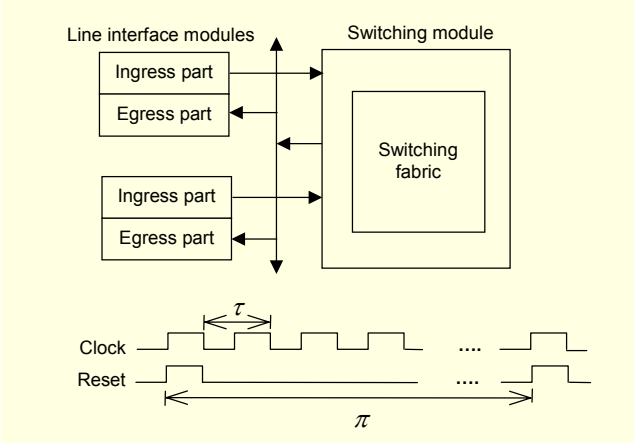

Fig. 1. Router architecture.

Round-robin algorithms represented by DRR also face a similar problem. The queuing processor should be aware of all the details of each flow and be involved in the signaling procedures. This makes the round-robin algorithms unsuitable for our architecture. If the timestamp of the PFQ algorithm can be calculated in the ingress part without knowledge of the output queue, we can make out a more efficient distributed algorithm. We need something new replacing the system virtual time for a distributed architecture.

We have studied the use of a periodic clock for traffic management [13], and reached an idea of sampling a counter triggered by a periodic clock signal to obtain the time reference. We call this type of time reference *hardware time*, in contrast to the *virtual time* of previous PFQ algorithms. The starting point of our contribution in this paper is providing a common clock signal to all the line interface modules as hardware time reference throughout the entire routing system, as shown in Fig. 1, instead of system virtual time. By doing so, we can achieve a distributed PFQ algorithm lowering the load of the queuing processors suitable for real-time traffic management of high speed links.

We suggest two ways to lower the demand on computing resources of the queuing processor: memory to keep the state of each connection, and processing power to calculate the system virtual time and the timestamp for each packet. We also lower the demand on computing resources for sorting the timestamps by improving the algorithm. First, we devise a scalable algorithm, DDQ, which can utilize hardware time and distributed processing.

In the previous paper [14] on DDQ, we presented a brief idea and desirable router architecture. Further developing mathematical models for the algorithm is the major goal of this paper. The main part of this paper is to provide analytical models and specific figures for the algorithm and architecture. These models include delay bounds of a single processor system and distributed architecture, the number of bins, and the effect of time difference between modules. The mathematical proofs on delay bounds and number of bins show the feasibility of the algorithm. These models justify the algorithm and confirm its competitiveness. We conclude that the ultimate goal of the DDQ application should be the CIOQ router, and a comparison to other PFQ algorithms for that application is made. We also notice that DDQ requires modification for CIOQ, and this modification will be our future work.

III. Basic DDQ Algorithm

The basic DDQ algorithm is based on timestamp calculation using the hardware time and a notion of the resolution of time. Like other PFQ algorithms, we calculate the timestamp of each real-time traffic packet from the packet size and the reserved rate [15] of the flow, and sort packets of the flows sharing an outgoing link in increasing order of the timestamps. Then, the packets are transmitted in that order, and the timestamp becomes equivalent to the transmission deadline or delay. In this way, we can provide delay bounds to real-time traffic.

Apart from previous PFQ, DDQ is based on the hardware time obtained by reading a counter and the fact that a very small difference between timestamps of each pair of packets would make sorting time consuming but meaningless. The clock signal to the counter is of a fixed rate and very high frequency. Thus, the hardware time is very accurate and has potential to replace the system virtual time of the previous PFQ.

If the difference of timestamps of two packets is very small, we had better ignore the difference and make the traffic control algorithm more efficient. Thus, we propose to digitize the timestamp, and allocate a first-in first-out (FIFO) queue for a digitized timestamp or delay. Then, the sorting is replaced with finding the digitized delay, and the traffic control algorithm becomes more efficient. The data structure representing the digitized delay is called a *bin*, following the similar work, bin sort fair queuing. We adopt the following notations for each flow to describe the DDQ algorithm:

$R.f$ forwarding rate (b/s) reserved for flow f
$L.f.i$ length (bits) of packet $p.f.i$
$A.f.i$ arrival time of packet $p.f.i$
$\overline{A}.f.i$ arrival time of packet $p.f.i$ in hardware time
$S.f.i$ service start time of packet $p.f.i$
$\overline{S}.f.i$ digitized start time of packet $p.f.i$
$J.f.i$ block bin index of packet $p.f.i$
$F.f.i$ timestamp of packet $p.f.i$
$\overline{F}.f.i$ digitized timestamp of packet $p.f.i$
$I.f.i$ ready bin index of packet $p.f.i$
$E.f.i$ exit time of packet $p.f.i$
$C.s$ speed of scheduler s

Inherently, DDQ is similar to VC. However, VC is based on the system virtual time representing the state of the output queue [5]. It is described by the following equations:

$$S.f.1 = A.f.1,$$
$$S.f.i = \max(A.f.i,\ F.f.(i-1)), \quad \text{for all } i, i > 2,$$
$$F.f.i = S.f.i + L.f.i/R.f, \text{ and}$$
$$E.f.i \le F.f.i + L_{\max}/C.s.$$

As the first step, we propose the pseudo VC service discipline described below using hardware time instead of system virtual time:

$$S.f.1 = \overline{A}.f.1,$$
$$S.f.i = \max(\overline{A}.f.i, F.f.(i-1)), \quad \text{for all } i, i > 2, \text{ and}$$
$$F.f.i = S.f.i + L.f.i/R.f.$$

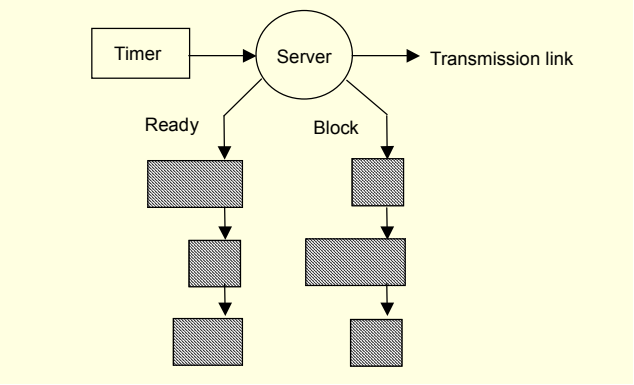

Fig. 2. Generic packet scheduler.

Figure 2 shows a generic packet scheduler [16]. The packet scheduler manages two lists of packets. The ready list is the sorted list of packets and is work-conserving. Thus, once a packet is queued in the ready list, the packet is transmitted as soon as possible. The block list is for rate control to reduce the buffer requirement in the downstream router. Actually, we have considered the per-flow queue architecture for a single processor system, but we found that in a distributed architecture a single sorted queue requires less information for each packet. Thus, we use single shared queues for the ready list and block list, respectively.

1. Ready List of DDQ

Digitization is another key idea to understand DDQ. We use the ceiling function to digitize the timestamp for the ready list.

The ceiling function of x, $\lceil x \rceil$, is the smallest integer not less than x. First, we divide the timestamp with digitization resolution, δ, and then take the ceiling function to get the digitized timestamp or digitized delay, $\overline{F}.f.i$:

$$\overline{F}.f.i = \delta \lceil F.f.i/\delta \rceil.$$

We use a 1 ms rule in this paper, which means that the digitization resolution, δ, is 1 ms. This rule is based on the assumption that 1 ms of additional delay would not affect the end-to-end QoS guarantee seriously.

Now, we can sort the ready list by the digitized timestamps. If the digitized timestamps of two packets are different, the transmission order is the same as when the ready list is sorted

by the virtual finish time. If the digitized timestamps are the same, the packets are served in a FIFO manner. Thus, we can queue all the packets of the same digitized delay to a data structure called a *bin* so that we can save time for sorting and devise a sorted list of bins, as shown in Fig. 3. A bin is shown as a box with the digitized delay index $\lceil F.f.i/\delta \rceil$ inside. Each bin has an integer variable of the index and a FIFO queue of packets. When a bin does not have any packet queued to it, the bin is not included in the ready list. When a packet is inserted into the ready list, the digitized delay index is calculated and the state of the bin is tested. If the bin is already in the ready list, the packet is simply appended to the bin. The state of a bin depends on the packet pointer. If the packet pointer is zero, the bin is in IDLE state and should be inserted in the ready list. This insert operation could be time consuming.

Thus, we propose to limit the number of ready bins. The limit comes from the maximum time for packetization at the source. For example, the first byte of the audio data should wait at the source until the whole audio packet is built. Some audio applications generate a packet every 10 ms. If the packetization period is too large, the delay at the source would also become too large, and it would affect the quality of the traffic. On the contrary, if the packetization period is too small, the source should transmit packets too often, and the protocol overhead caused by the packet header would be unacceptable. We believe that the packetization period should be standardized eventually and the routing system should be optimized for the standard. We propose the maximum packetization period (Δ) be 10 ms in this paper.

If we confine the maximum packetization period to 10 ms, we need at least 10 ready bins with 1 ms of digitization resolution. The maximum number of bins will be derived later in this paper. The ready bin variables can be declared as an array variable with I_{max} entries and can be cyclically reused [18]. The index to the corresponding ready bin, *I. f. i*, is obtained by the following modulus operation:

$$I.f.i = \lceil F.f.i/\delta \rceil \bmod I_{max} .$$

Similarly, we can define the base ready bin index, $\bar{I}$, by the following modulus operation of the hardware time, T:

$$\bar{I} = \lceil T/\delta \rceil \bmod I_{max} .$$

The base bin is the lowest bin, and potentially the ready list starts with the base bin. However, if the base bin does not have any packet queued to it, the top bin of the ready list could be different from the base bin. This can happen when the time reaches the millisecond boundary.

Comparison of two bin indices in a ready list is complicated due to cyclic reuse of index numbers. In order to lower the complexity, we use the unfold $(x, \bar{I}, I_{max})$ function. This function gives the original bin index value, if the bin index, x, is not less than the base bin index. Otherwise, it gives the ready bin index added by the modulus.

In the following number sequence, the ready bin index of a packet is 15 (= x) and is compared to a bin with bin index 2 (= y). Assume that the modulus is 16: I_{max}=16. In the first sequence below, the base bin index of the ready list is 14.

13	14	15	0	1	2	3	4
	$\bar{I}$	x			y		

compare (15, 2, 14, 16)
= unfold (15, 14, 16) – unfold (2, 14, 16)
= 15 – (16+2) = -3

In the second sequence, the base bin index of the ready list is 0.

13	14	15	0	1	2	3	4
		x	$\bar{I}$		y		

compare (15, 2, 0, 16)
= unfold (15, 0, 16) – unfold (2, 0, 16)
= 15 – 2 = 13

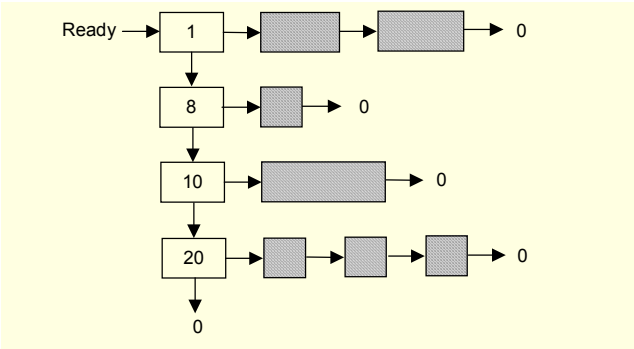

Fig. 3. Ready list of DDQ.

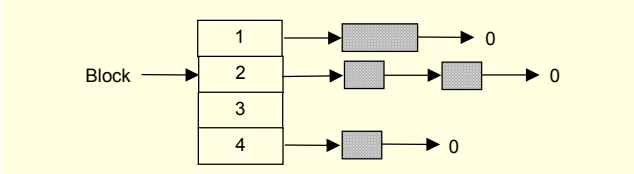

Fig. 4. Block list of DDQ.

2. Rate Control of DDQ

We use the interrupt model for rate control. The periodic event can also be implemented by polling the real-time clock, but under a multi-programming environment, the interrupt model is more accurate. We require a timer interrupt every 1 ms. Most computers use a 10 ms interrupt period for managing

time, but a 1 ms interrupt interval is also feasible with some overhead. The resolution of the periodic events is ε and could be the same as the resolution of the ready list, δ. In this paper, we use the 1 ms rule: $\delta = \varepsilon = 1$ ms. The periodic interrupt event takes place at every integer multiple of the resolution, $T = \varepsilon, 2\varepsilon, 3\varepsilon, \cdots$.

The block list is also implemented with an array variable of bins, as shown in Fig. 4. However, different from the ready list, the block list does not need to be a linked list of bins. When an interrupt is generated at $T = n\varepsilon$ (n is a non-negative integer), packets queued to the bin of index n are released at the interrupt event. Because the block list is non-work conserving, there could be a bin without any packet queued to it. This means that there is no packet to be released at the corresponding interrupt event.

The difference between the current time and the timestamp of the previous packet needs to be removed in the block list. However, in the DDQ algorithm, the difference is not completely removed. Only the difference can be maintained below the digitization resolution when the packet is released from the block list. The block bin index *J. f. i* is as follows:

$$\begin{aligned} &\overline{S}.f.i = \varepsilon \lfloor S.f.i/\varepsilon \rfloor, \\ &J.f.i = \lfloor S.f.i/\varepsilon \rfloor \bmod J_{\max}. \end{aligned} \tag{1}$$

When a packet arrives at a DDQ scheduler, a block bin index is calculated for the rate control test. The test also includes calculation of the base block bin index $\overline{J} = \lfloor \overline{A}.f.i/\varepsilon \rfloor \bmod J_{\max}$. If the two indices are the same, the rate control is not necessary. Otherwise, the packet is inserted into the block bin *J. f. i.*

The packets in the block bins *J. f. i* are released from the block list, when an interrupt event takes place at digitized service start time: $T = \overline{S}.f.i = \varepsilon \lfloor S.f.i/\varepsilon \rfloor$. At this time, all the packets queued in the block bin *J. f. i* are released and inserted into the ready list for transmission. This operation can take an unacceptably long amount of time, but can be made handy if all the released packets have the same timestamp value and thus are queued to the same ready bin. This can be achieved by standardizing the timestamp size. This type of standardization will be discussed in another paper. We concentrate on the similar features with other PFQ algorithms, developing the basis for future work.

3. Delay Bounds

In this section, we develop a relatively simple analytical model for the delay bounds of the DDQ algorithm. This model will be further developed for the distributed DDQ algorithm in the next section.

In a packet switched network, a link is modeled after the on-off server model. When there is no packet in the ready list, the link is idle and does not transmit any information. This state corresponds to the off-state and we call this an *idle period*. When there is at least one packet in the ready list, the link transmits the packet at the speed of the link, *C.s*. This state corresponds to the on-state and is called a *busy period*. The packet is transmitted within *L. f. i/C. s.* The busy period here means the transmission of real-time traffic only. We can obtain the delay bounds of the DDQ algorithm using the idea of a pseudo VC.

Lemma 1. The delay bounds of the pseudo VC scheduler with single flow established exist.

Proof. Assume that packet *i* of flow *f* arrives at the scheduler and a busy period starts. The service start time of packet *i* is larger than or equal to the packet arrival time, $\overline{A}.f.i$, as follows:

$$\begin{aligned} &S.f.1 = \overline{A}.f.1, \\ &S.f.i = \max\left(\overline{A}.f.i, F.f.(i-1)\right), \quad \text{for } i,\ i \ge 2, \\ &S.f.i \ge \overline{A}.f.i \quad . \end{aligned}$$

The timestamp of the pseudo VC scheduler is as follows:

$$\begin{aligned} &F.f.i = S.f.i + L.f.i/R.f \ , \\ &F.f.i \ge F.f.(i-1) + L.f.i/R.f \ , \quad \text{for } i,\ i \ge 2 \ . \end{aligned} \tag{2}$$

Assume that packet *j* of flow *f* arrives at the scheduler while the busy period continues. Here, *j* is larger than *i* $(j > i)$. The timestamp of packet *j* is determined by

$$\begin{aligned} &F.f.1 = \overline{A}.f.1 + L.f.1/R.f, \\ &F.f.j = \max\{\overline{A}.f.j, F.f.(j-1)\} + L.f.j/R.f \ \text{ for all } j,\ j \ge 2, \\ &F.f.j = F.f.(j-1) + L.f.j/R.f, \text{and} \\ &F.f.j = F.f.i + \sum_{k=i+1}^{j} L.f.k/R.f. \end{aligned}$$

From (2), we have

$$\begin{aligned} F.f.j &= \overline{A}.f.i + \sum_{k=i}^{j} L.f.k/R.f, \\ E.f.j &= A.f.i + \sum_{k=i}^{j} L.f.k/C.s. \end{aligned} \tag{3}$$

In order to derive the exit time, we need to know in depth the behavior of the hardware time by the periodic clock with period τ, as shown in Fig. 1. We consider the real time and the hardware time synchronized at each rising edge of the clock signal. There exists a sampling error, which is always less than the period τ. Therefore, we have the following exit time:

$$E.f.j < \overline{A}.f.i + \tau + \sum_{k=i}^{j} L.f.k/C.s \ .$$

With a delay bound condition known as rate admission control ($C.s \geq R.f$), we have the following delay bounds:

$$E.f.j < F.f.j + \tau .$$

In a packet switched network, we cannot stop the transmission of a packet in the middle of transmission. Thus, the busy period of real-time traffic can start after transmission of the best-effort traffic. As the size of a packet is finite, we can have the delay bounds, and the additional delay is also limited by the transmission delay of the largest packet.

$$E.f.j < F.f.j + \tau + L_{\max}/C.s \tag{4}$$

Because the sampling error is less than the clock period and the period is very small compared to the timestamp size, we can ignore the effect of the sampling error. The timestamp size is usually 10 ms, but the clock period is 1 μs typically.

Definition 1. The amount of traffic of flow *f* arriving since the busy period start time *A. f. i* and having a timestamp less than or equal to *F.f.j* is defined as the *traffic contribution* of flow *f* by the pseudo VC server and is denoted as *W.f.*(*A. f. i, F.f.j*).

From (3), the traffic contribution of flow f from i to j is $\sum_{k=i}^{j} L.f.k$.

$$\sum_{k=i}^{j} L.f.k \leq R.f(F.f.j - A.f.i),$$

$$W.f.(A.f.i, F.f.j) = \sum_{k=i}^{j} L.f.k \leq R.f\left(F.f.j - A.f.i\right). \tag{5}$$

Corollary 1. The delay bounds of the DDQ scheduler with single flow established exist.

Proof. Similarly, we can obtain the delay bounds of the DDQ algorithm. We assume that a busy period starts with packet *i* of flow *f*, and packets from *i* to *j* arrive at the ready list of a DDQ scheduler during the same busy period. The timestamp of packet *j* of flow *f* is $\overline{F}.f.j = \delta\lceil F.f.j/\delta \rceil$.

Equation (3) also applies to the DDQ algorithm. Thus, we have

$$\overline{F}.f.j = \delta\lceil F.f.j/\delta \rceil \geq F.f.j \geq \overline{S}.f.i + \sum_{k=i}^{j} L.f.k/R.f. \tag{6}$$

From (1) and (4), we have delay bounds by the DDQ scheduler of less than or equal to the digitized timestamp:

$$E.f.j = \overline{S}.f.i + \sum_{k=i}^{j} L.f.k/R.f,$$
$$E.f.j < \delta\lceil F.f.j/\delta \rceil + \tau = \overline{F}.f.j + \tau.$$

In a packet switched network, a busy period of real-time traffic can start after the transmission of a best-effort traffic packet.

$$E.f.j < \overline{F}.f.j + L_{\max}/C.s + \tau . \qquad \square$$

Theorem 1. The delay bounds of the pseudo VC scheduler with multiple flows established exist.

Proof. Now we consider other flows sharing the scheduler. Assume that there is flow g with the reserved rate of $R.g$ among the flows sharing the scheduler, and packet j of flow g arrives at the scheduler during the same busy period starting with packet i of flow f. The pseudo VC scheduler stamps the flow g packets in the same way as flow f. Assume that there are N flows established along the scheduler including flows f and g. The exit time of packet j of flow g is

$$E.g.j \leq A.f.i + \sum_{n=1}^{N} W.n.(\overline{A}.f.i, F.g.j)/C.s + L_{\max}/C.s,$$
$$E.g.j \leq A.f.i + \sum_{n=1}^{N} \left(F.g.j - \overline{A}.f.i\right)R.n/C.s + L_{\max}/C.s.$$

With a delay bound condition known as rate admission control $\left(\sum_{n=1}^{N} R_n \leq C.s\right)$, we have the following delay bounds:

$$E.g.j \leq F.g.j + L_{\max}/C.s + \tau . \tag{7}$$

Equation (7) is ambiguous if a packet k of flow h with a timestamp larger than *F. g. j* is transmitted earlier than packet *j*. This can happen if packet *k* arrives earlier than the arrival time *A. g. j.* This means that no more packets whose timestamp is less than or equal to *F. g. j* exist in the ready list at time *E.h.k-L.h.k/C.s.* Therefore, the traffic contribution of all the flows decreases because there will be no more packets whose arrival time is less than or equal to *E.h.k-L.h.k/C.s* and the timestamp is less than or equal to *F. g. j.* The exit time of packet *k* of flow *g* is upper-bounded as follows:

$$E.g.j \leq E.h.k - L.h.k/C.s + \sum_{n=1}^{N} W.n.\{(E.h.k - L.h.k/C.s), F.g.j\}/C.s$$
$$- W.g.\{(E.h.k - L.h.k/C.s), F.g.j\}/C.s + L.h.k/C.s,$$

$$E.g.j < F.g.j + L_{\max}/C.s .$$

Therefore, (7) still holds. The transmission of packet *k* is equivalent to the idle state for packet *j*. If a packet with a timestamp less than or equal to *F. g. j* arrives and starts transmission, the state gets the equivalent to a busy period start. Then, (7) holds for any packet. $\square$

Theorem 2. The delay bounds of the DDQ scheduler with multiple flows established exist.

Proof. From Theorem 1, we can derive the delay bounds of the DDQ scheduler. Assume that packet *i* of flow *f* arrives at a DDQ scheduler at $A.f.i$ and gets a rate control test. The packet is released from the block list at $\overline{S}.f.i$ ($\geq A.f.i$). Insertion of the packet into the ready list is equivalent to sorting packets in the increasing order of digitized timestamps. Now considering other flows, assume that there is flow *g* with the reserved rate of $R.g$ among N flows sharing the scheduler, and packet *j* of flow *g* arrives at the ready list during the same busy period starting with packet *i* of flow *f*. From the definition of traffic contribution, the exit time of packet *j* of flow *g* is as follows:

$$E.g.j \leq \overline{S}.f.i + \sum_{n=1}^{N} W.n.\left(\overline{S}.f.i, \overline{F}.g.j\right)/C.s + L_{\max}/C.s + \tau .$$

With a delay bound condition known as rate admission control $\left(\sum_{n=1}^{N} R.n \leq C.s\right)$, we have the following delay bound:

$$E.g.j \leq \overline{F}.g.j + L_{\max}/C.s + \tau$$

The above equation gets ambiguous if packet *k* of flow *h* with a timestamp larger than $\overline{F}.g.j$ is transmitted earlier than packet *j*. This can happen if packet *k* arrives earlier than the service start time $\overline{S}.g.j$. This means that no more packets whose timestamp is less than or equal to $\overline{F}.g.j$ exist in the ready list at time $E.h.k - L.h.k/C.s$. This state is equivalent to the idle state for packet *j*. If a packet with a timestamp less than or equal to $\overline{F}.g.j$ arrives and starts transmission, the state gets the equivalent to a busy period start. Then, the above equation holds for any packet. See Theorem 1. □

4. Number of Bins

We proposed that the DDQ scheduler use a finite number of bin indices for ready and block lists. This was made possible by the rate control in the upstream scheduler and the source node. Here, we study the maximum numbers of ready and block bins, $I_{\max}$ and $J_{\max}$.

Deriving $I_{\max}$ is straightforward because the maximum packetization period is limited to Δ (10 ms). We also assume that the digitization resolution for the ready list is δ (1 ms). Thus, we need at least 10 ($=\Delta/\delta$) bins to store the various sizes of packets and one more bin due to imperfect rate control. Additionally, we have to consider that the waiting time in the ready list is $L.f.i/R.f + L_{\max}/C.s + \tau$. Thus, when the hardware time becomes $L.f.i/R.f$, some packets can still remain in the ready bin whose index is less than the base bin. This means that we need one more bin for the packets left behind the hardware time. The ready list can start with the bin indexed by $\left(\lceil T/\delta \rceil + I_{\max} - 1\right) \bmod I_{\max}$, where T is the hardware time. Therefore, the maximum number of bins required for the ready list is 12: $I_{\max}$ =12. All the successive packets are stored in the block list.

However, deriving the maximum number of bins in the block list is complicated. We show this in the following theorem.

Theorem 3. The number of bins for the block list of a DDQ scheduler is upper-bounded by 12:

$$\lfloor \{F.f.i.t - A.f.(i+1)t\}/\delta \rfloor \leq 11 = J_{\max} - 1 .$$

Proof. Deriving $J_{\max}$ is very important in configuring the DDQ scheduler. To obtain $J_{\max}$, we have a scenario. First of all, we have two schedulers, *s* and *t*. Scheduler *s* is the upstream scheduler and *t* is the downstream scheduler. If scheduler *s* transmits a packet, it arrives at scheduler *t*. There is a link between the two schedulers and it gives a propagation delay to each packet. However, we can ignore the propagation delay for our analysis because the delay is fixed and can be added when it is necessary. Considering packet *i* of flow *f* departing scheduler *s* at $E.f.i$, it arrives at scheduler *t* at $E.f.i$. The arrival time at scheduler *t* of the packet is denoted as $A.f.i.t$. Thus, we assume $E.f.i = A.f.i.t$ in the scenario.

We are interested in the maximum number of block bins for scheduler *t*. Thus, in the scenario we push as many packets as possible into scheduler *t* and postpone the transmission of the packets from scheduler *t* as long as possible. Then, we can maximize the amount of packets stored in scheduler *t*. Because the number of packets in the ready list is limited, the rest of the packets are all stored in the block list. Eventually, the amount of packets arriving at the scheduler will be the same as the amount of packets departing the scheduler. Thus, we can get the maximum number of block bins required for scheduler *t*.

We assume that several packets arrive at scheduler *s* at the same time: $A.f.1 = A.f.2 = A.f.3 = A.f.4 = A.f.i$. The first and second packets are transmitted back-to-back in order to push as many packets into scheduler *t* as possible.

$$\begin{aligned} A.f.1.t = E.f.1 &< \delta\left\lceil \left(\overline{A}.f.1 + L.f.1/R.f\right)/\delta \right\rceil + L_{\max}/C.s + \tau, \\ A.f.2.t = E.f.2 &= E.f.1 + L.f.2/C.s \end{aligned} \tag{8}$$

The third packet cannot be transmitted immediately after the second packet from scheduler *s*. It should wait in the block list until it is released at $\overline{S}.f.3$.

$$\begin{aligned} A.f.3.t &= E.f.3 < \overline{S}.f.3 + L.f.3/C.s, \\ A.f.3.t &= \varepsilon\lfloor F.f.2/\varepsilon \rfloor + L.f.3/C.s, \\ F.f.2 &= \overline{A}.f.1 + L.f.1/R.f + L.f.2/R.f . \end{aligned}$$

We are interested in the case where the packets since the

third are at least as large as δ in the service time. Thus, the depart time is achieved as the transmission time plus the rate control time.

$$A.f.3.t = \varepsilon\left\lfloor\left(\overline{A}.f.1 + \sum_{k=1}^{2} L.f.k/R.f\right)\Big/\varepsilon\right\rfloor + L.f.3/C.s,$$
$$A.f.i.t = \varepsilon\left\lfloor\left(\overline{A}.f.1 + \sum_{k=1}^{i-1} L.f.k/R.f\right)\Big/\varepsilon\right\rfloor + L.f.i/C.s \quad (9)$$

At scheduler *t*, transmission of the first packet should be delayed as long as possible in order to give back pressure to the successive packets. Thus, the size of the first packet should be the largest: $L.f.1/R.f = 10\delta$.

$$F.f.1.t = \overline{A}.f.1.t + L.f.1/R.f = \overline{A}.f.1.t + 10\delta,$$
$$F.f.i.t = \max\{\overline{A}.f.i.t, F.f.(i-1)t\} + L.f.i/R.f \text{ for all } i,\ i \geq 2$$

We are interested in the case where maximal rate control is required.

$$A.f.i.t < F.f.(i-1),$$
$$F.f.i.t = F.f.(i-1)t + L.f.i/R.f$$
$$= F.f.1.t + \sum_{k=2}^{i} L.f.k/R.f$$
$$= A.f.1.t + \sum_{k=1}^{i} L.f.k/R.f$$

From (8), we have

$$F.f.i.t = \delta\left\lceil\left(\overline{A}.f.1 + L.f.1/R.f\right)/\delta\right\rceil + L_{\max}/C.s + \sum_{k=1}^{i} L.f.k/R.f$$
$$= \delta\left\lceil\overline{A}.f.1/\delta\right\rceil + 10\delta + L_{\max}/C.s + \sum_{k=1}^{i} L.f.k/R.f. \quad (10)$$

From (9) and (10), we have the following for $i \geq 2$:

$$F.f.i.t - A.f.(i+1)t$$
$$= \delta\left\lceil\overline{A}.f.1/\delta\right\rceil + 10\delta + L_{\max}/C.s + \sum_{k=1}^{i} L.f.k/R.f$$
$$-\varepsilon\left\lfloor\left(\overline{A}.f.1 + \sum_{k=1}^{i} L.f.k/R.f\right)\Big/\varepsilon\right\rfloor - L.f.(i+1)/C.s.$$

Assume that $\sum_{k=1}^{i} L.f.k/R.f = n\delta + \alpha$, where *n* is an integer and $0 \leq \alpha < \delta = \varepsilon$.

$$F.f.i.t - A.f.(i+1)t$$
$$= \delta\left\lceil\overline{A}.f.1/\delta\right\rceil + n\delta + \alpha + 10\delta - \delta\left\lfloor\left(\overline{A}.f.1 + n\delta + \alpha\right)/\delta\right\rfloor$$
$$+ \{L_{\max} - L.f.(i+1)\}/C.s$$

Also assume that $\overline{A}.f.i = m\delta + \beta$, where *m* is an integer and $0 \leq \beta < \delta$.

$$F.f.i.t - A.f.(i+1)t = m\delta + \delta\left\lceil\beta/\delta\right\rceil + 10\delta + \alpha - m\delta$$
$$-\delta\left\lfloor(\alpha+\beta)/\delta\right\rfloor + \{L_{\max} - L.f.(i+1)/C.s\},$$

The above equation gives the maximum value when $\alpha + \beta < \delta$, $\alpha \neq 0$, and $\beta \neq 0$.

$$F.f.i.t - A.f.(i+1)t = 11\delta + \alpha + \{L_{\max} - L.f.(i+1)/C.s\},$$

$$\left\lfloor\{F.f.i.t - A.f.(i+1)t\}/\delta\right\rfloor \leq 11.$$

The quantity $F.f.i.t - A.f.(i+1)t$ is the time for rate control. By taking the index form of the quantity, we can get the potential number of block bins. If the rate control time is misaligned from the interrupt event for rate control, we need one more interrupt event. Thus, we can conclude that the number of block bins is upper-bounded by 12:

$$I_{\max} = J_{\max} = 12.$$

□

The maximum number of bins results from the finite packet size and digitization resolutions δ and ε . We can choose different values for these variables, but we reached this conclusion after various simulations and debate. If there is a different idea on these figures, follow the guideline here to obtain different values.

Because we use a finite number of bins, it is very simple to find a bin by its index. Otherwise, it could be very time consuming to parse all the bins in the ready or block lists. The benefit of having a finite size of a packet simplifies the scheduler design. We are also developing a large distributed routing system architecture with QoS features. Having finite packet size is also very helpful for real system development.

The previous result is based on the assumption that the upstream scheduler *s* is also a DDQ scheduler with a more flexible block list. This type of scheduler could be an access router. The core schedulers have a finite number of bins and finite buffer memory. In a multi-hop network, all the down stream schedulers should check the block bin number of each packet. If the block bin number exceeds the limited number, the packet is discarded so that the following downstream schedulers can operate with a finite number of bins.

IV. Large Scale QoS Router

Routers with many line interfaces have a modular architecture. The entire routing system is too large to make it as a whole piece. Hence, each module is made as a separate part

and assembled for operation. Also, each module has a separate processor, and the routing system is made as a distributed system, as shown in Fig. 1.

Each communication link is connected to a separate line interface module. Each packet arriving at the ingress part of the line interface module is forwarded to the switching module after being stamped by the DDQ stamping server. For this purpose, in the ingress part of the line interface module there is a stamping server of the DDQ scheduler. The rest of the DDQ scheduler is in the egress part of the output line interface module. As mentioned in the previous section, the stamping server is independent of the sorting server. Thus, we can separate the DDQ scheduler into a stamping server in the ingress line interface module, and a sorting server in the egress line interface module. In this way, we can distribute the workload of the DDQ algorithm into multiple processors in the line interface modules. Thus, we can raise the performance of the scheduler without raising the performance of each processor. This is the benefit of the distributed DDQ algorithm.

1. Ideal Distributed DDQ

As the first step of developing the architecture, we can consider an ideal routing system consisting of line interface modules and switching module. The hardware time of each module is identical, and transmission delay of a packet between any pair of modules is zero. This ideal model is helpful in presenting the distributed system architecture.

A packet is segmented into smaller cells with additional header used only in the router; the internal cell format is shown in Fig. 5. Each cell carries an output port number, ready bin index $(I.f.i.a)$, and block bin index $(J.f.i.a)$ in the header, calculated by the ingress processor on the line interface module, for example, *a*.

The ingress processor on the line interface module looks up the forwarding information base (FIB) for the incoming packet, determines the output port number, and stamps it on the temporary header. A conventional routing system needs only the output port number, but our QoS routing system requires a ready bin index and block bin index as well. These indices are

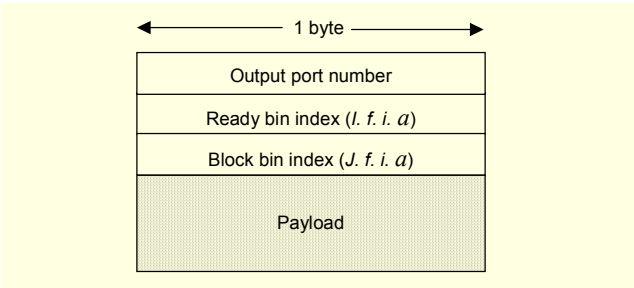

Fig. 5. Internal cell format at line interface module *a*.

computed by the DDQ algorithm, and the cells are forwarded to the switching module.

Once the cells are forwarded to the switching module and routed to the output line interface module, the egress processor tests the rate control. We can classify the test results by a comparison of the base block bin index $\bar{J} = \lfloor A.f.i/\varepsilon \rfloor \bmod J_{\max}$ to the block bin index in the header. Suppose that a packet is transferred from line interface module *a* to line interface module *b* via the switching module. The arrival time at line interface module *a* is $A.f.i.a$, and the block bin index in the header seen in line interface module *a* is $J.f.i.a$.

Case i) $\bar{J}.b = J.f.i.b$

When the packet arrives at line interface module *b*, the arrival time $A.f.i.b$ can be obtained by reading the hardware time. Obviously, the two arrival times $A.f.i.a$ and $A.f.i.b$ are different. The effect of the difference will be discussed later in this paper. Under the ideal distributed DDQ, the two arrival times are assumed to be the same. This leads to the same base block bin indices:

$$\bar{J}.b = \bar{J}.a = J.f.i.a = J.g.i.b \ .$$

In this case, rate control is not necessary, and the packet is immediately inserted into the ready list. Actually, the queuing processor does not need to read the hardware time at every event of packet arrival if the counter generates interrupts at $T = n\varepsilon$, where n is non-negative integer. Then, the queuing processor reads the hardware time only when the interrupt is generated. This reduces the burden due to the memory cycle or IO interface cycle resulting from the hardware time read. We can get the base block bin index $\bar{J}.b$ without knowing the exact arrival time $A.f.i.b$. The base block bin index in module *b* can be stored in the register of the queuing processor.

Case ii) $\bar{J}.b \neq J.f.i.b$

The arrival time at module *b* is greater than the arrival time at module *a*. Here, we assume again that the two arrival times are the same. As the base block bin index in module *b* is different from the block bin index on the header, $\bar{J}.b = \lfloor A.f.i.b/\varepsilon \rfloor \bmod J_{\max} \neq J.f.i.b$, the packet is inserted into the block bin $J.f.i.b$. The time increases, and eventually the base block bin index becomes $J.f.i.b$. Then, the interrupt from the counter is generated, and packets queued to the block bin are released from the block list and inserted into the ready list according to the ready bin index $I.f.i.b$ on the temporal packet header.

2. Effect of Different Time at Each Module

We presented a detailed operation of the distributed DDQ

under the assumption of identical time at each module and no transmission delay between any pair of modules. However, it is impossible to synchronize the time perfectly at every module, and the transmission delay is not zero. Does that mean a distributed DDQ cannot give delay bounds to real-time flows? We would like to show that there exist delay bounds with ignorable tolerance in spite of the inconsistency in time at each module and the different transmission delay caused by different packet sizes. First of all, relatively accurate time synchronization can be achieved by common clock distribution as shown in Fig. 6.

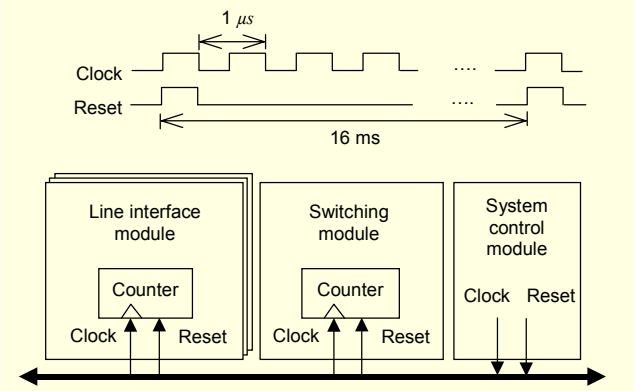

Fig. 6. Clock distribution for distributed DDQ.

Common clock distribution is a simple idea to synchronize the time of all the modules. We need two clock signals generated in the system control module and distributed to all the other modules. One signal is a 1 μs periodic clock signal. A finer clock was considered, but is meaningless for our traffic control purpose. One microsecond resolution seems accurate enough compared to a millisecond digitization resolution.

The other signal is a 16 ms reset signal. This reset signal makes each module hot swappable. Each module can be installed and replaced while the routing system is operating. The initial value of the counter in a new module is not known, but it is reset to zero in 16 ms. Thus, we can obtain time synchronization. The reset interval, 16 ms, comes from the number of bins in the switching module. Theorem 3 gives the number of block bins as 12. We add four more bins to both the ready list and block list. This is for a safety margin before the lowest and after the largest bins. We add two more bins just to make the calculation simpler.

Assume that the current time of module b is 6 ms: $T.b = 6$. Then, the block bin index from any line interface module $J.f.i.a$ should be larger than or equal to 6. However, there can be a line interface module slightly later than module b, and this can give a packet whose block bin index is 5: $J.f.i.a = 5$. Then, it is difficult to know if this packet requires rate control or not. This packet does not require rate control. This means that a safety margin is necessary and we need to modify the rate control test of the ideal distributed DDQ algorithm. If $J.f.i.b - \bar{J}.b = 0,1,or -15$, rate control is not necessary. Otherwise, rate control is necessary and the packet is queued to the block bin index $J.f.i.b$. This is based on the assumption that the time lag is very small. Because the clock signal comes from the same system control module, the time difference between modules would be hundreds of nanoseconds. Therefore, a 1 millisecond safety margin is large enough.

Similarly, the ready list also needs correction. The ready list can start with the ready bin whose index is

$$I.f.i.b = I.f.i.a - 1 \quad \text{or} \quad I.f.i.a + 15.$$

We have the extended maximum bin numbers for the distributed system making full use of 4 bits from the ready bin index field: $I_{\max}.D = J_{\max}.D = 16$.

3. Delay Bounds of Distributed DDQ

The effect of distributed processing of the DDQ algorithm on the delay bounds is an important factor for justifying the algorithm. The problem is the difference of time between modules and transmission delay from a line interface module to another line interface module. In Fig. 7, packet i of flow f arrives at the line interface module a at $A.f.i.a$ and is forwarded to module b. The arrival time at module b is $A.f.i.b$ by the module b time. The difference between the two arrival times incorporates the time difference between the two modules and transfer delay. We define the difference as θ. The time difference between the two modules is relatively small, hundreds of nanoseconds. We assume that the time difference of any pair of modules is within 200 ns. Thus, if there is another line interface module c, the time difference between module a and module c is also within 200 ns.

$$-200 \text{ ns} \le T.b - T.a \le 200 \text{ ns}$$
$$-200 \text{ ns} \le T.c - T.a \le 200 \text{ ns}$$
$$-200 \text{ ns} \le T.c - T.b \le 200 \text{ ns}$$

The transfer delay incorporates the transmission delay and the switching latency, and we ignore the switching latency because the cell size is very small. We assume that the arrival time difference θ is within 5.0 μs. The maximum arrival time

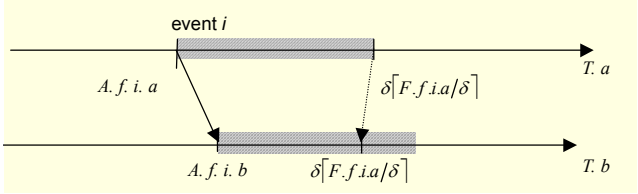

Fig. 7. Timing diagram of packets from a different module.

difference is very small compared to the digitization resolution: $\theta << \delta$.

Theorem 4. Delay bounds of the distributed DDQ scheduler with multiple flows established exist.

Proof. From Theorem 2, we can derive the delay bounds of the distributed DDQ scheduler. Assume that packet *i* of flow *f* arrives at line interface module *a* at *A. f. i. a* and is forwarded to module *b*. The packet arrives at module *b* at *A. f. i. b* and gets rate control. The packet is released from the rate control at $\overline{S}.f.i.b$ $(\geq A.f.i.b)$ and starts a busy period. Packets from *i* to *j* of flow *f* arrive at the same busy period. We ignore the sampling error because it is very small.

From (6), we have

$$\overline{F}.f.j.a \geq F.f.j.a \geq S.f.i.a + \sum_{k=i}^{j} L.f.k/R.f.$$

The amount traffic from packet *i* to *j* is defined as traffic contribution.

$$W.f.(A.f.i.a, F.f.j.a) = \sum_{k=i}^{j} L.f.k \leq R.f(F.f.j.a - S.f.i.a)$$

At the ready list of module *a*, the amount of traffic from packet *i* to *j* is less than or equal to the following traffic contribution:

$$\sum_{k=i}^{j} L.f.k \leq W.f.(\overline{S}.f.i.a, \overline{F}.f.j.a).$$

We are interested in the case where $\overline{S}.f.i.b \leq \overline{S}.f.i.a$ in order to get the delay bounds in module *b*:

$$\sum_{k=i}^{j} L.f.k \leq W.f.(\overline{S}.f.i.b, \overline{F}.f.j.a).$$

Assume that packets from *l* to *m* $(l \leq m)$ of flow *g* arrive at the ready list of module *b* from line interface module *c* during the same busy period starting with $\overline{S}.f.i.b$. The timestamp of packet *m* is less than or equal to $\overline{F}.f.j.a$. This amount of traffic from packet *l* to *m* is less than or equal to the traffic contribution of flow *g* from $\overline{S}.g.l.c$ to $\overline{F}.f.j.a\delta$ in module *c* as follows: $\sum_{k=l}^{m} L.g.k \leq W.g.(\overline{S}.g.l.c, \overline{F}.f.j.a)$.

Because packet *l* arrives at module *b* not earlier than $\overline{S}.f.i.b$, and the busy period starts at $\overline{S}.f.i.b$, $\overline{S}.f.i.b \leq \overline{S}.g.l.c$. The traffic contribution becomes

$$\begin{aligned} W.g.(\overline{S}.g.l.c, \overline{F}.f.j.a) &\leq W.g.(\overline{S}.f.i.b, \overline{F}.f.j.a) \\ &\leq W.g.(\overline{S}.f.i.a - \theta, \overline{F}.f.j.a) \\ &\leq R.g(\overline{F}.f.j.a - \overline{S}.f.i.a + \theta). \end{aligned}$$

If we assume that there are *N* flows established through module *b* including flow *f* and *g*, the exit time of packet *j* of flow *f* is upper-bounded as follows:

$$\begin{aligned} E.f.j.b &\leq \overline{S}.f.i.b + \sum_{n=1}^{N} W.n.(\overline{S}.f.i.b, \overline{F}.f.j.a)b/C.b + L_{\max}/C.b \\ &\quad - W.f.(\overline{S}.f.i.b, \overline{F}.f.j.a)b/C.b + \sum_{k=i}^{j} L.f.k \\ &\leq \overline{S}.f.i.b + \sum_{n=1}^{N} W.n.(\overline{S}.f.i.a - \theta, \overline{F}.f.j.a)/C.b + L_{\max}/C.b \\ &\leq \overline{S}.f.i.b + (\overline{F}.f.j.a - \overline{S}.f.i.a + \theta) \sum_{n=1}^{N} R.n/C.b + L_{\max}/C.b. \end{aligned}$$

With a delay bound condition known as rate admission control $\left(C.b \geq \sum_{n=1}^{N} R.n\right)$, we have the following delay bounds:

$$E.f.j.b \leq \overline{F}.f.j.a + L_{\max}/C.b + \theta.$$

This equation becomes ambiguous if packet *q* of flow *h* with a timestamp larger than $\overline{F}.f.j$ is transmitted earlier than packet *j*. This can happen if packet *q* arrives earlier than the digitized service start time $\overline{S}.f.j$. This state is equivalent to the idle state for packet *j*. The arrival and transmission of a packet with a timestamp less than or equal to $\overline{F}.f.j$ corresponds to the start of a new busy period. Then, the previous equation holds for any packet. See Theorem (1). □

The packet carrying the ready bin index $\overline{F}.f.j.a$ is transmitted from module *b* within the delay bound $\overline{F}.f.j.a + L_{\max}/C.b + \theta$. The additional delay θ is caused by the distributed DDQ algorithm and is very small compared to the digitization resolution. Therefore, we can conclude that the distributed processing idea lowers the hardware and software complexity of the queuing processor, but the additional delay is ignorable.

4. Comparison of PFQ Algorithms and CIOQ Router

As discussed above, the distributed DDQ algorithm shows comparable delay bounds to other PFQ algorithms. We summarize the latency, time complexity, and maximum number of queues of several PFQ algorithms mentioned in this paper including DDQ, as shown in [16]. In the case of distributed DDQ, the parameters at the egress module are provided.

In the case of DRR, the delay bounds are defined by the frame size (*F*) and the maximum amount of traffic of flow *i* that can be serviced during one frame (ϕ_i). DRR provides the best computing complexity, but the number of queues that should be managed by the scheduler is the largest. This is a very undesirable feature in a distributed system because each processor should have a copy of the details of all the flows.

Table 1. Latency, complexity, and number of queues of several PFQ algorithms.

Scheduler	Latency	Complexity	Maximum number of queues
VirtualClock	$L_i/R_i + L_{\max}/C$	O(log N)	1
Weighted fair queuing (WFQ)	$L_i/R_i + L_{\max}/C$	O(N)	1
Self-clocked fair queuing (SCFQ)	$L_i/R_i + (N-1)L_{\max}/C$	O(log N)	1
Worst-case fair queuing (WF2Q)	$L_i/R_i + L_{\max}/C$	O(log N)	2
Deficit round-robin (DRR)	$(3F - 2\phi_i)/C$	O(1)	N
DDQ	$\approx L_i/R_i + L_{\max}/C$	O(16)	2
Distributed DDQ	$\approx L_i/R_i + L_{\max}/C$	O(16)	2

This is critical when the number of flows is very large as on a high-speed link, and the scheduler should be distributed on many places in the routing system. We will discuss more about three stage-distributed architectures below.

Finally, we need to mention a more recently developed router architecture utilizing CIOQ [19]. It is known that this architecture can emulate WFQ with a speed-up of 2, which means that the speed of the switch output is twice the input link speed. Compared to the link speed of an output queuing architecture requiring the aggregate rate of all the input links, the CIOQ architecture is much improved in that it can use a lower speed buffer memory. This architectural improvement was made possible by the scheduler in the switching module sending back a grant to the serviced line interface module. This architecture works well with single priority, but scheduling gets more complicated with a multi-priority scheme. For example, a packet with lower priority at the switching module cannot be exchanged by a packet with higher priority at the line interface module if they are from the same input link. For this purpose, we need to standardize the timestamp to have single priority. This requirement was mentioned to simplify rescheduling packets released from the rate control. This requirement has to be solved before we go deeper into a high-speed router architecture. We will show how to satisfy this requirement in another paper.

In addition, because this architecture has distributed queues at all the ingress line interface modules, the switching module, and the egress line interface modules, it is difficult to get the system virtual time to apply the PFQ algorithm. DDQ has a strong point in that the hardware time is available at any module in the routing system. Thus, the DDQ algorithm has a good potential to be used for a large scale QoS router. In this architecture, the DRR algorithm is not suitable due to the large number of queues that each scheduler should manage and the signaling overhead for each queue. DDQ does not require any information of each flow at the second and third stages of the schedulers, another of its strong points.

V. Conclusion

We presented the DDQ queuing discipline for real-time traffic control in a packet switched network. The DDQ algorithm uses hardware time by sampling a counter. The clock to the counter is very small, and the effect of sampling error is ignorable. The delay bounds guaranteed by the DDQ scheduler are close to those by most PFQ schedulers. The difference is less than the digitization resolution – in this paper we assume 1 ms for the resolution. This amount of difference is believed to be acceptable.

The importance of DDQ is that we can make it as a distributed form. Thus, many processors can be used for the implementation of the algorithm. This distributes the computing complexity of the queuing processor into other line interface modules and lowers the hardware and software complexity of the queuing processor. This is our goal for devising DDQ. The analytical model for the delay bounds of the distributed architecture is the main contribution of this paper with the finite number of bins required. By developing this model, we provided the basis to improve the DDQ algorithm with a standard timestamp and applied it to a CIOQ architecture. Though the mathematical models presented here are simple, they were very helpful in understanding the scheduler behaviors and designing the system architecture and simulation program. The simulation results are very good with around 1.6 ms of maximum delay at 99% of traffic offered to a link. Compared to 10 ms of inter packet delay, this maximal delay is too good, requesting more review. We are preparing a paper specialized in the simulation and modification of the DDQ algorithm, and are already confident about the excellent performance and promising application.

References

[1] John B. Nagle, "On Packet Switches with Infinite Storage," *IEEE Trans. on Comm.*, vol. 35, iss. 4, Apr. 1987, pp. 435-438.

[2] Bob Branden, David Clark, and Scott Shenker, *Integrated Services Architecture,* IETF RFC 1633, June 1994.

[3] 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 Trans. on Networking*, vol. 1, no. 3, 1993, pp. 344-357.

[4] Lixia Zhang, "VirtualClock: A New Traffic Control Algorithm for

Packet Switching Network," *ACM SIGCOMM '90*, 1990, pp. 19-29.

[5] Norival R. Figueira and Joseph Pasquale, "An Upper Bound on Delay for the VirtualClock Service Discipline," *IEEE/ACM Trans. on Networking*, vol. 3, no. 4, Aug. 1995, pp. 399-408.

[6] S. Jamaloddin Golestani, "A Self-Clocked Fair Queuing Scheme for Broadband Applications," *IEEE INFOCOM '94*, 1994, pp. 636-646.

[7] Jon C. R. Bennett and Hui Zhang, "WF^2Q : Worst-Case Fair Weighted Fair Queuing," *IEEE INFOCOM '96*, Mar. 1996, pp. 120-128.

[8] Shun Y. Cheung and Corneliu S. Pencea, "BSFQ: Bin Sort Fair Queuing," *IEEE INFOCOM '02*, 2002, pp. 1640-1649.

[9] M. Shreedhar and G. Varghese, "Efficient Fair Queuing Using Deficit Round Robin," *ACM SIGCOMM '95*, 1995, pp. 231-242.

[10] S. Moon, J. Rexford, and K. Shin, "Scalable Hardware Priority Queue Architectures for High-Speed Packet Switches," *IEEE Trans. on Computer*, vol. 49, no. 11, 2000, pp. 1215-1227.

[11] P. Kuacharoen, M. Shalan, and V. Mooney, "A Configurable Hardware Scheduler for Real-Time Systems," *Int'l Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA'03)*, 2003, pp. 96-101.

[12] Chao and Uzun, "An ATM Queue Manager with Multiple Delay and Loss Priorities," *IEEE GLOBECOM '92*, 1992, pp. 308-313.

[13] Kicheon Kim, "Flow Aggregation of Rage Controlled Round-Robin Scheduler," *ETRI J.*, vol. 26, no. 4, Aug. 2004, pp. 351-359.

[14] Kicheon Kim, "Distributed Digitized Delay Queuing (DDQ) for Large Scale QoS Router," *Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'04)*, June 21-24, 2004, pp. 1183-1190.

[15] Bob Branden, Lixia Zhang, Steve Berson, Shai Herzog, and Sugih Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification," *IETF RFC 2205*, Sept. 1997.

[16] Kicheon Kim and David Hutchison, "Flowmeter for QoS Provision in Packet Switched Network," *IEE Electronics Lett.*, vol. 34, no. 1, 1998, pp. 21-22.

[17] Fabio M. Chiussi, Andrea Francini, and Joseph G. Kneuer, "Implementing Fair Queuing in ATM Switches – Part 2: The Logarithmic Calendar Queue," *IEEE GLOBECOM '97*, 1997, pp. 519-525.

[18] Anujan Varma and Dimitrios Stiliadis, "Hardware Implementation of Fair Queuing Algorithms for Asynchronous Transfer Mode Networks," *IEEE Comm. Magazine*, Dec. 1997, pp. 54-68.

[19] Shang-Tse Chuang, Ashish Goel, Nick McKeown, and Balaji Prabhakar, "Matching Output Queuing with a Combined Input/Output-Queued Switch," *IEEE J. on Selected Areas in Comm.*, June 1999, pp. 1030-1039.

Kicheon Kim received the BS from Hanyang University, Seoul, Korea in 1987 and the MS from the Korea Advanced Institute in Science and Technology (KAIST) in 1989. He received the PhD degree from Lancaster University, Lancaster, UK, in 1997. He worked in Electronics and Telecommunications Research Institute (ETRI) as a Member of Research Staff from 1989 to 1994. He worked on the design of the first high speed Internet service in Korea while he was working in Thrunet, Seoul, Korea from 1998 to 1999. He was also a Member of the Faculty Staff in Ajou University, Suwon, Korea, from 2002 to 2005. Currently, he is working for Korean Intellectual Property Office (KIPO). He is interested in real-time traffic quality of service and contents sales for the Internet.