Flow Aggregation of Rate Controlled Round-Robin Scheduler

Kicheon Kim

Flow aggregation is a scalable method to provide quality of service (QoS) guarantees to a large number of flows economically. A round-robin scheduler is an efficient scheduling algorithm. We investigate flow aggregation using a round-robin scheduler and propose the use of periodic timer interrupts for rate control of the round-robin scheduler. The proposed flow aggregator is a single-stage scheduler compared to Cobb's two-stage flow aggregator consisting of an aggregator and non-aggregating scheduler. It is possible to implement flow aggregation in the existing routers with only a software upgrade. We also present a simulation study showing the delay behaviors of the proposed algorithm.

Keywords: Packet scheduler, flow aggregation, rate control, IntServ, DiffServ, QoS.

I. Introduction

When fair queuing [1] was first introduced for providing a delay bound and reserved rate to real-time traffic, the packet scheduler worked on a per-session or per-flow basis. A series of fair queuing algorithms were devised improving efficiency and fairness [2], [3]. Fair queuing was proposed for the *integrated services* (IntServ) quality of service (QoS) mechanism in the Internet standards. However, as the link speed increases, the number of flows sharing a link also increases proportionally. So, per-flow packet scheduling has become uneconomical for high-speed links. The Internet standard introduced *differentiated services* (DiffServ) to improve scalability of the packet scheduling algorithm, and flow aggregation was proposed as an efficient solution to reduce the number of network states related to packet scheduling.

This paper is about flow aggregation for an affordable QoS solution. We think that the Internet QoS should be deployed from a simple QoS mechanism for loose real-time applications such as internet TV broadcasting and internet cyber education and gradually upgraded. We expect to use flow aggregation with a round-robin scheduler as the first phase of QoS deployment. The proposed flow aggregation is slightly different from the existing flow aggregation method and is suitable for a hardware-based scheduler or large scale schedulers based on approximation.

In section II, we present a brief introduction to fair queuing and flow aggregation. Section III shows a packet transmission experiment which gives a notion on traffic control timing granularity. Section IV presents a round-robin packet scheduler with rate control. In section V, we investigate the delay bounds of the constituent flows of the proposed aggregating scheduler. Section VI shows a simulation study on the delay behaviors of

Manuscript received Dec. 2, 2003; revised June 7, 2004.

Kicheon Kim (phone: +82 31 219 1840, email: kckim@ajou.ac.kr) is with the School of Information and Communication, Ajou University, Suwon, Korea.

the packet scheduler devised in this paper. In section VII, we draw our conclusion.

II. Fair Queuing and Flow Aggregation

When fair queuing was first introduced [1], it was a *round-robin scheduler* serving multiple packet queues in a circular order. It was expected to provide firewalls between real-time traffic and guarantee a reserved rate to each real-time traffic flow. However, the round-robin scheduler was not further developed for some time.

Instead, another type of scheduler was introduced, called VirtualClock [2]. The main contribution of VirtualClock was that it provided hints on the delay bounds of real-time packet flows by sorting packets according to the virtual clock stamp on the packets. The virtual clock stamp is now widely understood as the virtual finish time of 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 first packet. The service start time of the successive packet is then obtained by taking the larger value between the virtual finish time of the previous packet and the packet arrival time. The packet queue is sorted in the increasing order of the virtual finish time of each packet. So, we can classify this type of scheduler as a *sorted-queue scheduler*. The delay bound of the VirtualClock scheduler was derived later [4] and is equivalent to the virtual finish time of each packet.

A theoretical analysis of the fair queuing scheduler was first presented in generalized processor sharing (GPS) [3]. GPS is also a sorted-queue scheduler and similar to VirtualClock in many ways, but it is different in regard to virtual finish time calculation. The virtual finish time is obtained by simulating a fluid server. The scheduler serves each backlogged flow by the weighted share. The virtual time of the system should be recalculated at every change in the backlogged flows. When the flows are serviced at a rate faster than the reserved rate, the system virtual time gets faster than the real time. One possible drawback of this type of scheduler is a lack of capability for the rate control of each flow.

Self clocked fair queuing (SCFQ) [4] is also a sorted-queue scheduler similar to GPS and was suggested as a method to obtain the system virtual time from the virtual finish time of the packet receiving service. This scheduler is more efficient than GPS. So, it is suitable for a high speed link but provides larger delay bounds than VirtualClock or GPS.

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

The next step was an introduction of flow aggregation. The basic idea of flow aggregation is binding multiple flows into a single aggregate flow. The aggregate flow is treated as a single flow in packet scheduling. A packet scheduling method utilizing flow aggregation was presented in Aggregated Flow Fair Queuing (AFFQ) [9]. Later, a thorough theoretical analysis on flow aggregation was presented by Cobb [10]. He showed that quality of service guarantees can be preserved in spite of flow aggregation.

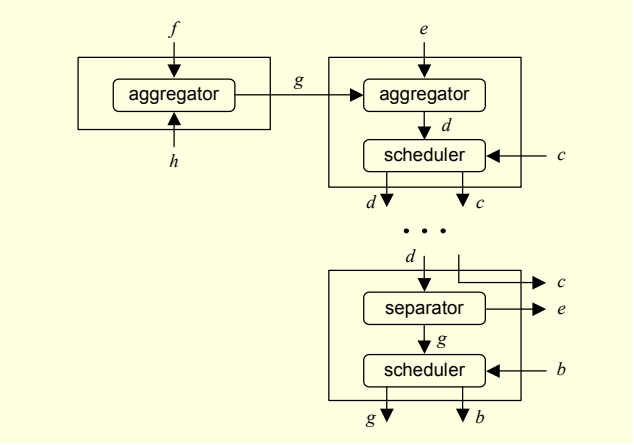

Fig. 1. Scheduler network for Cobb's flow aggregation.

Figure 1 shows Cobb's flow-aggregation network model. An aggregator is a packet scheduler similar to a WF^2Q scheduler. Flows e and g are aggregated into flow d by an aggregator. Flows c and d need not be aggregated if the two flows are forwarded into different directions at the immediate downstream router. So, flows d and e are scheduled into a single link, but not aggregated. This means that a two-stage scheduler network is necessary at a router as shown in Fig. 1. This brings about an additional delay. The delay of flow g at the aggregator is $L.g.i/R.g$, and an additional delay at the non-aggregation scheduler is $L.g.i/(R.g+R.e)$. The aggregator should perform a tight rate control such as WF^2Q. Otherwise, any flow serviced at a faster rate than the reserved rate can cause an unexpected large delay at the following scheduler because the scheduler guarantees only the reserved rate, $R.d = R.g + R.e$. The separator does not cause any additional delay. Cobb's flow aggregation is implemented by simulating the scheduler network as shown in Fig. 1. This requires high flexibility in the scheduler design. However, while the fair queuing schedulers are fast, they are not so flexible in

supporting a two-stage scheduler network. For these schedulers, we may need to change the flow aggregation method.

There are many different types of round-robin schedulers, but DRR is the first round-robin scheduler analytically studied [5]. Compared to a sorted-queue scheduler, a round-robin scheduler is much more efficient. The computing complexity of a sorted-queue scheduler is $O(\log N)$ where N is the number of flows. However, the computing complexity of a round-robin scheduler is $O(1)$. However, the delay behavior is better with a sorted-queue scheduler. The DRR scheduler is characterized by round size, T. The amount of traffic which a DRR scheduler is allowed to transmit each round is called quantum $Q.f$, and defined as the product of the round size and reserved rate; $Q.f = TR.f$. Thus, the delay bound of a DRR scheduler is larger than or equal to T. A sorted-queue scheduler can give smaller delay bounds than a DRR scheduler. However, at the early stage of QoS deployment an efficient round-robin scheduler can be deployed. In this case, flow aggregation for a round-robin scheduler should be considered.

Another important research about fair queuing is on the hardware-based packet scheduler [12], [13]. The performance of the software implementation of fair queuing algorithms may be limited. However, in hardware implementation, massive parallelism is employed for improving the sorting performance using a concurrent comparison. However, a hardware-based scheduler is not so flexible in adjusting the scheduler to perform a frequent establishment and release of flows. Flow aggregation makes the situation even worse. There is also a problem in the speed of deployment of QoS mechanisms in the Internet. We can not deploy such mechanisms all at once, but must gradually upgrade. We think that QoS mechanisms should be deployed by upgrading the existing routers in the current Internet from the entry level QoS. Our first requirement is an affordable solution for loose real-time services such internet TV broadcasting and cyber education services, which do not require tight interactivity. We defined this type of QoS as the entry level QoS guarantee. The scheduler should be very simple, and the number of flows can be minimized by flow aggregation. We combine a round-robin scheduler and flow aggregation for this purpose. Our work starts with simple experiments described in the following section.

III. Traffic Control Timing Granularity

Our traffic control algorithm is based on an experiment on traffic control timing granularity. In the experiment, we tried to transmit packets from a computer every 5 ms. We programmed various computers to schedule the packet transmission process to transmit a packet and then sleep for the remaining 5 ms. However, it was impossible to achieve a finer granularity than 10 ms in the packet transmission interval. Two packets were transmitted every 10 ms on some computers such as Sun workstations. There were also computers transmitting four packets every 20 ms as shown in Fig. 2. This is because most computers use periodic timer interrupts to maintain their clocks. Thus, the packet transmission process wakes up every 10 ms. The conventional processor is not suitable for achieving a precise timing granularity of less than 10 ms especially under a multitasking environment.

However, we can devise a rate control mechanism with the periodic timer interrupts and a credit variable. With the rate control mechanism, a flow can transmit a packet only when the credit of the flow is larger than or equal to the packet size. Whenever the flow transmits a packet, the credit is decreased by the packet size. If the size of the packet is larger than the remaining credit, the flow is blocked and cannot transmit any more. The timer interrupt increases the credit to the product of the reserved rate, $R.f$, and the interrupt interval, T. This amount of information, $TR.f$, is defined as quantum $Q.f$. Each packet should not be larger than the quantum size, or it is discarded.

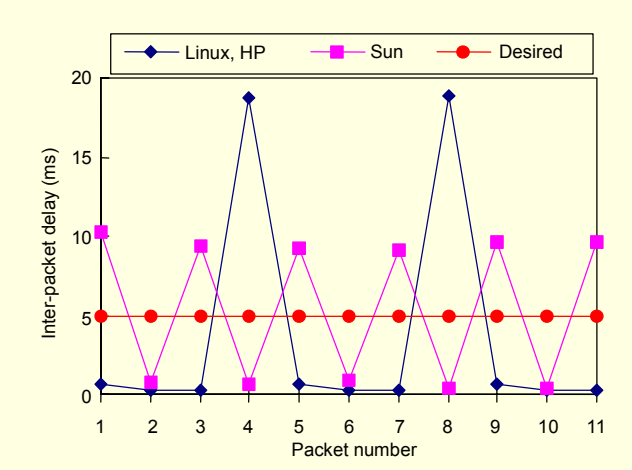

Fig. 2. Inter-packet delay with a 5-ms desired delay.

IV. Packet Scheduler

We have presented a packet scheduler based on the data structure of traffic-related variables [8]. The data structure was named as the flowmeter. The flowmeter-based packet scheduler is shown in Fig. 3. Now, we combine the rate control mechanism of the previous section with the packet scheduler similar to the DRR [3].

In case there are multiple flowmeters for a link, it is not efficient to update the credit variables of all the flowmeters every

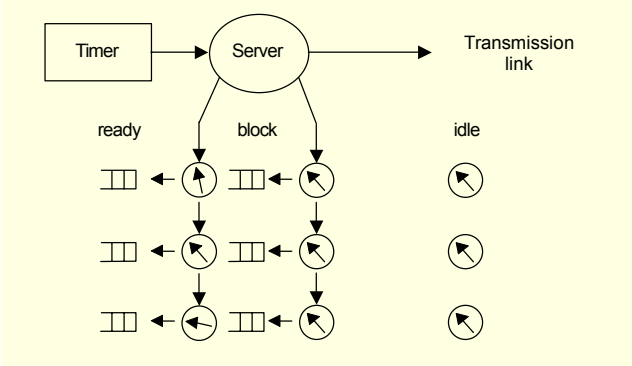

Fig. 3. A flowmeter-based packet scheduler with timer interrupts.

time the timer interrupt is generated. Instead, each flowmeter has a clock variable, `last_event_time`. When we need to calculate the credit of a flowmeter, this clock variable is compared to `current_time`, which is the master clock variable of a link and counts the number of timer interrupts. If the two clock variables are different, the credit is refreshed to the quantum of the flowmeter, and `last_event_time` is set to `current_time`. With this scheme, we can calculate the credit of a flowmeter only when it is required. Thus, the packet scheduler can control the rates of multiple flows efficiently using the conventional processor.

A flowmeter has head and tail pointers for a first-in first-out (FIFO) packet queue. The incoming packets are queued to the corresponding flowmeter. When there is no packet queued to a flowmeter, the flowmeter is *idle*. An idle flowmeter does not belong to any list and is isolated. The initial state of a flowmeter is also idle. When a new packet arrives and the packet is not larger than the credit of the flowmeter, the flowmeter is appended to the *ready* list as shown in Fig. 3. If the packet is larger than the credit, the flowmeter is appended to the *block* list. These two lists are of the FIFO service discipline.

When the *current* flowmeter of a link is empty, the server picks up the top flowmeter from the ready list and compares the `last_event_time` of the flow-meter to the system clock time, `current_time`. If the two clocks are different, the credit is refreshed to the quantum stored in the flowmeter and the `last_event_time` of the flowmeter is changed to the system clock time. Then, the server transmits packets from the flowmeter while there is enough credit. When there is not enough credit, the flowmeter is appended to the block list. The interrupt from the timer increases the system clock time and moves all the flowmeters in the block list to the ready list. In order to save time, the two lists are concatenated, not actually moving the flowmeters one by one.

We can increase the credit when we move the flowmeters from the block list to the ready list. However, this can make the timer interrupt service routine very time-consuming. Thus, we calculate the credit when the flowmeter is picked up as the current flowmeter. Once the current flowmeter is picked up, the flow is allowed to transmit packets as long as there is enough credit. Each time a packet is transmitted, the credit decreases by the packet size. If there is no packet in the flowmeter, it becomes idle and the server picks up the next flowmeter from the ready list.

One delicate problem of this credit operation is the boundary behavior. This is when the timer interrupt is generated while a flow is being served. In general, a flowmeter is appended to the block list when a quantum of traffic for the flowmeter is transmitted and the flow is waiting for a timer interrupt. In this case, the `last_event_time` of the flowmeter and `current_time` of the link are the same. If these two clocks are different, we define this as the boundary behavior and the flowmeter is appended to the ready list instead of the block list. When the timer interrupt is generated, all the flowmeters in the block list are already moved to the ready list before the boundary flow completes the transmission of packets. Even though the flowmeter is appended to the ready list, all the other flows can be served within the timer interrupt interval, T. Therefore, there is no problem to guarantee the reserved rate of each flow, but the rate of the boundary flow can reach up to twice the reserved rate. This can cause queuing of the traffic in the downstream scheduler and increase the theoretical delay bound. We will discuss more about the theoretical delay bound in the next section.

A finite state machine representation of the packet scheduler is shown in Fig. 4. In this way, we can provide delay bounds and a

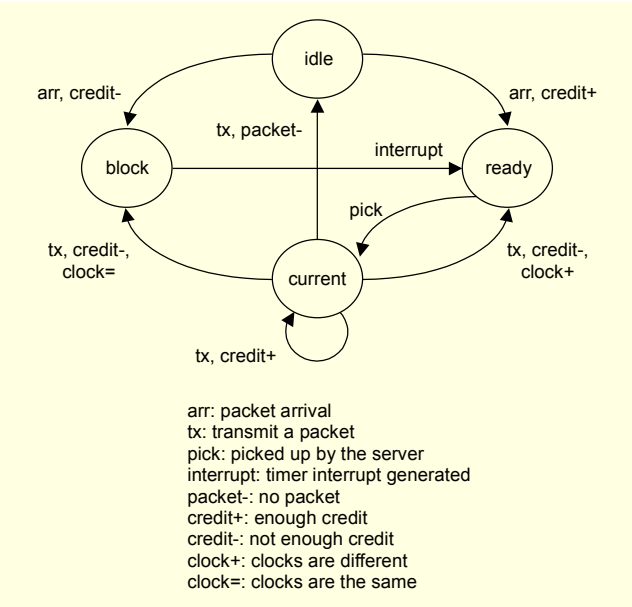

Fig. 4. A finite state machine representation of the packet scheduler.

rate control function to real-time flows. Our packet scheduler requires a computing complexity of $O(1)$ and serves all the flows at least once every timer interrupt interval, T. The resource reservation should meet the rate admission control condition, $\sum_f R.f \le C$, where C is the outgoing link capacity.

The next step is about how to implement flow aggregation with the proposed round-robin scheduler. We considered a different network model from Cobb's flow aggregation. In Fig. 5, flows `f` and `g` are two separate flows in the upstream scheduler, but these two flows are aggregated and share the input and output links in the downstream scheduler. The two flows also share the same FIFO queue in the downstream packet scheduler. In this way, we can reduce the number of flows in the downstream scheduler and implement flow aggregation without using a two-stage scheduler network of Cobb's flow aggregation. One of the important advantages of our flow aggregation model is that the aggregator is not required in the upstream scheduler. Flow separation is performed by the switch fabric before the packet scheduler. By applying the proposed rate-control scheme in the upstream scheduler, we can achieve a delay bound in the downstream scheduler. An analysis on the delay bound is presented in the next section.

We can devise two different protocol solutions for supporting flow aggregation. First, we can consider IP tunneling. This is a term for encapsulating the packets of flows `f` and `h` with an additional common IP header when the packets leave the upstream scheduler. The two flows are then naturally treated as a single flow in the downstream scheduler. For flow separation, the packets are de-capsulated from the additional IP header when they leave the router. Then the two flow packets have different IP headers. Thus, they are treated as two different flows in the downstream router. One possible drawback of this IP tunneling scheme is the protocol overhead caused by the additional IP header.

Second, we can consider label switching. Assume a flow label consists of virtual path identifier (VPI) and virtual channel identifier (VCI) fields. The VPI of flow f is 0 and the VPI of flow h is 1 in the upstream scheduler. The labels of the packets of the two flows are changed in the upstream router and the two flows are rate controlled as shown in Fig. 5. These two flows can have the same VPI value in the downstream scheduler, though the VCI values are different for flow separation in the further downstream routers. The downstream scheduler performs queuing and scheduling based on VPI. Therefore, the two flows are treated as a single flow. Currently, the Internet does not support label switching, but multiprotocol label switching (MPLS) is being widely studied for the future Internet.

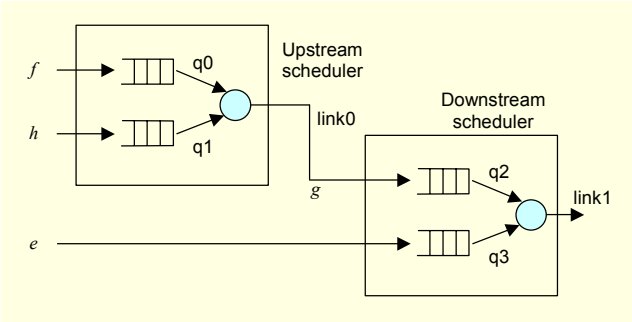

Fig. 5. Proposed flow aggregation network model.

V. Delay Bound

We are interested in achieving deterministic delay bounds of the constituent flows in the down stream scheduler shown in Fig. 5. In the upstream scheduler, flows f and h are two separate flows. In the downstream scheduler, the two flows share the input and output links and the packet queue. The two flows form an aggregate flow, g, in the downstream scheduler. This means that the two flows are heading to the same further downstream scheduler, which is connected to the output link of the downstream router. Flows f and h are constituent flows of flow g. Flow g is not a constituent of any other flow. Therefore, we say that flow g is a root flow. Flows f and h are root flows in the upstream scheduler, too.

The DRR scheduler guarantees the throughput of each root flow. However, the proposed rate-controlled round-robin scheduler is trickier. Therefore, we apply a few conditions in this paper. The amount of traffic served by the round-robin scheduler each round is determined by the quantum size, $Q.f = TR.f$. Each packet should not be larger than the quantum size. If any packet is larger than the quantum, the packet can not be served because the credit of the flow is not accumulated after multiple rounds. Thus, a packet larger than the quantum is discarded.

We have another condition in this paper. Because the packets are served in a FIFO manner, if there is a packet larger than the remaining credit, the packet can block the transmission of the following packets smaller than the remaining credit. We call this *large packet blocking*, and because of it, the reserved bandwidth can not be fully utilized. Therefore, the delay bound could get larger. If all the packets are of the same size, large-packet blocking does not take place and we can achieve a tight delay bound. The ATM switching and transmission systems use a fixed-size packet, called an ATM cell. Therefore, a rate controlled round-robin scheduler is a good match with ATM. Thus, we will discuss the delay bound only for a fixed-size packet network in this paper. The rate controlled round-robin packet scheduler with a variable-size packet is still under study.

Theorem: A flow aggregator implemented with a rate controlled round-robin scheduler in a fixed-size packet network provides a delay bound of twice the round size at the immediate downstream scheduler.

Proof: The upstream scheduler can transmit packets only while there is enough credit. Once the credit of a flow is refreshed, the flow can transmit up to a quantum of traffic. The interval between the first and second quantum of traffic varies from zero to twice the round size, or 2*T*. This can be explained by using the timing diagram of Fig.6.

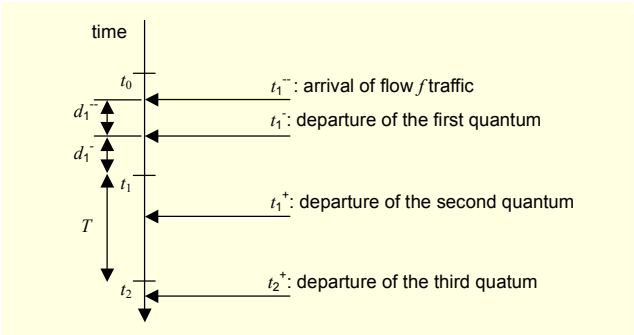

Fig. 6. Timing diagram.

Suppose that flow *f* is the only flow consisting of aggregate flow g and has just been established through the upstream and downstream schedulers as shown in Fig. 5, and that several quanta of flow *f* traffic arrives at the upstream scheduler at t_1^{--} as shown in Fig. 6. The last bit of the first quantum of flow *f* traffic is transmitted at t_1^{-}. We ignore the propagation delay between the upstream and downstream routers. Thus, as soon as the last bit of the first quantum of flow *f* traffic is transmitted, it arrives at the downstream scheduler. The propagation delay is constant and we can add the delay when it is necessary. The root flow is served at least once every round time, *T*. Therefore, the delay between t_1^{--} and t_1^{-} is d_1^{--}, and it is upper bounded by the frame size, *T*, plus the transmission delay of a maximum size packet. The last term could be caused by the best-effort traffic. Thus, we have

$$d_1^{--} = t_1^{-} - t_1^{--} \le T + L_{\max}/r_{up},$$

where r_{up} is the transmission rate of the upstream scheduler.

After the transmission of the first quantum of flow *f* traffic, the flowmeter of flow *f* does not have enough credit to transmit the next quantum of traffic. Therefore, it is moved to the block list and waits for a timer interrupt. The first timer interrupt after the transmission of the first quantum of traffic is generated at t_1. The waiting time between t_1^{-} and t_1, d_1^{-}, is less than the timer interrupt interval, *T*. The second quantum of flow *f* traffic is transmitted at t_1^{+}. The delay between t_1 and t_1^{+} is d_1^{+}, and it is as large as d_1^{--}. In addition, the timer interrupt is generated at t_2 and the third quantum of flow *f* traffic is transmitted at t_2^{+}. The delay between t_2 and t_2^{+} is d_2^{+}, and it is also as large as d_1^{--}.

When the second quantum of flow *f* traffic arrives at the downstream scheduler, it may find that the first quantum of flow *f* traffic is still queued there. The interval between the arrival of the first quantum and the arrival of the second quantum of flow *f* traffic at the downstream scheduler is $d_1^{-} + d_1^{+}$, and it has an upper bound of 2*T*. However, the lower bound is not determined. It could be very small, almost zero. The interval between the arrival of the first quantum and the second interrupt is $d_1^{-} + T$, and it is larger than *T*. This means that before the second interrupt, the downstream scheduler transmits at least a quantum of flow *f* traffic. Thus, the third quantum of flow *f* traffic can not see the first quantum queued in the downstream scheduler. The interval between the arrival of the first quantum and the *i*-th interrupt is $d_1^{-} + (i-1)T$. The amount of traffic arriving in the downstream scheduler until the *i*-th interrupt is $iQ.f$ at maximum. This gives the lower bound to the amount of traffic transmitted from the downstream scheduler when the *i*-th interrupt is generated in the upstream scheduler. That is, $(i-1)Q.f$, $i \ge 2$. So, when the interrupt is generated in the upstream scheduler, the amount of traffic queued in the downstream scheduler can reach up to $Q.f$. After the interrupt in the upstream scheduler, another quantum of flow *f* traffic can be transmitted to the downstream scheduler. Thus, the amount of traffic which can be queued in the downstream scheduler is limited to twice the quantum, or $2Q.f$. Therefore, the maximum latency of flow *f* traffic in the downstream scheduler is $2T + L_{\max}/r_{down}$, where r_{down} is the transmission rate of the downstream scheduler.

We can increase the number of flows sharing the flowmeter in the downstream scheduler. The amount of traffic of aggregate flow *g* queued in the downstream scheduler is denoted as *W.g*, and it has the upper bound of the sum of twice the quantum of each constituent flow;

$$W.g \le \sum_{f \in g} 2Q.f = 2\sum_{f \in g} TR.f = 2T\sum_{f \in g} R.f = 2TR.g.$$

The upper bound is equivalent to twice the quantum of the aggregate flow. Therefore, the queued traffic experiences a delay of twice the timer interrupt interval plus the transmission delay of a maximum-size packet. We can ignore the last term, especially when the link speed is fast. Thus, each constituent flow is guaranteed to have 2*T* of the delay bound and *R.g* of the throughput in the downstream scheduler.

On the other hand, the delay bound achieved by our aggregation packet scheduler is larger than the delay bound of the per-flow packet scheduler. This is due to a kind of boundary behavior. Two quanta of traffic belonging to a flow can be queued in the downstream scheduler before other constituent flows send packets to the downstream scheduler. Therefore, we name this queuing behavior caused by the flow aggregation as *double queuing*, and the delay behavior due to double queuing is studied in the next section. □

The proposed flow aggregator can be defined as a single-stage aggregator, compared to Cobb's scheduler network model of Fig. 1 where there are an aggregator and non-aggregating scheduler at a node performing flow aggregation. This difference can be compared using an example network as shown in Figs. 7(a) and 7(b). There are five nodes along the routing path of flow *f*. Flows *f* and *g* are aggregated into flow *h*. Figure 7(a) shows flow aggregation using the proposed single-stage scheduler. Figure 7(b) shows flow aggregation using Cobb's two-stage scheduler network. We use a rate controlled round-robin scheduler for the aggregator and non-aggregating scheduler.

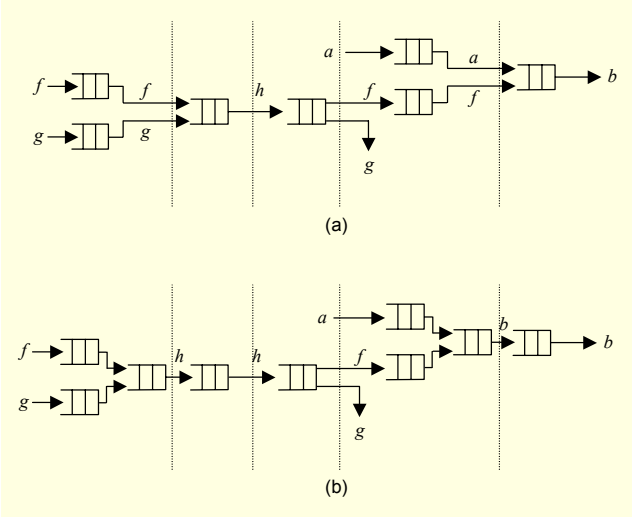

Fig. 7. (a) Flow aggregation by a single-stage scheduler, (b) Flow aggregation by a two-stage scheduler.

With the single-stage flow aggregator shown in Fig. 7(a), the flow aggregation takes place in the down stream scheduler. The maximum delay of periodic packet flow *f* after the second node is 3*T*. The delay bound at the first node is *T*, and the delay bound at the second node is 2*T* according to the theorem above.

With the two-stage flow aggregator shown in Fig. 7(b), the flow aggregation takes place in the first node. The delay at the first aggregator of the first node is zero because the output link of the first aggregator is regarded as infinitive. The first aggregator is implemented by software, but it makes the programming tricky. Furthermore, a hardware-based scheduler is not so flexible to reflect the change of the established flows. The delay bound at the non-aggregating scheduler of the first node is 2*T* according to the theorem above. After being rate controlled, the delay bound at the second node becomes *T*. Therefore, the maximum delay of periodic packet flow *f* after the second node is 3*T*.

Both flow aggregators give comparable delay bounds. The proposed single-stage aggregator does not require changing the scheduler or router architecture. This is the same as the scheduler which does not support flow aggregation. Thus, we can introduce QoS features and flow aggregation with the existing routers. We can also use flow aggregation with a hardware-based scheduler without any modification. However, the benefits of flow aggregation are all maintained.

At the fourth node, flow separation takes place. Before the flow aggregation with another flow, *a*, flow *f* is rate controlled. Thus, any change in the inter-packet delay picked up along the routing path does not affect the delay bound of the other constituent flow in both cases in Figs. 7(a) and 7(b).

VI. Simulation

In order to show the effectiveness of our rate control scheme for the QoS guarantees, we present a simulation study for the network shown in Fig. 5.

We have two different types of real-time traffic, video and audio. We assume a real-time video traffic of 25 video frames per second at a 932.8 kb/s rate and a real-time audio traffic of 42.4 kb/s. The audio traffic is equivalent to traffic transmitting an ATM cell every 10 ms. The video traffic is equivalent to traffic transmitting 22 ATM cells every 10 ms. The video traffic is generated every 40 ms and the audio traffic every 10 ms. The video traffic arrives in a burst and is rate-controlled by the timer interrupts occurring every 10 ms. When the audio traffic is multiplexed with the video traffic, the audio traffic is affected by the video traffic and shows a delay variation. This is the main point of the simulation study.

The output link of the upstream scheduler is relatively faster than that of the downstream scheduler. In the simulation, we chose a 100-Mbps link between the upstream and downstream schedulers. The output link speed of the downstream scheduler is selected as 2.12 Mbps. This speed is selected so that 90% of the channel bandwidth is reserved for real-time traffic. This link is relatively slow and could be a wireless link. The cell rate of the link is 50 cells per 10 ms. In Fig. 5, flows *f* and *e* are video traffic, and flow *h* is audio traffic.

Without the rate control scheme, the audio traffic experiences a varying delay in the downstream scheduler as shown in Fig. 8. With the rate control scheme, the traffic to the downstream scheduler is regulated so that the influence to the audio traffic

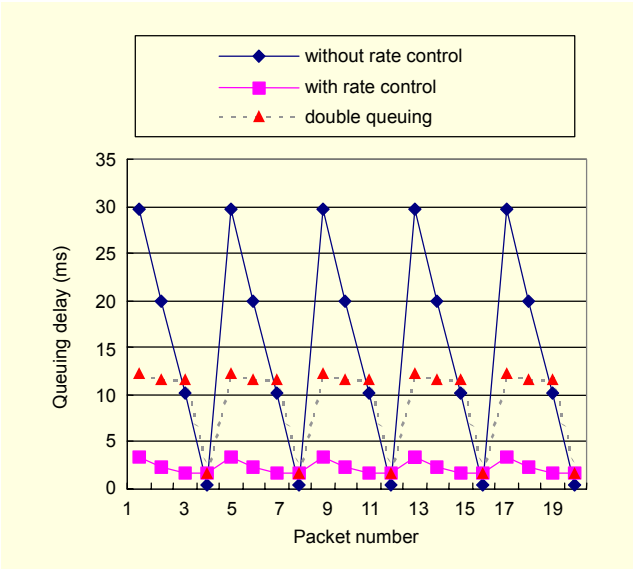

Fig. 8. Queuing delay of an aggregate flow.

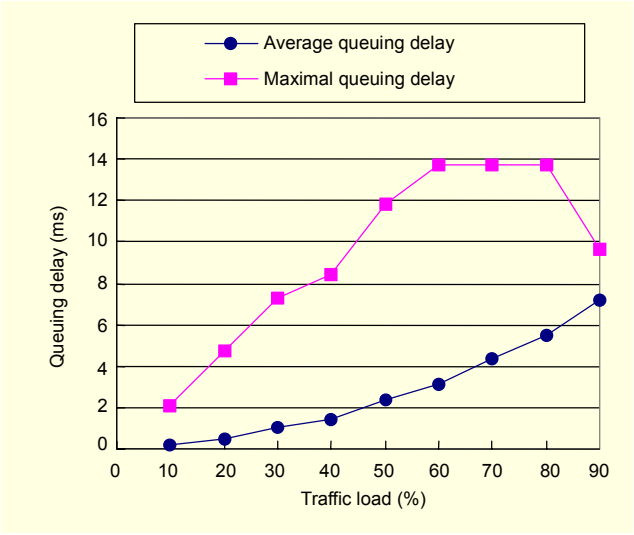

Fig. 9. Queueing delay at various traffic loads.

by the video traffic is decreased. The maximum delay of the audio traffic is theoretically 20 ms, but in our simulation the delay is below 10 ms. Improvement in the delay behavior is dramatic and the playback application at the destination hardly experiences a lack of data.

We assume a periodic generation of the traffic for the simulation. Therefore, by changing the offset time of the traffic, we could make double queuing take place as shown in Fig. 8. Due to double queuing, the delay can be larger than 10 ms. However, the time window for double queuing is very small. Only when traffic is generated near the timer interrupt does double queuing take place. A more statistical behavior of double queuing is studied by another simulation.

For the network shown in Fig. 5, each stream of traffic is generated by an exponentially distributed random variable. The quantum of each queue is set to 90% of the link speed of the downstream scheduler. Thus, the quanta of flows *f*, *h*, and *e* are 9328, 424, and 9328, respectively. The average packet generation interval for each traffic is 10 ms. Simulation runs are performed for 10 seconds at each traffic load. The queuing delays get stable after a few seconds of simulation runs. As shown in Fig. 7, the maximal queuing delay of flow *g* in Fig. 5 exceeds 10 ms due to double queuing. We noticed that the maximal queuing delay at a 90% load is smaller than the maximal delays at loads of 50 through 80%. This is because at a 90% traffic load the upstream queues always have packets. Thus, flows *f* and *g* are served by turns, and this reduces the frequency of double queuing and lowers the maximal queuing delay. Meanwhile, the average queuing delay of flow *g* increases proportionally to the traffic load. We can reduce the maximal queuing delay by reducing the time window for double queuing. This will be our future work.

VII. Conclusion

In this paper, we introduced periodic timer interrupts to control the rate of a real-time flow and to modify a round-robin scheduler to be synchronized to the timer interrupt for providing each flow with the reserved rate and delay bound. Real-time flows are merged to form an aggregate flow in the downstream scheduler, reducing the network states. The theoretical delay bound of each constituent flow is $2T$ at the immediate downstream scheduler. The simulation study shows that the rate control is a good method to limit the queued traffic and delay at the immediate downstream scheduler. We can provide real-time flows with QoS guarantees economically through a standardization of packetization and the timer interrupt period.

Given that our flow aggregation does not require a separate aggregator and non-aggregating scheduler at a router, flow aggregation with only a single-stage scheduler is beneficial especially with a hardware-based scheduler and large-scale scheduler such as Bin Sort Fair Queuing. The proposed single-stage flow aggregation can be implemented with a sorted-queue scheduler as well. It will be interesting to compare the delay behaviors of the single-stage flow aggregator of the sorted-queue scheduler to Cobb's flow aggregator.

Another important aspect of our contribution is that a rate controlled round-robin scheduler is easy to implement and provides a delay bound and flow aggregation without a change of the current router architecture. Thus, it can help the QoS mechanism deployed earlier, though it is not in the best form.

References

[1] John B. Nagle, "On Packet Switches with Infinite Storage," *IEEE Trans. on Comm*., 1994, pp. 435-438.

[2] Lixia Zhang, "VirtualClock: A New Traffic Control Algorithm for Packet Switching Network," *ACM SIGCOMM*, 1990, pp. 19-29.

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

[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, 1995.

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

[7] Jon C. R. Bennett and Hui Zhang, "WF2Q: Worst-case Fair Weighted Fair Queuing," *IEEE INFOCOM*, 1996, pp. 120-128.

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

[9] Salil S. Kanhere and Harish Sethu, "Fair, Efficient and Scalable Scheduling Without Per-Flow State," *IEEE Int'l Conf. on Performance, Computing, and Comm*., 2001, pp. 181-187.

[10] Jorge Arturo Cobb, "Preserving Quality of Service Guarantees in Spite of Flow Aggregation," *IEEE/ACM Trans. on Networking*, vol.10, no.1, 2002, pp. 43-53.

[11] Michael G. Hluchyj and Mark J. Karol, "Queuing in High-Performance Packet Switching," *IEEE J. on Selected Areas in Comm*., vol.6, no.9, 1988, pp. 1587-1597.

[12] 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.

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

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

[15] Huirong Fu and Edward W. Knightly, "A Simple Model of Real-Time Flow Aggregation," *IEEE/ACM Trans. on Networking*, vol.11, no.3, 2003, pp. 422-435.

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. From 1999 to 2002, he was the President of FlowTec, a venture company in the area of QoS and contents sales in the Internet. He has been working as a Member of Faculty Staff in Ajou University, Suwon, Korea since 2002. He is interested in supporting realtime traffic in the Internet.