Time-Shift Scheduling: Fair Scheduling of Flows in High Speed Networks

Jorge A. Cobb
Department of Computer Science
University of Houston
Houston, TX 77204-3475
cobb@cs.uh.edu

Mohamed G. Gouda Amal El Nahas
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
gouda@cs.utexas.edu amal@cs.utexas.edu

Abstract

We present a scheduling protocol, called Time-Shift scheduling, to forward data packets from multiple input flows to a single output channel. Each input flow is guaranteed a predetermined forwarding rate and an upper bound on packet delay. The protocol is an improvement over existing protocols because it satisfies the properties of low delay, fairness, and efficiency, while existing protocols fail to satisfy at least one of these properties. In Time-Shift scheduling, each flow is assigned an increasing timestamp, and the packet chosen for transmission is taken from the flow with the least timestamp. The protocol features the novel technique of time shifting, in which the scheduler's real-time clock is adjusted to prevent flow timestamps from increasing faster than the real-time clock. This bounds the difference between any pair of flow timestamps, thus ensuring the fair scheduling of flows.

1. Introduction

Consider a computer network with point-to-point communication channels. Each output channel of a computer is equipped with a scheduler. The input to the scheduler is a set of flows, where each flow is a sequence of data packets received from some input channel of the computer. When an output channel becomes idle, the task of its scheduler is to determine which received packet should be the next one to forward over the channel.

A particular type of scheduling protocols, which we call guaranteed-rate schedulers, forward data packets from each flow at a designated rate. Examples of these scheduling protocols can be found in [24]. In all these protocols, the source of a flow finds a network path that leads to its desired destination. Then, the source notifies each scheduler in the path about its desired forwarding rate. Each scheduler determines if it has enough available bandwidth in its output channel to forward the data packets from the new flow. The new flow is accepted if and only if all schedulers in the path accept the new flow.

Due to the reservation of bandwidth, the network can provide service guarantees to each flow, such as end-to-end packet delays, provided the rate of the flow does not exceed the agreed-upon rate. These service guarantees are of particular importance to real-time applications, such as interactive audio and video [11].

The desirable properties of guaranteed-rate schedulers are the following.

1. Rate-dependent delay: Let r be an input flow of a scheduler, and the reserved rate of r be R.r bits/sec. Assume r is also the sole input to a constant rate server that forwards the bits of each packet of r at a precise rate of R.r bits/sec. The delay of each packet of r through the scheduler should be at most the delay of the same packet through the constant rate server.
2. Efficiency: The time to enqueue a received packet or to dequeue a packet for transmission is O(log(N)), where N is the number of input flows of the scheduler.
3. Fairness: A flow should not be "punished" if it temporarily exceeds its reserved rate to take advantage of unused bandwidth in the channel. In addition, the unused bandwidth should be shared among the flows in proportion to their reserved rates.

The rate-dependent delay property guarantees to each flow that the upper bound on its packet delay depends solely on its reserved rate, and not on other factors, such as the number of flows sharing the scheduler or the rate reserved by other flows. The efficiency property is desirable due to the high bandwidth requirements expected from future applications of guaranteed-rate scheduling.

The fairness property is desirable, because it may be normal for some flows to violate their reserved packet rate. Examples of such flows are file transfers and multi-resolution video [18]. The sources of these flows may reserve enough forwarding rate from the network to receive a minimum quality of service. If the source of a flow detects that additional bandwidth is available, then it generates data packets at a rate higher than that which it reserved, in order to take advantage of the unused bandwidth. If the source detects that no additional bandwidth is available, it reduces its sending rate. (There are several techniques by which a source can detect if additional bandwidth is available, see for example [16] and [21]). Thus, because some flows may be of adjustable rate, the sched-

0-8186-7453-9/96 $05.00 © 1996 IEEE

uler should share the unused bandwidth in a fair manner among all flows that currently have packets queued in the scheduler.

Some scheduling protocols are inadequate for adjustable rate flows because they are not work-conserving [10] [13] [17]. That is, they serve each flow at exactly the rate it has reserved, and will not forward additional packets of the flow even if the outgoing channel is idle. These schedulers do not allow adjustable rate flows to take advantage of any additional bandwidth, and thus do not satisfy the fairness property.

Other scheduling protocols assign a timestamp to each packet, and packets are forwarded in increasing timestamp order. These protocols are work conserving, that is, the output channel is never idle as long as its packet queue is non-empty. However, some of these schedulers are unfair [23], in the sense that they "punish" a flow if it sends data packets at a rate higher than its reserved rate. Other schedulers are fair [14] [20], but they are either inefficient or do not have rate-dependent delay.

In this paper, we introduce a new scheduling protocol, called Time-Shift scheduling. The protocol is based on packet timestamps, is work-conserving, and satisfies all the desirable properties mentioned above. Time-shift scheduling is based on the novel technique of time-shifting, in which the scheduler's real-time clock is periodically adjusted to prevent packet timestamps from increasing faster than the real-time clock. This ensures fairness by placing an upper bound on the difference between the timestamps of any pair of flows.

Notation: we use quantifications of the form

$$(\min u : 0 \le u \le 2 : u^2)$$

to denote the minimum of 0^2, 1^2, and 2^2. If the range of the dummy variable u is omitted, all values in the type of u are included.

2. Flow Timestamps

In this section, we consider the scheduling protocols of Virtual Clock [23], Weighted Fair Queuing [16] [19] [20], and Self-Clocking Fair Queuing [14], and discuss their strengths and weaknesses. In these protocols, a timestamp is assigned to each received data packet, and packets are forwarded in increasing timestamp order.

We have shown in [4] that assigning a timestamp to a packet when it becomes the head of the queue of its flow is cleaner and more efficient than assigning it a timestamp when it is received. The former technique is called flow timestamps, since only one timestamp is maintained per flow. Thus, we base Time-Shift scheduling on flow timestamps. To maintain a single timestamp paradigm throughout the paper, the flow timestamp versions of the above protocols are discussed.

A *flow* is an infinite sequence of data packets received by a scheduler. A scheduler receives data packets from N distinct flows. For each flow, the scheduler stores the received packets in a separate FIFO queue. We say that a flow is *active* if its queue in the scheduler is non-empty.

We adopt the following notation, where p is a packet from flow r.

- queue.r queue of received packets from flow r.
- R.r forwarding rate in bits/sec. reserved for flow r.
- L.r packet length (in bits) of the head of queue.r.
- F.r timestamp of flow r.
- L.p packet length (in bits) of packet p.
- F.p value of F.r when p is the head of queue.r.
- Lmax upper bound on packet length for all flows.
- C capacity in bits/sec. of the output channel

Whenever the output channel becomes idle, the scheduler chooses a packet from those it has received, and forwards the packet to the output channel. The goal of the scheduler is to forward the packets of each flow r at an average rate of at least R.r. Since all N flows share the output channel, the following constraint is necessary.

$$\sum_{r=0}^{N-1} R.r \le C \qquad (1)$$

Assume a packet p from flow r is received. Before the packet is appended to queue.r, the scheduler checks if r is active. If it is not active, it updates F.r as follows.

$$F.r := \max(g.p, F.r) + L.p/R.r$$

In this assignment, g.p is some quantity related to packet p. The value chosen for g.p varies from one scheduling protocol to another.

When the output channel becomes idle, the scheduler chooses among the active flows the one with the smallest timestamp. Let r be this flow. Then, the next packet from flow r is removed from queue.r and forwarded to the output channel. If flow r remains active, its flow timestamp is updated as follows.

$$F.r := F.r + L.r/R.r$$

We next consider the case of Virtual Clock scheduling, which is defined by choosing g.p as the arrival time of p to the scheduler. Virtual Clock has the property of rate-dependent delay, as proven in [4] [22]. In particular, the exit time of p is at most F.p + Lmax/C. The scheduler is also efficient, requiring only O(log(N)) operations to enqueue or dequeue a packet. However, the scheduler is unfair, as illustrated by the following example.

A scheduler has two flows, r and s, each with a reserved rate of 1 Kbit/sec., a packet size of 1 Kbit, and an output channel bandwidth of 2 Kbit/sec.. Consider the following events.

From time 0 up to time 100 sec., packets from flow r arrive at a rate of 2 packets/sec., and no packet is received from flow s. Thus, at time 100, all packets from r are forwarded to the output channel, F.r = 200, and F.s = 0.

Next, from time 100 sec. up to time 200 sec., packets from both flows are received at the rate of 2 packets/sec.. Note that, when the next packet from flow s is received, F.s = 101. Also, when the next packet from flow r is received, F.r = 201. Since F.r is much larger than F.s, no packet from r is forwarded to the output channel until 100

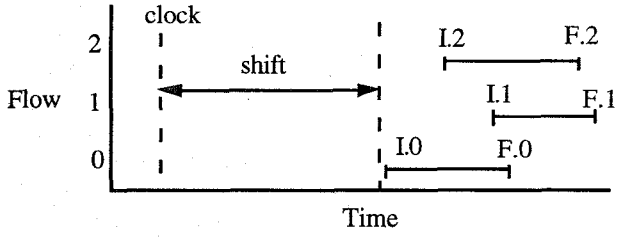


Figure 1: flow timestamps drift away from the clock.

packets from s are forwarded, i.e., until time 150 sec.. In effect, r is denied service for 50 sec. because it previously took advantage of bandwidth unused by s.

This unfairness does not occur in the Weighted Fair Queuing protocol. Furthermore, the bound on packet delay is similar to that of Virtual Clock, and thus it also satisfies the rate-dependent delay property. In Weighted Fair Queuing, the timestamp of a packet is the time at which the packet would exit a virtual server. The input to the virtual server are the same input flows of the scheduler, and the virtual server shares its unused bandwidth among all active flows in proportion to their reserved rates. The value of g.p is complicated and takes O(N) time to compute.

The high complexity of Weighted Fair Queuing led to the introduction of Self-Clocking Fair Queuing. In this scheduler, g.p = F.q, where q is the packet being sent over the output channel at the time p is received. The time to enqueue or dequeue a packet is O(log(N)).

This protocol is fair in the following sense. For any pair of active flows r and s,

$$| F.r - F.s | \leq \max(L.r/R.r, L.s/R.s)$$

In this way, no flow can have a timestamp that is significantly greater than the timestamp of other flows. Thus, a flow that takes advantage of free bandwidth cannot be punished. On the other hand, in Virtual Clock scheduling, | F.r - F.s | is unbounded.

Unfortunately, the delay of packets increases as follows. Let the scheduler have 91 flows, each with a packet size of 1 Kbit. Let flow 0 have a rate of 100 bits/sec., and flows 1 through 90 have a rate of 10 bits/sec.. Let the output channel's bandwidth be 1 Kbit/sec.. Assume that one packet from each of flows 1 through 90 arrive at time 0. The timestamp of each of these packets is 100. Next, when the first of these packets is being forwarded, a packet from flow 0 arrives. The timestamp of this packet is 110. This packet must wait until all 90 packets with timestamp 100 exit the scheduler before itself may exit. Hence, the delay of this packet is 90 seconds, as opposed to a delay of only 10 seconds that it would incur in Virtual Clock or Weighted Fair Queuing scheduling.

Note that the delay of flow 0 is related the rate of the other flows. If the 90 flows with a rate of 10 bits/sec. are replaced by 900 flows with a rate of 1 bit/sec., the delay of flow 0 increases by a factor of 10.

We conclude that each of the three scheduling protocols discussed satisfy only two of the three desired properties. In the next section, we present a scheduling protocol that satisfies all three properties.

3. Time-Shift Scheduling

We next present the intuition behind Time-Shift scheduling, followed by its formal definition. Its delay and fairness properties are given in later sections.

From its definition, F.r can be viewed, intuitively, as the time at which the packet at the head of the queue of flow r should be forwarded. That is, it should be forwarded L.r/R.r seconds later than the forwarding time of the previous packet from the same flow. Thus, we may define the "ideal arrival time" of the packet at the head of the queue of flow r as follows.

$$I.r = F.r - L.r/R.r$$

That is, if the packet should be forwarded at time F.r, and the flow is abiding by its reserved rate R.r, then, ideally, the packet should be received at time I.r.

Consider Figure 1, in which flows 0 through 2 are active, and the dashed line on the left indicates the current value of the real-time clock. The flow timestamp F.r is an indicator of how much service has been given to flow r. If F.r >> clock, then packets from flow r have been forwarded at a rate greater than R.r. This could occur if the flow is trying to take advantage of unused bandwidth. In the figure, all three flows have their flow timestamp significantly greater than the current value of the clock.

Assume an additional flow, flow 3, has been inactive for some time, but becomes active once again. Assume g.p is the arrival time of packet p. Then, when flow 3 becomes active, its flow timestamp is updated as follows.

$$F.3 := \max(clock, F.3) + L.3/R.3$$

Thus, F.3 = clock + L.3/R.3, which is significantly smaller than the flow timestamps of the other active flows. This implies that only packets from flow 3 are forwarded until F.3 reaches a value greater than the flow timestamps of the other flows.

To remedy this, F.3 should be given a value as close as possible to F.0, F.1, and F.2. Since F.3 is derived from the real-time clock, the clock should be close to F.0, F.1 and F.2. To do this, either we increase the value of the real-time clock, or we reduce the value of each flow timestamp by an equal amount. We take the former approach, because the latter requires O(N) time.

The remaining question is how much to advance the clock. If the clock is advanced beyond the minimum of the ideal arrival times, then the flow that becomes active has a timestamp larger than the timestamps of the active flows, and may be delayed excessively by these flows. If the clock is advanced to a value smaller than the minimum of the ideal arrival times, then the flow that becomes active has a timestamp smaller than the timestamps of the active flows, and it may unfairly delay the

```
process Time-Shift scheduler
inputs
ch_idle   :  is the output channel is idle?
R.r       :  rate of flow r
variables
r         :   0 ≤ r < N
queue.r   :  packet queue of flow r
L.r       :  length of packet at head of queue.r
F.r       :  timestamp of flow r
p         :  data packet
L.p       :  length of packet p
ShiftClock :  adjustable real-time clock
begin
  receive p from any r      →
        if queue.r = empty  →
            ShiftClock := max(ShiftClock, Imin);
            F.r := max(ShiftClock, F.r) + L.p/R.r
        [] queue.r ≠ empty  →  skip
        fi;
        queue.r := append(queue.r, p)
[]
  ch_idle ∧ (∃ s : : queue.s ≠ empty)  →
        r := least_ts(F);
        p := head(queue.r);
        send p to output channel;
        queue.r := tail(queue.r);
        if queue.r ≠ empty
                →  F.r := F.r + L.r/R.r
        [] (∀ s : : queue.s = empty)
                → ShiftClock := max(ShiftClock, F.r)
        [] (∃ s : : queue.s ≠ empty) ∧ queue.r = empty
                →  skip
        fi
end
```

Figure 2: Time-Shift scheduler

active flows. Thus, we choose to advance the clock to the minimum of the ideal arrival times of the active flows.

Because the scheduler advances the clock, i.e., it "shifts" the clock to the right, it is called a Time-Shift scheduler. We refer to the adjustable real-time clock by the name ShiftClock[1].

When a packet p is received from an inactive flow r (i.e., queue.r is empty), ShiftClock is updated as follows,

$$\text{ShiftClock} := \max(\text{ShiftClock}, \text{Imin})$$

where Imin is the minimum ideal arrival time of all active flows. That is,

$$\text{Imin} = (\min s : \text{queue.s} \neq \textbf{empty} : \text{I.s})$$

Furthermore, the timestamp of r is updated as follows.

$$\text{F.r} := \max(\text{ShiftClock}, \text{F.r}) + \text{L.p/R.r}$$

When a packet from an active flow r is forwarded, F.r is increased by L.r/R.r, provided flow r remains active.

We next present in Figure 2 the definition of the Time-Shift scheduler, using the notation of [15].

The process has two inputs from its environment. The first is ch_idle, which indicates if the output channel is idle. It becomes false when the scheduler sends a packet p to the channel, and becomes true L.p/C seconds afterwards. The second input is the rate reserved for each flow.

We assume the following. Variable ShiftClock is an adjustable real-time clock. It increases automatically with the progression of time. Also, executing an action takes zero time, i.e., ShiftClock remains constant while an action is executed unless an assignment statement in the action changes its value. Finally, ShiftClock does not advance while the packet queue is non-empty and the channel is idle, i.e., the next packet to forward is chosen immediately after the channel becomes idle.

The process contains two actions. In the first action, a packet is received from a flow. If the flow becomes active, then a time shift is performed, i.e., ShiftClock is increased, if necessary. Also, the flow timestamp is updated.

In the second action, when the output channel is idle and there are still packets to forward, the active flow with the smallest timestamp is obtained from function least_ts(F), and a packet from this flow is forwarded. The flow's timestamp is updated if the flow remains active. If no active flows remain, ShiftClock is updated so that it is greater than the flow's timestamp. This is necessary to prove fairness in Section 5.

Implementing this protocol requires two ordered queues: one for the flow timestamps, and one for the ideal service times. Inserting or removing an element from either of these takes O(log(n)) time, where n is the number of active flows. Thus, the desired efficiency is achieved.

The restriction on the reserved rates given by (1) above is sufficient, provided the rate assigned to each flow remains fixed. However, this is not always the case, because the application generating data packets for the flow may terminate, and the flow may be reassigned to a new application that reserves a different rate. Thus, a restriction is needed to indicate when can the reserved rate of a flow be reassigned to another flow.

The restriction we choose is the following. We say that a flow r is *live* if either ShiftClock ≤ F.r or r is active. The rate of a flow r can be reassigned to another flow if flow r is no longer live. Thus, the following is required to be an invariant of the reserved rates.

$$(\sum r : \text{ShiftClock} \leq \text{F.r} \vee \text{queue.r} \neq \textbf{empty} : \text{R.r}) \leq C \quad (2)$$

[1] If the clock of the scheduler cannot be advanced, the scheduler may increment a variable, say shift, instead of incrementing the clock. Any reference to ShiftClock is then substituted by the expression clock + shift.

4. Delay Bound in Time-Shift Schedulers

In this section, we show that the Time-Shift scheduler has a bound on packet delay no greater than the bound on packet delay of a Virtual Clock scheduler or a Weighted Fair Queuing scheduler.

We say that packet p *exits the scheduler* L.p/C seconds after p is forwarded to the output channel, i.e., when the output channel becomes idle after receiving p from the scheduler.

Henceforth, any reference to time refers to the value of ShiftClock. For example, the expression "at time T" refers to the state of the system when ShiftClock = T.

The delay bound for a Time-Shift scheduler follows from the following theorem.

Theorem 1

In a Time-Shift scheduler, for every active flow r,

$$\text{ShiftClock} \leq F.r + (Lmax - L.r)/C$$

Proof

Let p be the packet currently at the head of the queue of flow r. Let T be the time when p became the head of the queue of flow r, and let S be the latest time, no later than T ($S \leq T$), such that one of the following was true.

a) A time-shift occurred (i.e., ShiftClock was advanced, and S = ShiftClock = Imin after the increase).
b) Some packet q was forwarded, where $F.q > F.p$
c) All flow queues were empty.

In all three cases, at time S, for all active flows s,

$$S \leq I.s \vee F.p < F.s \qquad (3)$$

In (a), this holds because after the time-shift, $S = Imin \leq I.s$. In (b), the packet q chosen for transmission had $F.q > F.p$, and all active flows s have $F.s \geq F.q > F.p$. In (c), there are no active flows and (3) holds trivially.

Assume $F.p < S$. From (3), and $I.s < F.s$, for all active flows s at time S,

$$F.p < F.s$$

Furthermore, when any flow s becomes active after S,

$$F.p < S < F.s$$

Hence, $F.p < S$ is impossible, because flow r is active at time T, $S \leq T$, with F.p = F.r. We must have instead that $S \leq F.p$.

From the definition of S, only packets from flows s with $F.s \leq F.p$ are forwarded after S and before p is forwarded. From Lemma 1 below and from relation (3), these packets can be no more than $(F.p - S)\cdot C$ bits, and hence, p exits the scheduler no later than the following time.

$$S + (F.p - S) + Lmax/C$$

Thus, p is forwarded to the channel no later than time

$$F.p + Lmax/C - L.p/C.$$

The term Lmax/C is needed because for cases (a) and (b) we did not count the packet currently being forwarded at time S. When p is the head of the queue of flow r, we have F.r = F.p, and this continues to hold until p is forwarded. Thus, packet p is forwarded to the output channel no later than time $F.r + (Lmax - L.r)/C$, and at this time either flow r becomes inactive, or F.r increases, which implies the statement of the theorem.

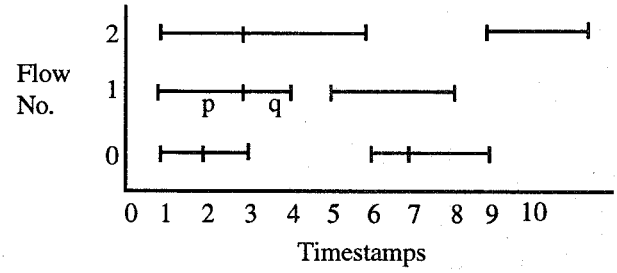


Figure 3: Timestamps contained by each packet

Lemma 1

If $U \leq V$, and at time T, for all active flows r,

$$U \leq I.r \vee V < F.r$$

then, starting from time T, the packets forwarded to the output channel with a timestamp at most V sum to a total of at most $(V - U)\cdot C$ bits.

Proof Sketch

For a packet p of flow r, define $I.p = F.p - L.p/R.r$.

We associate a timestamp with each "bit" b of packet p. We allow b to be a real number in the interval [0, L.p]. For a bit b in packet p, its timestamp equals $I.p + b/R.r$. Thus, the first bit has a timestamp equal to I.p, and the last bit has a timestamp equal to F.p. We say that timestamp t is contained by packet p if a bit in the packet has timestamp t, i.e., $I.p \leq t \leq F.p$. A flow r contains timestamp t if some packet of r contains t.

For example, consider Figure 3. In this figure, the ideal arrival time and timestamp of each packet are denoted by a small vertical line. In flow 1, for its first packet, p, we have I.p = 1 and F.p = 3. Similarly, we have I.q = 3 and F.q = 4. Thus, all timestamp values from 1 up to 4 are contained by flow 1.

The proof consists of first showing that if timestamp t is contained by packet p, then flow r is live at time t, and furthermore, the value of R.r used to compute the timestamp of p is the same value that R.r has at time t. Thus, for any interval (s, t), if flow r contains all values in this interval, then the same rate R.r is used to timestamp all packets of r that contain bits in this interval. Furthermore, the bits of flow r whose timestamps are in this interval sum to at most $(t - s)\cdot R.r$ bits.

Finally, using relation (2), it follows that, for any interval (U, V), the bits with a timestamp in this interval from packets of any flow add to at most $(V - U)\cdot C$ bits. The lemma then follows trivially.

The details of the proof may be found in [5].

♦

Theorem 1 implies that each packet p will exit the scheduler before ShiftClock reaches the value of the timestamp of p plus Lmax/C. It has been shown that the exit time of a packet in Virtual Clock scheduling is also at most the packet's timestamp plus Lmax/C [4] [8] [22]. These two bounds are not directly comparable, because the exit time in Time-Shift scheduling is measured with respect to a clock that is adjusted forward, while in Virtual Clock the exit time is measured with respect to a clock that is never adjusted.

To show that the delay bound in Time-Shift scheduling is at most the delay bound of Virtual Clock, consider the following alternative protocol. When a flow becomes active and the clock is less than Imin, the scheduler subtracts Imin - clock from all flow timestamps (whether active or not). Thus, Imin = clock is accomplished by reducing Imin rather than increasing the clock.

Note that the difference between the real-time clock and any flow timestamp remains the same in both algorithms. Hence, the difference between any pair of flow timestamps is also the same, which implies that the alternative scheduler forwards packets in the same order as the original. Note also that Theorem 1 places a bound on the difference between a flow timestamp and the real-time clock. Thus, this bound also holds for the alternative scheduler, and each packet is forwarded by the time indicated in its timestamp plus Lmax/C.

Finally, because the alternative scheduler reduces the flow timestamps, it is easy to show that the timestamp of each packet in the alternative scheduler is at most the timestamp of the same packet in a Virtual Clock scheduler. Thus, the delay bound for a Time-Shift scheduler is at most the delay bound of a Virtual Clock scheduler.

The delay bound of Weighted Fair Queuing is similar to that of Virtual Clock [19]. Hence, the delay bound of Time-Shift scheduling is also no greater than the delay bound of Weighted Fair Queuing.

5. Fairness in Time-Shift Schedulers

To ensure fairness, the Time-Shift scheduler should allocate any unused bandwidth to all active flows in proportion to their reserved rates. Equivalently, for any pair of flows, as long as both flows remain active, the ratio of the number of packets forwarded from each flow should be the same as the ratio of their reserved rates. Note that this holds if the difference between the flow timestamps of the pair of flows is tightly bounded.

The fairness of the Time-Shift scheduler is stated by the following theorem.

Theorem 2

Of the active flows with an ideal arrival time equal to Imin, let τ be the flow with the greatest timestamp. If r and s are active flows, then

$$| F.r - F.s | \leq L.\tau/R.\tau + \max(L.r/R.r, L.s/R.s) + Lmax/C$$

Proof Sketch

We prove the theorem by showing that the scheduler preserves the following invariant. For every flow r, one of the following conditions hold.

a) r is active, and $I.\tau \leq I.r \leq F.\tau + Lmax/C$

b) r is inactive, and

$$F.r \leq ShiftClock \ \vee \ F.r \leq Imin$$

c) r is inactive, and $I.\tau \leq F.r \leq F.\tau$

The proof that the protocol preserves the invariant may be found in [5].

We next show that the invariant implies the statement of the theorem. Below, we only need case (a) of the invariant. Cases (b) and (c) are needed to show that case (a) holds when an inactive flow becomes active.

Consider two active flows r and s. From case (a) above,

$$I.\tau \leq I.r \leq F.\tau + Lmax/C$$

$$I.\tau \leq I.s \leq F.\tau + Lmax/C$$

From the definition of ideal arrival time, we obtain the following upper and lower bounds for F.r and F.s, which imply the theorem.

$$I.\tau < F.r = I.r + L.r/R.r \leq I.\tau + L.\tau/R.\tau + L.r/R.r + Lmax/C$$

$$I.\tau < F.s = I.s + L.s/R.s \leq I.\tau + L.\tau/R.\tau + L.s/R.s + Lmax/C$$

♦

The bound of Theorem 2 on the relative value of two flow timestamps prevent flows that become active from "hogging" the output channel and denying service to flows that have exceeded their reserved rates. The Virtual Clock protocol has no similar bound.

The above bound is close to, but not as tight as, the bound provided by Self-Clocking Fair Queuing. However, the fairness bound above is achieved in conjunction with a delay bound that is significantly better than the delay bound of Self-Clocking Fair Queuing.

Notice also that part (a) of the invariant in the proof of Theorem 2 and Lemma 1 combined imply that the packet currently at the head of the queue of flow r will exit the scheduler within

$$L.\tau/R.\tau + L.r/R.r + Lmax/C$$

seconds. This bound holds regardless of whether flow r has abided by its reserved rate or not.

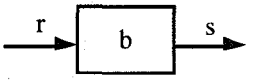
6. End to End Delay Bounds

We present the end to end delay bound for a flow traversing multiple Time-Shift schedulers. To do so, we borrow some results from Flow Theory [2] [3], which we overview next. Proofs for the theorems may be found in [5].

A *flow* r is an infinite sequence r.0, r.1, r.2, . . . , of non-negative real numbers. Informally, we divide time

into small and fixed sized intervals which we call instants. Each r.i represents the number of bits that travel in flow r at instant i. We denote the sequence r.0, r.1, . . . , r.i by r.(0,i).

As a flow traverses a network, it experiences queuing delays, represented by flow operators. A *flow operator* has an input flow r and an output flow s. At the ith instant, the operator inputs r.i, outputs s.i, and stores the remainder in an internal buffer. The content of the buffer at the ith instant is denoted b.i. The infinite sequence b.0, b.1, b.2, . . . , is the *buffer flow of* the flow operator.



Formally, a flow operator with input flow r, output flow s, and buffer flow b is defined, for every i, $i \geq 0$, as follows.

$$s.i \text{ is contained in the interval } F.(r.(0,i), b.(i-1), i)$$

$$b.i = b.(i-1) + r.i - s.i$$

where b.(-1) = 0, and F is a function, called the *operation* of the flow operator, that returns an interval of real numbers. The output of a flow operator should be no greater than its input, i.e., its operation is restricted as follows.

$$0 \leq s.i \leq r.i + b.(i-1)$$

Formal definitions for the buffer capacity and delay of a flow operator with respect to a given input flow may be found in [2] [3].

We model the changes experienced by a flow as it traverses a path in a computer network by a linear network of flow operators. A linear network consists of a sequence of flow operators, (f.0, f.1, . . . , f.(n-1)), where the input to each flow operator is the output of the previous flow operator. The input and output flows of the linear network are the input flow of f.0 and the output flow of f.(n-1), respectively.

The first flow operator we introduce is the R-limiter, where R is a positive real number. The operation of an R-limiter is defined as follows

$$s.i = \begin{cases} R & \text{if } b.(i-1) + r.i \geq R \\ b.(i-1) + r.i & \text{if } b.(i-1) + r.i < R \end{cases}$$

where r is the input flow, s is the output flow, and b is the buffer flow of the R-limiter. Basically, the R-limiter is a constant-rate server that forwards its input flow to its output flow at exactly the rate R.

Let a Virtual Clock scheduler have an input flow r with a reserved rate R. The timestamp of a packet p from flow r is equivalent to the time at which p exits an R-limiter whose input flow is r. In Virtual Clock, packet p exits by the time indicated in its timestamp (plus Lmax/C), i.e., by the time it would exit the R-limiter (plus Lmax/C).

Because the delay in a Time-Shift scheduler is at most that of Virtual Clock, packet p from flow r in a Time-Shift scheduler also exits by the time it would exit an R-limiter with input flow r (plus Lmax/C).

This behavior can be represented by an R-filter. An R-filter is a flow operator that never delays its flow longer than the delay it would experience in an R-limiter. That is, if an R-limiter and an R-filter have the same input flow, then, the sum of any prefix of the output flow of the R-filter is at least the sum of the same length prefix of the output flow of the R-limiter.

Let bl be the buffer flow of an R-limiter with input flow r. The operation of an R-filter is as follows.

$$s.i = \max((bf.(i-1) + r.i)\cdot X.i,\ bf.(i-1) + r.i - bl.i)$$

where r is the input flow, s is the output flow, bf is the buffer flow of the R-filter, and each X.i is an undetermined real value in the closed interval [0, 1].

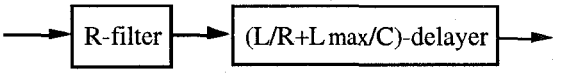
Theorem 3

A linear network of two R-filters is equivalent to a single R-filter.

◆

By induction over the number of R-filters, any linear network of R-filters is identical to a single R-filter.

From the definition of an R-filter, an R-filter can be used to represent the behavior of a single Time-Shift scheduler. However, in an R-filter, the first bit of a packet of size L may exit up to L/R instants earlier than the last bit of the packet. These bits should exit together, since packets are indivisible units of data and must be transmitted as a whole. Furthermore, a packet in a Time-Shift scheduler may exit up to Lmax/C seconds after it would exit an R-limiter. Therefore, we represent a single Time-Shift scheduler with the following linear network, where L is the maximum packet size of the flow.



A D-delayer, where D is positive integer, is a flow operator that may delay its input flow in an arbitrary manner by at most D instants. Its formal definition can be found in [3] [5].

Theorem 4

Let d be a D-delayer, and f be an R-filter. If the linear networks (d,f) and (f,d) have the same input flow, then:

1. Each output flow of (d,f) is an output flow of (f,d).
2. The buffer capacity of (f,d) is at most the buffer capacity of f plus D·R.

◆

A flow traversing n Time-Shift schedulers is represented by a linear network consisting of n (R-filter, (L/R+Lmax/C)-delayer) pairs. Using Theorems 3 and 4, it is easy to show that the delay of this linear network is at most the delay of a single R-limiter followed by an (n·(L/R+Lmax/C))-delayer (recall that the delay of an R-filter is at most that of an R-limiter).

The delay through the R-limiter is dependent upon the burstiness of the flow. There are several ways to bound

the burstiness of a flow [2] [6] [13]. One of these is the (m,R)-uniform property [2] below.

Let m be a positive integer and R be a positive real number. A flow r is (m,R)-uniform iff, for every j, $j \geq 0$,

$$\sum_{i=j}^{j+m-1} r.i \leq m \cdot R$$

The delay of an (m,R)-uniform flow through an R-limiter is at most m, and the buffer capacity is at most m·R. Thus, the total end-to-end delay is at most

$$m \cdot R + n \cdot (L/R + Lmax/C).$$

Furthermore, from Theorems 3 and 4, the buffer capacity of the jth scheduler is at most

$$m + j \cdot (L + Lmax \cdot R/C).$$

The upper bound on the end-to-end delay of a series of Time-Shift schedulers derived in the section is the same upper bound on the end-to-end delay for a series of Virtual Clock or Weighted Fair Queuing schedulers [8] [12] [19]. Thus, the Time-Shift scheduler achieves the same end-to-end delay, while at the same time being fair and efficient.

7. Concluding Remarks

An additional attempt to improve the fairness of Virtual Clock has been presented in [1]. Three techniques were proposed. As stated by the author, the first technique does not preserve the rate-dependent delay. The second technique resets the Virtual Clock value of all flows to zero when the scheduler has no packets to forward, which can be shown to preserve the rate-dependent delay. However, during periods of heavy load, especially in slow links, all flows may not become inactive often, and thus the unfairness of Virtual Clock prevails. The third technique resets to zero the Virtual Clock value of an individual flow when it becomes inactive. Again, this does not achieve fairness unless all flows become inactive simultaneously.

In Time-Shift scheduling, the end-to-end delay increases by L/R + Lmax/C with each hop in the path to the destination. It is possible to decrease this delay for some flows at the expense of either increasing the delay of other flows or leaving some bandwidth unreserved. This has already been done in other protocols that do not ensure fairness (e.g., [7] [9] [10]). In a future paper, we plan to investigate if this reduction in delay is possible while at the same time ensuring that the scheduling protocol is fair.

References

[1] Bernstein G., "Reserved Bandwidth and Reservationless Traffic in Rate Allocating Servers", *Computer Communication Review*, pp. 6-24, July 1993.

[2] Cobb J., Gouda M., "Flow Theory: Verification of Rate-Reservation Protocols", *IEEE International Conference on Network Protocols*, 1993.

[3] Cobb J., Gouda M., "Flow Theory", in revision for the *IEEE/ACM Transactions on Communications*, 1996.

[4] Cobb J., Gouda M., El-Nahas A., "Flow Timestamps", *Annual Joint Conf. on Information Sciences*, 1995.

[5] Cobb, J., *Flow Theory and The Analysis of Timed-Flow Protocols*, Ph.D. Thesis, The University of Texas at Austin, May 1996.

[6] Cruz R.L., ``A Calculus for Network Delay, Part I: Network Elements in Isolation", *IEEE Tran. on Information Theory*, Vol. 37, No. 1, January 1991.

[7] Figueira N, Pasquale J., "Leave-in-Time: A New Service Discipline for Real-Time Communications in a Packet-Switching Data Network", *1995 SIGCOMM Conference*, p. 207.

[8] Figueira N. R., Pasquale J., "An Upper Bound on Delay for the Virtual Clock Service Discipline", *IEEE/ACM Transactions on Networking*, Vol. 3, No. 4, Aug. 1995.

[9] Ferrari D., Verma D., "A Scheme for Real-Time Channel Establishment in Wide-Area Networks", *IEEE Journal of Selected Areas in Communication*, Vol 8, No. 4, pp. 368-379, April 1990.

[10] Ferrari D., Zhang H., "Rate Controlled Static Priority Queueing", *IEEE INFOCOM Conference*, 1993.

[11] Gall D., "A Video Compression Standard for Multimedia Applications", *Communications of the ACM*, Vol. 34, No. 4, April 1991.

[12] Goyal P, Lam S., Vin H., "Determining End-to-End Delay Bounds in Heterogeneous Networks", *NOSSDAV Workshop*, 1995.

[13] Golestani S. J., "A Stop-and-Go Queueing Framework for Congestion Management", *ACM SIGCOMM Conference*, 1990.

[14] Golestani S. J., "A Self-Clocking Fair-Queueing Scheme for Broadband Applications", *IEEE INFOCOM Conference*, 1994.

[15] Gouda M., "Protocol Verification Made Simple", *Computer Networks and ISDN Systems*, Vol. 25, 1993, pp. 969-980.

[16] Keshav S., "A Control Theoretic Approach to Flow Control", *1991 ACM SIGCOMM Conference.*

[17] Kalmanek C. R., Kanakia H., and Keshav S., "Rate Controlled Servers for Very High-Speed Networks", *IEEE GLOBCOMM, Conference*, 1990.

[18] Kanakia H., Mishra P., "A Hop-by-Hop Rate Based Congestion Control Scheme", *1992 ACM SIGCOMM Conference.*

[19] Parekh A. K. J., "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks", MIT Report # LIDS-TH-2089.

[20] Parekh A. K. J., Gallager R., "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, pp. 344-357, June 1993.

[21] Ramakrishnan K. K., Raj J., "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks", *ACM Trans. on Computer Systems*, Vol 8, No. 2, pp. 158-181.

[22] Xie G., Lam S., "Delay Guarantee of Virtual Clock Server", *IEEE/ACM Transactions on Networking*, December 1995.

[23] L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for Packet-Switched Networks", *ACM Trans. on Computer Systems*, Vol. 9, No. 2, May 1991.

[24] Zhang H., Keshav S., "Comparison of Rate-Based Service Disciplines", *ACM SIGCOMM Conf.*, 1991.