Universal Timestamp-Scheduling for Real-Time Networks

Jorge A. Cobb

Department of Computer Science
Mail Station EC 31
The University of Texas at Dallas
Richardson, TX 75083-0688
jcobb@utdallas.edu

Abstract

Consider a network of computers interconnected by point-to-point communication channels. Before generating network packets, each source in the network reserves a fraction of the packet rate of each output channel in the path to its destination. We define a family of scheduling protocols, called *Universal Timestamp-Scheduling*, to forward packets in this network such that all members of the protocol family provide the same upper bound on packet delay as Virtual Clock scheduling. That is, the packets from a source will exit the output channel of a computer no later than the time they would exit an output channel whose rate equals the source's reserved rate and whose input is exclusively the packets from this source. The protocol family is called *universal* because it encompasses a wide variety of protocols. To show this, we prove that many scheduling protocols in the literature are members of the protocol family, and thus provide the above guarantee. In addition, we show that the protocols in the literature have only considered one side of the spectrum of possible scheduling protocols, and that there is another side of the spectrum that deserves attention and remains to be investigated.

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 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 transmit over the output channel.

A particular type of scheduling protocols, which we call guaranteed-rate schedulers, forward packets from each flow at a designated rate. Examples of these scheduling protocols can be

found in [18][20]. In all of 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 packet rate. Each scheduler determines if it has enough available bandwidth in its output channel to transmit the packets of 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 quality of service guarantees to each flow, such as end-to-end packet delays. These service guarantees are of particular importance to real-time applications, such as interactive audio and video [9].

Some guaranteed-rate schedulers assign a timestamp to each incoming packet, and forward these packets to the output channel in order of increasing timestamp. Thus, when a packet is received, it is stored in a queue ordered by increasing timestamp. When the output channel becomes idle, the packet with the smallest timestamp is forwarded to the output channel. Examples of these schedulers are Virtual Clock [6,17,19], Weighted Fair Queuing[13, 14], Time-Shift Scheduling [2] and Frame-Based Fair Queuing [16].

The method used to assign timestamps to packets differs from one protocol to another. However, they all share a similar structure. In this paper, we reveal the strong relationship among these protocols, by defining a family of scheduling protocols, called *Universal Timestamp-Scheduling*, and by showing that the above protocols are all members of this family.

The protocol family is based on simulating a virtual server. The virtual server assigns timestamps to each bit of a packet, and it forwards bits in increasing timestamp order. When a flow becomes backlogged in the virtual server, the timestamp of its first bit is computed in accordance to the progress of the virtual server. The difference among the members of the family is the value chosen for the initial timestamp.

We show that all members of the protocol family guarantee to each flow its reserved packet rate. In addition, they guarantee an upper bound on the delay of each packet similar to the delay bound in Virtual Clock scheduling. Finally, we show that the end-to-end delay of each packet is also bounded. Since the protocols mentioned earlier are all members of the protocol family, they all possess these properties.

To show the generality of the protocol family, we show that the above protocols in the literature correspond to one end of the spectrum of the Universal Timestamp-Scheduling family, and that there is another end of the spectrum that is worthy of attention but has not been investigated. In this end of the spectrum, if a flow has not generated packets for a certain amount

of time, the flow will try to reclaim the bandwidth it has lost during its period of inactivity. However, in doing so *it will not infringe upon the basic deadline guarantees of other flows*.

The organization of the paper is as follows. In Section 2, we present the general structure of all members of the protocol family. In Section 3, we present the notation used to specify our protocols. In Section 4, we introduce the concepts of bit timestamps and bit deadlines, which are the foundation of the virtual server. In Section 5, we present the Universal Timestamp-Scheduling family of protocols. In Section 6, we prove the delay bound for all members of the protocol family. The end-to-end delay of the protocol family is given in Section 7. In Section 8, we show that existing protocols are members of the protocol family. Finally, in Section 9, we present related work, and we discuss the end of the spectrum of the Universal Timestamp-Scheduling family which remains to be investigated.

2. Basic Protocol Structure

In this paper, we present a family of scheduling protocols for a computer network. Before doing so, we first present the structure that is common to all members of the family, and then in later sections we describe the difference between members of the protocol family.

A *computer network* consists of a set of computers interconnected via point-to-point bi-directional channels. A *flow* in a computer network is a potentially infinite sequence of packets generated by the same source and having the same destination in the network.

When a new flow wishes to join the network, the network finds a path from the source of the flow to the destination of the flow. Then, the network reserves a fraction of the bit rate of each channel along the path, and assigns this rate to the new flow. Finally, the source of the flow is given permission to generate packets at the rate reserved for the source. The chosen path and reserved rate of the flow remain fixed throughout the lifetime of the flow.

Each output channel of a computer is equipped with a scheduler, as shown in Figure 1. From the input channels, the scheduler receives packets from flows whose next hop to the destination is the output channel of the scheduler. Whenever its output channel becomes idle, the scheduler chooses a received packet and forwards the packet to the output channel.

The rate at which the scheduler forwards the packets of a flow must be bounded from below by the reserved rate of the flow. To guarantee this minimum packet rate, the scheduler assigns a timestamp to each received packet. The timestamp is a function (among other things) of the

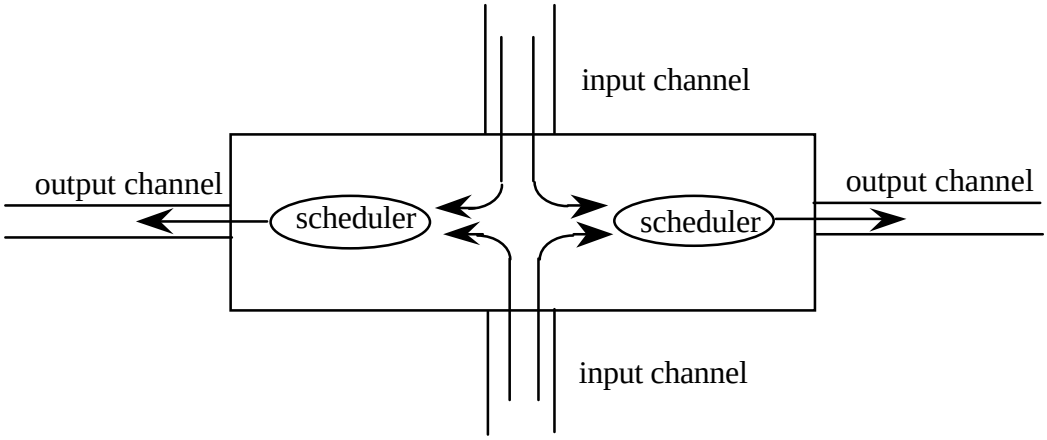

Figure 1: A computer with two input channels and two output channels.

flow's reserved rate. The scheduler forwards the received packets in order of increasing timestamp.

We say that a packet is *forwarded* to the output channel when the first bit of the packet is being transmitted by the output channel. We say a packet has *exited* the output channel if its last bit has been transmitted by the output channel. We say a packet is *in* the output channel if it has been forwarded to the output channel but has not exited the output channel.

The scheduler maintains a separate FIFO queue for the received packets from each flow. An interval of time [t1, t2] is said to be a *busy period* if the output channel is continuously sending packets throughout the interval, and the output channel is idle immediately before t1 and immediately after t2.

We adopt the following notation:

- clock real-time clock of the scheduler.
- N maximum number of flows allowed by the scheduler.
- C bandwidth (bits/sec) of the output channel.
- R.i reserved rate of flow i (bits/sec).
- queue.i queue of received packets of flow i.
- p.i.n nth packet received from flow i.
- A.i.n value of clock when p.i.n is received (sec).
- L.i.n packet size of p.i.n (bits).
- L_{max} upper bound on packet length over all flows.
- E.i.n time when packet p.i.n exits the output channel (sec).

When a scheduler receives a packet from a flow, the packet is assigned a timestamp, and stored in the FIFO queue of the flow. When the output channel becomes idle, the scheduler examines the timestamp of the packet at the head of each queue, and chooses the packet with the smallest timestamp. This packet is dequeued and forwarded to the output channel.

The goal of the scheduler is to forward the packets of each flow i at an average rate of at least R.i. Since all N flows share the output channel, the following constraint is necessary.

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

The timestamp T.i.n assigned to each packet p.i.n is calculated as follows.

$$T.i.n := S.i.n + L.i.n/R.i \qquad (2)$$

In the above equation, $S.i.n \geq T.i.(n-1)$, and its exact value is protocol dependent; the choice of S.i.n is what distinguishes one member of the protocol family from another.

For example, if we choose for S.i.n the value max(A.i.n, T.i.(n-1)), then the resulting protocol is known as Virtual Clock. It can be easily shown by induction that, in the Virtual Clock protocol, T.i.n equals the time at which p.i.n would exit an output channel whose sole input flow is flow i and the channel's rate is exactly R.i.

Furthermore, it has been shown that, in Virtual Clock protocol, each packet exits the scheduler by the time indicated in its timestamp (plus the small constant L_{max}/C) [6,17]. Thus, the timestamp in the Virtual Clock protocol may be viewed as a bound on the exit time of a packet.

The above bound on the exit time of a packet is desirable, since the bound depends solely on the reserved rate of the flow. Thus, the packets of a flow will not experience large delays at a scheduler unless the source of the flow exceeds its reserved packet rate.

If a scheduling protocol has the same upper bound on exit time as the Virtual Clock protocol, we say that the protocol has *rate-proportional delay*.

Other timestamp protocols have different choices for the value of S.i.n, and thus the timestamps assigned to the packets are different than the timestamps assigned in Virtual Clock, and the timestamps may no longer be viewed as an exit deadline. Nonetheless, many of these protocols have rate-proportional delay, i.e., their upper bound on the exit time of a packet is at most the upper bound on the exit time of the same packet in Virtual Clock scheduling.

3. Protocol Notation

In this section, we present our notation to formally specify scheduling protocols, and we use the notation to specify the basic protocol structure of the previous section.

We define the behavior of a scheduler process by a set of global constants, a set local inputs, a set of local variables, and a set of actions. Actions are separated from each other with the symbol [], using the following syntax:

begin *action* [] *action* [] . . . [] *action* **end**

Each action is of the form *guard* $\rightarrow$ *command*. A guard is either a boolean expression involving the local variables of its process, or a receive statement of the form **receive** p.i.n. A command is constructed from sequencing (;), conditional (**if fi**), and iterative (**do od**) constructs that group together **skip**, assignment, and **forward** statements of the form **forward** p.i.n. Similar notations for defining network protocols are discussed in [11] [12].

An action is said to be *enabled* if its guard is either a boolean expression that evaluates to true, or a receive statement of the form **receive** p.i.n, and there is a packet that may be received from some input flow.

An execution step of a protocol consists of choosing any enabled action from any process, and executing the action's command. If the guard of the chosen action is a receive statement **receive** p.i.n, then, before the action's command is executed, the flow number of the packet is stored in variable i, and the packet number is stored in variable n.

Protocol execution is fair, that is, each action that remains continuously enabled is eventually executed.

On occasions, we make use quantifications of the form:

$$(\oplus\ x : R(x) : B(x))$$

Above, $\oplus$ is a commutative and associative binary operator, such as max, min, Σ (summation), $\forall$ (conjunction), or $\exists$ (disjunction). R(x) is a function defining the range of values for the dummy variable x, and B(x) is a function defining the value given as an operand to $\oplus$. For example,

$$(\min\ x : 1 \leq x \leq 3 : x^2)$$

denotes the minimum of 1^2, 2^2, and 3^2 . If the range of the dummy variable x is omitted, all values in the type of x are included.

process Scheduler

constants

N	:	**integer**	/* maximum number of flows */
C	:	**real**	/* rate of output channel */

inputs

R.i	:	**real**	/* reserved rate of flow i */

variables

queue.i	:	**packet sequence**	/* packet queue of flow i */
T.i.n	:	**real**	/* timestamp of packet p.i.n */
clock	:	**real**	/* real-time clock */
idle	:	**real**	/* finishing time of current packet transmission */

begin

receive p.i.n → **compute** S.i.n;
T.i.n := S.i.n + L.i.n/R.i;
enqueue(queue.i, p.i.n)

▯

clock ≥ idle ∧ (∃ j :: queue.j ≠ **empty**) →
p.i.n := **dequeue**(queue);
forward p.i.n;
idle := clock + L.i.n/C

▯

clock < idle → clock := **advance**(clock, idle)

▯

clock ≥ idle ∧ (∀ j :: queue.j = **empty**) →
clock := **advance**(clock, ∞)

end

Figure 2: Scheduler Process

We are now ready to present the code for the scheduler process, which is given in Figure 2. We model the real-time clock with a variable of type real. This variable is incremented at certain points in the code to represent the natural progression of real-time. We make the simplifying assumption that executing an action takes zero time.

In the first action, a packet is received, its timestamp is computed, and it is appended to the queue of its flow. The second action is enabled when the output channel is idle and there is a non-empty queue. If this is the case, function **dequeue**(queue) finds the packet whose timestamp is the smallest, and removes it from the queue of its flow. This packet is then forwarded to the output channel.

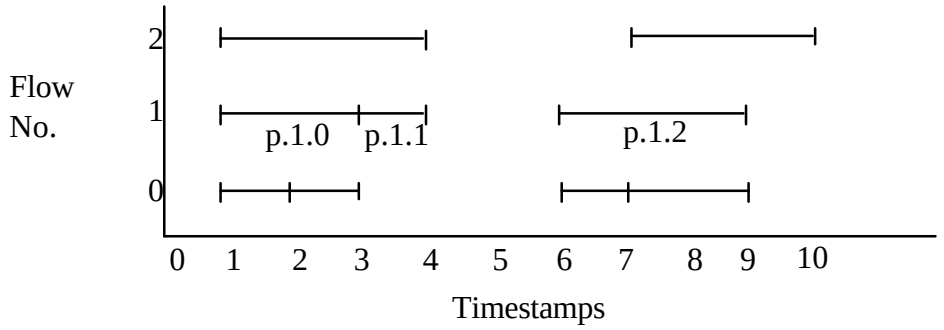

Figure 3: Timestamps contained by each packet

In the third action, if the output channel is not idle, then the clock may be advanced, but not beyond the time when the output channel becomes idle again. This is necessary to ensure that when the channel becomes idle again the next packet will be sent immediately. In this action, function **advance**(clock, idle) chooses any value in the interval (clock, idle] to represent the normal advancement of the clock.

In the final action, if the output channel is idle and all queues are empty, then the clock may advance its value without any constraints.

4. Bit Timestamps and Deadlines

In this section, we broaden the concept of packet timestamps to bit timestamps [2], that is, we define a timestamp to each bit of a packet, and the timestamp of the packet is the timestamp of the last bit of the packet. Also, we associate a deadline to each bit of a packet, and the deadline of the packet is the deadline of the last bit of the packet.

4.1 Bit Timestamps

We next introduce the concept of bit timestamps [2]. We assume that bits are of arbitrarily small size, and a timestamp is defined for each bit as follows. Let b be a real number in the interval [0, L.i.n]. We define the *bit timestamp* of "bit" b of packet p.i.n as:

$$S.i.n + b/R.i \qquad (3)$$

Thus, S.i.n is the timestamp of the first bit of the packet (b = 0) and T.i.n is the timestamp of the last bit of the packet (b = L.i.n). In this way, if at time S.i.n the packet is given to a constant rate server with rate R.i, then bit b of p.i.n exits the server at the time indicated by (3) above.

We say that bit timestamp t is *contained* by packet p.i.n if a bit in the packet has timestamp t, i.e., $S.i.n \le t \le T.i.n$. Flow i contains timestamp t if some packet of i contains t.

For example, consider Figure 3. In this figure, the timestamps of the first and last bits of each packet are indicated by a vertical line. In flow 1, for its first packet, p.1.0, the timestamp of its first bit is 1 and the timestamp of its last bit is 3. For the second packet, p.1.1, the timestamp of its first bit is 3 and the timestamp of its last bit is 4, and for the third packet, p.1.2, its first bit has timestamp 6 and its last bit has timestamp 9. Thus, all timestamp values from 1 up to 4 and from 6 up to 9 are contained by flow 1. Also, no flow contains timestamp 5.

4.2 Bit Deadlines

To ensure rate proportional delay, we associate with each packet p.i.n a deadline D.i.n, which is equal to the timestamp the packet would receive under Virtual Clock scheduling. That is,

$$D.i.n := \max(A.i.n, D.i.(n-1)) + L.i.n/R.i$$

Therefore, a packet scheduler has *rate-proportional delay* if the exit time of each packet p.i.n is at most D.i.n plus a small constant.

We will often need to refer to the deadline of the bit of flow i whose timestamp is t. Thus, D.i.t, where t is a timestamp, denotes the deadline of the bit of flow i whose timestamp is t. Let timestamp t be contained by packet p.i.n, and let bit b, $0 \leq b \leq L.i.n$, be the bit of p.i.n whose timestamp is t. Then, D.i.t is defined as follows.

$$D.i.t = \max(A.i.n, D.i.(n-1)) + b/R.i$$

That is, the deadline of a bit is simply the timestamp the bit would receive in a Virtual Clock scheduler. Thus, in Virtual Clock, the deadline of a bit and its timestamp are equivalent.

5. The Universal Timestamp-Scheduling Family

In this section, we define a family of scheduling protocols, which we name *Universal Timestamp-Scheduling* protocols. Each protocol in the family assigns a timestamp to each received packet according to Assignment (2), and forwards the packets in order of increasing timestamp. However, the protocols differ in their choice of S.i.n. In this section, we define bounds on the possible values used for S.i.n. Regardless of which value is chosen within these bounds, the scheduling protocol will have rate-proportional delay.

To represent the entire behavior of the protocol family, we define a predicate that must be satisfied by the value chosen for S.i.n. We later show that the protocols of Virtual Clock, Weighted Fair Queuing, Time-Shift Scheduling, and Frame-Based Fair Queuing choose values

for S.i.n that satisfy this predicate, and thus, all these protocols are members of the protocol family.

The possible values for S.i.n are derived from the behavior of a virtual server, which we describe next.

5.1 Virtual Server

The virtual server receives as input the same set of flows as the packet scheduler, and forwards the packets of these flows to an output channel whose bandwidth is also C. In addition, each packet is assigned a timestamp. The virtual server also assigns timestamps to packets using Assignment (2).

The difference between the packet scheduler of the previous section and the virtual server is that the virtual server is able to forward a single bit at a time, rather than a whole packet at a time. The virtual server forwards bits in order of increasing bit timestamp. Although the virtual server is not implementable, its behavior can be computed by the scheduler, because the input flows of the virtual server are also the input flows of the scheduler.

Let [t, t'] be an interval such that, if a flow contains a bit timestamp in this interval, then the flow contains all timestamps in this interval; i.e., no flow partially contains the timestamps in this interval. Let B be the set of flows that contain all the bit timestamps in this interval. From the way in which timestamps are assigned to bits, the time required for the virtual server to forward all the bits in this interval is:

$$\frac{\left(\sum i : i \in B : R.i\right) \cdot (t' - t)}{C}$$

Note that this is similar to the way in which bits are forwarded by the virtual server used in Weighted Fair Queuing [14]. However, the behavior of both virtual servers is quite different. In Weighted Fair Queuing, timestamps are not assigned to bits. Therefore, if a flow has bits remaining to be forwarded, then bits from this flow will continuously be forwarded until no more bits of the flow remain. This is not the case in our virtual server, because if all the bits of flow i have timestamps that are smaller than all the bits of flow j, then all the bits of flow i will be forwarded before any bit of flow j is forwarded.

Since bits are forwarded in increasing timestamp order, we define the following.

- W.i timestamp of the next bit to be forwarded from flow i

- T.i largest bit timestamp of flow i; i.e., T.i = T.i.n, where p.i.n is the last packet received from flow i.
- Z timestamp of the next bit to be forwarded by the virtual server.
- F.i.n time when the last bit of p.i.n exits the virtual server.

If W.i < T.i, there are some bits from flow i that remain to be forwarded. If W.i = T.i, all bits from flow i have been forwarded. Thus, if there is an i such that W.i < T.i, then Z is the minimum over all these i, that is,

$$Z = (\min i : W.i < T.i : W.i)$$

On the other hand, if there is no i such that W.i < T.i, then we assign to Z a value greater than the maximum over all timestamps, that is,

$$Z > (\max i : : T.i)$$

We next give a small example of the behavior of the virtual server. Assume the virtual server has an output channel bandwidth of 10 bits/sec, and three flows, where R.0 = R.1 = 2.5 bits/sec, and R.2 = 5 bits/sec. Flows 0 and 1 have two packets each, and flow 2 has a single packet. The size of each packets is 100 bits. The timestamps contained by each flow are shown in Figure 4(a). For the first packet of flow 0, its first bit has timestamp 0 sec. and its last bit has timestamp 40 sec., and for the second packet of flow 0, the first bit has timestamp 40 sec. and the last bit has timestamp 80 sec.. The timestamps of the packets of flow 1 are the same as those of flow 0. The single packet of flow 2 has a timestamp of 20 sec. for its first bit and a timestamp of 40 sec. for its last bit.

The behavior of the virtual server is shown in Figure 4(b). From time 0 up to time 10, the bits from flow 0 and flow 1 with timestamps 0 up to 20 are transmitted over the output channel, for a total of 50 bits from each flow. Thus, at time 10 , W.0 = 20, W.1 = 20, and Z = 20. At time 10, the first bit of the packet from flow 2 begins to be transmitted, since the timestamp of this bit is also 20. Thus, for the next 20 seconds, the virtual server transmits the bits with timestamps 20 up to 40 from all three flows, and the last bit of the first packet of each flow exits at time 30. Therefore, at time 30, W.0 = 40, W.1 = 40, W.3 = 40, and Z = 40. Then, since only the second packets from flow 0 and 1 remain, these packets are transmitted during the next 20 seconds. Hence, at time 50, W.0 = 80, W.1 = 80, W.2 = 40, and Z > 80.

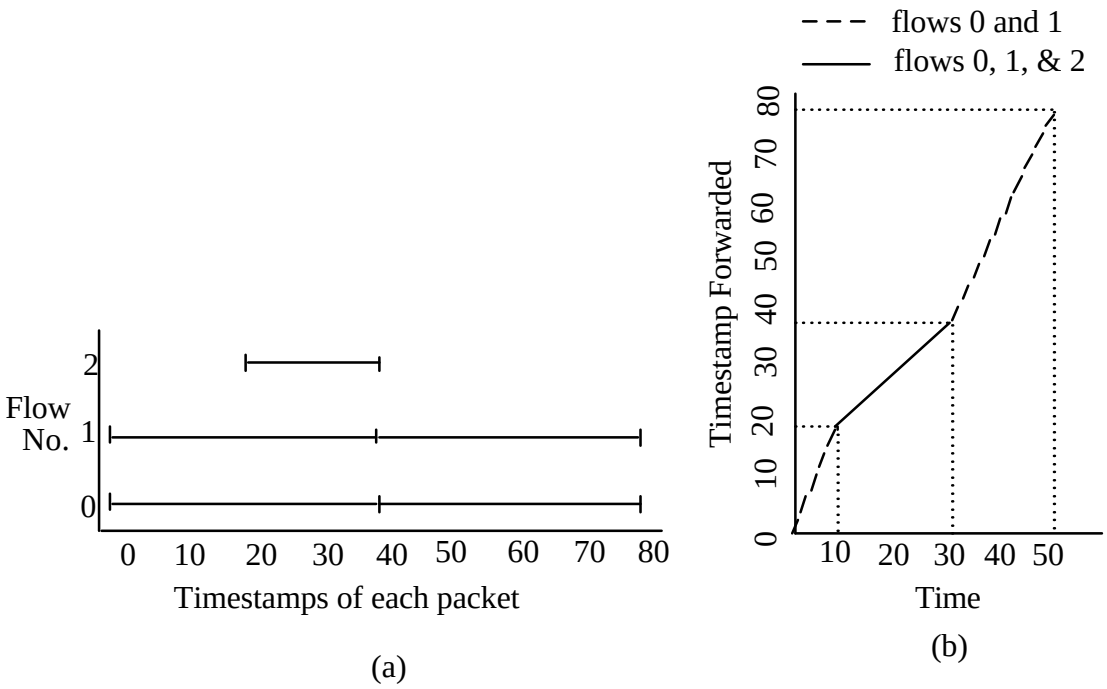

Figure 4

5.2 Virtual Server Specification

We next present the specification of the virtual server, which is given in Figure 5. There are four new variables, StartS, Z, B, and upd.

StartS is a lower bound on the value chosen for S.i.n. Z is the timestamp of the next bit to be transmitted, and B is the set of flows which still have bits to be transmitted and whose next bit to be transmitted has timestamp equal to Z.

process Virtual Server

constants

N	:	**integer**	/* maximum number of flows */
C	:	**real**	/* rate of output channel */

inputs

R.i	:	**real**	/* reserved rate of flow i */

variables

T.i.n	:	**real**	/* timestamp of packet p.i.n */
T.i	:	**real**	/* largest timestamp of flow i */
W.i	:	**real**	/* timestamp of next bit to be transmitted of flow i */
Z	:	**real**	/* next bit timestamp to be transmitted */
B	:	**set of integer**	/* set of flows whose next bit will be transmitted next */
upd	:	**real**	/* timestamp where the membership in B changes */
clock	:	**real**	/* real-time clock */
StartS	:	**real**	/* low bound for S.i.n */

begin

receive p.i.n → **if** W.i = T.i.(n-1) → **choose** S.i.n;
W.i := S.i.n;
update StartS
▯ W.i < T.i.(n-1) → S.i.n := T.i.(n-1)

fi;
T.i.n := S.i.n + L.i.n/R.i;
T.i := T.i.n;
Z := (min i : W.i < T.i : W.i);
B = set {i | W.i = Z ∧ W.i < T.i};
upd := min { (min i : i ∈ B : T.i), (min i : i ∉ B ∧ W.i < T.i : W.i) }

▯

Z < upd → Z := **advance**(Z, upd);
clock := clock + (Σ i : i ∈ B : (Z - W.i)·R.i)/C;
do for each i ∈ B
W.i := Z
od

▯

Z = upd → B := set { i | W.i < T.i ∧ W.i = (min i : W.i < T.i : W.i)};
if B = **empty** → Z := **advance**((max i :: T.i), ∞)
▯ B ≠ **empty** → Z := (min i : W.i < T.i : W.i);
upd := min { (min i : i ∈ B : T.i),
(min i : i ∉ B ∧ W.i < T.i : W.i) }
fi

▯

Z > (max i :: T.i) → clock := **advance**(clock, ∞)

end

Figure 5: Virtual Server

Variable upd (update) contains the timestamp such that when Z = upd, the membership in set B changes. This will occur when either a flow i in B does not have any more bits to transmit, or a new flow i should be added to B. Thus, upd is the minimum of two values: the minimum of all T.i for each i in B, and the minimum of W.i such that i is not in B and W.i < T.i.

The specification consists of four actions.

In the first action, a packet is received. Before assigning a timestamp to the packet, the value of S.i.n is computed, and the lower bound StartS is updated (in the next section, we explain in detail how S.i.n is computed and how StartS is updated). After the packet is assigned its timestamp, Z, B, and upd are recomputed because their values may have changed.

In the second action, if there are still bits to be forwarded from the flows in B without causing a change in the membership of B, i.e., if Z < upd, then we arbitrarily choose how many more of these bits to transmit. Thus, Z is updated accordingly, and the amount of time needed to transmit these bits is added to the clock.

In the third action, if the membership of B is about to change, i.e., if Z = upd, then it is time to update B. If the new value of B is empty, then there are no more bits to transmit, and Z is set to a value greater than the maximum of all bit timestamps. If B is not empty, then Z and upd are recomputed.

In the last action, if there are no more bits to transmit, the clock is advanced an arbitrary amount of time.

5.3 Computing the Starting Timestamp

What remains to be defined for the virtual server is the way in which S.i.n is chosen. If bits from flow i remain to be forwarded (W.i < T.i), then S.i.n is assigned T.i.(n-1). On the other hand, if no bits remain to be forwarded (W.i = T.i), then there is some flexibility in the choice of S.i.n.

We would like to identify a range of values for S.i.n such that choosing any one of them ensures rate proportional delay for the virtual server. This range of values should be as large as possible, to provide the flexibility to represent many protocols in the literature as members of our protocol family.

We need to compute a value for S.i.n that will ensure a rate proportional delay in the virtual server at the bit level. That is, each bit should exit the virtual server by the time indicated by its bit deadline.

To guarantee that a bit from flow j with timestamp W.j exits by its deadline, we simply count, for each flow k, where W.k < W.j, the maximum number of bits that could have a timestamp at most W.j and are yet to be transmitted. These bits are:

$$(W.j - \max(W.k, Z)) \cdot R.k$$

To obtain the amount of time required to transmit these bits, simply divide by C. The virtual server should have enough time to transmit all these bits from every flow k before the deadline of bit W.j of flow j expires. Thus, the following predicate should always hold in the virtual server:.

$$(\forall\, j : Z \le W.j :$$

$$\frac{\left(\sum k : W.k < W.j : (W.j - \max(W.k, Z)) \cdot R.k\right)}{C} \le D.j.(W.j) - clock \qquad (4)$$

)

We refer to the above predicated as the *safety predicate*. We will later show that if the above predicate holds, it continues to hold as bits are transmitted, and thus, each bit exits no later than its deadline.

However, assume packet p.i.n is received, and no more bits remain to be transmitted from flow i. If S.i.n is chosen such that S.i.n < Z, then Z is assigned S.i.n. This change in Z could invalidate the safety predicate. Thus, we restrict the value chosen for S.i.n such that the safety predicate holds after p.i.n is assigned a timestamp. This will ensure that the safety predicate holds at all times.

In addition, we restrict S.i.n to be at least T.i.(n-1), to prevent bits from different packets from having the same timestamp.

We next present an example to illustrate the safety predicate. Assume we have three flows, i, j, and k, where R.i = R.j = R.k = 100 bits/sec., and C = 300 bits/sec. Initially, W.i, W.j, W.k, and Z are all zero. Also, assume for simplicity that all packets are of size 100 bits and initially clock = 0.

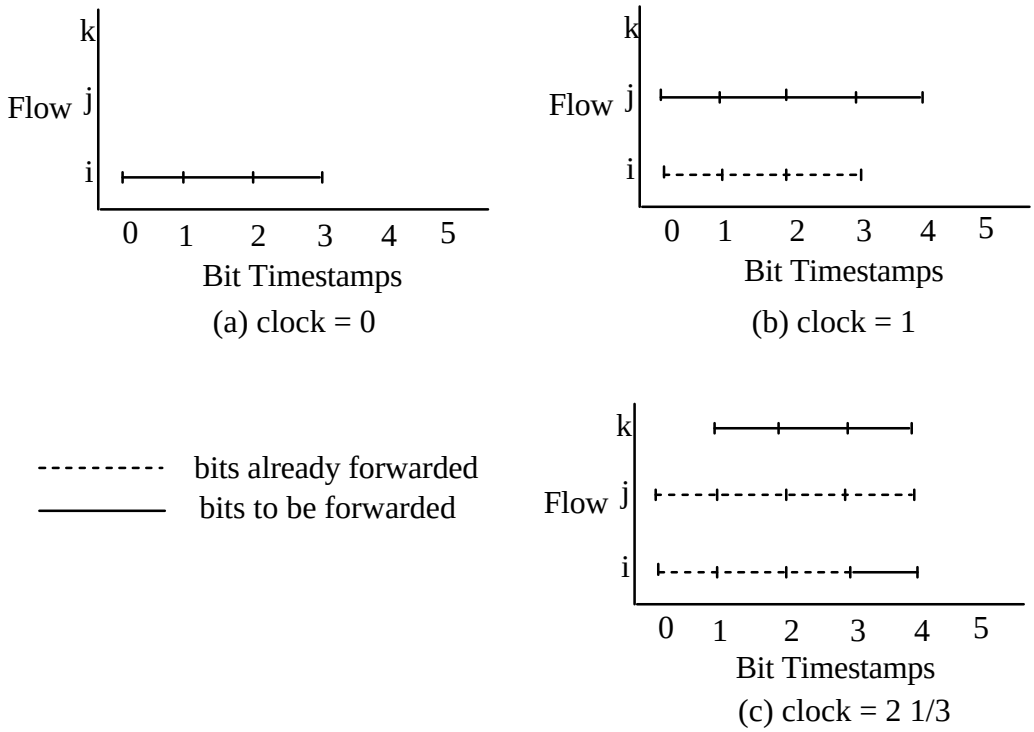

Figure 6

At time 0, three packets are received from flow i. The virtual server chooses S.i.0 = 0, and thus, the last packet receives a timestamp equal to 3, and receives a deadline also equal to 3. This is indicated in Figure 6(a). It takes one second for the virtual server to transmit all three packets. After transmitting the last bit, Z = 3 and clock = 1.

Then, when clock = 1, four packets are received from flow j. Note that S.j.0 may be assigned zero without violating the safety predicate. Thus, assume we choose S.j.0 = 0. The last packet of j receives a timestamp of 4 and a deadline of 5, as shown in Figure 6(b). The virtual server takes 1 1/3 sec. to transmit these packets. After transmitting the last bit, Z = 4 and clock = 2 1/3.

Next, three packets are received from flow k, and then one packet is received from flow i. The value of zero is no longer valid for S.k.0. This is because the timestamp of the first bit of the next packet of i is 3, and there is not enough time to transmit all bits of flow k with timestamps 0 to 3 before the deadline of the next bit of i, which also equals 3, expires. Namely,

$$(W.i - \max(W.k, 0))\cdot R.k/C = (3 - 0)\cdot 100/300 = 1 \; > \; D.i.(W.i) - clock = 3 - 7/3 = 2/3$$

In fact, the smallest value possible for S.k.0 that satisfies the safety predicate is one. Thus, assume we choose S.k.0 = 1. Thus, the timestamp of the last packet of k is 4, its deadline is 5 1/3, the timestamp of the packet of i is 4, and its deadline is also 4.

A packet scheduler belongs to the Universal Timestamp-Scheduling family if it assigns to S.i.n a value equal to the value chosen by the virtual server. It can be shown that the two restrictions given earlier on the value chosen for S.i.n will ensure rate-proportional delay in the virtual server. However, they do not ensure that the packet scheduler will have rate-proportional delay. To accomplish this, we will require the following additional restriction on S.i.n.

We define a non-decreasing lower bound on the value of S.i.n, called StartS. The value chosen for S.i.n must be at least StartS. After W.i is assigned S.i.n, StartS is updated. Its new value is the smallest value, no smaller than the previous value of StartS, such that

$$(\forall\ j : StartS \le W.j : \frac{\left(\sum k : W.k < W.j : (W.j - \max(W.k, StartS)) \cdot R.k\right)}{C} \le D.j.(W.j) - clock \qquad (5)$$

)

That is, if any future packets are given bit timestamps whose values are at least StartS, then all bits currently in the virtual server will exit no later than their deadlines.

The above predicate is known as the *start predicate*, because it gives a lower bound on the starting value of the timestamps of a flow.

Notice that (4) is the same as (5) if we replace Z by StartS. Thus, we will later show that StartS is always at most Z.

In summary, we have three requirements for the value chosen for S.i.n:

a) $T.i.(n-1) \le S.i.n$.

b) $StartS \le S.i.n$.

c) The safety predicate holds after the packet receives its timestamp.

We will show in a later section that these requirements guarantee that the packet scheduler also has rate-proportional delay.

6. Upper Delay Bound

In this section, we prove that both the virtual server and the Universal Timestamp-Scheduler have rate-proportional delay. We first address the virtual server, and then the packet scheduler.

6.1 Delay Bound of The Virtual Server

We begin with a few lemmas. In the following, the expression "at time t" refers to the state of the virtual server when variable clock has the value t.

Lemma 0 - After its first packet is received, each flow i will always contain all timestamps in the interval [W.i, T.i].

Proof

The proof is by induction over the packet number of flow i. For the base case, p.i.0 is received and after it is timestamped, W.i = S.i.0 < T.i.0 = T.i. When W.i increases, it is not increased beyond T.i, and T.i does not change until the second packet is received. Thus, flow i contains all timestamps in the interval [W.i, T.i].

For the induction step, assume after p.i.(n-1) is received, flow i contains all timestamps in [W.i, T.i.(n-1)].

If W.i < T.i = T.i.(n-1), then S.i.n is assigned T.i.(n-1), and afterwards, W.i < T.i.(n-1) = S.i.n < T.i.n = T.i. Hence, from the induction hypothesis, flow i contains all bit timestamps in the interval [W.i, T.i].

If W.i = T.i, then after p.i.n is timestamped, W.i = S.i.n < T.i.n = T.i, and hence, flow i contains all bit timestamps in the interval [W.i, T.i].

Furthermore, W.i will not increase beyond T.i, and T.i does not change until the next packet is received. Thus, the theorem holds.

♦

Lemma 1 - Let $t' < t$, and let Z' and Z be the values of Z at times t' and t, respectively. If during the time interval [t' , t], the virtual server has bits to forward, and Z does not decrease, then

$$t - t' \leq Z - Z'$$

Proof:

Consider the second action of the virtual server, which is where the clock is advanced when the server is not idle, and consider the second assignment in this action.

$$clock := clock + (\Sigma\ i : i \in B : (Z - W.i) \cdot R.i)/C$$

At this point, each W.i has the value of Z'. Thus, we have,

$$clock := clock + (\Sigma\ i : i \in B : (Z - Z') \cdot R.i)/C$$

From Relation (1),

$$(\Sigma\, i : i \in B : (Z - Z')\cdot R.i)/C = (\Sigma\, i : i \in B : R.i)\cdot(Z - Z')/C \leq C\cdot(Z - Z')/C = Z - Z'$$

Thus, the value of Z increases at least as fast as the value of clock.

♦

Lemma 2

a) Assume $W.i = Z < T.i$ and $D.i.(W.i) \geq$ clock. Then, as long as $W.i = Z < T.i$ holds (i.e., as long as Z increases monotonically and flow i has bits to send), we have $D.i.(W.i) \geq$ clock.

b) If for some constant s, $W.i < s \leq T.i$, then eventually $s \leq W.i$.

Proof

a) From Lemma 1, if Z increases monotonically, it increases at least as fast as real-time. Hence, W.i increases at least as fast as real time. From the definition of bit timestamps, if flow i contains all timestamps in the interval [t', t], then those bits corresponding to these timestamps are exactly $(t - t')\cdot R.i$ bits. Thus, since the deadline of a bit is the time at which the bit exits a constant rate server with rate R.i, then $D.i.t - D.i.t' \geq t - t'$.

Note that as long as $W.i = Z < T.i$, flow i contains the timestamp W.i as Z increases. Thus, from the above, D.i.(W.i) also increases at least as fast as real-time, and $D.i.(W.i) \geq$ clock continues to hold.

b) To increase W.i beyond s, the virtual server transmits bits whose timestamps are at most s. These bits can be at most $(\Sigma\, j :: R.j\cdot s) = (\Sigma\, j :: R.j)\cdot s \leq C\cdot s$ bits, which take at most s seconds to forward. Hence, within at most s seconds W.i will increase beyond s.

♦

Lemma 3 - If before executing the first action, $StartS \leq Z$, and also, the safety predicate is true, then, after executing the first action, $StartS \leq Z$, and the safety predicate remains true.

Proof

First, we must show that there is at least one value of S.i.n that can be chosen such that: $T.i.(n-1) \leq S.i.n$, $StartS \leq S.i.n$, and the safety predicate holds after p.i.n is timestamped. We then have to show that after StartS is updated, $StartS \leq Z$.

If $W.i < T.i$, then S.i.n is assigned T.i.(n-1). In this case, the safety predicate still holds since neither W.i nor D.i.(W.i) have changed. Furthermore, when StartS is updated, StartS cannot be greater than Z, because $StartS \leq Z$ held before the action is executed, and (5) holds if we replace StartS by Z.

If W.i = T.i (i.e., W.i = T.i.(n-1)), then consider assigning to S.i.n the max(T.i.(n-1), Z). Regardless of which of these two values is greater, W.i = S.i.n after executing the action.

If T.i.(n-1) ≤ Z, S.i.n is chosen to be Z. It is easy to check that the safety predicate still holds, since max(Z, W.i) = max(Z, S.i.n) does not change, and D.i.(W.i) = D.i.(S.i.n) ≥ clock. Since StartS ≤ Z before executing the action, StartS ≤ S.i.n, and since the safety predicate continues to hold, StartS ≤ Z after StartS is updated.

Assume now Z < T.i.(n-1) = W.i. Thus, S.i.n is assigned T.i.(n-1), and W.i does not change. Also, because Z < T.i = T.i.(n-1), there is another flow j, where Z = W.j < T.j, and thus Z does not change value. Finally, if the safety predicate holds, it is impossible for Z < W.i if the deadline of bit W.i has expired. This implies that D.i.(W.i) does not change. Since none of W.i, D.i.(W.i), or Z changes, the safety predicate still holds. Furthermore, since the safety predicate holds and Z does not change value, StartS ≤ Z still holds when StartS is updated.

♦

Lemma 4 - If the safety predicate holds before executing the second, third, or fourth action, then the safety predicate will remain true after executing the action.

Proof

In the second action, some bits are forwarded. Also, variables Z, clock, and every W.i in B are increased accordingly.

In the safety predicate, consider any flow j. We are required to show that

$$\frac{\left(\sum k : W.k \le W.j : (W.j - \max(W.k, Z)) \cdot R.k\right)}{C} \le D.j.(W.j) - clock$$

If before the action is executed, W.j < Z , then W.j < Z after executing the action, and we have no proof obligation for j.

If before the action is executed, W.j = Z, we have two choices. If W.j = T.j, then j is not in B, and after the action is executed, W.j < Z, and thus we have no proof obligation for j. If Z = W.j < T.j, then after the action is executed, Z = W.j ≤ T.j, and the left hand side of our obligation is 0. Notice that the left hand side is always at least zero, thus, since the predicate held before the action is executed, D.j.(W.j) ≥ clock before execution. From Lemma 2(a), D.j.(W.j) ≥ clock continues to hold after the action is executed.

Finally, if Z < W.j before the action is executed, then j is not in B, and after the execution, Z ≤ W.j. Thus, W.j and D.j.(W.j) did not change. Let us look at the changes to the left and right

hand sides of our proof obligation above. The left hand side decreases since Z increased, and the right hand side decreases since clock increases by

$$\frac{\left(\sum k : k \in B : R.k\right) \cdot (Z - Z')}{C}$$

where Z' is the value of Z before the execution. Consider any flow k, and see how much both sides change due to k.

If k is in B, then k's contribution to reducing the left hand side is (Z - Z')· R.k/C, which is the same as k's contribution to the increase in the clock, so both sides remain balanced.

If k is not in B, then k does not contribute to changes to the right side. Also, W.k does not change, and we have two cases.

a) if Z' < W.k, then upd ≤ W.k, and Z cannot increase beyond W.k, and max(W.k, Z) remains the same, so the left side does not change.

b) if W.k ≤ Z', then the left side reduced by (Z - Z')· R.k/C without any contribution to the right side, so the predicate continues to hold.

Consider now the third action. It does not change any W.i, but it may increase Z. Notice that increasing Z preserves (5), so the safety predicate continues to hold.

The last action increases the clock, but only if there are no bits to transmit, in which case Z > W.i for all i, and the safety predicate holds trivially.

♦

Corollary 1 - At all times, the safety predicate is true and StartS ≤ Z.

Proof

From Lemma 3, the first action preserves StartS ≤ Z and the safety predicate. From Lemma 4, the remaining actions preserve the safety predicate. Since Z increases and StartS remains constant in these actions, then StartS ≤ Z is also preserved by these actions.

♦

Theorem 1 For any j, if W.j < T.j, then clock ≤ D.j.(W.j).

Proof

As long as W.j < T.j we have, from the definition of Z, Z ≤ W.j. From Corollary 1 and (4),

$$\frac{\left(\sum k : W.k \le W.j : (W.j - \max(W.k, Z)) \cdot R.k\right)}{C} \le D.j.(W.j) - clock$$

Since Z ≤ W.j, the left-hand-side is always at least zero, and thus clock ≤ D.j.(W.j).

♦

Theorem 1 states that each bit in the virtual server exits before its deadline expires. Let us now examine the behavior of the packet scheduler.

6.2 Delay Bound of Packet Scheduler

A Universal Timestamp-Scheduler is a packet scheduler that chooses a value for S.i.n equal to the value of S.i.n in the virtual server. Thus, both the packet scheduler and the virtual server will assign equal timestamps to the same packet. Furthermore, the deadline of a packet in both the packet scheduler and the virtual server are also the same. What remains to be shown is that the packet scheduler has rate proportional delay, that is, that each packet will exit by its deadline plus a small constant, namely, L_{max}/C. To show this, we take advantage of the behavior of the virtual server.

Lemma 5 - Let t, $t \leq A.i.n$, be the latest time when all packet queues of a Universal Timestamp-Scheduler are empty. If, from time t up to time E.i.n, the scheduler only forwards packets with timestamps at most T.i.n, then

$$E.i.n \leq F.i.n$$

Proof

Since the input flows of both the virtual server and the packet scheduler are the same and their output channel's have equal rate, then the times when all queues are empty coincide in both systems. Thus, at time t, the virtual server has no more bits to forward.

At time F.i.n, $A.i.n < F.i.n$, the virtual server has forwarded all bits with timestamps at most T.i.n. If $F.i.n < E.i.n$, this implies the packet scheduler has forwarded at least one packet with timestamp at most T.i.n, other than p.i.n, to the output channel before F.i.n, and not all bits (if any) of this packet have been forwarded by the virtual server by time F.i.n. Let p.j.m be the first of these packets to be forwarded by the packet scheduler.

Consider the arrival of p.j.m. It cannot be that $A.j.m < F.i.n$, because by time F.i.n the virtual server forwarded all bits with timestamps at most T.i.n, and we are given that $T.j.m \leq T.i.n$ and $F.i.n < F.j.m$. On the other hand, $F.i.n \leq A.j.m$ is impossible, since p.j.m is forwarded by the scheduler before F.i.n. Thus, p.j.m cannot exist, and $E.i.n \leq F.i.n$.

♦

Theorem 2 - In a Universal Timestamp-Scheduler, for all i and n,

$$E.i.n \leq D.i.n + L_{max}/C.$$

Proof

Consider the latest time t, $t \leq A.i.n$, such that the packet scheduler had all packet queues empty, or it forwarded a packet with a timestamp greater than T.i.n.

If t corresponds to when the queues were empty, then from Lemma 5, $E.i.n \leq F.i.n$, and from Theorem 1, $E.i.n \leq F.i.n \leq D.i.n$.

If the time corresponds to when a packet with greater timestamp was forwarded, then note that at time t there are no packets with timestamps smaller than T.i.n. Thus, all packets with timestamps smaller than T.i.n that are forwarded before p.i.n will arrive after time t.

Consider the virtual server when p.i.n is received and is assigned a timestamp. From Corollary 1, after the first action is executed and from the definition of Z, $StartS \leq Z \leq W.i \leq S.i.n$.

From the definition of StartS, Relation (5) holds after executing the action, and recall that StartS is non-decreasing. Thus, all future bits to be forwarded (whether they have been received already or not) with timestamps at most W.i take no more than D.i.(W.i) - A.i.n (clock = A.i.n) seconds to transmit. From the definition of the deadline of a bit, and since flow i contains all bit timestamps in the range [W.i, T.i.n] (Lemma 0), then $D.i.n \geq D.i.(W.i) + (T.i.n - W.i)$. Furthermore, due to (1), all bits (from any flow) with timestamps in the range [W.i, T.i.n] take at most T.i.n - W.i seconds to transmit. Thus, starting at time A.i.n, all bits (whether received or not) to be transmitted by the virtual server with timestamps at most T.i.n take at most D.i.n - A.i.n seconds to transmit.

Let Y be the set of packets with timestamps at most T.i.n (including p.i.n) received after t and forwarded by the scheduler up until p.i.n is forwarded. From time t up to time A.i.n, some of the bits in Y may have been transmitted by the virtual server. The remaining bits of Y are contained in the calculation above made at time A.i.n. Thus, starting at time t, the virtual server has enough time to transmit all bits in Y before the deadline of p.i.n expires. Since the virtual server and the packet server have an output channel with rate C, the packet scheduler has enough time also. However, since at time t a packet is being transmitted with timestamp greater than T.i.n, by time $D.i.n + L_{max}/C$ the scheduler will transmit all bits in Y.

♦

We therefore conclude that any Universal Timestamp-Scheduler has rate-proportional delay. Thus, when a new scheduling protocol based on packet timestamps is designed, the designer

needs only to show that the protocol belongs to the family of Universal Timestamp-Scheduling protocols, and as a corollary the protocol has rate-proportional delay.

7. End-to-End Delay Bound

Consider the network path of a flow from its source to its destination, and let the number of schedulers in this path be K. Let R be the reserved rate of the flow. Furthermore, assume each of the computers uses a scheduling protocol that has rate-proportional delay. In [1, 3, 10], it has been shown that, for a source whose flow of packets is constrained by a leaky bucket of size b and rate R, the end-to-end delay of a packet along its path is at most

$$b/R + (K-1)\cdot(L/R) + K\cdot(L_{max}/C)$$

In the above, L is the maximum packet size of the flow, and L_{max} is the maximum packet size allowed along the path.

Since all protocols in the Universal Timestamp-Scheduling family have rate-proportional delay, then a source whose network path consists of schedulers of this family has the above upper bound on end-to-end packet delay.

8. Existing Family Members

In this section, we show the flexibility of our family of Universal Timestamp-Scheduling protocols by showing that many scheduling protocols in the literature belong to the family.

Assume the packet scheduler chooses S.i.n as follows:

$$S.i.n := \max(V, T.i.(n-1)) \qquad (6)$$

where V is a value that increases at least as fast as real-time. That is, if at some point in time, V - clock $\geq \alpha$, for some constant α, then this will continue to hold forever. Furthermore, assume we are given that $V \leq Z$ always holds. We next show that if this is the case, the value of S.i.n is the same as in the virtual server.

We begin with a lemma.

Lemma 6 - Consider a flow i and packet p.i.n, such that W.i = T.i when S.i.n is received. Assume the virtual server chooses S.i.n to be contained in the range [T.i.(n-1), max(Z, T.i.(n-1))]. (i.e., eliminate the restriction of the safety predicate and StartS). Then, if the packet scheduler chooses S.i.n according to (6), then this is also a possible choice of S.i.n in the virtual server.

Proof

Consider first when W.i < T.i. In this case, the virtual server has no choice and assigns S.i.n = T.i.(n-1). Since V ≤ Z ≤ W.i, then V < T.i = T.i.(n-1), and S.i.n = T.i.(n-1) also in the packet scheduler.

Consider now when W.i = T.i. If V < T.i.(n-1), then the packet scheduler assigns S.i.n = T.i.(n-1), which is valid also in the virtual server. If T.i.(n-1) ≤ V, then S.i.n = V in the packet scheduler. Since V ≤ Z, then V is a valid choice of S.i.n in the virtual server.

♦

We thus have that the timestamps chosen by the packet scheduler may also be chosen by the virtual server.

Since V increases as fast as real-time, the timestamp of a packet may be viewed as a pseudo-deadline, i.e., a packet should exit the scheduler before V reaches a value greater than its timestamp. It is easy to show by induction that, after packet p.i.n is timestamped, for any bit timestamp t contained by p.i.n,

$$t - V \leq D.i.t - clock \qquad (7)$$

Furthermore, since V increases at least as fast as the clock, the above always holds.

We next show that choosing S.i.n as in (6) ensures that the safety predicate holds and that StartS ≤ S.i.n. Thus, any packet scheduler choosing S.i.n as in (6) above is a member of the Universal Timestamp-Scheduling family.

Theorem 3 If a packet scheduler assigns timestamps according to (6), then S.i.n satisfies all three conditions required by the virtual server.

Proof

We have shown in Lemma 6 that the packet scheduler and the virtual server will choose the same value of S.i.n if we eliminate the restriction of the safety predicate and StartS. What we show next is that these two restrictions are also satisfied if we choose S.i.n according to (6).

We have to show that the value chosen for S.i.n is at least StartS, and that the safety predicate holds afterwards. To do so, we also show that StartS ≤ V always holds. Notice that V is non-decreasing, and StartS increases only in the first action. Thus, assume StartS ≤ V, and consider the first action.

Assume the received packet is assigned its timestamp and StartS has not been updated. Consider any j, where W.j ≥ V. From Relations (1) and (7),

$$(\sum k : W.k < W.j : (W.j - \max(V, W.k))\cdot R.k/C) \leq (W.j - V)\cdot C/C = W.j - V \leq D.j.(W.j) - clock$$

Therefore, in Relation (5), if we replace StartS by V, then the relation still holds. Because of this, and because StartS ≤ V before executing the action, then the value of StartS after the action is at most V.

Also, since V ≤ Z, then Relation (5) with StartS replaced by V is stronger than the safety property, and the safety property also holds.

Finally, since S.i.n := max(V, T.i.(n-1)), and StartS ≤ V, then StartS ≤ S.i.n.

We therefore have that S.i.n satisfies all three requirements of the virtual server.

♦

Consider for example the case of the Virtual Clock protocol [19]. In this case, the value of V is simply chosen to be the real-time clock, that is, V = clock, and Relation (6) above reduces to

$$S.i.n := \max(clock, T.i.(n-1))$$

Thus, in this case, $\alpha = 0$. What we need to show is that V ≤ Z. This is easily shown as follows. Because the sum of the reserved rates of all the flows is at most the capacity of the output channel of the virtual server, the rate of increase of Z is at least the rate of increase of the real-time clock. Furthermore, when Z is decreased, it is set to S.i.n, where p.i.n is the latest packet received. Finally, from the above relation, S.i.n ≥ clock, and hence Z ≥ clock.

Therefore, V = clock ≤ Z at all times, and from Theorem 3, Virtual Clock belongs to the Universal Timestamp Scheduling family of protocols.

Showing that Weighted Fair Queuing [13, 14] belongs to this family is also easy. In this case, we choose V to be equal to Z at all times, hence, V ≤ Z trivially holds. We need to show that V increases at least as fast as real-time, but as we argued above, since the capacity of the virtual server is not overallocated, Z (which equals V) increases at least as fast as real-time.

Finally, we would like to show that Time-Shift Scheduling [2] belongs to this familty of protocols. In Time-Shift Scheduling, V is actually an adjustable clock, which increases at the same rate as any real-time clock, except that on special ocasions its value is adjusted forward, as follows. Let S_{min} be the minimum value of all the S.i.n values of all packets currently in the queue (including the one currently in transmission). Whenever the scheduler detects that V < S_{min}, it sets V := S_{min}.

From the definition of V, V increases at least as fast as real-time. The only thing we need to show is that V ≤ Z at all times. We have shown earlier that Z increases at least as fast as real-time. Thus, as long as V is not adusted forward and Z is not decreased, V ≤ Z holds.

When Z is decreased, is because a new packet p.i.n is received with S.i.n < Z. The new value of Z will be S.i.n. From the timestamp formula, S.i.n ≥ V, and hence, the new value of Z ≥ V.

When V is increased, it's new value is S_{min}. Thus, we only need to show that $S_{min} \leq Z$. This is easy to see, because the virtual server forwards bits in increasing *bit* timestamp order, while the packet scheduler forwards bits in increasing *paket* timestamp order. Therefore, the minimum bit timestamps in the packet scheduler are always at most the minimum bit timestamps in the virtual server.

Hence, from Theorem 3, Time-Shift scheduling belongs to the Universal Timestamp Scheduling family.

In [15], it was shown that Frame-Based Fair Queuing has properties similar to those needed for Relation (6), and the argument is not repeated here.

Therefore, for all of these protocols, rather than proving directly that they have the rate-proportional delay property, which may not be an easy task, one simply needs to prove that they belong to the Universal Timestamp Scheduling family, and the rate-proportional delay property of the protocol becomes an immediate corollary of this proof.

9. Related and Future Work

All protocols in the literature that assign timestamps to packets and exhibit rate-proportional delay compute timestamps according to some virtual function V defined in (6) above. Function V has the restriction that it must always increase as fast as real-time, and it must be at most Z. Protocols in the literature differ from one another in their specific choice of V.

For each flow i, let p.i.n denote the packet at the head of its queue, and let S.i denote S.i.n. Also, let S_{min} denote the minimum of all the S.i such that the queue of flow i is not empty in the packet scheduler. In [2, 3], we have shown that, if we choose V to increase at least as fast as real time, and do not allow V to lag behind S_{min}, then all protocols which choose V in this range have rate-proportional delay.

In [15, 16], it was shown that the values of V can be extended to include Z. Thus, any value of V that increases as fast as real-time and is at most Z, as required in (6) above, guarantees rate-proportional delay. To show this, the authors of [15, 16] present a virtual server similar to the one we presented in this paper.

In this paper, we have significantly extended the spectrum of protocols that satisfy the rate proportional delay. To do so, we defined a virtual server that in addition to taking into

consideration the timestamp of a packet, it also takes into consideration the *deadline* of a packet. Furthermore, we extended the concepts of packet timestamps and packet deadlines to include bit timestamps and bit deadlines. Most importantly, we have *eliminated* the virtual function V, and we defined a *weak* safety predicate that must be satisfied by the initial timestamp S.i.n to guarantee rate-proportional delay.

Due to the weakness of the safety predicate, a broad range of values are possible for S.i.n. For example, we have shown that S.i.n does not need to increase as fast as real-time. To see this, simple take a look at the example in Section 5.3. Here, the values chosen for S.j.0 and S.k.0 are much smaller than real-time. In addition, it is easy to obtain examples in which S.i.n is given a value greater than max(Z, T.i.(n-1)) without violating the safety property. Finally, if a protocol uses a virtual function V, once a packet p.i.n receives an S.i.n value greater than T.i.(n-1), then no other packet p.j.m received after p.i.n can receive an S.j.m value less than S.i.n. This, however, is possible to do without violating the safety predicate, and thus the virtual function V is an unnecessary restriction.

Therefore, we have defined a much broader spectrum of possible values for S.i.n than those allowed by (6). This allows the protocol designer a greater flexibility in choosing S.i.n, while at the same time preserving rate-proportional delay.

In particular, in the same way that a myriad of protocols which increase S.i.n at least as fast as real-time have been developed, we believe that there is also a myriad of protocols in which S.i.n does not increase as fast as real-time that may be developed, and are worthy of investigation.

For example, consider a protocol in which the value chosen for S.i.n is always the smallest value possible that satisfies the safety predicate. In this case, if a flow has not generated packets for a certain amount of time, the flow will try to reclaim the bandwidth it has lost during its period of inactivity. However, in doing so *it will not infringe upon the basic deadline guarantees of other flows.*

If we assume that the source of the flow has paid money for its reserved bandwidth, it is sensible to assume that the source would like, if possible, to recapture some of the bandwidth it did not use during a period of idleness. Other protocols, such as [2, 14, 16, 20], distribute the unused bandwidth among the flows which still have packets to be forwarded, and do not allow a source that has been idle for some time to recuperate its unused bandwidth. Which of these two choices is more appropriate will depend on the nature of the applications supported. For

example, in mobile computing, a user may become disconnected from the network due to the unreliable wireless link, not because the user is idle. In this case, the user may want to recuperate as much of the bandwidth it failed to use while disconnected, without violating the deadlines of other users.

Computing the minimum value of S.i.n that satisfies the safety predicate takes O(N) time. It would be interesting to obtain protocols which approximate this value with lower complexity, such as O(log(N)) time, in the same way that approximations to Weighted Fair Queuing [2, 16] compute the value of S.i.n in O(log(N)) or O(1) time rather than O(N).

We have so far consider only the case of rate-proportional delay. It would be interesting to determine if a similar family of protocols may be obtained that have more flexible delay assignments. That is, the contribution to the end-to-end packet delay from each hop in the path to the destination is not forced to be L/R. This is known as rate-independent delay. Examples of protocols with rate independent delay include [5,8].

References

[1] Cobb J., Gouda M., "Flow Theory", *IEEE/ACM Transactions on Communications,* October 1997.

[2] Cobb J., Gouda M., El-Nahas A., "Time-Shift Scheduling: Fair Scheduling of Flows in High-Speed Networks", *Proc. of the IEEE Int'l Conf. on Network Protocols,* 1996. Also, in *IEEE /ACM Transactions on Networking,* June 1998.

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

[4] Cruz R.L., ``A Calculus for Network Delay, Part I: Network Elements in Isolation", *IEEE Transactions on Information Theory*, Vol. 37, No. 1, January 1991, p. 114-131.

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

[6] 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.

[7] Figueira N., Pasquale J., "A Schedulability Condition for Deadline-Ordered Service Disciplines", *IEEE/ACM Transactions on Networking,* Vol. 5, No. 2, April 1997.

[8] Ferrari D., Verma D., "A Scheme for Real-Time Channel Establishment in Wide-Area Networks", *IEEE Journal of Selected Areas in Communication*, 8(4):368-379, April 1990.

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

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

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

[12] Gouda M., *The Elements of Network Protocols*, textbook in preparation.

[13] Keshav S., "A Control Theoretic Approach to Flow Control", *Proceedings of the 1991 ACM SIGCOMM Conference*.

[14] Parekh A. K. J., Gallager R., "A generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case", *IEEE/ACM Transactions on Networking*, 1(3):344-357, June 1993.

[15] Stiliadis D., Varma A., "Rate Proportional Servers: A Design Methodology for Fair Queueing Algorithms", University of California at Santa Cruz, Computer Science Department technical report number UCSC-CRL-95-58.

[16] Stiliadis D., Varma A., "Design and Analysis of Frame-Based Fair Queuing: A New Traffic Scheduling Algorithm for Packet Switched Network", *ACM SIGMETRICS*, 1996.

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

[18] Zhang H., "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks", *Proceedings of the IEEE*, Vol. 83, No. 10, Oct. 1995.

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

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