Flow Timestamps

Jorge A. Cobb Amal El Nahas Mohamed G. Gouda

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

Abstract

We consider the problem of a packet multiplexor that receives packets from multiple input flows, then forwards them to a single output channel. The multiplexor ensures a predefined forwarding rate for each flow. This is achieved as follows. The multiplexor assigns a timestamp to each received packet. When the channel becomes idle, the multiplexor forwards the packet with the smallest timestamp. In this paper, we show that the multiplexor efficiency can be improved without sacrificing packet delay by maintaining one timestamp per flow rather than one timestamp per packet. We show that this approach improves the computational efficiency of two known multiplexing methods, namely, virtual clock and self-clock.

1. Introduction

Virtual circuit computer networks, such as ATM [4], transfer a flow of data packets from a source computer to a destination computer via a virtual circuit. This virtual circuit is established along a network path between the source and destination computers.

Since multiple virtual circuits may share a single output channel of an intermediate computer, a multiplexing algorithm determines the forwarding order of packets along the channel. Some algorithms use packet timestamps to decide the forwarding order [1] [2]. In this paper, we suggest an alternative method that maintains one timestamp per flow, i.e., per virtual circuit, rather than one timestamp per packet. We show that this method improves the computational efficiency without sacrificing the upper bounds on packet delay.

2. Flow Multiplexing

A flow is an infinite sequence of packets that are received by a multiplexor.

The multiplexor receives packets from N distinct flows. A packet is denoted p.(i,k) if it is the kth packet the multiplexor receives from flow i. Note that i is of type 0 . . N-1, and k is a non-negative integer.

We adopt the following notation.

- L.(i,k) is the length in bytes of packet p.(i,k).
- L_{max} is an upper bound on all L.(i,k).
- A.(i,k) is the arrival time at the multiplexor of packet p.(i,k).
- E.(i,k) is the exit time from the multiplexor of packet p.(i,k).
- R.i is the rate in bytes/second associated with flow i.

The multiplexor forwards the received packets, one by one, into a single output channel. Thus, whenever the output channel becomes idle, the multiplexor chooses a packet from those it has received, and forwards the packet along the channel.

The goal of the multiplexor is to forward the packets of each flow i at an average rate of at least R.i bytes/second. Since the N flows share the output channel, the following constraint holds, where C is the capacity in bytes/second of the output channel.

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

3. Multiplexing using Packet Timestamps

A well-known method to multiplex the packets of multiple input flows into a single output channel is to use packet timestamps. When a packet is received, it is assigned an integer, known as the packet's timestamp, then the packet is stored in a queue. This packet queue is shared by all flows, and the packets it contains are ordered by increasing timestamp. When the output channel becomes idle, the packet with the smallest timestamp is removed from the head of the queue and sent over the channel.

Let F.(i,k) be the timestamp of packet p.(i,k). The timestamps of packets are computed such that the channel's bandwidth is shared among the flows according to their rates. In particular, the timestamp F.(i,k) of each p.(i,k) is computed according to the following equation.

$$F.(i,k) = \max(q.(i,k), F(i,k-1)) + L.(i,k)/R.i \quad (1)$$

In this equation, q.(i,k) is some quantity related to packet q.(i,k). Different methods differ in their choices of q.(i,k). We consider two specific choices for q.(i,k) in Sections 5 and 6 below.

In any case, the algorithm of the multiplexor is as follows.

variables
T.i : timestamp of last packet received from flow i;
R.i : rate of flow i;
queue : received packets;
i : 0 . . N-1;
k : non-negative integer

begin
rcv p.(i,k) →
T.i := max(q.(i,k), T.i) + L.(i,k)/R.i;
queue := **insert**(queue, p.(i,k), T.i)

▯

idle ∧ queue ≠ **empty** →
p.(i,k) := **head**(queue);
queue := **tail**(queue);
send p.(i,k)

end

This algorithm consists of two actions. In the first action, a packet is received from any flow i, its timestamp is computed, then the packet is inserted into the queue according to the computed timestamp. In the second action, when the output channel becomes idle, the packet with the smallest timestamp is removed from the queue and sent over the channel.

The following theorem shows the relation between the exit time E.(i,k) of packet p.(i,k) and the computed timestamp of p.(i,k).

Theorem 1
If q.(i,k) ≥ A.(i,k), then for every i and k,

$$E.(i,k) \leq F.(i,k) + L_{max}/C$$

Proof
Let t be the latest time, t ≤ A.(i,k), such that queue was empty, or the packet being sent had a timestamp larger than F.(i,k). Because q.(i,k) ≥ A.(i,k), t ≤ A.(i,k) < F.(i,k) holds. Also, any packet sent after t and before p.(i,k) is sent must have a timestamp of at most F.(i,k).

Consider any flow j. If at time t flow j has packets in the queue, then their timestamps are greater than F.(i,k), and no packet of j is sent until p.(i,k) is sent. If at time t flow j has no packets in the queue, then the first packet of j arriving after t, say p.(j,l), has a timestamp F.(j,l) ≥ q.(j,l) + L.(j,l)/R.j. Since q.(j,l) ≥ A.(j,l) ≥ t, F.(j,l) ≥ t + L.(j,l)/R.j.

Note that, for any l, if F.(j,l) = F.(j,l-1) + Δ, then L.(j,l) ≤ Δ∗R.j. Hence, after time t, the packets of flow j that have a timestamp of at most F.(i,k) add to at most (F.(i,k) - t)∗R.j bytes. Summing over all flows, the packets arriving at or after t whose timestamp is at most F.(i,k) are at most (F.(i,k) - t)∗C bytes. Hence, these packets are sent out in at most F.(i,k) - t seconds. Since there might be a packet being sent at time t with a timestamp greater than F.(i,k), packet p.(i,k) exits no later than

$$t + (F.(i,k) - t) + L_{max}/C$$

End of Theorem 1

4. Multiplexing using Flow Timestamps

From equation (1), it is easy to see that F.(i,k) > F.(i,k-1), which implies that packets from the same flow are sent in the order in which they were received. Thus, when a queued packet

p.(i,k) has a timestamp smaller than any other queued packet from flow i, it continues to have the smallest timestamp until it is sent, regardless of future packet arrivals from flow i.

This observation suggests an alternative scheme, in which only one timestamp per flow is maintained. The packet chosen for transmission is the earliest received packet from the flow with the smallest timestamp.

In this case, the algorithm of the multiplexor is as follows.

```
variables
    U.i     :  timestamp of flow i;
    R.i     :  rate of flow i;
    queue.i :  received packets from flow i;
    i       :  0 . . N-1;
    k       :  positive integer
begin
    rcv p.(i,k)     →
        if queue.i = empty      →
            U.i := max(q.(i,k), U.i) + L.(i,k)/R.i
        ▯ queue.i ≠ empty       →   skip
        fi
        queue.i := append(queue.i, p.(i,k))
▯
    idle ∧ queue ≠ empty        →
        i := min_flow(U);
        p.(i,k) := head(queue.i);
        queue.i := tail(queue.i);
        send p.(i,k);
        if queue.i ≠ empty      →
            p.(i,k) := head(queue.i);
            U.i := U.i + L.(i,k)/R.i
        ▯ queue.i = empty       →   skip
        fi
end
```

Each flow has a separate first-in-first-out queue of packets, and variable U.i stores the timestamp of flow i.

When packet p.(i,k) is received, if the queue of flow i is not empty, then U.i is not updated. If this queue is empty, then U.i is updated as follows.

$$U.i := \max(q.(i,k), U.i) + L.(i,k)/R.i$$

In the second action, min_flow(U) returns the flow number whose timestamp is smallest and its queue is non-empty. Thus, the packet to send is taken from the queue of this flow. If more packets of this flow remain, its timestamp is increased by L.(i,k)/R.i, where p.(i,k) is the next packet of the flow.

Flow timestamps have two advantages over packet timestamps.

First, the processing time to send a packet is smaller. This is because the number of queued packets is at least the number of flows with non-empty queues, and hence, finding the minimum flow timestamp is often faster than finding the minimum packet timestamp.

Second, the processing time to receive a packet is also smaller. This is because the flow timestamp is not updated when the flow's queue is not empty. On the other hand, the packet timestamp is always computed upon the packet's arrival, requiring the ordered queue to be updated each time.

To compare flow timestamps with packet timestamps, define G.(i,k) as the timestamp of flow i when packet p.(i,k) becomes the head of the queue of flow i. From the algorithm above, it follows that each G.(i,k) satisfies the following equations. If the queue of flow i is empty at time A.(i,k), then

$$G.(i,k) = \max(q(i,k), G.(i,k-1)) + L.(i,k)/R.i \quad (2)$$

If the queue of flow i is not empty at time A.(i,k), then

$$G.(i,k) = G.(i,k-1) + L.(i,k)/R.i \quad (3)$$

The flow timestamps defined by (2) and (3) are also related to the exit time of the packet as follows.

Theorem 2

If $q.(i,k) \geq A.(i,k)$, then for every i and k,

$$E.(i,k) \leq G.(i,k) + L_{max}/C$$

Proof

Using G.(i,k) instead of F.(i,k), the same argument for the proof of Theorem 1 holds for Theorem 2.

End of Theorem 2

Thus, flow timestamps also provide an upper bound on the exit time of each packet. The relationship between flow timestamps and

packet timestamps is given by the following theorem.

Theorem 3
For every i and k,

$$G.(i,k) \leq F.(i,k)$$

Proof Sketch
This is easily shown using induction on the definitions of G.(i,k) and F.(i,k).

End of Theorem 3

From Theorem 3, if a multiplexor uses flow timestamps instead of packet timestamps, its upper bound on packet delay does not increase. Thus, the multiplexor gains processing efficiency without sacrificing packet delay.

We next consider flow timestamps for virtual clock and self-clock multiplexing.

5. Virtual Clock Multiplexing using Flow Timestamps

Virtual clock multiplexing [7] assigns packet timestamps according to equation (1). Its choice of q.(i,k) is q.(i,k) = A.(i,k). Thus, timestamp F.(i,k) of packet p.(i,k) is as follows.

$$F.(i,k) = \max(A.(i,k), F.(i,k-1)) + L.(i,k)/R.i$$

From Theorem 1, the exit time is as follows.

$$E.(i,k) \leq F.(i,k) + L_{max}/C$$

Using flow timestamps, flow timestamp G.(i,k) is as follows. If the queue of flow i is empty at time A.(i,k), then

$$G.(i,k) = \max(A.(i,k), G.(i,k-1)) + L.(i,k)/R.i$$

Otherwise,

$$G.(i,k) = G.(i,k-1) + L.(i,k)/R.i$$

From Theorem 2, the exit time is as follows.

$$E.(i,k) \leq G.(i,k) + L_{max}/C$$

6. Self-Clock Multiplexing using Flow Timestamps

Self-clock multiplexing [3] also assigns packet timestamps according to equation (1). Its choice of q.(i,k) is the timestamp being sent at time A.(i,k). Thus, timestamp F.(i,k) of packet p.(i,k) is as follows,

$$F.(i,k) = \max(F.(j,l), F.(i,k-1)) + L.(i,k)/R.i$$

where packet p.(j,l) is being sent at time A.(i,k).

Using flow timestamps, the flow timestamp G.(i,k) is as follows. If the queue of flow i is empty at time A.(i,k), then,

$$G.(i,k) = \max(G.(j,l), G.(i,k-1)) + L.(i,k)/R.i$$

where packet p.(j,l) is being sent at time A.(i,k). Otherwise,

$$G.(i,k) = G.(i,k-1) + L.(i,k)/R.i$$

It is interesting to note that G.(i,k) = F.(i,k), because of the following. It is easy to show that the timestamps of packets sent by the multiplexor are always in increasing order. Thus, at time A.(i,k), if the queue of flow i is not empty, then q.(i,k) ≤ F.(i,k-1), which implies G.(i,k) = F.(i,k).

Before applying Theorem 1 or 2, we need to show that q.(i,k) ≥ A.(i,k). Due to space limitations, we leave this exercise for the reader. Thus, from Theorem 2, the exit time is as follows.

$$E.(i,k) \leq G.(i,k) + L_{max}/C$$

7. Concluding Remark

We also suspect that another multiplexing method, namely, weighted fair-queuing [2] [5], could benefit from flow timestamps. The packet timestamps in this method satisfy equation (1), and, although computing q.(i,k) is somewhat complex, it appears that q.(i,k) ≥ A.(i,k). If this is the case, Theorem 2 provides a delay bound for the case of flow timestamps.

References

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

[2] A. Demers, S. Keshav, S. Shenkar, ``Analysis and Simulation of a Fair Queueing Algorithm'', *Proceedings of the ACM SIGCOM,* 1989.

[3] S. J. Golestani, ``A Self-Clocking Fair-Queueing Scheme for Broadband Applications'', *Proceedings of the IEEE INFOCOM*, 1994.

[4] F. Halshal, *Data Communications, Computer Networks and Open Systmes,* 3rd Ed., Addison Wesley, 1992.

[5] A. K. J. Parekh, ``A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks'', MIT Technical Report No. LIDS-TH-2089.

[6] H. Zhang and Srinivasan Keshav, ``Comparison of Rate-Based Service Disciplines'', *Proceedings of the ACM SIGCOMM*, 1991.

[7] L. Zhang, ``Virtual Clock: A New Traffic Control Algorithm for Packet-Switched Networks'', *ACM Trans. on Computer Systems*, **9**, 2, May 1991.