Preserving Quality of Service Guarantees in Spite of Flow Aggregation

Jorge A. Cobb
Department of Computer Science
The University of Texas at Dallas
Richardson, TX 75083-0688

Abstract

We investigate the preservation of quality of service guarantees to a flow of packets in the presence of flow aggregation. In flow aggregation, multiple flows, known as the constituent flows, are merged together resulting in a single aggregate flow. Packet schedulers located after the network point where the aggregation occurred are aware of the aggregate flow, but are unaware of its constituent flows. In spite of this, we show that quality of service may still be guaranteed to the constituent flows if the aggregation is performed fairly. Furthermore, contrary to intuition, the quality of service guaranteed to a flow may be greater under flow aggregation than in the case where no aggregation is performed.

1 . Introduction

Consider a computer network that consists of a set of computers interconnected with point-to-point communication channels.

A *flow* in the computer network is a potentially infinite sequence of packets generated by the same source and having the same destination in the network.

Each output channel of a computer is equipped with a scheduler process, 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.

One type of schedulers, known as guaranteed-rate schedulers, guarantee that the packets of each flow will be forwarded at a designated rate. Examples of these scheduling protocols can be found in [16,17].

In all these protocols, the following steps are taken to reserve bandwidth for a new flow. First, the network finds a path from the source of the flow to the destination of the flow. Then, the network reserves for the flow a fraction of the bandwidth of each output channel along the path. The new flow is admitted into the network only if there is enough available bandwidth along the entire path.

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 [5].

In this paper, we investigate the effects of aggregating multiple flows, known as the constituent flows, into a single aggregate flow. Once the aggregation is done, the remaining schedulers along the path will have no knowledge of the constituent flows of the aggregate flow, and will treat the aggregate flow as a single flow whose rate is the sum of the reserved rates of the constituent flows.

One practical implementation of flow aggregation is virtual paths in virtual circuit networks [12]. Multiple virtual circuits may be combined into a single virtual path. Schedulers along the virtual path are aware only of the virtual path, and are unaware of the virtual circuits that constitute the virtual path. Thus, a virtual circuit may be viewed as a constituent flow, and a virtual path may be viewed as an aggregate flow.

The purpose of flow aggregation is to improve the efficiency of the schedulers and to simplify the management of flows. For example, buffer management is simplified, because only one queue per aggregate flow is required, rather than one queue per constituent flow. Furthermore, rerouting an aggregate flow in the event of a failed channel along its path is much more efficient than rerouting each of the constituent flows individually.

In this paper, we examine if it is possible to provide quality of service guarantees to the constituent flows. In particular, we consider end-to-end delay guarantees. We show that if the aggregation of flows is performed fairy, then an upper bound on end-to-end delay is guaranteed for the constituent flows, even though schedulers are unaware of these flows. Furthermore, we show that, contrary to intuition, the upper bound on end-to-end delay for the constituent flows may be *lower* under flow aggregation than in the case where no aggregation is performed.

The paper is organized as follows. In Section 2, we review how packet timestamps are computed to provide bounded packet delays at a scheduler. In Section 3, we review the concept of rate-proportional schedulers, which will be the foundation for computing end-to-end delays. In Section 4, we define flow aggregation and present examples. In Section 5, we identify the property that the aggregation must satisfy in order to preserve a low end-to-end delay. In Section 6, we show how to implement

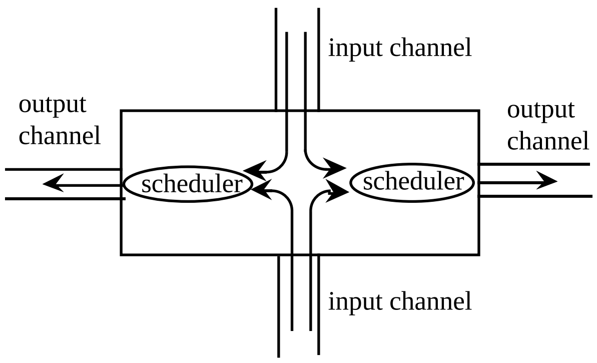

Figure 1: A computer with input channels and output channels.

schedulers that satisfy this property. Future work is given in Section 7.

A note on notation: throughout the paper we use quantifications of the form

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

Above, $\oplus$ is a commutative and associative operator, such as +, -, max, min, $\forall$ (conjunction), or $\exists$ (disjunction). R(x) is a boolean 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 R(x) is omitted, all values in the type of x are included.

2. Timestamp scheduling

We next review the use of timestamps in guaranteed rate schedulers.

To guarantee that the packets of a flow are forwarded at a rate of at least the rate reserved for the flow, the scheduler assigns a timestamp to each received packet. The timestamp is a function, among other things, of the flow's reserved rate. Then, the scheduler forwards the packets in order of increasing timestamp. Below, we examine in more detail how this timestamp is computed.

We adopt the following notation for a scheduler.

• N	number of input flows of the scheduler.
• R.f	forwarding rate (in bits/sec.) reserved for flow f.
• p.f.i	ith packet received from flow f, $i \geq 0$.
• A.f.i	arrival time into the scheduler of packet p.f.i.
• T.f.i	Timestamp assigned to packet p.f.i.
• L.f.i	packet length (in bits) of packet p.f.i.
• E.f.i	exit time of packet p.f.i, i.e., when the output channel finishes forwarding packet p.f.i.
• $L.f_{max}$	upper bound on packet length for flow f.
• L_{max}	upper bound on packet length for all flows.
• C	capacity in bits/sec. of the output channel of the scheduler.

```
process  scheduler
inputs
idle        :  is the output channel idle?
R.f         :  rate of flow f
variables
f           :  flow, 0 . . N-1
i           :  packet index (integer)
queue.f     :  packet queue of flow f
L.f.i       :  length of packet p.f.i
T.f.i       :  timestamp of packet p.f.i
V           :  virtual time function
begin
  receive p.f.i from any f          →
          update(V);
          T.f.i := max(V, T.f.(i-1)) + L.f.i/R.f;
          append(p.f.i, queue.f)

▯ idle ∧ (∃ f : : queue.f ≠ empty)   →
          f := least(queue);
          p.f.i := head(queue.f);
          forward p.f.i;
          queue.f := tail(queue.f);
end
```

Figure 2: Timestamp Scheduler

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

$$(+\ f : 0 \leq f < N : R.f) \ \leq\ C$$

Each packet timestamp T.f.i is computed as follows.

$$T.f.i := \max(V, T.f.(i-1)) + L.f.i/R.f$$

In this assignment, T.f.-1 is defined to be zero, and V is a function, known as the virtual time function, whose value monotonically increases with time.

The protocols of Virtual Clock [14,15], Weighted Fair Queuing [10,11], Self-Clocking Fair Queuing [7], and Time-Shift Scheduling [3], among others ([1,13]) all use the above formula to compute packet timestamps. The difference between these protocols is the value chosen for V. For example, in Virtual Clock scheduling [15], V is the arrival time of packet p.f.i, while in Self-Clocking Fair Queuing [7], V is the timestamp of the packet currently in the output channel.

Therefore, a scheduler based on assigning timestamps to packets may be defined as in Figure 2 (using the notation presented in [8] and [9].)

The inputs to the scheduler are the reserved rate of each flow, and a boolean flag indicating if the output channel is idle at this moment.

The scheduler consists of two actions. In the first action, the scheduler receives a packet from a flow. The scheduler first updates the value of V to make sure it is current before the timestamp is computed. Then, the timestamp of the packet is computed, and the packet is appended to the queue of its flow. Notice that T.f.i > T.f.(i-1), and thus, maintaining the queue of flow f in first-in-first-out order also maintains the queue of f in increasing timestamp order.

In the second action, the scheduler detects that the output channel is currently idle, and that packets remain to be forwarded. Then, function **least**(queue) examines the packet at the head of the queue of each flow, and returns the flow whose packet timestamp is the smallest. Then, the scheduler removes the packet at the head of the queue of this flow and forwards the packet to the output channel.

3. Rate proportional schedulers

We define the *rate-proportional deadline*, D.f.i, of the ith packet received from flow f as follows.

a) D.f.0 = A.f.0 + L.f.0/R.f
b) D.f.i = max(A.f.i, D.f.(i-1)) + L.f.i/R.f, where i > 0.

In essence, the rate proportional deadline of a packet is the time at which the packet would exit a constant rate server whose rate is R.f bits/sec. and whose sole input is f.[1]

We say that a scheduler is a *rate proportional scheduler* [2,6] with scheduling constant α, $\alpha > 0$, iff, for all f and i,

$$E.f.i \le D.f.i + \alpha$$

That is, each packet will exit the scheduler at a time at most the rate-proportional deadline of the packet plus α.

Many timestamp protocols, such as Virtual Clock, Self-Clocking Fair Queuing, Weighted Fair Queuing, Time-Shift Scheduling, among others, have been shown in the literature to be rate-proportional schedulers [3, 11, 14, 15, 1, 13]. Thus, all these schedulers guarantee a deadline to each packet that is independent of the arrival patterns of packets from other flows, and depends solely on the arrival patterns of packets from its own flow.

A flow will traverse a path of multiple schedulers before it reaches its destination. Some values associated with a packet vary from one scheduler to another along this path, and thus we make explicit the scheduler in question by the following change in the notation.

• A.s.f.i	arrival time of packet p.f.i at scheduler s
• T.s.f.i	timestamp of packet p.f.i at scheduler s
• D.s.f.i	rate-proportional deadline of packet p.f.i at scheduler s
• E.s.f.i	exit time of packet p.f.i from scheduler s
• α.s	scheduling constant of scheduler s.

The following theorem has been shown independently and using different proof techniques in [2] and [6].

[1]Notice that the rate-proportional deadline of a packet is the same as the timestamp of the packet under Virtual Clock scheduling. Thus, Virtual Clock scheduling forwards packets in order of increasing rate proportional deadlines.

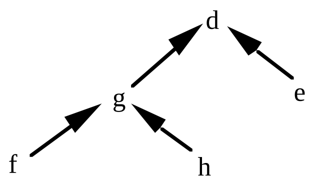

Figure 3: Aggregation of flows into flow aggregates.

Theorem 1
Let the path of flow f have k rate-proportional schedulers, $t_1, t_2, \ldots, t_k$, $k > 1$. Then,

$$E.t_k.f.i \le D.t_1.f.i + (k-1)\cdot L.f_{max}/R.f + (+\ j : 1 \le j \le k : \alpha.t_j)$$

$$D.t_k.f.i \le D.t_1.f.i + (k-1)\cdot L.f_{max}/R.f + (+\ j : 1 \le j < k : \alpha.t_j)$$

♦

Therefore, the end-to-end delay of the packets of a flow is composed of two parts. The delay encountered due to the burstiness of the flow, which is represented by the $D.t_1.f.i$ term above, and an extra delay of $L.f_{max}/R.f + \alpha$ for each hop in the path of the flow. The latter delay is non-trivial, and may be significant for flows with long paths to their destinations.

4. Flow aggregation

We next introduce the new concepts of simple and aggregate flows.

A *simple flow* is a potentially infinite sequence of packets generated by the same source and having the same destination in the network. That is, the flows considered in earlier sections were simple flows.

An *aggregate flow* f of level k is a sequence of packets with the following additional properties:

a) If k = 0, f is a simple flow.
b) If k > 0, then f is the merging of the packets from at least two aggregate flows of level less than k, of which at least one is of level k - 1.

If the packets of aggregate flow f, where $f \ne g$, are also packets of aggregate flow g, then f is a *constituent* of g. Aggregate flow f is an *immediate constituent* of aggregate flow g if f is a constituent of g and the level of g is one greater than the level of f. We say that flow g is the *root* of flow f if f is a constituent of g, and g is not a constituent of any other flow.

For example, consider Figure 3, where f, h, and e are simple flows, i.e., aggregate flows of level 0. Assume f and h are aggregated together to form aggregate flow g, and also g and e are aggregated together to form aggregate flow d. Then, the immediate constituents of g are f and h, and the immediate constituents of d are g and e. The level of g is one, and the level of d is two. Furthermore, d is the root of g, e, f, and h.

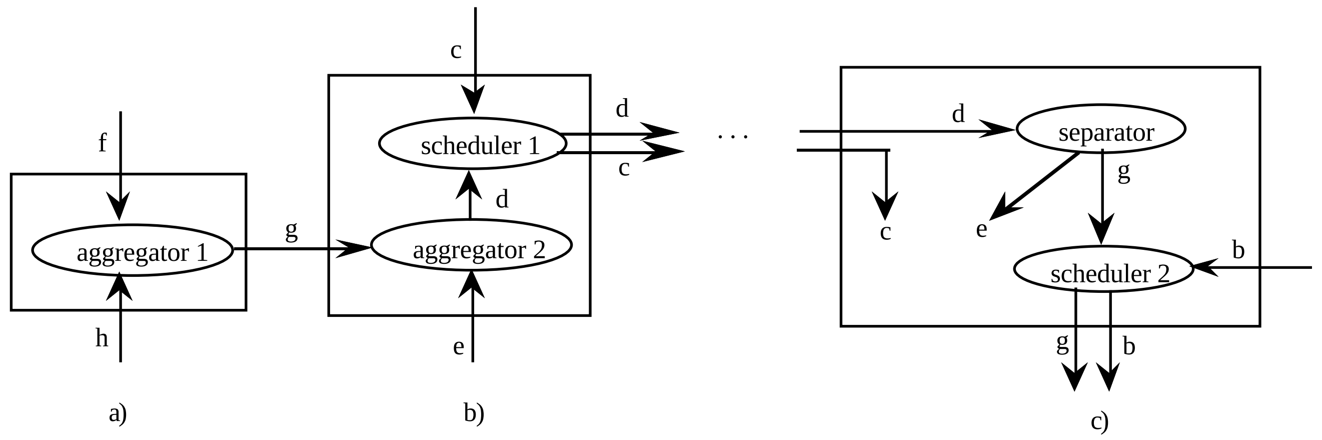

Figure 4: Example of aggregate flows

The *reserved rate* R.g of aggregate flow g is the sum of the reserved rates of the immediate constituent flows of g. Thus, inductively, the reserved rate of aggregate flow g is the sum of the rates of all the simple flows which are constituents of g. In the example, R.d = R.g + R.e, and R.g = R.f + R.h.

At a scheduler, the rate proportional delay of each packet of aggregate flow g is calculated in the same manner as if g were a simple flow with reserved rate R.g.

As mentioned in the introduction, the objective of this paper is to investigate the possibility of aggregating multiple flows into a single flow, and determine if the quality of service provided to the constituent flows is maintained by the scheduler, even though the scheduler is only aware of the aggregate flow and not the individual constituent flows.

Thus, the input to a scheduler is now a set of aggregate flows. For each aggregate flow g, the scheduler is unaware of the constituent flows contained by g. It simply schedules the packets of g as if g were a simple flow with reserved rate R.g.

A scheduler which receives as inputs a set of aggregate flows and produces as output a single aggregate flow whose immediate constituents are the input aggregate flows is called an *aggregator.*

A *separator* is a process that receives as input an aggregate flow, and produces as output the immediate constituents of the input flow. We assume a separator introduces no additional packet delay. Separators are the only processes which are aware of the immediate constituents of an aggregate flow.

Consider for example Figure 4. In Figure 4(a), the first computer receives two input flows, f and h. These flows are given as input to an aggregator, which produces aggregate flow g. Flow g is then the input to the computer in Figure 4(b), and is aggregated with flow e to produce flow d. Flow d and another input flow, c, are given as input to a scheduler, which forwards these two flows (without aggregating them) to the output channel.

Then, flows c and d traverse multiple computers until they arrive to the computer in Figure 4(c). Here, flow d is separated into its constituent flows, e and g. The destination of flows c and e is this computer, so they are not forwarded further. However, flow g must be forwarded further to its destination, so it is given as input to a scheduler. The scheduler forwards flow g and another input flow b to the output channel, and so on.

Later on in the path of g, before the destinations of flows f and h are reached, flow g will be separated into its constituent flows f and h.

Several points are worthy of being stressed. First, the root of a flow may change as it traverses the network. For example, flow g is the root of flow f when g is between the computers in Figures 4(a) and 4(b). However, when g is aggregated into flow d in Figure 4(b), the root of f becomes d. Then, after d is separated into e and g in Figure 4(c), g is once again the root of f.

Note also that the level of a flow does not change as it traverses the network. E.g., the level of flow g is 1 and the level of f is 0 throughout the network. In addition, a flow is always separated only into its immediate constituents. For example, flow f cannot be separated from flow d, since flow f is not an immediate constituent of flow d.

Finally, in some cases, flows will be aggregated and then directly forwarded to the output channel (Figure 4(a)), while in other cases, flows will be aggregated and then given as input to another scheduler before being forwarded to the output channel (Figure 4(b)). In the latter case, both the aggregator and the next scheduler are in the same computer. The aggregator may be viewed as having an output channel of infinite capacity, that makes the packet forwarded by the aggregator immediately available to the next scheduler.

5. Fair aggregators

We now address the type of behavior that is needed from an aggregator in order to retain quality of service guaran-

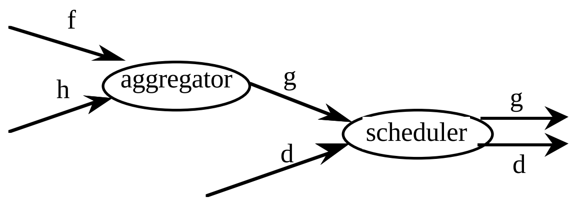

Figure 5: Fair Aggregators

tees to the constituent flows even though the succeeding schedulers are unaware of them.

Consider Figure 5. The scheduler is not aware that g consists of two flows, f and h. Thus, the scheduler only guarantees that it will forward the packets of g at a rate of R.g, where R.g = R.f + R.h.

Assume that packets from h arrive at a rate greater than R.g, and no packets are received from flow f. If the aggregator simply forwards these packets to the scheduler as soon as they arrive, it is possible that they will cause a significant increase in the queue of g at the scheduler. Then, if packets from f are received and forwarded to the scheduler, they will be placed at the end of the queue of g. Thus, the next packet forwarded from flow f will not exit the scheduler until the queue of g empties.

The above is undesirable, since it violates the quality of service guarantees of flow f. That is, if the scheduler were aware of flows f and h, the next packet of f would experience a delay of at most $L.f_{max}/R.f + \alpha$. Therefore, the aggregator must not forward packets from h at a rate greater than R.g when no packets from f are present.

Assume now that both input flows are generating packets at a rate greater than their reserved rate. Assume the aggregator forwards packets from f at a rate of exactly R.f, but it forwards the packets from h at a rate greater than R.h. Notice that this still satisfies the definition of a rate-proportional scheduler, since the packets of f will exit the aggregator by their rate-proportional deadline.

Also in this case, it is possible that the queue of g will grow significantly, since packets of g arrive at a rate greater than R.g = R.f + R.h. Thus, the packets of f will be delayed excessively at the scheduler, because in the queue of g there is an excessive number of packets of h in between any two packets of f.

To prevent the above, the aggregator should forward packets from f and h in a fair manner. That is, the ratio of bits forwarded from h vs. the number of bits forwarded from f should be kept as close as possible to R.h/R.f.

The above desired behavior for an aggregator is captured succinctly by the following definition.

Definition 1
Let s be an aggregator, f be an aggregate input flow of s, and g be the aggregate output flow of s. Let g be an input flow of scheduler t. Furthermore, let packet p.f.i = p.g.j, i.e., the ith packet of f is the jth packet of g. We say that s is a *fair aggregator*, iff s is a rate-proportional scheduler, and

$$D.t.g.j \le D.t.f.i + \beta.s$$

for some aggregating constant β.s.

♦

Thus, the rate-proportional deadline of packet of an immediate constituent flow increases by at most β.s when the flow of the packet is aggregated with other flows. Furthermore, since an aggregator is a rate-proportional scheduler, packet p.f.i exits aggregator s no later than time $D.s.f.i + \alpha.s$.

We will see next how this ensures an upper bound on end-to-end delay similar to the upper bound when flows aren't aggregated. We first consider the case when a simple flow goes only through a single aggregator, i.e., the level of the root flow is at most one. We then consider the more general case when the root may have any level.

Theorem 2
Let f be an input flow of a fair aggregator s with scheduling constant α.s and aggregating constant β.s. Let g be the output flow of the aggregator, and let g pass through k rate-proportional schedulers, $t_1, t_2, \ldots, t_k$. Then,

$$E.t_k.f.i \le D.s.f.i + L.f_{max}/R.f + (k-1)\cdot L.g_{max}/R.g + \beta.s + \alpha.s + (+\ j : 1 \le j \le k : \alpha.t_j)$$

Proof:

Consider a packet p.f.i of f. Assume that p.f.i = p.g.j, i.e., the jth packet of g is the ith packet of f. From the definition of a fair aggregator

$$D.t_1.g.j \le D.t_1.f.i + \beta.s$$

Since the aggregator is a rate-proportional scheduler, and from Theorem 1,

$$D.t_1.g.j \le D.t_1.f.i + \beta.s \le D.s.f.i + L.f_{max}/R.f + \alpha.s + \beta.s$$

From Theorem 1

$$E.t_k.g.j \le D.t_1.g.j + (k-1)\cdot L.g_{max}/R.g + (+\ j : 1 \le j \le k : \alpha.j)$$

Combining the above two relations,

$$E.t_k.g.j \le D.s.f.i + L.f_{max}/R.f + (k-1)\cdot L.g_{max}/R.g + \beta.s + \alpha.s + (+\ j : 1 \le j \le k : \alpha.j)$$

♦

Theorem 3
Let f be an input to a fair aggregator s with scheduling constant α.s and aggregating constant β.s. Let g be the output of the aggregator, and p.f.i = p.g.j. Let g pass through multiple schedulers, the first one being t, and the last one being u. Also, assume

$$E.u.g.j \le D.t.g.j + \varphi$$

If after u, flow f or flow g is given as input to scheduler v, then

$$D.v.f.i \le D.s.f.i + 2 \cdot L.f_{max}/R.f + \alpha.s + \beta.s + \varphi$$

Proof

From Theorem 1 and s is a rate-proportional scheduler,

$$D.t.f.i \le D.s.f.i + L.f_{max}/R.f + \alpha.s$$

From definition of an aggregator,

$$D.t.g.j \le D.t.f.i + \beta.s$$

Since p.f.i = p.g.j, from the assumption in the theorem, and the above two relations

$$\begin{aligned} E.u.f.i = E.u.g.j &\le D.t.g.j + \varphi \\ &\le D.t.f.i + \beta.s + \varphi \\ &\le D.s.f.i + L.f_{max}/R.f + \alpha.s + \beta.s + \varphi \end{aligned}$$

Thus, the schedulers from s up to u, from the point of flow f, may be viewed as a single rate-proportional scheduler whose scheduling constant equals

$$L.f_{max}/R.f + \alpha.s + \beta.s + \varphi$$

From Theorem 1, we obtain the desired result.

$$D.v.f.i \le D.s.f.i + 2 \cdot L.f_{max}/R.f + \alpha.s + \beta.s + \varphi$$

♦

Corollary 1
Let f be an input to a fair aggregator s with scheduling constant α.s and aggregating constant β.s. Let g be the output of the aggregator, and let g pass through k rate-proportional schedulers, $t_1, t_2, \ldots, t_k$. After t_k, f is separated from g and is given as input to scheduler v. Then,

$$D.v.f.i \le D.s.f.i + 2 \cdot L.f_{max}/R.f + (k-1) \cdot L.g_{max}/R.g + \beta.s + \alpha.s + (+\ j : 1 \le j \le k : \alpha.j)$$

Proof

Let p.g.j = p.f.i. From Theorem 1,

$$E.t_k.g.j \le D.t_1.g.j + (k-1) \cdot L.g_{max}/R.g + (+\ j : 1 \le j \le k : \alpha.j)$$

From Theorem 3 (replacing t by t_1 and u by t_k).

$$D.v.f.i \le D.s.f.i + 2 \cdot L.f_{max}/R.f + (k-1) \cdot L.g_{max}/R.g + \beta.s + \alpha.s + (+\ k : 1 \le j \le k : \alpha.j)$$

♦

The crucial point of the above theorems is that quality of service may be provided to a flow even though it is aggregated with other flows. In particular, when flow f is aggregated into flow g, the rate-proportional deadline of a packet p.f.i increases by $L.g_{max}/R.g + \alpha.j$ along the jth hop in the path. On the other hand, if it is not aggregated, its delay increases by $L.f_{max}/R.f + \alpha.j$.

Note that R.g > R.f, and assume $L.f_{max} \approx L.g_{max}$. Then, *the delay using flow aggregation may actually be smaller than the delay when flow aggregation is not used.* Thus, this yields the counter-intuitive result that aggregating flows may reduce the packet delay incurred due to long network paths, even though the scheduler is unaware that an input flow is an aggregation of multiple flows.

Notice, however, that the delay increases by β.s, due to the fair aggregator. However, if β.s is not very large, and the number of schedulers after the aggregator is significant, the overall effect may be a reduction in packet delay when compared to the case of no aggregation.

We next consider the general case when the root of a flow may have any level.

Theorem 4
Let f traverse k schedulers, $t_1, t_2, \ldots, t_k$, k > 2, any of which may be an aggregator. Then, for any i, 0 ≤ i,

$$\begin{aligned} E.t_k.f.i \le D.t_1.f.i &+ (+\ x : 1 \le x < k : L.r_{xmax}/R.r_x) + \\ &(+\ x : 1 \le x \le k : \alpha.t_x) + \\ &(+\ x : 1 \le x < k \wedge t_x \text{ is an aggregator} : \beta.t_x) \end{aligned}$$

$$\begin{aligned} D.t_k.f.i \le D.t_1.f.i &+ (+\ x : 1 \le x < k\text{-}1 : L.r_{xmax}/R.r_x) \\ &+ L.f_{max}/R.f + (+\ x : 1 \le x < k : \alpha.t_x) + \\ &(+\ x : 1 \le x < k \wedge t_x \text{ is an aggregator} : \beta.t_x) \end{aligned}$$

where

a) r_x is the root flow of flow f when f is the input to t_x, provided f does not go through a separator between t_x and t_{x+1},

b) r_x is the root flow of f when f is the input to t_{x+1}, provided f goes through at least one separator between t_x and t_{x+1}.

♦

The proof of Theorem 4 is deferred to the full paper due to space restrictions.

Theorem 4 allows us to calculate the exit time and rate-proportional delay of each packet of a flow at each of the schedulers along its path. Note that the rate-proportional delay of a packet of flow f increases by $L.r_{max}/R.r$ at each hop, where r is the root of flow f at the scheduler. Thus, the network can provide quality of service guarantees to flow f. Furthermore, since R.r > R.f, the network may be able to guarantee an end-to-end delay to flow f using flow aggregation that is significantly lower than the end-to-end delay guaranteed to flow f without flow aggregation.

We next present an example of the application of Theorem 4. Consider Figure 6. In this example, flow f is aggregated into flow g at aggregator t_2, and flow g is aggregated into flow h at aggregator t_4.

We next calculate $E.t_4.f.i$, $D.t_4.f.i$, $E.t_6.f.i$, $D.t_6.f.i$. First notice that since there is a separator after t_5, then r_5 equals g, and not h. Thus, from Theorem 4,

$$E.t_4.f.i \le D.t_1.f.i + 2 \cdot L.f_{max}/R.f + L.g_{max}/R.g + \alpha.t_1 + \alpha.t_2 + \alpha.t_3 + \alpha.t_4 + \beta.t_2$$

$$D.t_4.f.i \le D.t_1.f.i + 3 \cdot L.f_{max}/R.f + \alpha.t_1 + \alpha.t_2 + \alpha.t_3 + \beta.t_2$$

$$E.t_6.f.i \le D.t_1.f.i + 2 \cdot L.f_{max}/R.f + 3 \cdot L.g_{max}/R.g + \alpha.t_1 + \alpha.t_2 + \alpha.t_3 + \alpha.t_4 + \alpha.t_5 + \alpha.t_6 + \beta.t_2 + \beta.t_4$$

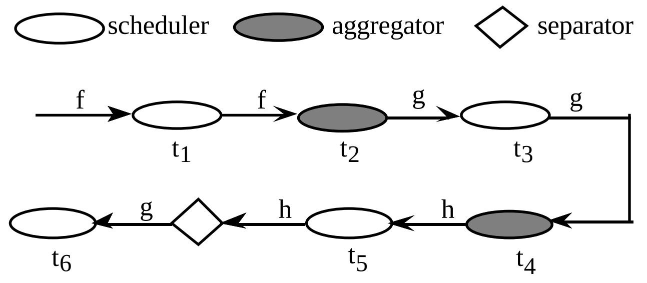

Figure 6: Calculating exit times and delays.

$$D.t_6.f.i \leq D.t_1.f.i + 3 \cdot L.f_{max}/R.f + 2 \cdot L.g_{max}/R.g + \alpha.t_1 + \alpha.t_2 + \alpha.t_3 + \alpha.t_4 + \alpha.t_5 + \beta.t_2 + \beta.t_4$$

6. Basic fair aggregators

In this section, we provide a simple and straightforward technique to construct fair aggregators.

In the previous section, we mentioned that if flows f and h are aggregated into a flow g, then the aggregator should not forward packets from flow f faster than R.g if there are no packets available from flow h. In addition, if packets from both flows are available and the aggregator forwards packets at a rate faster than R.g, then the packets of both f and h should be forwarded in relative proportion to their rates.

Let g be the output flow of the aggregator, and the channel capacity of the aggregator be C. A simple technique to construct an aggregator that satisfies both of the above requirements is the following. Consider a fictitious rate proportional scheduler whose output channel has capacity R.g and has the same input flows as the aggregator. The aggregator assigns the same timestamp to each packet equal to the timestamp of the same packet in the fictitious scheduler. After the aggregator forwards a packet of length L, the aggregator does not forward another packet until L/R.g seconds later, even though the packet takes only L/C seconds to transmit.

We call a fair aggregator constructed from the above technique a *basic fair aggregator.* Note that the above actually defines a whole family of basic fair aggregators, one family member for each possible type of rate-proportional scheduler emulated to assign timestamps to packets. E.g., we could define a Virtual Clock basic fair aggregator, a Weighted Fair Queuing basic fair aggregator, etc..

Note that the first of the two requirements is met since the packets are forwarded at a rate of R.g. In addition, since the aggregator does not forward packets at a rate greater than R.g, the second requirement is irrelevant. Finally, since the aggregator timestamps packets in the same way that a rate-proportional scheduler with output channel capacity of R.g, then the aggregator is also a rate-proportional scheduler with the same scheduling constant.

```
idle ∧ (∃ f : : queue.f ≠ **empty**) ∧ next ≤ clock   →
        f := **least**(queue);
        p.f.i := **head**(queue.f);
        **forward** p.f.i;
        next := clock + L.f.i/R.g;
        queue.f := **tail**(queue.f);
```
Figure 7: Basic fair aggregator

The above requires only a simple change to the code of Figure 2. One new variable, next, keeps track of the time when the next packet may be forwarded, and input R.g specifies the rate of the output flow. All other variables and inputs remain the same. Only the second action needs to be modified, and it is shown in Figure 7. The guard of the action is strengthened so that a packet is not forwarded until next ≤ clock. If a packet p.f.i is forwarded, the time when the next packet may be forwarded is calculated as next := clock + L.f.i/R.g.

We next prove that a basic fair aggregator satisfies the fair aggregator property. We begin with a lemma whose proof is deferred to the full paper due to space restrictions.

Lemma 1

Let g be the output flow of a basic fair aggregator, and t be the scheduler after the aggregator. Let p.g.i, . . . , p.g.k, i < k, be a sequence of packets of flow g such that D.t.g.i = A.t.g.i + L.g.i, and for each j, i < j ≤ k,

$$D.t.g.j = D.t.g.(j-1) + L.g.j/R.g$$

That is, A.t.g.j ≤ D.t.g.(j-1). Let p.g.m, i ≤ m ≤ k, be the largest packet in the sequence. Then,

$$D.t.g.(k-1) - A.t.g.k = L.g.m/C - L.g.k/C \leq L_{max}/C$$

♦

Theorem 5

Let v be a rate-proportional scheduler with scheduling constant α.v and output channel rate R.g. Let s be a basic fair aggregator with output flow g and output channel capacity C (R.g ≤ C), and s assigns timestamps to packets equal to the timestamps they would be assigned by v. Then,

a) s is a rate-proportional scheduler
b) α.s = α.v
c) $\beta.s = L_{max}/C$

Proof

Because s forwards each the next packet after L.p/R.g seconds elapse from forwarding the previous packet p, and the output channel rate of v is R.g, then each packet p is forwarded by s at exactly the same time it is forwarded by v. Furthermore, since R.g ≤ C, p will exit the output channel of s no later than the exit time of p in v, and thus s is also a rate-proportional scheduler, and also α.s = α.v.

Let t be the scheduler after s, f be an input flow of s, and p.f.i = p.g.k.

If k = 0, then also i = 0, and

$$D.t.g.0 \quad = \quad A.t.g.0 + L.g.0/R.g$$

$$
\begin{aligned}
&= A.t.f.0 + L.f.0/R.g \\
&< A.t.f.0 + L.f.0/R.f \\
&= D.t.f.0
\end{aligned}
$$

If k > 0, then from Lemma 1,

$$D.t.g.(k\text{-}1) - A.t.g.k \le L_{max}/C$$

Since D.t.g.k = max(D.t.g.(k-1), A.t.g.k) + L.g.k/R.g and the above,

$$D.t.g.k \le A.t.g.k + L.g.k/R.g + L_{max}/C$$

Since R.g > R.f, and p.g.k = p.f.i,

$$D.t.g.k < A.t.f.i + L.f.i/R.f + L_{max}/C$$

Note that for any packet p.f.i, D.t.f.i ≥ A.t.f.i + L.f.i/R.f. Thus,

$$D.t.g.k < D.t.f.i + L_{max}/C$$

Thus, $\beta.s = L_{max}/C$.

♦

Note that if the aggregator is internal (as in the case of aggregator 2 in Figure 4) then $C = \infty$, and $\beta.s = 0$.

Consider a basic fair aggregator with two input flows f and h, and one output flow g. The maximum rate at which it can forward packets is R.f + R.h = R.g. However, the definition of a fair aggregator may be satisfied by forwarding packets at a rate higher than R.g, provided packets are available from all input flows, and these packets are merged fairly. In the full version of the paper, we will present and prove correct more sophisticated fair aggregators that may forward packets at a rate higher than the reserved rate of their output flow.

7. Concluding remarks.

The advantages of flow aggregation are simplified scheduling and management of flows, and in some cases, a reduction in the end-to-end delay. The disadvantage is that flow aggregation has to be performed by a non-work conserving scheduler. In the full version of the paper, we will define more sophisticated flow aggregators which, although still non-work conserving in some instances, are more able to make use of unused bandwidth in the output channel than the basic fair aggregator we presented here.

Some scheduling protocols (e.g., [4,18] among others) guarantee to each flow f a per-hop delay of δ.f, chosen by the source of f, rather than the usual per-hop delay of $L.f_{max}/R.f$. This is accomplished by enhancing the algorithm to compute packet timestamps to take into consideration the δ.f per-hop delay, and accepting a new flow only if it passes a schedulability condition. In the full paper, we will show how a more flexible delay of this type can be guaranteed to a flow even in the presence of flow aggregation.

References

[1] Bennet J.C.R., H. Zhang, "Hierarchical Packet Fair Queuing Algorithms", in *Proceedings of the ACM SIGCOMM 1996.*

[2] Cobb J., Gouda M., "Flow Theory", *IEEE/ACM Transactions on Communications,* Jan 1998.

[3] Cobb J., Gouda M., El-Nahas A., "Time-Shift Scheduling: Fair Scheduling of Flows in High-Speed Networks", *IEEE/ACM Transactions on Communications,* June 1998.

[4] 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.

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

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

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

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

[9] Gouda M., *The Elements of Network Protocols*, Wiley Publishers, 1998.

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

[11] 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.

[12] Tananbaum A., *Computer Networks,* third edition, Prentice Hall, 1996.

[13] Suri S., Varghese G., "Leap-Forward Virtual Clock: A New Fair Queuing Scheme with Guaranteed Delays and Throughput Fairness", *Proceedings of the INFOCOM 1997 conference.*

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

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

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

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

[18] Zheng Q., Shin K.G., "On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks", *IEEE Transactions on Communications,* Vol 42, No. 2/3/4, Feb./March/April 1994.