

# Preserving Quality of Service Guarantees In Spite of Flow Aggregation

Jorge Arturo Cobb

**Abstract**— We investigate how quality of service may be guaranteed to a flow of packets in the presence of flow aggregation. For efficiency, multiple flows, known as the constituent flows, are merged together resulting in a single aggregate flow. After the network node where the aggregation occurs, packet schedulers are aware of the aggregate flow, but are unaware of its constituent flows. In spite of this, we show that quality of service may be guaranteed to the constituent flows, provided the aggregation is performed fairly. When the delay bound of a flow is de-coupled from the flow's reserved rate, flow aggregation preserves the delay bound. When the delay bound of a flow is coupled to the flow's reserved rate, flow aggregation preserves, and in some cases improves, the delay bound.

**Keywords**— Quality of Service, Real-Time Scheduling, Flow Aggregation.

## 1 Introduction

Consider the problem of transferring packets from a real-time application across a computer network. This problem has been studied extensively, resulting in the development of guaranteed-rate schedulers. That is, scheduling protocols which guarantee a minimum bandwidth and a maximum packet delay to each application. Examples of these protocols may be found in [29][30].

In guaranteed-rate schedulers, resources, such as bandwidth and buffer space, are reserved for each application along the entire path to its destination. If not enough resources are available, the application's request for a network connection is denied. Due to the reservation of resources, the network can provide service guarantees to each application, provided the rate of the application does not exceed the agreed-upon rate. These service guarantees are necessary for real-time applications, such as interactive audio and video [14].

Let us denote by a flow the sequence of packets generated by a single application. 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 have no knowledge of the constituent flows of the aggregate flow. They simply treat the aggregate flow as a single flow whose rate is the sum of the reserved rates of the constituent flows.

One of many possible implementations of flow aggregation is a virtual path in virtual circuit networks [26]. Mul-

iple virtual circuits may be combined into a single virtual path. Schedulers are aware of the virtual path, but are unaware of the virtual circuits constituting 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. Scheduling efficiency is improved, because the number of flows is reduced. Finally, rerouting an existing aggregate flow in the event of a failed channel is much more efficient than rerouting each of the individual constituent flows.

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 fairly, then an upper bound on end-to-end delay is guaranteed to the constituent flows, even though schedulers are unaware of these flows. If the per-hop delay is de-coupled from the reserved rate, we show that the end-to-end delay with flow aggregation is similar to the end-to-end delay without flow aggregation. In addition, if the per-hop delay is coupled with the reserved rate, we show that the end-to-end delay with flow aggregation may be lower than the end-to-end delay without flow aggregation.

The paper is organized as follows. In Section 2, we present the definition of packet deadlines and the type of packet schedulers we will be using. In Section 3, we present our definition of flow aggregation and flow aggregators. In Section 4, we present the properties necessary for a flow aggregator to preserve the quality of service to its input flows, and in Section 5, we present two possible implementations of aggregators. In Section 6, we show how flow aggregation may be applied to a core network. Finally, in Section 7, we present related work and concluding remarks.

## 2 Packet Deadlines

Before discussing flow aggregation, we discuss how deadlines are assigned to each packet arriving to a scheduler. We begin with some definitions and notations.

A computer network consists of a set of computers connected via point-to-point communication channels. A *flow* in a computer network is a sequence of packets generated by a single application.

Jorge Arturo Cobb is with the Department of Computer Science, The University of Texas at Dallas, Richardson, Texas, 75083-0688. E-mail: jcobb@utdallas.edu.

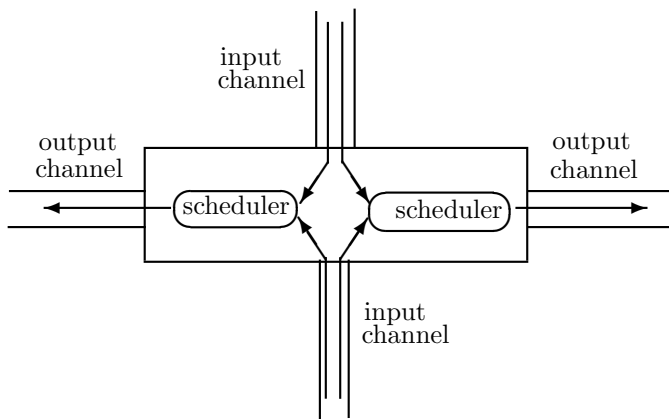


Figure 1: Output channels and their schedulers.

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. The scheduler maintains a first-in-first-out queue for each flow. 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 at least the reserved rate of the flow.

We say a packet is *forwarded* to the output channel when the first bit of the packet is being transmitted by the channel. A packet *exits the scheduler* when the last bit of the packet is transmitted by the output channel. The *path* of flow  $f$  is the sequence of schedulers traversed by  $f$  from its source to its destination.

We adopt the following notation for each flow  $f$  and each scheduler  $s$  along the path of  $f$ .

- $R.f$  forwarding rate (bytes/sec.) reserved for  $f$ .
- $p.f.i$   $i^{\text{th}}$  packet of flow  $f$ ,  $i \geq 1$ .
- $L.f.i$  length (bytes) of packet  $p.f.i$ .
- $A.s.f.i$  arrival time to scheduler  $s$  of packet  $p.f.i$ .
- $E.s.f.i$  exit time of packet  $p.f.i$  from scheduler  $s$ .
- $L_{max}.f$  upper bound on packet length for flow  $f$ .
- $L_{max}.s$  upper bound on packet length for all flows at scheduler  $s$ .
- $C.s$  capacity (bytes/sec.) of the output channel of scheduler  $s$ .

Consider a scheduler  $s$ , an input flow  $f$  of  $s$ , and a packet  $p.f.i$ . We define the *start-time*  $S.s.f.i$  and *finish-time*  $F.s.f.i$  of  $p.f.i$  as follows. Assume  $s$  were to forward the packets of  $f$  at exactly the rate  $R.f$ , as if  $f$  were its only input flow and  $C.s$  were equal to  $R.f$ . Then,  $S.s.f.i$  is the time at which  $p.f.i$  is forwarded to the output channel of  $s$ , and  $F.s.f.i$  is the time at which  $p.f.i$  exits scheduler  $s$ . The formal definition follows.

**Definition 1** Let  $f$  be an input flow of scheduler  $s$ .

$$S.s.f.1 = A.s.f.1.$$

$$S.s.f.i = \max(A.s.f.i, F.s.f.(i-1)), \text{ for all } i, i > 1.$$

$$F.s.f.i = S.s.f.i + L.f.i/R.f, \text{ for all } i, i \geq 1.$$

□

Since flow  $f$  reserves a rate  $R.f$  from scheduler  $s$ , one way to define the deadline of packet  $p.f.i$  is for  $p.f.i$  to exit  $s$  no later than  $F.s.f.i + \alpha.s$  for some small constant  $\alpha.s$  (usually,  $\alpha.s = L_{max}.s/C.s$ ). That is,  $s$  forwards the packets of  $f$  at least as fast as a constant-rate server does. We refer to this type of deadline as *rate-proportional*. With a rate-proportional deadline, the per-hop delay of each packet of  $f$  is at most  $L_{max}.f/R.f + \alpha.s$  at each scheduler  $s$  along its path [16][17]. This delay bound is coupled with the reserved rate of the flow. To decrease the per-hop delay, a greater rate must be reserved.

Another approach for assigning deadlines to packets is the real-time channel model [31]. Here, each flow  $f$  is assigned a pair of values,  $(\delta.f, T.f)$ , and the packet size of the flow is fixed. Packets from flow  $f$  are assumed to arrive with an inter-packet separation time of at least  $T.f$ , and the scheduler guarantees their per-hop delay to be at most  $\delta.f$ . Thus, the deadline of each packet of  $f$  is its arrival time plus  $\delta.f$ . This deadline is flexible, since each flow chooses its  $\delta.f$  value, as opposed to a rate-proportional deadline, where the deadline is coupled with the reserved rate. Due to this deadline flexibility, a non-trivial schedulability test must be performed to determine if the scheduler can satisfy the  $(\delta, T)$  values chosen by each flow. Necessary and sufficient conditions for the schedulability of this model may be found in [31].

In this paper, we choose a more flexible packet deadline based on the packet's start-time (similar to the deadline chosen in [13]), as follows.

**Definition 2** A scheduler  $s$  is a start-time scheduler iff, for every input flow  $f$  of  $s$  and every  $i$ ,  $i \geq 1$ ,

$$E.s.f.i \leq S.s.f.i + \delta.s.f.i$$

for some constant  $\delta.s.f.i$ .

□

Throughout the paper, we assume all schedulers are start-time schedulers. Start-time deadlines are flexible enough to represent both rate-proportional deadlines and real-time channel deadlines. If we choose  $\delta.s.f.i = L_{max}.f/R.f + \alpha.s$ , start-time and rate-proportional deadlines are equal. Real-time channel deadlines are obtained from start-time deadlines by making packets of constant size, setting  $T.f = L.f/R.f$ , and preventing packet  $p.f.i$  from being considered for scheduling until the system's clock equals  $S.s.f.i$ .

Since the start-time of a packet determines its exit time from a scheduler, we must examine how the start-time of a packet changes from one scheduler to the next.

**Theorem 1** Let  $s$  and  $t$  be two consecutive start-time schedulers along the path of flow  $f$ . For all  $i$ ,  $i \geq 1$ ,

$$S.t.f.i \leq S.s.f.i + \Delta.s.f.i$$

where  $\Delta.s.f.i = \max\{\delta.s.f.x \mid 1 \leq x \leq i\}$ .

□

A theorem similar to Theorem 1 was proven in [13], and it also follows from the results in [9]. From this bound on the increase of the start-time of a packet, we obtain a bound on end-to-end packet delay.

**Theorem 2** Let  $t_1, t_2, \dots, t_k$  be a sequence of start-time schedulers traversed by flow  $f$ . For all  $i, i \geq 1$ ,

$$S.t_k.f.i \leq S.t_1.f.i + \sum_{x=1}^{k-1} \Delta.t_x.f.i$$

$$E.t_k.f.i \leq S.t_k.f.i + \delta.t_k.f.i$$

□

The first part of the theorem follows from induction on the length  $k$  of the sequence of schedulers. The second part follows from the definition of a start-time scheduler.

Since rate-proportional delay can be represented using start-time delay, Theorem 2 slightly generalizes the end-to-end theorems using rate-proportional delay reported in [16] and [17].

Due to the flexibility in the assignment of start-time deadlines, a schedulability condition is necessary to determine if these deadlines may be satisfied. Schedulability tests for start-time deadlines, similar to the tests in [31], are given in [8].

### 3 Flow Aggregation

We next describe the aggregation of multiple flows into a single flow. In the next section, we show the end-to-end delay bounds achievable under flow aggregation.

A *simple* flow is a potentially infinite sequence of packets generated by an application. That is, the flows considered in earlier sections were simple flows.

A flow  $f$  is a *constituent* of flow  $g$  if the packets of  $f$  are a subset of the packets of  $g$ .

A scheduler that receives as inputs a set of flows  $f_1, f_2, \dots, f_n$ , and produces as output a single aggregate flow  $g$ , by merging the packets of the input flows, is called an *aggregator*. Flows  $f_1, f_2, \dots, f_n$  are said to be the *immediate constituents* of flow  $g$ . Note that any constituents of flows  $f_1, f_2, \dots, f_n$  are also constituents of  $g$ . The reserved rate,  $R.g$ , of aggregate flow  $g$  is the sum of the reserved rates of the immediate constituent flows of  $g$ .

A scheduler whose set of input flows is the same as its set of output flows is called a *non-aggregating scheduler*.

For each input flow  $g$  of a scheduler, the scheduler is unaware of the constituent flows contained by  $g$  (or simply chooses to ignore them). It thus schedules the packets of  $g$  as if  $g$  were a simple flow with reserved rate  $R.g$ , i.e., the start and finishing times of each packet of  $g$  are defined as if  $g$  were a simple flow with rate  $R.g$ .

We assume all aggregators are start-time schedulers. Thus, every input packet  $p.f.i$  will exit an aggregator  $s$  no later than time  $S.s.f.i + \delta.s.f.i$ .

A *separator* is a process that receives as input an aggregate flow, and produces as output the set of immediate constituents of the input flow. We assume a separator

causes no packet delay. Separators are the only processes aware of the immediate constituents of a flow.

We say that flow  $g$  is the *root* of flow  $f$  at some point in the network if  $f$  is a constituent of  $g$ , and  $g$  is not a constituent of any other flow.

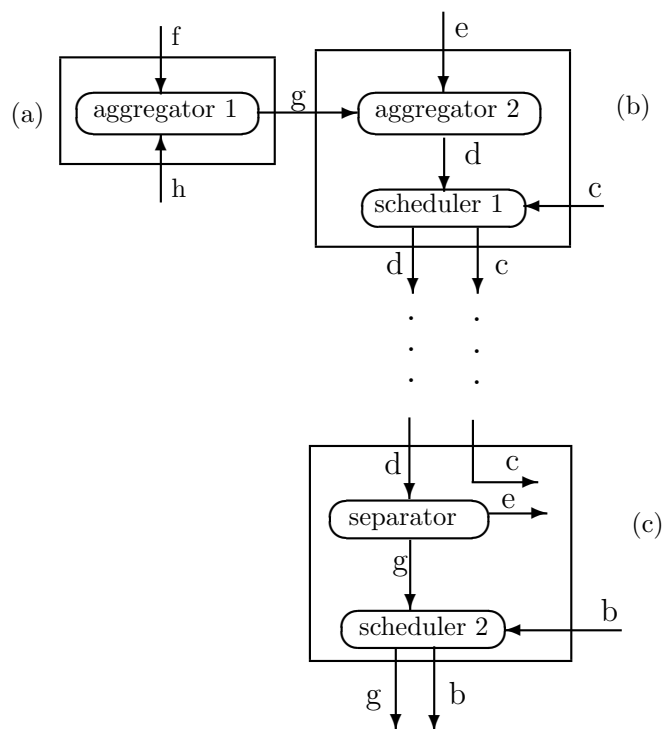


Figure 2: Example of aggregate flows.

Consider for example Figure 2. In Figure 2(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 the input to the computer in Figure 2(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 2(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 Figure 2(a) and Figure 2(b). However, when  $g$  is aggregated into flow  $d$  in Figure 2(b), the root of  $f$  becomes  $d$ . Then, after  $d$  is separated into  $e$  and  $g$  in Figure 2(c),  $g$  is once again the root of  $f$ . 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 2(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 2(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, which makes the packet forwarded by the aggregator immediately available to the next scheduler.

## 4 Fair Aggregation

In this section, we investigate the end-to-end delay of a flow  $f$  as it traverses a network containing flow aggregators and separators. In order to guarantee a low end-to-end delay to each flow, the behavior of flow aggregators must be restricted. We explore this restriction below.

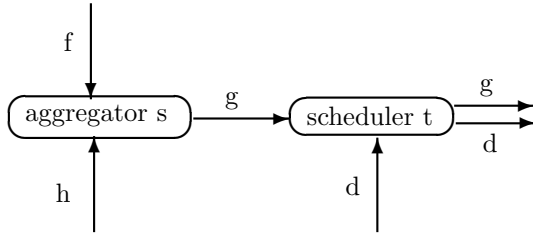


Figure 3: Fair aggregators.

Consider Figure 3. Scheduler  $t$  is not aware that its input flow  $g$  is the aggregation of flows  $f$  and  $h$ . Hence,  $t$  only guarantees that the exit time of each packet  $p.g.j$  is at most  $S.t.g.j + \delta.t.g.j$ , and makes no guarantees about the exit times of packets from  $f$  and  $h$ .

Since the exit time from scheduler  $t$  depends on the start-time of flow  $g$ , we would like to bound the start-time of  $g$  at  $t$  as if no aggregation had occurred. For example, assume instead that  $s$  is a non-aggregating scheduler, and thus,  $f$ ,  $h$ , and  $d$  are individual input flows of scheduler  $t$ . Then, from Theorem 1 and  $s$  being a start-time scheduler,

$$S.t.f.i \leq S.s.f.i + \Delta.s.f.i$$

Let  $p.g.j = p.f.i$ , i.e., the  $j^{\text{th}}$  packet from  $g$  is the  $i^{\text{th}}$  packet from  $f$ . We would like to obtain a similar relationship between  $S.t.g.j$  and  $S.s.f.i$  as the one above between  $S.t.f.i$  and  $S.s.f.i$ . This will bound the exit time from  $t$  of  $p.f.i$  as a function of  $S.s.f.i$ . We choose the bound as follows.

$$S.t.g.j \leq S.s.f.i + \lambda.s.f.i \quad (1)$$

for some constant  $\lambda.s.f.i$ ,  $\lambda.s.f.i \geq 0$ . From (1),  $t$  being a start-time scheduler, and  $p.f.i = p.g.j$ , we conclude,

$$E.t.f.i$$

$$\begin{aligned} &= E.t.g.j \\ &\leq S.t.g.j + \delta.t.g.j \\ &\leq S.s.f.i + \lambda.s.f.i + \delta.t.g.j \end{aligned}$$

Therefore, bound (1) above can be used to compute the exit time of  $p.f.i$  at scheduler  $t$  even though scheduler  $t$  is not aware of flow  $f$ .

### Definition 3

1. An aggregator  $s$  is fair, iff, for every immediate constituent  $f$  of its output flow  $g$ , and for all  $i$ ,  $i \geq 1$ , there exists a constant  $\lambda.s.f.i$  such that,

$$S.t.g.j \leq S.s.f.i + \lambda.s.f.i$$

where  $t$  is the next scheduler of  $g$ , and  $p.g.j = p.f.i$ .

2. A non-aggregating scheduler  $s$  is fair, iff, for every input flow  $f$ , and for all  $i$ ,  $i \geq 1$ , there exists a constant  $\lambda.s.f.i$ , such that,

$$S.t.f.i \leq S.s.f.i + \lambda.s.f.i$$

where  $t$  is the next scheduler of  $f$ .

□

In Section 5, we show how fair aggregators can be constructed. Note that for a non-aggregating scheduler  $s$ , if  $s$  is a start-time scheduler, then  $\lambda.s.f.i \leq \Delta.s.f.i$  (from Theorem 1). Thus, start-time non-aggregating schedulers are fair.

### Definition 4

1. Let  $s$  be a fair aggregator with input flow  $f$ . For all  $i$ ,  $i \geq 1$ ,

$$\Lambda.s.f.i = \max\{\lambda.s.f.k \mid 1 \leq k \leq i\}$$

2. Let  $s$  be a start-time scheduler with input flow  $f$ . For all  $i$ ,  $i \geq 1$ ,

$$\Lambda.s.f.i = \Delta.s.f.i$$

□

As mentioned in Section 3, all aggregators are assumed to be start-time schedulers. However, this does not imply that they are fair, as we show next.

Consider Figure 3, and assume that both  $f$  and  $h$  are generating packets at a rate greater than their reserved rate. The aggregator forwards packets from  $f$  at exactly the rate  $R.f$ , but it forwards the packets from  $h$  at a rate greater than  $R.h$ . Notice that the aggregator satisfies the definition of a start-time scheduler, since the packets of  $f$  will exit the aggregator close to their start-times.

However, it is possible that the queue of  $g$  will grow without bound at scheduler  $t$ , 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 scheduler  $t$ , 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 be fair. That is, it should forward packets from  $f$  and  $h$  in a manner that ensures relation (1) holds, and also ensures a similar relation between  $h$  and  $g$ . We show below that, by restricting all aggregators to be fair, an upper bound on the end-to-end delay of each flow, similar to the bound of Theorem 2, holds in the presence of flow aggregation.

**Definition 5** Let  $r$  be the root flow of  $f$  at scheduler  $s$ , and  $p.r.j = p.f.i$  for some  $i$  and  $j$ ,  $i \geq 1$ ,  $j \geq 1$ . Then,

$$rp.(s.f.i) = p.r.j$$

We refer to  $rp.(s.f.i)$  as the root packet corresponding to  $p.f.i$  at  $s$ .

**Theorem 3** Let  $f$  be an input flow of scheduler  $t_1$ , and let  $f$  traverse schedulers  $t_1, t_2, \dots, t_k$ . Each of these schedulers is a start-time scheduler and is fair. Then,

$$S rp.(t_k.f.i) \leq S.t_1.f.i + \sum_{x=1}^{k-1} \Lambda.rp.(t_x.f.i)$$

$$E.t_k.f.i \leq S rp.(t_k.f.i) + \delta.rp.(t_k.f.i)$$

*Proof*

Since we assume all schedulers are start-time schedulers, the bound on  $E.t_k.f.i$  follows from the definition of a start-time scheduler.

Let  $depth.g$  be the maximum number of nested aggregations a flow  $g$  goes through along its path, and let  $length.g$  be the number of schedulers in the path traversed by  $g$ . Notice that  $depth.g \leq length.g$ , since the aggregation depth can only increase at an aggregator, and an aggregator is also a scheduler.

We can order all pairs  $(d, n)$ , where  $d$  is the depth of a flow and  $n$  is the length of its path, under the total order  $\preceq$  as follows:

$$(d, n) \prec (d', n') = d < d' \vee (d = d' \wedge n < n')$$

$$(d, n) \preceq (d', n') = ((d, n) \prec (d', n')) \vee (d = d' \wedge n = n')$$

Therefore, the set of pairs  $(d, n)$  form a well-formed set under  $\preceq$ , since there are no infinite descending chains.

The proof is based on induction over the pair  $(d, n)$ . For the base case, we consider pair  $(1, 1)$ , which is the only pair without a predecessor under  $\preceq$ . In this case, the flow goes through only one scheduler,  $t_1$ , and the bound on  $S rp.(t_1.f.i)$  reduces to

$$S rp.(t_1.f.i) \leq S.t_1.f.i$$

which is trivial since  $rp.(t_1.f.i) = t_1.f.i$ .

For the induction case, we assume that Theorem 3 holds for all flows of depth less than  $depth.f$ , and for all flows at depth  $depth.f$  (including  $f$ ) for the first  $k-1$  schedulers along the path of the flow.

Consider Figure 4, which gives us three scenarios for the induction step. In scenario (a), the input to  $t_{k-1}$  is  $f$ , and the input to  $t_k$  is also  $f$ . From the induction hypothesis,

$$S rp.(t_{k-1}.f.i) \leq S.t_1.f.i + \sum_{x=1}^{k-2} \Lambda.rp.(t_x.f.i)$$

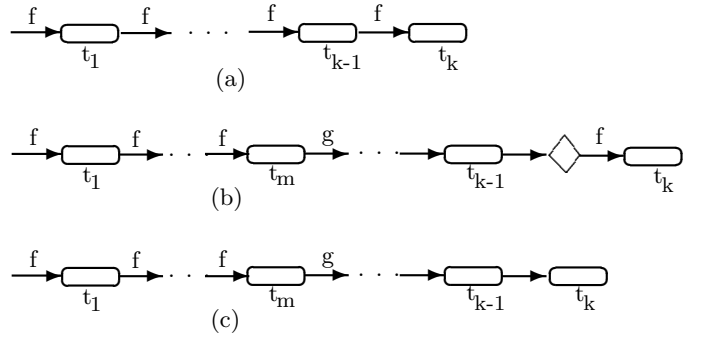


Figure 4: Induction scenarios.

Since  $t_{k-1}$  is a fair scheduler, then

$$S rp.(t_k.f.i) \leq S rp.(t_{k-1}.f.i) + \lambda.rp.(t_{k-1}.f.i)$$

$$\leq S.t_1.f.i + \sum_{x=0}^{k-2} \Lambda.rp.(t_x.f.i) + \Lambda.rp.(t_{k-1}.f.i)$$

which yields the desired result.

Consider now case (b). Here, the input to  $t_{k-1}$  is a flow whose depth is less than the depth of  $f$ , i.e., it contains  $f$  as a constituent flow. The output of  $t_{k-1}$  is then given to one or more separators which separate  $f$ , which is then given as input to  $t_k$ .

Let  $t_m$  be the last aggregator on the path of  $f$  whose input is  $f$ . Thus,  $g$  goes through  $t_{m+1}$  up to  $t_{k-1}$ . Let  $p.g.j = p.f.i$ . Note that the root flow of  $g$  is the same as that of  $f$ . Since the length of the path of  $g$  is less than  $k$ , then, by the induction hypothesis,

$$S rp.(t_{k-1}.g.j) \leq S.t_{m+1}.g.j + \sum_{x=m+1}^{k-2} \Lambda.rp.(t_x.g.j)$$

$$= S.t_{m+1}.g.j + \sum_{x=m+1}^{k-2} \Lambda.rp.(t_x.f.i) \quad (2)$$

Also, from the induction hypothesis,

$$S.t_m.f.i = S rp.(t_m.f.i) \leq S.t_1.f.i + \sum_{x=1}^{m-1} \Lambda.rp.(t_x.f.i) \quad (3)$$

Since  $t_m$  is a fair aggregator,

$$S.t_{m+1}.g.j \leq S.t_m.f.i + \lambda.t_m.f.i \leq S.t_m.f.i + \Lambda.t_m.f.i \quad (4)$$

We assume all schedulers are start-time schedulers. Thus,  $t_{k-1}$  is also a start-time scheduler. If  $t_{k-1}$  is an aggregator, this is irrelevant, since immediately after  $t_{k-1}$  its output flow will be separated into its constituent flows. Hence, we will consider  $t_{k-1}$  as a non-aggregating start-time scheduler. From (2) we conclude,

$$E.rp.(t_{k-1}.g.j) \leq S.t_{m+1}.g.j + \sum_{x=m+1}^{k-2} \Lambda.rp.(t_x.f.i) + \delta.rp.(t_{k-1}.g.j)$$

Combining this with relation (4), and  $p.f.i = p.g.j$ ,

$$\begin{aligned} & E.rp(t_{k-1}.f.i) \\ & \leq S.t_m.f.i + \Lambda.t_m.f.i + \\ & \quad \sum_{x=m+1}^{k-2} \Lambda.rp(t_x.f.i) + \delta.rp(t_{k-1}.f.i) \end{aligned}$$

Note that  $rp(t_m.f.i) = p.f.i$ . Thus,  $\Lambda.t_m.f.i = \Lambda.rp(t_m.f.i)$ , and

$$E.t_{k-1}.f.i \leq S.t_m.f.i + \sum_{x=m}^{k-2} \Lambda.rp(t_x.f.i) + \delta.rp(t_{k-1}.f.i)$$

Thus, the whole path from  $t_m$  up to  $t_{k-1}$  can be viewed as a single start-time scheduler  $P$ , whose  $\delta.P.f.i$  value equals,

$$\sum_{x=m}^{k-2} \Lambda.rp(t_x.f.i) + \delta.rp(t_{k-1}.f.i)$$

Note that  $\Lambda$  is an increasing function with each successive packet, and thus,  $\Delta.P.f.i$ , i.e., the maximum of  $\delta.P.f$  over all packets of  $f$  up to and including  $i$ , is as follows.

$$\Delta.P.f.i \leq \sum_{x=m}^{k-2} \Lambda.rp(t_x.f.i) + \Delta.rp(t_{k-1}.f.i) \quad (5)$$

Since  $t_{k-1}$  is a start-time scheduler, from Definition 4,  $\Lambda.rp(t_{k-1}.f.i) = \Delta.rp(t_{k-1}.f.i)$ . Hence,

$$\Delta.P.f.i \leq \sum_{x=m}^{k-1} \Lambda.rp(t_x.f.i)$$

From Theorem 1 and  $\Delta.P.f.i$  above,

$$S.t_k.f.i \leq S.t_m.f.i + \sum_{x=m}^{k-1} \Lambda.rp(t_x.f.i)$$

Combining the above with (3), and  $rp(t_k.f.i) = p.f.i$ ,

$$S.rp(t_k.f.i) = S.t_k.f.i \leq S.t_1.f.i + \sum_{x=1}^{k-1} \Lambda.rp(t_x.f.i)$$

which is the desired result.

Case now case (c). In case (c), from  $t_m$  to  $t_k$ , flow  $f$  is not its own root. From the induction hypothesis,

$$\begin{aligned} & S.rp(t_k.g.j) \\ & \leq S.t_{m+1}.g.j + \sum_{x=m+1}^{k-1} \Lambda.rp(t_x.g.j) \\ & = S.t_{m+1}.g.j + \sum_{x=m+1}^{k-1} \Lambda.rp(t_x.f.i) \end{aligned}$$

Note that (3) and (4) also hold in case (c). Hence, combining the above with (3) and (4),

$$\begin{aligned} & S.rp(t_k.g.j) \\ & \leq S.t_1.f.i + \sum_{x=1}^{m-1} \Lambda.rp(t_x.f.i) + \\ & \quad \Lambda.t_m.f.i + \sum_{x=m+1}^{k-1} \Lambda.rp(t_x.f.i) \end{aligned}$$

The desired result follows from  $rp(t_m.f.i) = p.f.i$ .

□

By comparing Theorem 2 and Theorem 3, we see that the end-to-end delay bound obtained via flow aggregation is similar to the end-to-end delay bound obtained without flow aggregation. The per-hop delay of packet  $p.f.i$  at a scheduler  $t$  is  $\Delta.t.f.i$  without flow aggregation. On the other hand, with flow aggregation, its per-hop delay is  $\Lambda.rp(t.f.i)$ . Note that if  $t$  is a non-aggregating scheduler, then  $\Lambda.rp(t.f.i) = \Delta.rp(t.f.i)$ . Therefore, the per-hop delay of  $f$  at  $t$  is the same as the per-hop delay of its root flow at  $t$ . This implies that only flows with similar delay requirements must be aggregated together, since the delay is determined by the root flow. This restriction is not too rigid, since it is unlikely that the per-hop delay requirements of flows will be diverse.

As mentioned in Section 2, when using start-time delay, a schedulability test is necessary to determine if the delay requirement of each flow may be satisfied by the scheduler. The obvious question to ask is the following. Assume each flow has a per-hop delay requirement, and assume that all the flows are schedulable in a network without aggregation. Then, if we aggregate together flows with the same per-hop delay requirement, would the schedulability test still be satisfied? In [8], we confirmed that the schedulability test is still satisfied, and thus, the schedulability region does not decrease using flow aggregation.

Consider now flow aggregation using rate-proportional deadlines. In [7], we presented a theorem for the end-to-end delay with rate-proportional deadlines (this theorem is a corollary of Theorem 3). Examples of schedulers implementing these deadlines are Virtual Clock [27][28], PGPS [18][21], and Time-Shift Scheduling [4][25][10]. Recall that with rate-proportional deadlines, each packet  $p.g.j$  of an input flow  $g$  would exit scheduler  $s$  no later than  $F.s.g.j + \alpha.s$ , where  $\alpha.s$  is a constant (usually equal to  $L_{max}.s/C.s$ ). The per-hop delay of flow  $f$  at scheduler  $s$  without flow aggregation is  $L_{max}.f/R.f + L_{max}.s/C.s$ , and with flow aggregation it is  $L_{max}.g/R.g + L_{max}.s/C.s$ , where  $g$  is the root flow of  $f$  at  $s$ . Notice that if  $f$  is a constituent flow of  $g$ , then  $R.f < R.g$ , and the delay with aggregation is smaller than the delay without aggregation, provided  $L_{max}.g \approx L_{max}.f$ . Thus, for rate-proportional delay, aggregation not only improves efficiency, but also *decreases end-to-end delay*.

The discussion above focused on the per-hop delay at non-aggregating schedulers under flow aggregation. We still need to consider the delay encountered by a flow at an aggregating scheduler. Below, we consider how to implement fair aggregators with a small aggregation constant  $\lambda$ .

## 5 Implementing Fair Aggregators

In this section we present two fair aggregators, the basic fair aggregator and the greedy fair aggregator.

## 5.1 Basic Fair Aggregator

In Section 4, we mentioned two requirements for a fair aggregator. First, if flows  $f$  and  $h$  are aggregated into a flow  $g$ , then the aggregator should not forward packets from  $f$  faster than  $R.g$  when there are no packets available from  $h$ . Second, if packets from both flows are available, and the aggregator forwards packets at a rate greater than  $R.g$ , then the packets of both  $f$  and  $h$  should be forwarded in proportion to their rates.

Let  $s$  be an aggregator,  $g$  be the output flow of  $s$ , and the channel capacity of  $s$  be  $C.s$ . A simple technique to construct  $s$  to satisfy the above requirements is as follows. Consider a fictitious start-time scheduler  $v$ , whose output channel has capacity  $R.g$ , and whose set of input flows is the same as that of  $s$ . Aggregator  $s$  forwards each packet to its output channel at exactly the same time the fictitious scheduler  $v$  would forward the same packet. Thus, since the output channel of  $v$  has capacity  $R.g$ ,  $s$  cannot forward the packets at a rate higher than  $R.g$ , even though its channel has capacity  $C.s$ , where  $C.s \geq R.g$ .

We call an 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 start-time scheduler  $v$ . E.g., we could define a Virtual Clock basic fair aggregator, a Weighted Fair Queuing basic fair aggregator, etc..

The first of the two requirements for a fair aggregator mentioned above is met, since the packets are forwarded at a rate at most  $R.g$ . In addition, since the basic fair aggregator does not forward packets at a rate greater than  $R.g$ , the second requirement is irrelevant. Finally, since the basic fair aggregator forwards each packet at the same time as a start-time scheduler, then the basic fair aggregator is also a start-time scheduler.

We next prove that a basic fair aggregator is indeed fair. We begin with a lemma.

**Lemma 1** *Let  $s$  be a basic fair aggregator,  $g$  be the output flow of  $s$ , and  $t$  be the scheduler after  $s$ . For all  $k, k > 1$ ,*

$$F.t.g.(k-1) \leq A.t.g.k + L_{max}.g/C.s$$

*Proof*

Let  $p.g.i, \dots, p.g.k, i < k$ , be a sequence of packets of flow  $g$  such that  $F.t.g.i = A.t.g.i + L.g.i/R.g$ , and for each  $j, i < j \leq k$ ,

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

That is,  $A.t.g.j \leq F.t.g.(j-1)$ . If no such  $i$  exists, then  $F.t.g.(k-1) < A.t.g.k$ , and we are done.

Let  $p.g.m, i \leq m \leq k$ , be the largest packet in the sequence. We will show that

$$\begin{aligned} & F.t.g.(k-1) - A.t.g.k \\ & \leq L.g.m/C.s - L.g.k/C.s \\ & \leq L_{max}.g/C.s \end{aligned}$$

The proof is based on induction on  $k$ . Consider  $k = i+1$  for the base case. Packet  $p.g.i$  is chosen for transmission at

the aggregator  $L.g.i/C.s$  seconds before arriving at  $t$ , and from the definition of a basic fair aggregator,  $p.g.(i+1)$  begins transmission at the aggregator at least  $L.g.i/R.g$  seconds after  $p.g.i$  is chosen for transmission, and arrives to  $t$   $L.g.(i+1)/C.s$  seconds later. Thus,

$$\begin{aligned} & A.t.g.(i+1) \\ & \geq A.t.g.i - L.g.i/C.s + L.g.i/R.g + \\ & \quad L.g.(i+1)/C.s \end{aligned}$$

We are given that  $F.t.g.i = A.t.g.i + L.g.i/R.g$ , and hence,

$$\begin{aligned} & F.t.g.i - A.t.g.(i+1) \\ & \leq A.t.g.i + L.g.i/R.g - (A.t.g.i - L.g.i/C.s + \\ & \quad L.g.i/R.g + L.g.(i+1)/C.s) \end{aligned}$$

Eliminating terms,

$$F.t.g.i - A.t.g.(i+1) \leq L.g.i/C.s - L.g.(i+1)/C.s$$

Since  $L.g.i \leq L.g.m$ , and  $k = i+1$ , we obtain the desired result.

For the induction step, we are given that

$$\begin{aligned} & F.t.g.(k-2) - A.t.g.(k-1) \\ & \leq L.g.m/C.s - L.g.(k-1)/C.s \end{aligned}$$

Again, from the definition of a basic fair aggregator,

$$\begin{aligned} & A.t.g.k \\ & \geq A.t.g.(k-1) - L.g.(k-1)/C.s + L.g.(k-1)/R.g \\ & \quad + L.g.k/C.s \end{aligned}$$

Since  $F.t.g.(k-1) = F.t.g.(k-2) + L.g.(k-1)/R.g$ , we have,

$$\begin{aligned} & F.t.g.(k-1) - A.t.g.k \\ & \leq F.t.g.(k-2) + L.g.(k-1)/R.g - (A.t.g.(k-1) - \\ & \quad L.g.(k-1)/C.s + L.g.(k-1)/R.g + L.g.k/C.s) \end{aligned}$$

From the hypothesis,  $F.t.g.(k-2) \leq A.t.g.(k-1) + L.g.m/C.s - L.g.(k-1)/C.s$ . Thus,

$$\begin{aligned} & F.t.g.(k-1) - A.t.g.k \\ & \leq A.t.g.(k-1) + L.g.m/C.s - L.g.(k-1)/C.s + \\ & \quad L.g.(k-1)/R.g - (A.t.g.(k-1) - L.g.(k-1)/C.s + \\ & \quad L.g.(k-1)/R.g + L.g.k/C.s) \end{aligned}$$

Finally, reducing we obtain,

$$F.t.g.(k-1) - A.t.g.k \leq L.g.m/C.s - L.g.k/C.s$$

□

**Theorem 4** *Let  $s$  be a basic fair-aggregator,  $f$  one of its input flows,  $g$  its output flow, and  $v$  be the fictitious scheduler emulated by  $s$ . Then,*

$$\begin{aligned} E.s.f.i & \leq S.s.f.i + \delta.v.f.i - (L.f.i/R.g - L.f.i/C.s) \\ S.t.g.j & \leq S.s.f.i + \delta.v.f.i - (L.f.i/R.g - L.f.i/C.s) + \\ & \quad L_{max}.g/C.s \end{aligned}$$

where  $t$  is the scheduler after  $s$ , and  $p.g.j = p.f.i$ .

*Proof*

Consider first  $E.s.f.i$ . Since  $s$  forwards packet  $p.f.i$  at the same time as  $v$ , and the output channel of  $s$  has capacity  $C.s$ , then the bound on the exit time of  $p.f.i$  from  $s$  is the same as in  $v$ , except for the difference in the time required to transmit packet  $p.f.i$  itself due to the difference in the capacity of the two channels.

Consider next  $S.t.g.j$ .

From Lemma 1, for all  $j, j > 1$ ,  $A.t.g.j \geq F.t.g.(j-1) - L_{max}.g/C.s$ . Thus, from Definition 1,

$$S.t.g.j \leq A.t.g.j + L_{max}.g/C.s$$

Furthermore, since  $v$  is a start-time scheduler, and the output channel rate of  $s$  is  $C.s$ , we have,

$$A.t.g.j \leq S.s.f.i + \delta.v.f.i - (L.f.i/R.g - L.f.i/C.s) \quad (6)$$

Hence,

$$\begin{aligned} & S.t.g.j \\ \leq & S.s.f.i + \delta.v.f.i - (L.f.i/R.g - L.f.i/C.s) + \\ & L_{max}.g/C.s \end{aligned}$$

Consider  $j = 1$ . In this case,  $S.t.g.j = A.t.g.j$ . Thus, using (6), which holds for all  $j, j \geq 1$ ,

$$S.t.g.j \leq S.s.f.i + \delta.v.f.i - (L.f.i/R.g - L.f.i/C.s)$$

□

Thus, we can construct a fair aggregator from any start-time scheduler using the technique above. Notice that if the start-time scheduler  $v$  being emulated is work-conserving, then an input flow  $f$  of the aggregator  $s$  will be allowed to forward packets at a rate greater than  $R.f$ , but not at a rate greater than  $R.g$ . Thus, if the scheduler  $v$  distributes unused bandwidth in a fair manner among its input flows, such as in [4][10][25] then aggregator  $s$  will also distribute unused bandwidth (up to rate  $R.g$ ) among its input flows.

## 5.2 Greedy Fair Aggregators

As discussed in the previous section, the basic fair aggregator cannot forward packets at a rate faster than  $R.g$ , where  $g$  is its aggregate output flow. For completeness, we next discuss an aggregator which can forward packets at a rate faster than  $R.g$ . We call this aggregator *greedy*. However, the higher rate may be maintained only as long as all input flows have an arrival rate greater than their reserved rates. This could occur if all input flows use a mechanism to detect congestion from the network, such as packet pair [18], and the output channel's rate is greater than the sum of the reserved rates of all the input flows.

Let  $g$  be the output flow of a greedy aggregator  $s$ . Let  $v$  be a fictitious start-time scheduler, whose input flows are the same as those of  $s$ , and whose output channel has a rate equal to  $R.g$ . We assume scheduler  $v$  assigns timestamps to packets, is work conserving, and forwards packets in order of increasing timestamp.

Aggregator  $s$  assigns to each packet the same timestamp it would receive at scheduler  $v$ . Also,  $s$  has access to a real-time clock, and it maintains a variable,  $D$ , whose initial value is zero. After packet  $p.f.i$  is forwarded by  $s$ ,  $D$  is increased as follows.

$$D := \max(\text{clock}, D) + L.f.i/R.g$$

Aggregator  $s$  forwards each packet as soon as possible, provided at least one of the following conditions holds before forwarding the packet:

1.  $\text{clock} \geq D$ ,
2. all input flows have at least one packet buffered in the scheduler

We next show that  $s$  is a start-time scheduler and also a fair aggregator. We begin with a lemma.

**Lemma 2** *Let  $s$  be a greedy fair aggregator, and let  $v$  be the scheduler emulated by  $s$ . Let  $p.s.i$  and  $p.v.i$  be the  $i^{\text{th}}$  packet forwarded by  $s$  and  $v$ , respectively.*

- Both  $s$  and  $v$  forward packets in the same order, i.e., for all  $i, i \geq 1$ ,  $p.s.i = p.v.i$ .
- For all  $i, i \geq 1$ ,  $s$  forwards  $p.s.i$  to its output channel no later than  $v$  forwards  $p.v.i$  to its output channel.
- For all  $i, i \geq 1$ ,  $p.v.i$  exits scheduler  $v$  at the time contained in  $D$  immediately after  $p.s.i$  is forwarded by  $s$ .

*Proof*

The proof is based on induction over the number of packets forwarded by  $s$ . For the base case, consider  $p.s.1$ . Since at the beginning,  $\text{clock} \geq 0 = D$ ,  $s$  will forward its first received packet immediately (i.e.,  $p.s.1$  is the first packet received). After forwarding the packet,  $D := \max(D, \text{clock}) + L.s.1/R.g = \text{clock} + L.s.1/R.g$ . Because  $v$  is work conserving,  $v$  will forward its first received packet also immediately (i.e.  $p.s.1 = p.v.1$ ), and the packet will exit at time  $\text{clock} + L.v.1/R.g = \text{clock} + L.s.1/R.g$ , which is the new value of  $D$  at  $s$ .

For the induction step, assume the lemma holds for the first  $k$  packets. Consider the system when  $s$  chooses its next packet,  $p.s.(k+1)$  to be forwarded. Let this time be  $T$ , and let  $D'$  be the value of  $D$  at time  $T$  (i.e., before  $s$  forwards  $p.s.(k+1)$  and updates  $D$ ).

From the induction hypothesis, at time  $T$ ,

- $p.s.k = p.v.k$
- $p.v.k$  exits  $v$  at time  $D'$ .
- $p.s.k$  is forwarded by  $s$  no later than the time it is forwarded by  $v$  (and hence, it exits  $s$  no later than it exits  $v$ ).

Our proof obligations are as follows.

1.  $p.s.(k+1) = p.v.(k+1)$ .
2.  $p.v.(k+1)$  exits  $v$  at time  $T + L.s.(k+1)/R.g$  if  $T > D'$  or at time  $D' + L.s.(k+1)/R.g$  if  $T \leq D'$ .
3.  $s$  forwards  $p.s.(k+1)$  no later than  $v$  forwards  $p.v.(k+1)$ .

Assume first  $T > D'$ . We know  $p.s.k$  exits  $s$  no later than  $D'$ . After time  $D'$ , since  $\text{clock} \geq D'$ ,  $s$  can forward

the next packet, but it does not do so until time  $T$ , where  $T > D'$ . Hence, this implies the queue of  $s$  is empty after forwarding  $p.s.k$ , and no packets are received by  $s$  from time  $D'$  up to time  $T$ . Thus,  $p.s.(k + 1)$  arrives at  $s$  at time  $T$ .

From the hypothesis,  $p.s.k = p.v.k$ , and  $p.v.k$  exits  $v$  at time  $D'$ . Hence, the queue of  $v$  is also empty from  $D'$  up to  $T$ . Since  $v$  is work-conserving, it will forward the next packet received, namely,  $p.s.(k + 1)$ . Thus,  $p.s.(k + 1) = p.v.(k + 1)$  (proof obligation 1), and both are forwarded at the same time by  $s$  and  $v$  (proof obligation 3). After  $p.s.(k + 1)$  is forwarded at time  $T$ ,  $D$  is updated to  $T + L.s.(k + 1)/R.g$ , which equals  $T + L.v.(k + 1)/R.g$ , which is the time  $L.v.(k + 1)$  exits  $v$  (proof obligation 2).

Consider next  $T < D'$ . This implies that  $s$  has a packet available from every input flow at time  $T$ . Therefore, since packets timestamps from the same flow are increasing, and  $p.s.(k + 1)$  was chosen to be forwarded, then  $p.s.(k + 1)$  has a timestamp less than or equal to all packets  $p.s.j$ , where  $j > k + 1$ .

From the induction hypothesis, the first  $k$  packets forwarded by  $s$  and  $v$  are the same, and packet  $p.v.k$  (i.e.  $p.s.k$ ) exits  $v$  at time  $D'$ . Since  $T < D'$ , at least one packet from each flow is available at  $v$  at time  $T$ . From the discussion above,  $p.s.(k + 1)$  has the smallest timestamp of all these packets, and will be forwarded by  $v$  (i.e.  $p.s.(k + 1) = p.v.(k + 1)$ ) at time  $D'$ ,  $D' > T$  (proof obligations 1 and 3). Since  $v$ 's output channel has rate  $R.g$ ,  $p.v.(k + 1)$  will exit at time  $D' + L.v.(k + 1)/R.g$  which equals  $D' + L.s.(k + 1)/R.g$  (proof obligation 2).

Finally, consider  $T = D'$ . From the induction hypothesis,  $p.v.k$  exits scheduler  $v$  at time  $D'$ . This implies both  $p.s.(k + 1)$  at  $s$  and  $p.v.(k + 1)$  at  $v$  are chosen to be forwarded at time  $T$ . Since the first  $k$  packets forwarded at  $s$  and  $v$  are the same, at time  $T$  the minimum timestamp packet is the same in both systems. Hence, the same packet is forwarded in both systems and at the same time (proof obligations 1 and 3). The new value of  $D$  in  $s$  will be  $D' + L.s.(k + 1)/R.g$  which equals  $D' + L.v.(k + 1)/R.g$ , which is the time at which packet  $p.v.(k + 1)$  exits scheduler  $v$  (proof obligation 2).  $\square$

**Theorem 5** *Let  $s$  be a greedy fair aggregator,  $f$  be one of its input flows, and  $g$  its output flow. Furthermore, let  $v$  be the fictitious scheduler emulated by the greedy fair aggregator. Then,*

$$E.s.f.i \leq S.s.f.i + \delta.v.f.i - (L.f.i/R.g - L.f.i/C.s)$$

$$S.t.g.j \leq S.s.f.i + \delta.v.f.i - (L.f.i/R.g - L.f.i/C.s) + L_{max}.g/C.s$$

where  $t$  is the scheduler after  $s$ , and  $p.g.j = p.f.i$ .

*Proof*

From the construction of a basic aggregator, a basic aggregator forwards each packet at exactly the same time as the emulated scheduler  $v$ . Furthermore, the greedy aggregator  $s$ , from Lemma 2, forwards packets in the same order and no later than  $v$ . Hence, the greedy aggregator

forwards packets in the same order and no later than a basic fair aggregator which also emulates  $v$ . The upper bound on  $E.s.f.i$  follows from Theorem 4. Also, the start-time of a packet, from Definition 1, cannot increase if the arrival time of the packet decreases. Hence, the upper bound on  $S.t.g.j$  also follows from Theorem 4.  $\square$

From the above theorem, a greedy aggregator is a start-time scheduler, and it is also fair.

## 6 Flow Aggregation in a Core Network

In this section, we present a practical example of flow aggregation while preserving quality of service. Consider the inter-network in Figure 5, which consists of a core network and a set of access networks. The purpose of the core network is to route messages between the access networks, and the access networks are attached to the core network via access routers. This model is similar to the one presented in [24].

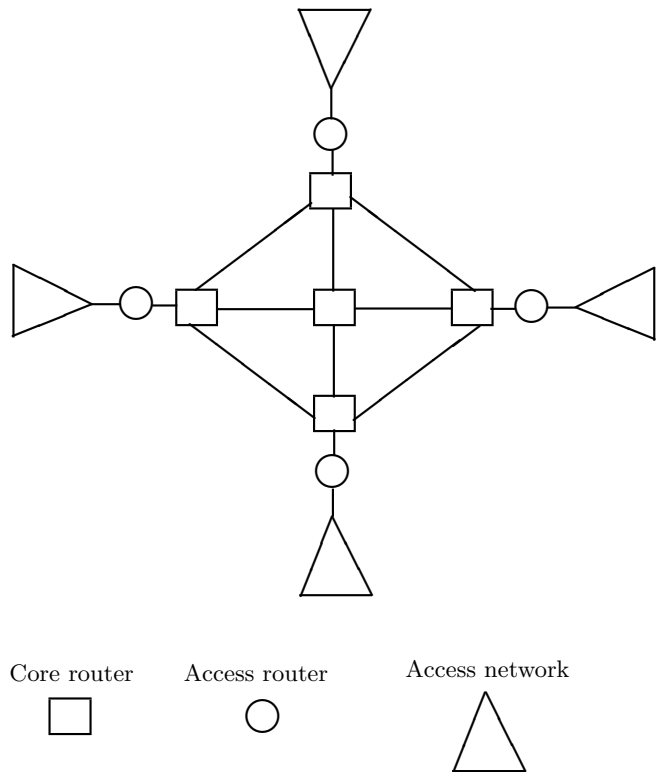


Figure 5: Flow aggregation in a core-network.

Our goal is to reduce the number of flows visible by each of the routers in the core network. In particular, each core router should receive on each input link at most  $N$  flows, where  $N$  is the number of access routers attached to the core-network (i.e., the number of exit points from the core network).

To this end, assume flows are managed individually by

the access network, and are managed as aggregate flows in the core network. The access router aggregates flows entering the core network according to each exit point of the core network. E.g., if flows  $f$  and  $h$  enter the core network through the same access router  $r$  and exit the core network through the same access router  $s$ , then  $f$  and  $h$  are aggregated into a single flow at router  $r$  before being forwarded to the core router.

Each core router will receive from each input link at most  $N$  flows, one for each possible access router in the core network. For each output link, the core router will aggregate together into a single flow all flows which exit the core network via the same access router. Note that in this way, each core router is guaranteed to receive at most  $N$  flows from each input link.

Each access router will receive from the core network at most one flow. This flow is an aggregation of all the flows which exit the core network at this particular access router. The access router will then separate all the individual flows from this aggregate flow, and continue to route them in its access network.

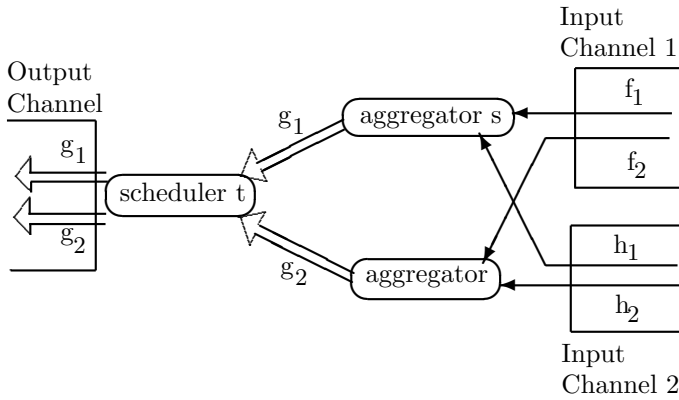


Figure 6: Aggregation at a core router.

Consider as an example a simple case in which a core router has two input channels, and two access routers are reachable along the path of one of its output channels. Figure 6 depicts the scheduling components needed for this output channel. From one input channel, the core router receives flows  $f_1$  and  $f_2$ , which are destined to access router 1 and access router 2, respectively. From the other input channel, the core router receives flows  $h_1$  and  $h_2$ , which are destined to access router 1 and access router 2, respectively. Since flows  $f_1$  and  $h_1$  are destined to the same access router, they are aggregated together, resulting in flow  $g_1$ . Similarly, flows  $f_2$  and  $h_2$  are aggregated together, resulting in flow  $g_2$ . Flows  $g_1$  and  $g_2$  are then given to the scheduler of the output channel, which is a non-aggregating scheduler.

Consider flow  $f_1$ . If aggregator  $s$  is either a basic fair aggregator or a greedy fair aggregator, then from Theorems 4 and 5,

$$S.t.g_1.j$$

$$\leq S.s.f_1.i + \delta.v.f_1.i - (L.f_1.i/R.g_1 - L.f_1.i/C.s) + L_{max}.g_1/C.s$$

where  $p.g_1.j = p.f_1.i$ , and  $v$  is the start-time scheduler emulated by the aggregator.

Since aggregator  $s$  is internal to the router, we assume that  $C.s$  is infinity, and hence, the last two terms above are zero.

$$S.t.g_1.j \leq S.s.f_1.i + \delta.v.f_1.i - L.f_1.i/R.g_1$$

Consider when scheduler  $v$  is similar to Virtual Clock scheduling [27][28] or Time-Shift scheduling [10]. In this case,

$$\begin{aligned} & \delta.v.f_1.i \\ = & \\ & L.f_1.i/R.f_1 + L_{max}.g_1/C.v \\ = & \\ & L.f_1.i/R.f_1 + L_{max}.g_1/R.g \end{aligned}$$

Combining this with the above,

$$\begin{aligned} & S.t.g_1.j \\ \leq & S.s.f_1.i + L.f_1.i/R.f_1 + L_{max}.g_1/R.g \\ & - L.f_1.i/R.g_1 \end{aligned}$$

Let  $u$  be the next scheduler after  $t$ . In our core network model,  $u$  would be the aggregator at the entrance of the next core router. If scheduler  $t$  is a Virtual Clock scheduler or a Time-Shift scheduler, then

$$\delta.t.g_1.j = L.g_1.j/R.g + L_{max}.t/C.t$$

From Theorem 1 and the above two relations,

$$\begin{aligned} & S.rp(u.f.i) \\ = & S.u.g_1.j \\ \leq & S.s.f_1.i + L.f_1.i/R.f_1 - L.f_1.i/R.g_1 \\ & 2 \cdot L_{max}.g_1/R.g + L_{max}.t/C.t \end{aligned}$$

Therefore, the increase in the start-time of the root packet  $p.f.i$  at the core router is

$$L.f_1.i/R.f_1 - L.f_1.i/R.g_1 + 2 \cdot L_{max}.g_1/R.g + L_{max}.t/C.t \quad (7)$$

We next examine if this increase is practical. We will make the simplifying assumptions that all flows will have the same maximum packet size and all core routers have an output channel of capacity  $C$ .

The increase in start-time is inversely proportional to  $R.g_1$ . Since  $R.g_1 \geq R.f_f$ , the above increase is maximum when  $R.g_1 \approx R.f_1$ , that is, the rate of  $f_2$  is very small compared to that of  $f_1$ . In this case,  $L.f_1.i/R.f_1 \approx L.f_1.i/R.g_1$  and  $L_{max}/R.g_1 \approx L_{max}/R.f_1$ . Hence, from (7), the increase in the start-time reduces to

$$2 \cdot L_{max}/R.f_1 + L_{max}/C \quad (8)$$

If the core router were simply a start-time scheduler with no aggregation such as Virtual Clock, the increase in the start-time of  $f_1$  would be

$$L_{max}/R.f_1 + L_{max}/C$$

With our worst-case assumption of  $R.g_1 \approx R.f_1$ , the additional increase caused by flow aggregation is  $L_{max}/R.f_1$ . This increase in start-time is comparable to that of some techniques which attempt to reduce the implementation complexity of scheduling algorithms, such as the technique presented in [3], and furthermore, this worst-case scenario is highly unlikely in a core network which handles a large number of flows.

Consider another scenario. Assume that at the entrance of the network, flow  $f_1$  is aggregated with at least one more flow of the same rate. Then,  $R.g_1 \geq 2 \cdot R.f_1$ . In this case, from (7), the increase in the start-time at the first core router is at most

$$L.f_1.i/R.f_1 + L_{max}/R.f_1 + L_{max}/C$$

In subsequent hops, even if the resulting flow  $g_1$  is aggregated with other flows whose rate is arbitrarily small, from (8) (replacing  $f_1$  for  $g_1$ ), the per-hop increase in the start-time of  $f_1$  is at most

$$2 \cdot L_{max}/R.g_1 + L_{max}/C$$

which is at most

$$L_{max}/R.f_1 + L_{max}/C$$

Hence, the per-hop increase in the start-time of  $f_1$  is the same as the per-hop increase in start-time without aggregation, except for an additional increase of  $L_{max}/R.f_1$  at the initial router. Therefore, aggregation does not significantly increase the start-time of  $f_1$ . Again, this is an unlikely scenario, since we expect a large number of flows to be aggregated with  $f_1$  along its path to its destination.

Finally, consider a more likely scenario. Assume  $R.g_1 \geq 10 \cdot R.f_1$ . From (7), the increase in start-time through the first router is at most

$$L.f_1.i/R.f_1 + \frac{1}{5} \cdot (L_{max}/R.f_1) + L_{max}/C$$

This is only 20% higher than the increase in start-time without aggregation. In subsequent hops, even if the resulting flow  $g_1$  is aggregated with flows of arbitrarily small rate, from (8) (replacing  $f_1$  for  $g_1$ ), the per-hop increase in the start-time of  $f_1$  is at most

$$2 \cdot L_{max}/R.g_1 + L_{max}/C$$

which is at most

$$\frac{1}{5} \cdot (L_{max}/R.f_1) + L_{max}/C$$

Hence, the per-hop increase in the start-time of  $f_1$  actually *decreases* with flow aggregation to about one fifth of value without aggregation.

We next consider the implementation complexity for the configuration in Figure 5.

Inserting or removing a packet into the priority queue of an aggregator takes  $O(\log m)$  time, where  $m$  is the number of input links. Inserting or removing a packet from the priority queue of scheduler  $t$  takes  $O(\log N)$  time, where  $N$  is the number of exit points of the core network. However, a performance problem occurs because the aggregator is not work conserving, and therefore, on occasions it has packets in its queue, but none can be made available to the scheduler at the time. If one packet from every aggregator becomes available to the scheduler at the same time, queuing these packets at the scheduler would require  $O(N \cdot \log N)$  time, which may be excessive.

The above problem, however, may be solved using a simple modification to the technique introduced in [23], which can be used to reduce the complexity to  $O(\log N)$  time. The technique would consist of having an array indexed by time. The entry corresponding to time  $t$  contains a priority queue of packets which become available at time  $t$ , and performing insertion/deletion operations into this array in a way similar to those in [23]. Introducing this array increases the per-hop delay by the granularity of time of the array index. This, in general, would be quite small, such as the amount of time to transmit one packet.

## 7 Related Work and Concluding Remarks

In this paper, we have defined the aggregation of multiple flows into a single flow, and how the end-to-end delay bound is preserved in spite of flow aggregation. The advantages of aggregation are simplified scheduling and signaling due to the reduction in the number of input flows to a scheduler. The disadvantages are that flow aggregation is performed by a non-work conserving scheduler, and that an aggregate flow is separated into all its immediate constituents. We believe it is possible to separate a constituent flow  $f$  from an aggregate flow  $g$  without exposing all the constituent flows of  $g$ . However, to do so, it is likely that aggregation will be required to be less flexible. In particular, it is likely that an aggregator should not forward the packets of an input flow at a rate higher than its reserved rate. We plan to investigate this in our future work.

In [24], a rate-proportional protocol is presented for a network core without per-flow state. This is advantageous, since no flow state needs to be maintained. However, the packet delay is the same as with per-flow state techniques, and hence, it is higher than the delay with flow aggregation.

In [11], a different approach is taken for the aggregation of flows. The delay-jitter of a flow  $f$  that is aggregated with other flows to form flow  $g$  is set to zero. Thus, the characteristics of  $f$  once it is separated from  $g$  are identical to its characteristics when it was aggregated into  $g$  (modulo an equal delay applied to all packets). The disadvantage of this approach is that  $f$  cannot take advantage of unused bandwidth and temporarily exceed its reserved rate  $R.f$ .

In our approach, a flow  $f$  can exceed its reserved rate  $R.f$  up to the reserved rate  $R.g$  of its parent flow, allowing for a better use of network bandwidth.

The traffic scheduling model used in [11] is more general than the model used in this paper. The model is based on service curves [1][2][5][6][12][19][20][22], a model which has matured in the last few years. We chose not to adopt this model in our paper, since our results cannot be extended to service curves in general. That is, under the general service curve model, if a flow  $f$  exceeds its service specification while being aggregated with other flows, the delay guarantees of the other flows may be violated. Therefore, we considered it more appropriate to present the results under the simpler model of start-time deadlines. However, there could be specific cases under the service curve model in which  $f$  may exceed its service specification without violating the guarantees of other flows. We plan to investigate this in our future work.

## Acknowledgment

The authors would like to thank the referees for their constructive criticism which improved the quality of this paper.

## References

- [1] Agrawal R., Rajan R., "Performance Bounds for Guaranteed and Adaptive Services", *Proceedings of the 34th Allerton Conference on Comm., Cont., and Comp.*, Moticello, IL, Oct. 1996.
- [2] Baccelli F., Cohen G., Olsder G., Quadrat J., *Synchronization and Linearity: An Algebra for Discrete Event Systems*, Wiley, 1992.
- [3] Bennett J.C.R., Stephens D.C., Zhang H., "High Speed, Scalable and Accurate Implementation of Packet fair Queuing Algorithms in ATM Networks", *Proceedings of the IEEE International Conference on Network Protocols*, pp. 7-14, 1997.
- [4] Bennett J.C.R., H. Zhang, "Hierarchical Packet Fair Queuing Algorithms", *Proceedings of the ACM SIGCOMM Conference*, Palo Alto, California, pp 143-156, 1996.
- [5] Chang C.S., "On Deterministic Traffic Regulation and Service Guarantees: A Systematic Approach by Filtering", *IEEE Transactions on Information Theory*, Vol. 44., pp. 1097-1110, 1998.
- [6] Chang C.S., Lin Y.H., "A General Framework for Deterministic Service Guarantees in Telecommunication Networks with Variable Length Packets", *Proceedings of the Sixth IEEE International Workshop on Quality of Service (IWQoS)*, pp. 49-58, 1998.
- [7] Cobb J., "Preserving Quality of Service Guarantees In-Spite of Flow Aggregation", *Proceedings of the Sixth IEEE International Conference on Network Protocols*, pp. 90-97, 1998.
- [8] Cobb J., "An In-Depth Look at Flow Aggregation", *Proceedings of the Seventh IEEE International Conference on Network Protocols*, pp. 127-134, 1999.
- [9] Cobb J., Gouda M., "Flow Theory", *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5, pp. 661-674, Oct. 1997.
- [10] Cobb J., Gouda M., El-Nahas A., "Time-Shift Scheduling-Fair Scheduling of Flows in High-Speed Networks", *IEEE/ACM Transactions on Communications*, Vol. 6, No. 3, pp. 274 -285, June 1998.
- [11] Cruz R. L., "SCED+: Efficient Management of Quality of Service Guarantees", *Proceedings of the IEEE INFOCOM Conference*, Vol. 2, pp. 625-634, March, 1998.
- [12] Cruz R.L., "Quality of Service Guarantees in Virtual Circuit Switched Networks", *IEEE Journal on Selected Areas of Communications*, Vol. 13, No. 6, pp. 1048-1056, Aug. 1995.
- [13] Figueira N., Pasquale J., "Leave-in-Time: A New Service Discipline for Real-Time Communications in a Packet-Switching Network", *Proceedings of the ACM SIGCOMM Conference*, pp. 207-218, Aug. 1995, Cambridge United States .
- [14] Gall D., "MPEG: A Video Compression Standard for Multimedia Applications", *Communications of the ACM*, 34(4), pp. 46-58, April 1991.
- [15] Gouda M., *The Elements of Network Protocols*, Wiley Publishers, 1998.
- [16] Goyal P, Lam S., Vin H., "Determining End-to-End Delay Bounds in Heterogeneous Networks", *Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video (NOSS-DAV)*, 1995.
- [17] Goyal P, Vin H., "Generalized Guaranteed Rate Scheduling Algorithms", *IEEE/ACM Transactions on Networking*, Vol. 5, No. 4, pp. 561-571, August 1997
- [18] Keshav S., "A Control Theoretic Approach to Flow Control", *Proceedings of the ACM SIGCOMM Conference*, pp. 3-15, Sept. 1991, Zurich Switzerland.
- [19] Le Boudec J.Y., "Application of Network Calculus to Guaranteed Service Networks", *IEEE Transactions on Information Theory*, Vol. 44, No. 3, pp 1087-1096, 1998.
- [20] Le Boudec J.Y., Thiran P., "Network Calculus Viewed as a Min-Plus System Theory Applied to Telecommunications", Technical report SSC/1998/016, available from <http://icawww.epfl.ch>.
- [21] 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.
- [22] Sariowan H., "A Service-Curve Approach to Performance Guarantees in Integrated Service Networks", Ph.D. Dissertation, UC San Diego, 1996.
- [23] Stiliadis D., Varma A., "A General Methodology for Designing Efficient Traffic Scheduling and Shaping Algorithms", *Proceedings of the Sixteenth IEEE INFOCOM Conference, 1997*, Vol. 1, pp. 326-335, April

- 1997.
- [24] Stoica I, Zhang H, “Providing Guaranteed Services without Per Flow Management”, *Proceedings of the ACM SIGCOMM Conference*, pp. 81-94, August 1999, Cambridge United States.
  - [25] Suri S., Varghese G., “Leap-Forward Virtual Clock: A New Fair Queuing Scheme with Guaranteed Delays and Throughput Fairness”, *Proceedings of the Sixteenth IEEE INFOCOM Conference*, Vol. 2, 557-565, April 1997.
  - [26] Tananbaum A., *Computer Networks*, third edition, Prentice Hall, 1996
  - [27] Xie G., Lam S., “Delay Guarantee of Virtual Clock Server”, *IEEE/ACM Transactions on Networking*, Vol. 3, No. 6, pp. 683-689, Dec. 1995.
  - [28] Zhang L., “Virtual Clock: A New Traffic Control Algorithm for Packet-Switching Networks”, *ACM Transactions on Computer Systems*, Vol. 9, No. 2, May 1991.
  - [29] Zhang H., Keshav S., “Comparison of Rate-Based Service Disciplines”, *Proceedings of the ACM SIGCOMM Conference*, pp. 113-121, Sept. 1991, Zurich Switzerland.
  - [30] Zhang H., “Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks”, *Proceedings of the IEEE*, Vol. 83, No. 10, pp. 1374-1396, Oct. 1995.
  - [31] 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, pp.1096-1105, Feb.-April, 1994.