

# INTRA-FLOW FAIRNESS IN WORK-CONSERVING FLOW AGGREGATION

Jorge A. Cobb    Zhe Xu  
Department of Computer Science  
The University of Texas at Dallas  
Richardson, TX 75083-0688  
{cobb, xuzhe}@utdallas.edu

## ABSTRACT

Currently, the Internet provides mainly best effort services. As multimedia applications emerge, a demand for quality of service guarantees follows. Packet-scheduling protocols, such as Virtual Clock and Weighted Fair Queuing, were designed to fulfill this demand by reserving network resources for each individual packet stream (also known as flows). This leads to a scalability problem in the core of the network, because routers are expected to maintain state information for each individual flow. One method to mitigate this problem is to merge together into a single flow those flows following a common path through the core. This drastically reduces the number of flows managed by a router.

Initial flow aggregation methods were non-work conserving. This prevents flows from taking advantage of unused bandwidth, that is, from exceeding their reserved rate when the network is lightly loaded. Recently, this was remedied with the introduction of work-conserving flow aggregation methods. However, these methods are unfair, i.e., a flow that takes advantage of unused bandwidth may temporarily be denied service from the network.

In this paper, we propose a flow aggregation method that is fair, and thus, flows are not punished if they exceed their reserved rate. We show that this new method still conserves the end-to-end delay guarantee of work-conserving flow aggregation. Moreover, because of its fairness, it also provides an end-to-end throughput guarantee.

## KEY WORDS

Quality of Service, Real-Time Scheduling, Flow Aggregation.

## 1 Introduction

While the best effort service provided by the Internet suffices for traditional applications, such as email and web browsing, real-time applications such as interactive audio and video require Quality of Service (QoS) guarantees from the network. In particular, they require the network to reserve bandwidth for the application and to provide a bounded end-to-end delay. To provide these QoS guarantees, a broad array of packet scheduling protocols have been developed, such as Virtual Clock (VC) [19, 21] and Weighted Fair Queuing (WFQ). These and others belong to

a large family of scheduling protocols, known as Guaranteed Rate (GR) schedulers [10], which provide a low end-to-end delay bound.

Let us denote by *flow* the sequence of packets generated by a single connection of a real-time application. To ensure packets are delivered on time, GR schedulers must reserve bandwidth along the end-to-end path of each flow. Therefore, routers must process signaling messages and maintain state for each individual flow. This has raised a scalability concern in the core of the network, due to the large number of flows traversing each router. To deal with this problem, two general methodologies have been proposed: dynamic packet state (DPS) [17, 11], and flow aggregation [3].

In DPS, instead of maintaining any per-flow information in routers, scheduling information is carried in the packet header. Then, each router updates the packet header at every hop (based solely on information in the header and local factors such as the arrival time of the packet). Packets from all flows are maintained in a single queue sorted by scheduling information in the header. However, this approach requires clocks to be synchronized, or links to have non-variable delay.

In this paper, we focus instead on flow aggregation [3], which does not have the above requirements on clocks and links. In addition, a lower end-to-end delay is obtained using flow aggregation [4]. In flow aggregation, multiple flows that following a common path are merged into a single aggregate flow. Routers after the point of aggregation are aware of the aggregate flow, but not of the individual flows, reducing significantly the amount of state that the router must maintain.

A scheduling protocol is *work-conserving* if it never allows its output channel to be idle when the packet queue is not empty, i.e., no time is wasted on the output channel. Initial flow aggregation methods were not work-conserving [3]. In particular, aggregate flows could not exceed their reserved rate. This prevents flows from taking advantage of unused bandwidth during times when the network is lightly loaded.

Recently, this was remedied with the introduction of work-conserving flow aggregation methods [18, 7]. However, each of these methods has its drawbacks. In [18], flows are leaky-bucket constrained, which in effect defeats the purpose of work-conservation. In [7], the method is un-

fair, in a similar way that some scheduling protocols, such as Virtual Clock, are unfair. That is, if a flow exceeds its reserved rate to take advantage of unused network bandwidth, then at a later time it may be denied service by the network in proportion to the excess bandwidth it consumed.

In this paper, we propose a flow aggregation method that is both work-conserving and fair. In consequence, flows are not punished if they exceed their reserved rate. We show that this new method still conserves the end-to-end delay guarantee of work-conserving flow aggregation. Moreover, because of its fairness, it also provides an end-to-end throughput guarantee.

Due to space restrictions, proofs are deferred to [20].

## 2 Background

### 2.1 Quality of Service Model

We base our QoS model on the models of [10, 5]. Each output channel of a computer is equipped with a scheduler, whose function is to schedule packets in an order which guarantees QoS to each input flow. We say a packet exits/arrives from/to a scheduler when the last bit of the packet is transmitted/received by the scheduler. For simplicity, we assume a propagation delay of zero.

Each flow is characterized by the bit-rate it reserves from the network, and its maximum packet size. We adopt the following notation for each flow  $f$  and each scheduler  $s$  along the path of  $f$ .

- $C^s$ : output channel bit rate of  $s$
- $R_f$ : bit rate reserved for flow  $f$
- $f.i$ :  $i^{\text{th}}$  packet of flow  $f$
- $A_{f,i}^s$ : arrival time of  $f.i$  at  $s$
- $E_{f,i}^s$ : exit time of  $f.i$  from  $s$
- $L_{f,i}$ : length of packet  $f.i$
- $L_f^{\text{max}}$ : maximum packet size of  $f$
- $L_{\text{max}}^s$ : maximum packet size at  $s$

We define the *guaranteed-rate (GR) finishing time*<sup>1</sup>  $F_{f,i}^s$  of packet  $f.i$  at scheduler  $s$  as follows. Assume  $s$  were to forward the packets of  $f$  at a constant bit-rate of exactly  $R_f$  bits/sec.. Then,  $F_{f,i}^s$  is the time at which the last bit of  $f.i$  is forwarded by  $s$ . More formally,

$$\begin{aligned} F_{f,1}^s &= A_{f,1}^s + L_{f,1}/R_f \\ F_{f,i}^s &= \max(A_{f,i}^s, F_{f,(i-1)}^s) + L_{f,i}/R_f, \quad \forall i, i > 1 \end{aligned} \quad (1)$$

A scheduler  $s$  is a *GR scheduler* [10] if each packet exits by the time indicated in its GR finishing time. That is, a GR scheduler provides to each flow  $f$  a guaranteed rate of  $R_f$ . More formally, a scheduler  $s$  is a GR scheduler iff, for every input flow  $f$  and every  $i, i \geq 1$ ,

$$E_{f,i}^s \leq F_{f,i}^s + \beta^s \quad (2)$$

<sup>1</sup>This value is also known as the guaranteed-rate-clock value in [10], and it is also equal to the timestamp assigned by a virtual-clock scheduler [19].

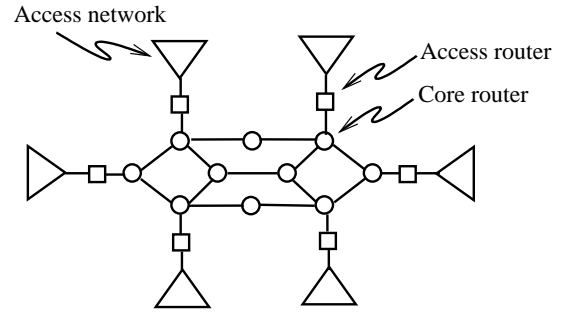


Figure 1. Core Network

for some constant  $\beta^s$ . We refer to  $\beta^s$  as the scheduling constant of  $s^2$ .

GR schedulers have the nice property that if a flow traverses a series of these schedulers, the end-to-end delay bound is determined by the GR finishing time at the first scheduler (plus a small per-hop delay). In a sense, *the series of schedulers, as a whole, guarantees a rate of  $R_f$  to each flow  $f$* . Their end-to-end delay bound is as follows [10, 5, 16]. Let  $s^1, s^2, \dots, s^k$  be a sequence of  $k$  GR schedulers traversed by flow  $f$ . For all  $i$ ,

$$F_{f,i}^k \leq F_{f,i}^{s^1} + (k-1) \cdot \frac{L_f^{\text{max}}}{R_f} + \sum_{x=1}^{k-1} \beta^{s^x} \quad (3)$$

$$E_{f,i}^k \leq F_{f,i}^{s^1} + (k-1) \cdot \frac{L_f^{\text{max}}}{R_f} + \sum_{x=1}^k \beta^{s^x} \quad (4)$$

### 2.2 Flow Aggregation

To reduce the amount of state managed by each router, multiple flows can be combined together to form a single aggregate flow [3, 4, 9, 14].

An *aggregate* flow  $g$  is obtained by merging, at a single point in the network, the packets of multiple flows  $f^1, f^2, \dots, f^n$ . In this case,  $f^1, f^2, \dots, f^n$  are said to be the *constituents* 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*. A *separator* is a process that receives as input an aggregate flow, and produces as output the set of constituent flows composing the aggregate flow.

The reserved rate,  $R_g$ , of aggregate flow  $g$  is at least the sum of the reserved rates of the constituent flows of  $g$ . Schedulers after the aggregation point are not aware of the constituents of an aggregate flow. At a later point in the network, the aggregate flow is separated again into its constituent flows.

We consider aggregation over a core network model, shown in Figure 1.<sup>3</sup> It consists of a network of core routers

<sup>2</sup>Typically  $\beta^s = L_{\text{max}}^s/C^s$ , such as the case of classical Virtual-Clock and Weighted-Fair Queuing protocols [13, 19].

<sup>3</sup>This network is similar to a SCORE network in [17, 11]. However,



Figure 2. Scheduler path followed by simple flow  $f$

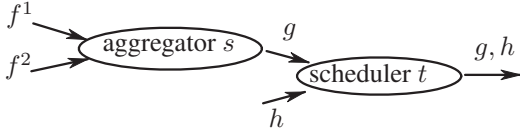


Figure 3. Need for fair aggregation

surrounded by access networks. Ingress/egress routers interface the core routers to the access networks.

We assume that all flows that traverse the core network via the same pair of ingress and egress routers are aggregated together at the ingress router. Therefore, the output of each ingress router are  $N - 1$  aggregate flows, one destined to each egress router, and the input to each egress router are  $N - 1$  aggregate flows, one from each ingress router. Egress routers separate each aggregate flow and the resulting simple flows continue into the access networks.

Due to our network model, we will address a single level of aggregation. This is illustrated in Figure 2. A flow  $f$  enters an aggregator  $s$ , and it is aggregated with other flows (not shown) to produce an aggregate flow  $g$ . Flow  $g$  traverses multiple GR schedulers,  $t^1, \dots, t^k$ , before reaching a separator  $u$ , after which the original flow  $f$  is restored. We say that  $f$  is a *simple* flow because it is not an aggregate of other flows, i.e., it has no constituents.

In our network model, aggregators and separators are internal to routers, i.e., their output remains within the router. Thus, their output channel capacity is, in principle, unbounded.

### 2.3 Work-Conserving Flow Aggregation

Aggregating flows together must be done with care if an end-to-end delay bound for each simple flow similar to the bound in (4) above is to be preserved. Consider as an example Figure 3, and assume there is a buildup of packets in the queue of flow  $g$  at scheduler  $t^k$ . Assume this buildup occurs because  $f^2$  is sending packets at a very high rate, while  $f^1$  has not generated packets for some time. When packets of  $f^1$  arrive, they will be placed at the end of the queue of  $g$  at  $t$ , which causes them to suffer a large queuing delay.

One way to address this problem is to ensure that aggregator  $s$  forwards the packets of  $g$  at a rate never greater

dynamic-packet state, and not flow aggregation, is assumed in SCORE networks.

<sup>4</sup>Recall that the output of an aggregator is internal to the router, and hence,  $s$  can forward the arriving packets of  $f^1$  and  $f^2$  to  $t$  with zero delay.

than  $R_g$  [3]. However, this causes the aggregator to be non-work conserving, which is undesirable. The other approach is to allow the packets from  $f^1$  to “jump” over packets from  $f^2$  in the queue of  $g$  at  $t$  [7]. To do this, packets from  $f^1$  and  $f^2$  must be given a priority label, and  $t$  maintains the queue of  $g$  sorted by this label.

The choice of label is important to ensure the packets of  $f^1$  and  $f^2$  are merged correctly at  $t$ . In [7], we choose to label each packet with its GR finishing time at the aggregator. I.e., each packet  $f^1.i$  is tagged with the value  $F_{f^1.i}^s$ . In this way, if  $f^2$  exceeds its reserved rate, and if  $f^1$  abides by its reserved rate, then the packets of  $f^2$  will have labels that are much larger than those of  $f^1$ , allowing the packets of  $f^1$  to be serviced on time at  $t$ .

The above, however, is not enough to provide a low delay bound to each simple flow. Consider again the case when packets of  $g$  have arrived at  $t$  at a rate greater than  $R_g$  (because the aggregator is work conserving and  $f^2$  exceeded its rate). Even though  $t$  is a GR scheduler, it can temporarily deny service to  $g$  (and hence to  $f^1$  and  $f^2$ ) without violating Equation (2) because  $g$  exceeded its rate  $R_g$ . This would be the case if, for example,  $t$  implements the Virtual Clock protocol, which has been known to have this behavior [19][21].

To solve this problem, only “fair” GR schedulers, e.g. WFQ [13] and WF<sup>2</sup>Q [2], where the amount of time a flow is not served is bounded by a small amount. This bound is known as the Worst-Case Fair Index (WFI), as defined in [2].

**Definition 1** A scheduler  $t$  provides to an input flow  $g$  a Worst-Case Fair Index (WFI) of  $W_g^t$  if for all time  $\tau$ , the delay of a packet arriving at time  $\tau$  is upper-bounded by

$$\frac{Q_g^t(\tau)}{R_g} + W_g^t$$

where  $Q_g^t(\tau)$  is the queue size of  $g$  at time  $\tau$ .

In this manner,  $g$  will be served at a rate of at least  $R_g$  at all times, except for an additional delay of at most  $W_g^t$ . This ensures the following end-to-end delay of  $f$  in Figure 2.

$$E_{f.i}^{t^k} \leq F_{f.i}^s + \sum_{x=1}^k W_g^{t^x} + (k-1) \frac{L_g^{max}}{R_g} \quad (5)$$

In [1, 2], it is shown that the WFI of a WF<sup>2</sup>Q scheduler  $t$  is bounded as follows.

$$W_g^t \leq \frac{L_g^{max}}{R_g} + \frac{L_{max}^t}{C^t}$$

We thus have that the end-to-end delay has a per-hop increase of approximately  $2 \cdot \frac{L_g^{max}}{R_g}$ . Since, in general,  $R_g \gg R_f$ , then the per-hop increase for work-conserving flow aggregation is significantly smaller than that without flow aggregation given earlier in Relation (4).

### 3 Fair Work-Conserving Aggregation

We now focus on the topic of the paper: fairness of simple flows. Even though we have chosen each scheduler in Figure 2 to be “fair” (i.e., with a small worst-case fair index), and even though a low delay bound is provided to each simple flow in Relation (5), fairness to each simple flow is not assured, as we discuss next.

#### 3.1 Unfairness of Simple Flows

Consider again Figure 3. Above, we focused on flow  $f^1$ , which is slower than  $f^2$ , and argued what is necessary to ensure the packets of  $f^1$  are not delayed excessively by the packets of the more aggressive  $f^2$ . Below, we focus our attention on  $f^2$ , and ensure that it is treated “fairly” even though it temporarily exceeded its reserved rate.

The behavior of  $f^2$  should not be considered malicious, because rate-adaptive applications perform in this way. They estimate whether the network has additional unused bandwidth through various methods, such as noticing that the end-to-end delay is small, and then increase their rate to take advantage of this bandwidth. When there is no more unused bandwidth, the application will reduce its rate down to what it originally reserved.

If we use the GR finishing time  $F$  as the label of each packet, then  $f^2$  can later be “punished” as follows. Assume  $f^1$  increases its packet rate, and this increase is greater than  $R_{f^1}$ . Because  $f^1$  has been idle or slow for some time, its packet labels will be significantly smaller than those of  $f^2$ . This may temporarily cause the packets of  $f^2$  to be forwarded from  $t$  at a rate less than  $R_{f^2}$ . Actually, if  $f^1$  exceeds the aggregate rate  $R_g$ , then  $f^1$  can temporarily prevent *any* packet of  $f^2$  from departing  $t$ .

This type of unfairness is typical in some scheduling protocols, for example, the Virtual Clock (VC) protocol [19][21]. In VC, packets are labeled with their GR finishing time,  $F_{f,i}$ , and they are forwarded in order of their label. Our work-conserving aggregator is similar to VC, i.e., it assigns the same label. It differs in that aggregators are internal to the router, and thus, packets have no delay at the aggregator (queuing delays occur only at the schedulers along the path).

The unfairness in VC (and our aggregator) occurs because computing the labels for a flow is independent of the packet arrivals of other flows. Other scheduling protocols, such as weighted-fair-queuing (WFQ) [13], are fair<sup>5</sup>. They accomplish this fairness by assigning labels to flows in such a way that labels from different flows are always close to each other. While doing this, they remain as GR schedulers, i.e., at all times they forward packets from each flow  $f$  at a rate of at least  $R_f$ .

We thus explore whether end-to-end delay bounds similar to those of Relations (4) and (5) can be obtained when the aggregator assigns packet labels in a similar way

as fair GR schedulers, such as WFQ. For the result to be as general as possible, we choose a broad family of GR schedulers, known as Rate-Proportional Schedulers [15]. This family includes both unfair protocols (such as VC), and fair protocols (such as WFQ), and also protocols whose fairness approximates that of WFQ, such as Time-Shift Scheduling [6], and many others.

#### 3.2 Rate-Proportional Schedulers

Rate-Proportional (RP) Schedulers [15] are based on the notion of a virtual server. The virtual server is not a real scheduler, it is used only for reference. Both the real scheduler and the virtual server have the same input flows and the same output channel capacity.

The difference between the real scheduler and the virtual server is as follows. Real schedulers are packet-oriented, i.e., they cannot interrupt the transmission of a packet, and must wait for the packet transmission to end before the transmission of the next packet may begin. Contrary to a real scheduler, the virtual server is “fluid”, that is, it is able to concurrently forward an arbitrarily small amount of data from any number of flows.

Associated with the virtual server is the notion of the scheduler’s virtual time  $V$  and also the virtual time  $v_f$  of each input flow  $f$ .  $V$  is a measure of the progress the virtual server, and it increases as bits are forwarded, while  $v_f$  increases only when bits from  $f$  are forwarded. Due to space limitation the reader is referred to [15] for more details on the virtual server.

Recall that the real RP scheduler is packet-oriented, and thus it cannot forward bits in the same way as the virtual server. Instead, it assigns to each packet a label with the value of  $V$  when the last bit of the packet is forwarded by the virtual server. Packets are forwarded to the output channel in the order of increasing label values.

The label  $T_{f,i}$  of packet  $f.i$  is computed as follows.

$$\begin{aligned} T_{f,1} &= V(A_{f,1}) + L_{f,1}/R_f \\ T_{f,i} &= \max(V(A_{f,i}), T_{f,(i-1)}) + L_{f,i}/R_f, \quad \forall i, i > 1 \end{aligned} \quad (6)$$

where  $V(A_{f,i})$  is the value of  $V$  when packet  $f.i$  arrives to the scheduler.

Note the similarity between Equation 6 and Equation 1. In effect, if  $V$  is simply the current real time, then  $T_{f,i} = F_{f,i}$ , and the RP scheduler is actually the original Virtual-Clock scheduler [19][21]. On the other hand, if  $V$  is derived from a virtual server that forwards bits from all flows concurrently, we have a Weighted-Fair-Queuing Scheduler [13].

#### 3.3 RP Aggregators

Consider again Figure 2. We next consider an aggregator whose behavior is simply as follows. Choose a particular virtual server whose input flows are the same as those of the

<sup>5</sup>At the expense of a more complex labeling method

aggregator, and let the output channel capacity of the virtual server be equal to the reserved rate  $R_g$  of the aggregate flow. For every packet received, the aggregator computes the virtual time at which the packet would exit the virtual server. Then, the packet is forwarded immediately to the first scheduler.

Note that the above aggregator is similar to RP schedulers in that it assigns labels to packets in the same manner as an RP scheduler. Thus, we refer to it as an *RP aggregator*. It differs from an RP scheduler, however, in its capacity. An RP scheduler has finite output channel capacity; we chose the specific value of  $R_g$  for our purposes, but it could be any capacity. On the other hand, the aggregator, being internal to the router, has infinite capacity and introduces no delay.

**Theorem 1** *Consider the scenario in Figure 2. Let the aggregator be an RP aggregator. That is, it assigns to each packet  $f.i$  a label  $T_{f,i}$  corresponding the virtual time when the packet finishes service in a virtual scheduler. Then, the end-to-end delay of flow  $f$  is as follows.*

$$E_{f,i}^{t_k} \leq \text{real-time}(T_{f,i}^s) + \sum_{x=1}^k W_g^{t_x} + (k-1) \frac{L_g^{max}}{R_g} \quad (7)$$

where  $\text{real-time}(T_{f,i}^s)$  is the real time when the virtual time  $V$  reaches the value  $T_{f,i}^s$ .

The above exit bound makes no assumption about the behavior of the schedulers along the path of  $f$ , other than providing a worst-case-fair index to each input flow. Equation (5) now becomes a corollary of Theorem 1, because  $T_{f,i} = F_{f,i}$  when the virtual scheduler chooses  $V$  to simply be equal to real time.

The difference between Equation (5) and the above theorem is the first term, i.e.,  $\text{real-time}(T_{f,i}^s)$ . Consider again Figure 3, where  $f^1$  is generating packets slowly, while  $f^2$  is exceeding its reserved rate. If the aggregator assigns the same labels as a fair protocol, such as Weighted-Fair-Queuing and its variants [13, 2], then, as long as the combined packet rate of  $f^1$  and  $f^2$  does not exceed  $R_g$ , flow  $f^2$  will not be punished for taking advantage of the unused rate of  $f^1$ .

Although the above does not appear to be a significant advantage, consider that in practice each aggregate flow in a core router may consist of hundreds of flows. Many of these flows may reserve a rate greater than their average rate in order to quickly transmit bursts of data. Thus,  $R_g$  is significantly greater than  $R_{f^2}$ , allowing  $f^2$  to significantly exceed its rate without being penalized. We more formally discuss this below.

### 3.4 Guaranteed Throughput

Besides end-to-end delay guarantee, the network could also offer a throughput guarantee. When a flow  $f$  reserves a certain rate  $R_f$  from the network, and sends bits at a rate

higher than  $R_f$ , it would expect bits to go through the whole network at the rate at least  $R_f$ . However, due to the scheduling algorithms running at each router, and each router only serves all flows one packet at a time, flow  $f$  may receive no service at all for a certain period of time, say  $\tau$ . Let us denote by  $B_f(t_1, t_2)$  the throughput guarantee from the network for flow  $f$  during an arbitrary time interval  $(t_1, t_2)$ , then the throughput guarantee is simply defined as follows.

$$B_f(t_1, t_2) = R_f(t_2 - t_1 - \tau)^+ \quad (8)$$

where  $(x)^+ = \max(x, 0)$ .

If the network could provide an upper bound for the “no service” time  $\tau$ , that means a throughput guarantee is provided. Let us consider a worst case scenario, where two consecutive packets,  $p_1$  and  $p_2$  from flow  $f$ , traverse the whole network. The first packet  $p_1$  goes through the whole network without experiencing any unnecessary delays, i.e., each router  $t$  serves this packet at its capacity,  $C_{out}^t$ . However, the second packet  $p_2$  is delayed for all possibilities, i.e., queuing delays, including delays within the aggregate flow  $g$  to which  $f$  belongs, and processing delays at each router. Then the difference between the exit times of these two packets would be the “no service” time upper bound. We only consider the case of all packets having the same size in this paper. Or formally, we have the following theorem.

**Theorem 2** *Assume packets have an equal size  $L$ . Let flow  $f$  be aggregated into flow  $g$  at aggregator  $s$ , and the reserved rate for  $g$  is  $R_g$ . Assume  $W_f^s$  is the WFI of the RP algorithm implemented in the aggregator, and each scheduler  $t_x$  of  $k$  schedulers has a WFI value of  $\frac{L}{R_g} + \frac{L}{C_{out}^x}$ . If the aggregate flow is served at a rate of at most  $R_g$ , then the network provides a throughput guarantee to flow  $f$  as follows,*

$$B_f(t_1, t_2) = R_f(t_2 - t_1 - \tau)^+$$

where

$$\tau = W_f^s + (2 \cdot k + 1) \cdot \frac{L}{R_g} \quad (9)$$

This result is only meaningful if the aggregator is fair, i.e., its WFI value  $W_f^s$  is limited. Of course, flow  $f$  should also be sending packets into the network at least at rate  $R_f$ .

The above upper bound of  $\tau$  is inversely proportional to the reserved rate of the aggregate flow, which is improved from the case of without flow aggregation. The upper bound of  $\tau$  is shown to be inversely proportional to the reserved rate of the individual flow in [12], where no flow aggregation is implemented.

However, the assumption of Theorem 2 also implies that the aggregator is non-work-conserving. When an individual flow  $f$  is served at a rate higher than the aggregate flow reserved rate  $R_g$ , tag values of its packets grow higher than the system virtual time of the aggregator algorithm. If packets from other flows have tag values close to the system virtual time of the aggregator algorithm, flow  $f$  may be

punished. Next, we discuss the throughput guarantee in work-conserving flow aggregation, in which case a simple flow  $f$  can exceed the aggregate rate  $R_g$ .

## 4 Increasing Throughput via Timestamp Reuse

### 4.1 Background

Assume that constituent flow  $f^2$  wishes to increase its packet rate beyond aggregate rate  $R_g$ . When flow  $f^2$  exceeds this unused bandwidth, the labels of its packets will begin to significantly exceed the virtual time  $V$  of the virtual scheduler emulated by the aggregator. This may cause unfairness against  $f^2$  at the schedulers downstream when  $f^1$  increases its packet rate.

In order for flow  $f^2$  to compete with flow  $f^1$ , its labels must be reduced. This cannot be done for packets already in transit. However, new packets from  $f^2$  may be given a label smaller than that indicated by Equation (6). In particular, the labels of earlier packets could be *reused*.

Label reuse was first introduced for dynamic packet state (DPS) networks [12]. In [8], we adapted label reuse for the case of work-conserving flow aggregation networks. Here, we adapt it for the more general approach of fair work-conserving flow aggregation, in which an aggregator can assign labels by emulating any of the virtual schedulers in the family of RP scheduling protocols [15].

Label reuse is performed as follows. If an earlier packet from  $f^2$  has exited the core network, its label may be reused. This is provided that this reuse does not interfere with the satisfaction of the deadlines of other packets (both from  $f^2$  and from other flows constituting  $g$ ). To learn that a packet has exited the core network, the egress router could send an acknowledgment to the ingress router indicating the departure of the packet.

### 4.2 Aggregate-Flow Labels

We assume each packet  $f.i$  of a constituent flow  $f$ , in addition to receiving a label  $T_{f.i}$  as defined in Section 3, is also assigned an *aggregate label*  $T_{g.j}$ , where  $f.i = g.j$ . This label is logical, not physical. That is, it is not actually carried in the packet; it is only used to simplify our reasoning below.

Recall that label  $T_{f.i}$  is the same label  $f.i$  would receive if it were the input to an RP scheduler with output channel capacity of  $R_g$ . Label  $T_{g.j}$  is simply defined to be the exit time of  $g.j$ , i.e., of  $f.i$ , from this reference RP scheduler.

We define  $I_{g.j}$  the time at which  $g.j$  begins to be forwarded by the reference RP scheduler, i.e.,

$$T_{g.j} = I_{g.j} + \frac{L}{R_g}$$

Labels from  $g$  may be reused, that is, multiple packets may be given the same aggregate label  $T_{g.j}$ . If this is the

case, then instead of multiple packets of  $g$  being sent, we consider this as the same packet  $g.j$  being sent multiple times. Otherwise, the definition of index  $j$  of flow  $g$  is not sound.

**Lemma 1** Consider Figure 2, and assume each packet of flow  $g$  is given an aggregate timestamp  $T_{g.j}$  as defined above. Any packet  $g.j$  may be resent if desired when the following two conditions hold:

- No other packet of  $g$  currently in the core network is using label  $T_{g.j}$ .
- $T_{g.j} \geq \text{clock} + \frac{L}{R_g}$ , where *clock* is the current real time.

Under these conditions, packets labeled with  $T_{g.j}$  have the following exit bound.

$$E_{g.j}^{t^k} \leq T_{g.j} + \sum_{x=1}^k W_g^{t^x} + (k-1) \frac{L}{R_g}$$

From what is known of RP schedulers [15],

$$T_{g.j} \leq \text{real-time}(T_{f.i}). \quad (10)$$

Hence, the above exit bound is the same as in Theorem 1.

### 4.3 Constituent-Flow Exit Bounds

Recall that  $T_g$  is only a logical label, and schedulers along the path of a constituent flow  $f$  actually sort packets by the constituent flow label  $T_f$ . Consider any two packets  $f^1.x$  and  $f^2.y$  from two constituent flows  $f^1$  and  $f^2$ . Assume  $f^1.x = g.j$  and  $f^2.y = g.k$ . If the following holds,

$$T_{f^1.x} \leq T_{f^2.y} \equiv T_{g.j} \leq T_{g.k}$$

then, sorting by aggregate label is the same as sorting by constituent flow label.

However, the above is not always true. Without loss of generality, assume instead that

$$T_{f^1.x} \leq T_{f^2.y} \wedge T_{g.j} > T_{g.k} \quad (11)$$

If these two packets are in the queue of a scheduler at the same time, then the scheduler will reorder them differently than their aggregate-flow label. Recall, however, that the aggregate-flow label is only a logical label. So in this case, we *exchange* the logical labels of these two packets. Since both packets are already in the queue and their lengths are the same, this does not affect the result of Lemma 1.

Packet  $f^1.x$  fares well in this exchange, because its new logical label  $T_{g.k}$  is smaller than before. However, packet  $f^2.y$  receives a new logical label  $T_{g.j}$  that is larger than its previous label. This will affect its exit bound from Lemma 1. This is actually of no consequence, because combining Equations (11) and (10), we obtain

$$T_{g.j} \leq \text{real-time}(T_{f^1.x}) \leq \text{real-time}(T_{f^2.y})$$

The desired exit bound is still preserved.

Finally, labels of constituent flows are reused, not logical labels. Therefore, the label reuse requirement  $T_{g,j} \geq \text{clock} + \frac{L}{R_g}$  of Lemma 1 must be translated to a requirement at least as strong but in terms of constituent flow labels. It can be shown [20] that needed requirement is

$$T_{f,i} \geq \frac{L}{R_g} + \frac{L}{R_{f,min}}$$

where  $R_{f,min}$  is the minimum of the reserved rates of all the constituent flows of  $g$ . We thus conclude with the following.

**Theorem 3** *Consider Figure 2, and assume the aggregator is an RP aggregator. Assume also that the aggregator can reuse a label  $T_{f,i}$  of flow  $f$  under the following conditions:*

- No other packet of  $f$  currently in the core network is using label  $T_{f,i}$ .
- $T_{f,i} \geq \frac{L}{R_g} + \frac{L}{R_{f,min}}$ , where *clock* is the current real time.

*Under these conditions, packets of  $f$  labeled with  $T_{f,i}$  have the following exit bound.*

$$E_{f,i}^{t_k} \leq \text{real-time}(T_{f,i}) + \sum_{x=1}^k W_g^{t_x} + (k-1) \frac{L}{R_g} \quad (12)$$

## 5 Guaranteed Throughput

If the above mentioned conditions are met, a simple flow can actually transmit packets at a rate greater than the aggregate flow reserved rate, and the network is still able to provide a throughput guarantee as follows.

**Theorem 4** *In a work-conserving flow aggregation network with tag-reuse technique incorporated, if a simple flow  $f$ , with reserved rate  $R_f$ , is aggregated into  $g$ , and  $g$  traverses fair schedulers  $t_1, t_2, \dots, t_k$ , and a scheduler  $t_x$  has the WFI value of  $W_g^{t_x}$ , the network guarantees a throughput to a simple flow stated in Equation (8), where  $\tau$  is*

$$\tau = D + \frac{L}{R_{min}} + (2 \cdot k) \cdot \frac{L}{R_g} + \sum_{x=1}^k \frac{L}{C^{t_x}} \quad (13)$$

*where  $D$  is the time needed for the acknowledgment to travel from the egress router to the ingress router, and  $R_{min}$  is the smallest reserved rate among the set of flows that share any of the schedulers with  $f$ , hence with aggregate flow  $g$ .*

Theorem 4 shows that the “no service” time is bounded from above. Intuitively, the upper limit includes two parts.  $D$  is the time for the acknowledgment to come to the ingress router so that the aggregator knows the label

might be eligible for reuse. The second part is the maximum delay a packet would experience through the network. If the time passed by the length of  $\tau$ , one of two things would happen.

- Label values of other flows have grown as large as that of the flow under consideration.
- The flow under consideration has a reusable label that is as small as those of other flows.

Again, we also need the assumption that the flow under consideration is sending packets into the network at a rate that is at least  $R_f$ .

Similar to the case of non-work-conserving flow aggregation, the upper bound of  $\tau$  is still inversely proportional to the aggregate reserved rate.

## 6 Concluding Remarks

We added fairness into flow aggregation by using fair scheduling algorithms to label packets for the purpose of sorting within aggregate flow. With this fairness, a simple flow can take advantage of the unused bandwidth of the aggregate and have a throughput guarantee. For those users who are willing to take advantage of the unused bandwidth beyond the aggregate flow at a higher cost, a tag-reuse technique can be incorporated. In both cases, the network provides both end-to-end delay and throughput guarantee to each simple flow.

The assumption for tag-reuse in this paper is that all packets have equal size. We speculate that the delay and throughput theorems still hold even if we drop this assumption. We will further investigate the variable packet size scenario and report the result in [20].

## References

- [1] J. C. Bennett and H. Zhang, “Hierarchical packet fair queueing algorithms,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 675–689, Oct. 1997.
- [2] —, “WF2Q: worst-case fair weighted fair queueing,” in *IEEE INFOCOM Conference*, 1996.
- [3] J. Cobb, “Preserving quality of service guarantees in spite of flow aggregation,” *IEEE/ACM Transactions on Networking*, vol. 10, no. 1, pp. 43–53, Feb. 2002.
- [4] —, “Scalable quality of service across multiple domains,” *Elsevier: Computer Communications*, vol. 28, no. 18, pp. 1997–2008, Nov. 2005.
- [5] J. Cobb and M. Gouda, “Flow theory,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 661–674, Oct. 1997.
- [6] J. Cobb, M. Gouda, and A.-E. Nahas, “Time-shift scheduling: Fair scheduling of flows in high-speed

- networks,” *IEEE/ACM Transactions on Networking*, vol. 6, no. 3, pp. 274–285, June 1998.
- [7] J. Cobb and Z. Xu, “Maintaining flow isolation in work-conserving flow aggregation,” in *Proc. of the IEEE GlobeCom Conf.*, 2005.
- [8] —, “Guaranteed throughput in work-conserving flow aggregation through deadline reuse,” in *Proceedings of the ICCCN Conference*, Oct. 2006.
- [9] H. Fu and E. W. Knightly, “A simple model of real-time flow aggregation,” *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, June 2003.
- [10] P. Goyal, S. Lam, and H. Vin, “Determining end-to-end delay bounds in heterogeneous networks,” in *Proc. of the NOSSDAV Workshop*, 1995.
- [11] J. Kaur and H. M. Vin, “Core-stateless guaranteed rate scheduling algorithms,” in *Proc. of the IEEE INFOCOM Conf.*, 2001.
- [12] —, “Core stateless guaranteed throughput networks,” in *Proc. of the IEEE INFOCOM Conf.*, 2003.
- [13] A. K. J. Parekh and R. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: The multiple node case,” *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, Apr. 1993.
- [14] J. Qiu and E. W. Knightly, “Measurement-based admission control with aggregate traffic envelopes,” *IEEE/ACM Transactions on Networking*, vol. 9, no. 2, Apr. 2001.
- [15] D. Stiliadis and A. Varma, “Rate proportional servers: A design methodology for fair queuing algorithms,” *IEEE/ACM Transactions on Networking*, Apr. 1998.
- [16] D. Stiliadis and A. Varma, “Latency rate servers: a general model for analysis of traffic scheduling algorithms,” *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 611–624, 1998.
- [17] I. Stoica and H. Zhang, “Providing guaranteed services without per-flow management,” in *Proc. of the ACM SIGCOMM Conference*, 1999.
- [18] W. Sun and K. G. Shin, “Coordinated aggregate scheduling for improving end-to-end delay performance,” in *Proc. of the IEEE Workshop on Quality of Service (IWQoS)*, 2004.
- [19] G. Xie and S. Lam, “Delay guarantee of the virtual clock server,” *IEEE/ACM Transactions on Networking*, pp. 683–689, Dec. 1995.
- [20] Z. Xu, “Scalable scheduling for quality of service guarantees,” Ph.D. dissertation, The University of Texas at Dallas, Fall, 2007.
- [21] L. Zhang, “Virtual clock: A new traffic control algorithm for packet-switched networks,” *ACM Transactions on Computer Systems*, vol. 9, no. 2, pp. 101–124, May 1991.