

# Guaranteed Throughput in Work-Conserving Flow Aggregation Through Deadline Reuse

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

**Abstract**—The Internet traditionally provides best effort service to all applications. While elastic applications are satisfied by this service, inelastic applications such as interactive audio and video suffer from end-to-end delay guarantees. Although Guaranteed Rate schedulers were developed to provide such guarantees, their scalability has been a concern because they maintain per-flow state. In an effort to reduce per-flow state, two methods have been proposed: stateless core networks and flow aggregation. Stateless core networks require no per-flow state at the routers, while flow aggregation maintains state for a small number of aggregate flows. Although flow aggregation maintains more state, it provides a lower end-to-end delay bound than stateless core networks.

The original proposals of these two techniques did not provide guaranteed throughput, that is, flows could be temporarily denied service if they exceeded their reserved rates at earlier times. Recently, guaranteed throughput has been incorporated into the stateless core model through the reuse of deadlines. This is similar to the deadline reuse found in earlier stateful protocols that provide guaranteed throughput.

In this paper, we propose adding deadline reuse to flow aggregation networks. In this way, guaranteed throughput can be achieved while maintaining a lower end-to-end delay bound. In addition, we revise the deadline reuse method for stateless core networks.

## I. INTRODUCTION

While the Internet’s best effort service is well suited for traditional applications, such as file transfer, email, and web browsing, it does not provide QoS (Quality of Service) guarantees needed by emerging applications, such as interactive audio and video. In order to provide QoS guarantees to a flow of packets, the network needs to reserve necessary resources at each router that runs a proper scheduling protocol. Many scheduling protocols have been developed for this task. An earlier survey may be found in [26].

In order to support QoS in current Internet, the IETF defines two architectures, namely, Integrated Services (IntServ)[3] and Differentiated Services (DiffServ)[13], [14]. IntServ is similar to the protocols overviewed in [26], such as Virtual Clock[24] and Weighted Fair Queuing[17]. In order to support QoS guarantees, a router maintains a per-flow state for each individual flow. Given the very large number of flows expected in the core routers of the Internet, maintaining state for each individual flow has been identified as a potential scalability problem. This prompted the introduction of DiffServ, in which

a few bits classify packets into a set of “per-hop behaviors”, and routers are unaware of the individual flows. However, the utilization level and end-to-end delay of DiffServ falls short from those provided by IntServ.

The difference in the performance of DiffServ and IntServ prompted the search for protocols that maintain little or no state while providing the same guarantees as the stateful IntServ approach. Two such approaches have been proposed: stateless core networks [22], [15] and flow aggregation networks [5], [18], [23], [9].

In stateless core networks, a core network maintains no per-flow state, while the surrounding access networks do maintain per-flow state. The access networks label packets with enough information, which is dynamically updated at each hop in the core, to ensure the core can schedule the packets to meet end-to-end deadlines, which are similar to the end-to-end deadlines in Virtual Clock. In flow aggregation, multiple flows are aggregated together before entering the core network. This reduces significantly, although does not totally eliminate, per-flow state at the core. In addition, due to aggregation, an end-to-end delay bound can be obtained that is lower than the end-to-end delay bound without aggregation.

The original proposals for these two state reduction techniques did not provide guaranteed throughput. That is, assume a flow exceeded its reserved rate temporarily, perhaps in an attempt to take advantage of bandwidth that is currently not being used by other flows. Then, at a later time the network may temporarily deny service to this flow and favor other flows. I.e., the flow is “punished” unfairly for taking advantage of unused bandwidth.

Providing guaranteed throughput in a stateful scheduling protocol, such as [8], [2], [4], has been studied in the past. It is accomplished by reducing the deadlines of flows that have exceeded their reserved rate, provided in doing so the deadlines of other flows are not violated. The stateless core network model has adopted a similar approach by reusing the deadlines of packets that cross the core network earlier than expected. In this manner, stateless core networks can provide guaranteed throughput [16].

In this paper, we propose adding deadline reuse to flow aggregation networks. In this way, guaranteed throughput can be achieved while maintaining a lower end-to-end delay

bound. In addition, we revise the deadline reuse method for stateless core networks.

This paper is organized as follows. The quality of service model used in the paper is reviewed in Section II. In Section III, we overview the stateless core model, and revise the guaranteed throughput method for this model. In Section IV, we overview the flow aggregation model, and introduce guaranteed throughput to this model. Finally, concluding remarks are given in Section V. Due to space restrictions, omitted proofs will be presented in [25].

## II. QoS MODEL

In this section, we define the quality of service model provided by the network. We base our service model on the models of [12], [20].

### A. Virtual Finishing Times and Guaranteed-Rate Schedulers

A *flow* is a sequence of packets generated by an application. Each output channel of a computer is equipped with a scheduler, whose function is to schedule packets in an order which guarantees quality of service 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 the propagation delay between schedulers is zero.

Each flow is characterized by its reserved packet rate 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 $s$

Consider a scheduler  $s$  and a flow  $f$ . We define the *guaranteed-rate finish 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 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, let  $f$  be an input flow of scheduler  $s$ . Then,

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

Assume scheduler  $s$  forwards the packets of  $f$  at a rate of at least  $R_f$ . Then, each packet  $f.i$  exits from  $s$  close to  $F_{f,i}^s$ . Schedulers with this property are known as *guaranteed-rate schedulers* [12]. More formally, a scheduler  $s$  is a guaranteed-rate (GR) scheduler if and only if, for every input flow  $f$  and every  $i, i \geq 1$ ,

<sup>1</sup>The virtual finishing time is also known as the guaranteed rate clock value in [12], and it is also equal to the timestamp assigned by a virtual clock scheduler [24].

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

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

Since the virtual finishing time of a packet determines its exit time from a scheduler, then a bounded end-to-end delay requires a bounded per-hop increase in the virtual finishing time. This bound is well known (it was shown in [12] and also follows from the results in [7], [20]) and is as follows. Let  $s^1, s^2, \dots, s^k$  be a sequence of  $k$  GR schedulers traversed by flow  $f$ . For all  $i$ ,

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

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

Notice that the above upper bound on end-to-end delay of packets from  $f$  is independent of the properties of other flows sharing the network with  $f$ . In addition, if the burstiness of  $f$  is bounded by a leaky bucket of rate  $R_f$  and bucket size  $B_f$  (i.e.,  $f$  is bounded by the envelope  $\Psi(\tau) = B_f + \tau \cdot R_f$ ), then  $f$  ‘‘pays at most once’’ for its burstiness. I.e., the only term in Relation (4) related to the burstiness of  $f$  is  $F_{f,i}^{s^1}$ .

Providing the above end-to-end bound requires each scheduler to have knowledge of each individual flow  $f$ , that is, to maintain a separate queue for the flow, and maintain scheduling state for each flow. It has been argued that maintaining per-flow state in the Internet is not scalable. Therefore, two approaches have been proposed to reduce the amount of per-flow state required: a) stateless core networks, and b) flow aggregation. Both approaches are able to provide end-to-end bounds similar to those in Relation (4) (in the case of flow aggregation, an even smaller bound). Below, we overview each of these approaches, and then investigate how fairness can be maintained across flows even though schedulers are not aware of each individual flow.

## III. STATELESS CORE NETWORKS

### A. Network Model and QoS Scheduling

The model of a stateless core network is shown in Figure 1. It consists of a core network, where each core router maintains no per-flow state. The core network is surrounded by access networks, which are attached to the core network via access routers. Per-flow state is maintained at the access networks and at the access router. Once a packet enters the core network, however, it must be scheduled without per-flow information.

The objective is to obtain an end-to-end delay bound similar to that of Relation (4). In this relation, the sequence of

<sup>2</sup>The value of  $\beta^s$  determines the type of delay guaranteed by  $s$ . If  $\beta^s > 0$  (typically  $L_{\text{max}}^s/C^s$ ), then it is *rate-dependent delay*, such as the delay bound provided by the Virtual-Clock and Weighted-Fair Queuing protocols [17], [24]. On the other hand, if  $\beta^s < 0$ , then we have *rate-independent delay*, such as the delay bound provided by the protocols in [27]. In this paper, we will focus on the former, i.e., on rate-dependent delay, where  $\beta^s > 0$ .

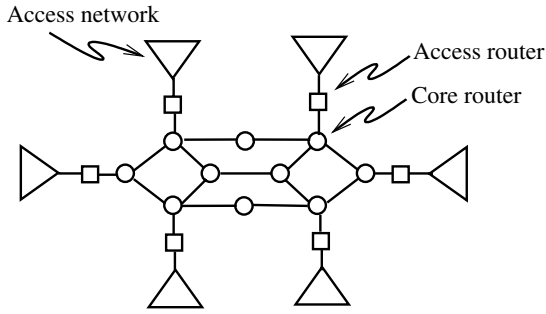


Fig. 1. Stateless Core Network

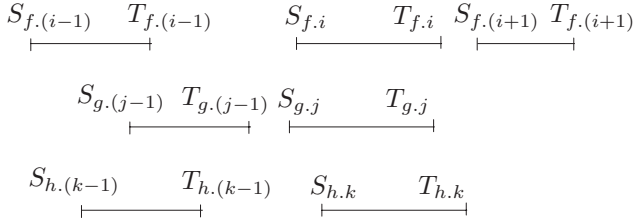


Fig. 2. Non-overlapping service intervals

schedulers correspond to the path of  $f$  from its ingress access router to its egress access router. Since no per-flow state is maintained at the core, the packet header must contain all information necessary for the core routers to schedule the packet. The approach taken in [11] is to assign to each packet  $f.i$  a *deadline*  $T_{f,i}^{s^k}$  at scheduler  $s^k$  along the path, where  $T_{f,i}^{s^k}$  is the upper bound on the virtual finishing time at scheduler  $s^k$  given in Relation (3), that is,

$$T_{f,i}^{s^k} = F_{f,i}^{s^1} + (k-1) \cdot \frac{L_f^{max}}{R_f} + \sum_{x=1}^{k-1} \beta^{s^x} \quad (5)$$

If each scheduler along the path forwards packets in order of their deadlines, then it can be shown that Relation (4) is preserved. To see this intuitively, let us define the *virtual start time*  $S_{f,i}^{s^k}$  as follows:

$$S_{f,i}^{s^k} = T_{f,i}^{s^k} - \frac{L_{f,i}}{R_f}$$

Assume that the *service interval*  $[S_{f,i}^{s^k}, T_{f,i}^{s^k}]$  of  $f.i$  does not overlap with the corresponding interval of any other packet of  $f$  (as shown in Figure 2). Intuitively,  $S_{f,i}$  is the time at which packet  $f.i$  requires to begin receiving service at a rate of  $R_f$  in order for the packet to exit by its deadline  $T_{f,i}$ . It is easy to show using the results of [10] that, if all packets arrive no later than their virtual start times, and the service intervals of a flow do not overlap, then all packets will exit by their deadlines plus  $\frac{L_f^{max}}{C}$ . The per-hop increase of  $L_f^{max}/R_f + \beta$  in the deadline of the packets of  $f$  ensures that, at the next hop, all packets arrive by their virtual start times and the service intervals do not overlap.

Note that the timestamp increases by  $L_f^{max}/R_f + \beta^{s^x}$  at each scheduler  $s^x$ . The first term is a constant, while the

second is known to scheduler  $s^x$ . Hence, these two values, along with the timestamp, can be carried in the packet and updated at every hop. In this manner, the timestamp is derived solely from the information in the packet, and no per-flow state is needed.

### B. Guaranteed Throughput

Equation (5) implies that the deadlines of a flow are independent of the deadlines of another flow (which is similar to the deadlines in the Virtual Clock protocol [24]). This allows the deadlines of a flow to grow significantly larger than those of another flow. In particular, as long as a flow generates packets without exceeding its reserved rate, then it is easy to see from (1) that a packet's deadline is close to its arrival time. However, if the reserved rate is exceeded, the deadlines will diverge from real time. In consequence, the well-known unfairness problem of Virtual Clock [5] also occurs here, as shown in the following example.

Consider two flows,  $f$  and  $g$ , that share the same path, and their combined reserved rate equals the rate of the channels along the path. Assume that, for a period of time,  $g$  generates few packets, and  $f$  takes advantage of this and exceeds its reserved rate. Notice that  $f$  is taking advantage of unused bandwidth, and should not be penalized for doing so. Thus, the deadlines of  $f$  become much greater than real time. If  $g$  then decides to exceed its reserved rate, it can temporarily deny service to  $f$  because  $g$ 's deadlines are close to real time, and hence, smaller than those of  $f$ . This continues to occur until the deadlines of  $g$  become closer to those of  $f$ .

The above denial of service to  $f$  is easily resolved in stateful protocols such as Weighted-Fair-Queuing and Time-Shift scheduling [8], [2]. However, it is more difficult to resolve in a stateless approach. In [16], Vin and Kaur propose the following method.

In order for flow  $f$  to compete with flow  $g$ , the deadlines of the packets of  $f$  must be reduced. This cannot be done for packets already in transit. However, new packets from  $f$  can be given a deadline smaller than that indicated by Equation (5). In particular, the deadlines of earlier packets could be reused. That is, if an earlier packet from  $f$  has exited the core network, its deadline may be reused. This is provided that this reuse does not interfere with the satisfaction of the deadlines of other packets (both from  $f$  and from other flows). To learn that a packet had exited the core network, the egress router could send an acknowledgment to the ingress router indicating the departure of the message.

The conditions under which a deadline  $T_{f,i}^{s^1}$  can be reused are as follows [16].

$$\forall f.j \in v : [S_{f,j}^{s^1}, T_{f,j}^{s^1}] \cap [S_{f,i}^{s^1}, T_{f,i}^{s^1}] = \emptyset \quad (6)$$

$$T_{f,i}^{s^1} - \frac{L_{f,i}}{R_f} \geq \tau \quad (7)$$

where  $v$  is the set of packets of  $f$  that are currently traversing the core network, and  $\tau$  is the current time. The first condition prevents the overlap of virtual service intervals within  $f$ .

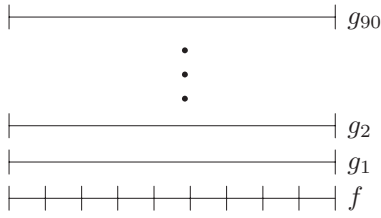


Fig. 3. Deadline Reuse Counter-Example

The second condition requires the virtual start time  $S_{f,i}^{s_1}$  to be greater than the current time. Since the virtual start time has not been reached yet, reintroducing this packet's deadline should not affect other packets. Therefore, as reported in [16], the exit bound of Relation (4) still holds in this case.

### C. Counter-Example to the Reuse Condition

Conditions (6) and (7) intuitively capture the requirements for deadline reuse. However, the second condition is not strong enough to satisfy the desired exit bound. Below, we provide a counter-example in which packets violate the exit bound in Relation (4). We then strengthen Condition (7), and show the correctness of the revised version.

Consider Figure 3. Assume we have a scheduler with 91 input flows: flow  $f$  and flows  $g_1$  through  $g_{90}$ . For simplicity, assume packet sizes are constant and equal among the flows, and the output channel rate is one packet per second. Let  $f$  reserve ten percent of the channel's bandwidth, and hence,  $\frac{L_f}{R_f} = 10$ , and let each of the  $g$  flows reserve one percent of the channel's bandwidth, and hence,  $\frac{L_g}{R_g} = 100$ . Assume ties are broken in favor of  $f$ , and that feedback about packets exiting the scheduler is received immediately.

At time zero, ten packets from flow  $f$ , and one packet from each of the 90  $g$  flows, arrive at the scheduler. According to Equation (1), the ten packets from  $f$  have deadlines 10, 20, ..., 100, while each packet of the other 90 flows has a deadline of 100. Thus, the ten packets from  $f$  exit the scheduler first, consuming 10 seconds from the clock. Note that Condition (6) is satisfied for all 10 packets because there are no packets of  $f$  available in the scheduler. However, since  $t = 10$ , Condition (7) is not satisfied for the first packet of  $f$  with deadline 10. Nonetheless, the remaining nine packets of  $f$  do satisfy Condition (7). Hence,  $f$  can reuse the deadlines of these nine packets.

We continue at time 10, when  $f$  has reintroduced deadlines 20, 30, ..., 100. All of these deadlines are smaller than those of the packets from the 90  $g$  flows. Hence, they exit the scheduler first, no later than time 19. At this point in time, note again that Condition (6) is satisfied for all nine packets of  $f$ , however, since  $t = 19$ , Condition (7) is not satisfied for the first packet of  $f$  with deadline 20. Nonetheless, the remaining eight packets of  $f$  do satisfy Condition (7). Hence,  $f$  can reuse the deadlines of these eight packets.

Hence, at time 19,  $f$  reintroduces deadlines 30, 40, ..., 100. This argument can be repeated multiple times, until  $f$

is no longer able to reuse any of the deadlines up to 100. This occurs at time  $t = 91$ . At this time, the 90 packets from the  $g$  flows, all with deadline 100, begin to be transmitted. The time at which the last of these packets exits will be 181. From Equation (4), the deadline of any of the  $g$  packets is  $F_{g,1} = A_{g,1} + \frac{L_g}{R_g} + \frac{L_{max}}{C} = 0 + 100 + 1 = 101$ , and hence the exit bound is violated by 80 seconds.

Note that, in our example, any deadline reuse by a packet from flow  $f$  will violate the delay guarantee of packets from flow  $g$  because the bandwidth of the scheduler has been fully allocated.

### D. Revised Deadline Reuse Condition

In this section, we revise the conditions for deadline reuse, and prove the correctness of the revised version. Condition (6) is strong enough and remains as is. We focus our attention on Condition (7).

In earlier work [8], we investigated fairness among flows in a stateful scheduler via deadline reuse. The scheduler assigned deadlines to packets in the same way as Virtual Clock, i.e., the deadline of a packet is its virtual finishing time,  $T_{f,i}^{s_k} = F_{f,i}^{s_k}$ . In this case, if a flow temporarily exceeds its reserved rate to take advantage of unused bandwidth, then its packet would receive deadlines much larger than the current time. On the other hand, other flows not exceeding their rate would receive deadlines close to the current time. To remedy this disparity, under certain conditions, the deadlines of all packets currently in the system would be reduced by an equal amount. This ensured fairness for new packets while preserving the scheduling order of packets currently in the queue.

For example, consider Figure 4a). Here, the virtual start times (and hence, also the deadlines) of the packets from flows,  $f$  and  $g$ , are larger than the current time  $\tau$ . If another flow,  $h$ , which has been inactive for some time, generates a new packet, then this packet would be given a virtual start time equal to the current time (and hence smaller than those of  $f$  and  $g$ ). To ensure the deadlines of  $h$  are close to those of  $f$  and  $g$ , the virtual starting times of both flows would be reduced by the amount  $(\min(S_f, S_g) - \tau)$ , as shown in Figure 4b)<sup>3</sup>.

The above reduction in deadlines ensures that the virtual start times of all queued packets are at least the current value of the clock. This can easily be checked in a stateful scheduler, but becomes a problem in a stateless scheduler. Instead, we take advantage of the following observation.

*Observation 1:* Let a packet with deadline  $T$  be chosen for transmission at a scheduler at time  $\tau$ . Then,

- 1) At time  $\tau$ , all queued packets have timestamps at least  $T$ .
- 2) If  $(T - \tau) \geq \left( \max_g : \frac{L_g^{max}}{R_g} \right)$ , then the virtual start time of every queued packet at time  $\tau$  is at least  $\tau$ .

From the above observation, if the deadline  $T$  of the packet being transmitted is large enough with respect to real-time,

<sup>3</sup>In practice, rather than reducing the deadlines of all packets, which would be too time consuming, the clock is advanced instead, yielding the same effect.

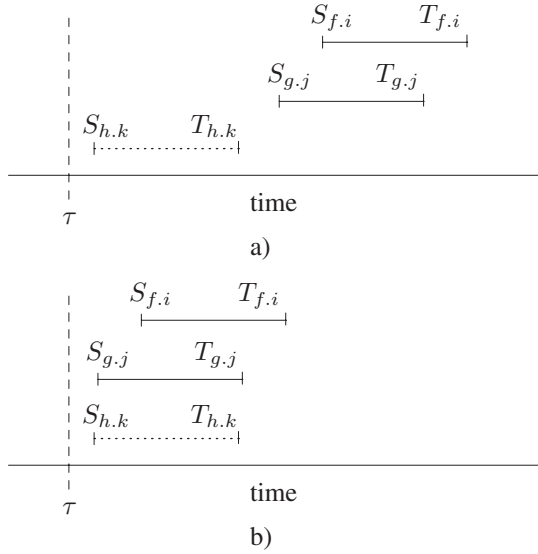


Fig. 4. Deadline Reuse in Stateful Scheduler

then all virtual start times are at least  $t$ . Hence, similar to our scheme for deadline reuse in a stateful scheduler [8], we should be able to reuse deadline  $T$  in this case.

Our revised conditions for the reuse of the deadline  $T_{f,i}^{s^1}$  of packet  $f.i$  are as follows.

$$\forall f.j \in v : [S_{f,j}^{s^1}, T_{f,j}^{s^1}] \cap [S_{f,i}^{s^1}, T_{f,i}^{s^1}] = \emptyset \quad (8)$$

$$T_{f,i}^{s^1} - \left( \max_{g \in w} : \frac{L_g^{max}}{R_g} \right) \geq \tau \quad (9)$$

where  $v$  is the set of packets of  $f$  still in the core,  $w$  is the set of flows that share any of the schedulers with  $f$ , and  $\tau$  is the current time.

We next address the correctness of these conditions. Before presenting the main theorem, we begin with a few lemmas. The proofs of the first two lemmas are straightforward and are omitted.

*Lemma 1:* Consider a set of packets from flow  $f$ . Define their deadlines at each scheduler  $s^k$  along their path as follows.

$$T_{f,i}^{s^k} = T_{f,i}^{s^1} + (k-1) \cdot \frac{L_f^{max}}{R_f} + \sum_{x=1}^{k-1} \beta^{s^x} \quad (10)$$

Assume the service intervals of these packets at the first scheduler  $s^1$  do not overlap. Then, for every scheduler  $s^k$  along the path of  $f$ , the service intervals of these packets at scheduler  $s^k$  do not overlap. ■

*Lemma 2:* If packet deadlines are assigned according to Equation (10), and if each packet of  $f$  departs each scheduler  $s^k$  by its deadline plus  $\beta^{s^k}$ , then it will arrive at scheduler  $s^{k+1}$  no later than its virtual start time at  $s^{k+1}$ . ■

*Lemma 3:* Consider a scheduler such that, at some time  $\tau$ , all queued packets have a virtual starting time at least  $\tau$ , and all packets arriving after  $\tau$  arrive no later than their virtual start time. Furthermore, for any flow, the service intervals of

its packets either queued at  $\tau$  or arriving after  $\tau$  are non-overlapping. Then, starting from time  $\tau$ , every packet of a flow  $f$  exits by its deadline plus  $\frac{L_g^{max}}{C}$ .

*Proof:* The proof is similar in structure to the well-known proof the Virtual Clock protocol [24] but differs in significant points.

Consider a packet  $f.i$  with deadline  $T_{f,i}$  that is either in the queue at time  $\tau$  or arrives after  $\tau$ , and let  $E_{f,i}$  be the exit time of  $f.i$ . Consider the latest time  $\tau'$ ,  $E_{f,i} > \tau' \geq \tau$ , that satisfies all of the following conditions.

- 1) for each queued packet whose deadline is at most  $T_{f,i}$ , the virtual start time is at least  $\tau'$ .
- 2) from  $\tau'$  until  $E_{f,i}$ , only packets with deadlines at most  $T_{f,i}$  are chosen for transmission.
- 3) the queue is never empty from  $\tau'$  until  $E_{f,i}$ .

First we must show that such  $\tau'$  exists. Notice that, from the assumptions of the lemma, the first condition holds at time  $\tau$ . Consider the latest time that the first requirement holds. Let  $x$  be this time. Thus, the first requirement does not hold after  $x$  until  $E_{f,i}$ .

We argue that if any of the remaining two conditions fail to hold at any time  $y$ , where  $E_{f,i} \geq y \geq x$ , then the first condition also holds at time  $y$ , which contradicts the definition of  $x$ , and hence, the two conditions hold from  $x$  until  $E_{f,i}$ .

Assume the queue becomes empty at time  $y$ . In this case, the first condition holds trivially because there are no packets in the queue. If a packet with deadline greater than  $T_{f,i}$  is chosen for transmission at time  $y$ , this implies that there are no packets in the queue with deadlines at most  $T_{f,i}$ , which satisfies the first condition as desired. Therefore, we conclude that  $\tau'$  exists.

From the definition of  $\tau'$ , and the assumption that all packets arrive by their virtual start time, we have that only packets with virtual start time at least  $\tau'$  and deadline at most  $T_{f,i}$  are chosen for transmission during the interval  $[\tau', E_{f,i}]$ . From the assumption that the service intervals of each flow do not overlap, the total number of bytes of any flow  $g$  with virtual start times at least  $\tau'$  and whose deadlines are at most  $T_{f,i}$  is at most

$$(T_{f,i} - \tau') \cdot R_g$$

Summing over all flows  $g$  sharing the scheduler, the bytes add to at most

$$\sum_g (T_{f,i} - \tau') \cdot R_g = (T_{f,i} - \tau') \cdot \sum_g R_g \leq (T_{f,i} - \tau') \cdot C \quad (11)$$

where  $C$  is the output channel's rate.

Recall that from  $\tau'$  onwards the queue is never empty, and therefore,  $(T_{f,i} - \tau') \cdot C$  bytes are transmitted during the interval  $[\tau', T_{f,i}]$ . From Relation (11), there is enough time to transmit all bytes with deadlines at most  $T_{f,i}$  by time  $T_{f,i}$ . However, we did not count in Relation (11) the packet currently being transmitted at time  $\tau'$  (if any), so packet  $f.i$  will exit no later than time  $T_{f,i} + \frac{L_g^{max}}{C}$ . ■

*Theorem 1:* Consider a stateless core network where deadlines are initially assigned according to Equation (5), and deadlines are reused according to Conditions (8) and (9) above. Then, the exit bound given in Relation (4) holds.

*Proof:* First note that, from Lemma 1, the only way two packets from the same flow can have overlapping service intervals is when  $s^1$  reuses deadlines. Furthermore, from Condition (8), if a deadline is reused, then its service interval does not overlap with any other packet of the same flow still in the network.

We assume that any packet that arrives at a scheduler later than its virtual start time will be dropped. Note, however, that because of Lemma 2, if a packet exits on time, then it will not be dropped at the next node. Thus, under this assumption, we will argue that all packets will exit by their deadline. This in turn implies that no packet is dropped, and the assumption can thus be removed. We next argue that an arbitrary packet  $f.i$  will exit all schedulers by its deadline plus  $\frac{L^{max}}{C}$ .

Assume packet  $f.i$  arrives at a scheduler. Assume that no packet with a reused timestamp exits the scheduler before  $f.i$  exits. Then, at time zero, the conditions of Lemma 3 hold trivially, because the queue is empty at time zero, and no reused timestamp exits before  $f.i$  exits. Hence, from the lemma,  $f.i$  will exit by its deadline plus  $\frac{L^{max}}{C}$ , as desired.

Assume however that one or more packets exit before  $f.i$  and their deadlines were later reused. From these packets, let  $q$  be the packet that departed the latest from the scheduler, and let  $\tau$  be the time it is chosen for transmission. Let  $\tau'$  be the time when  $T_q$  is reused at  $s^1$ . Hence, from Condition (9),

$$T_q \geq \tau' + \left( \max_g : \frac{L_g^{max}}{R_g} \right) \geq \tau + \left( \max_g : \frac{L_g^{max}}{R_g} \right).$$

Note that, at time  $\tau$  there are no packets in the queue with deadlines less than  $T_q$ , because  $q$  was chosen for transmission. Hence, at time  $\tau$ , all packets in the queue of the scheduler have a virtual start time greater than  $\tau$ . From the definition of  $q$ , no other timestamp will be reused before  $f.i$  exits. This satisfies the conditions of Lemma 3. Hence,  $f.i$  will exit by its deadline plus  $\frac{L^{max}}{C}$ .

This argument can be applied on a simple induction proof over the length of the path of  $f$ . Hence, all packets of  $f$  (or of any other flow) exit each scheduler on time and arrive to each scheduler on time. Hence, our assumption of dropping packets that arrive late can be removed, and the theorem holds. ■

#### IV. FLOW AGGREGATION

We next consider a different approach to reduce state in the network core: flow aggregation. That is, multiple flows can be combined together to form a single aggregate flow [5], [6], [11], [18]. We first overview earlier results on flow aggregation, and then introduce throughput guarantees in flow aggregation via deadline reuse.

##### A. Network Model and End-to-End Delay

An *aggregate* flow  $g$  is obtained by merging, at a single point in the network, the packets of multiple *constituent* flows

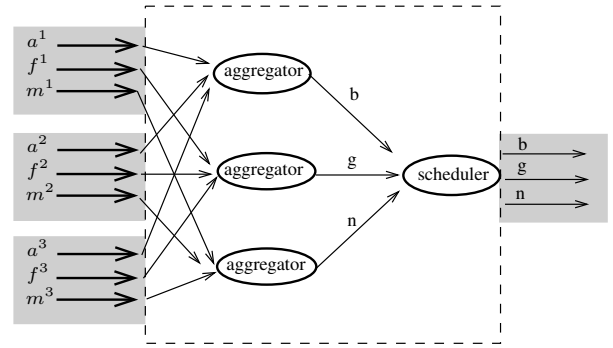


Fig. 5. Ingress router configuration

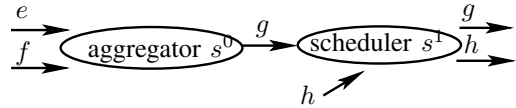


Fig. 6. Need for fair aggregation

$f^1, f^2, \dots, f^n$ . The reserved rate,  $R_g$ , of aggregate flow  $g$  is at least the sum of the reserved rates of the constituent flows of  $g$ . A scheduler whose input is the set of constituent flows and whose output is the single aggregate flow is said to be an *aggregating* scheduler. Regular schedulers after the aggregation point are not aware of the constituents of the aggregate flow. At a later point in the network, the aggregate flow is separated again into its constituent flows via a *separator*. We assume separators do not introduce any delay.

We consider aggregation over the same core network of Figure 1. We assume that all flows entering and exiting the core network via the same ingress and egress routers are aggregated together at the ingress router. This results in an internal structure of an ingress router as shown in Figure 5, where the router has three input channels and one output channel. Input flows leading to the same egress router (e.g., flows  $f^1, f^2$  and  $f^3$ ) are aggregated into a single flow (i.e.,  $g$ ) before being transmitted to the output channel, along with other aggregate flows, via a scheduler.

Note that aggregators are internal, and thus their output channel capacity is, in principle, unbounded. Hence, we assume  $C^t = \infty$  for any aggregator  $t$ .

We therefore consider a flow  $f$  that first reaches an ingress router, whose aggregator,  $s^0$ , aggregates  $f$  with other flows that exit via the same egress router. Let  $g$  be the output aggregate flow of  $s^0$ . Flow  $g$  then traverses the schedulers  $k$  of core routers,  $s^1, s^2, \dots, s^k$  before reaching its egress router, where it is separated into its constituents. Note that schedulers  $s^1, s^2, \dots, s^k$  are unaware of the constituent flow  $f$ .

We assume all schedulers, aggregating or not, are GR schedulers. However, this is not sufficient to guarantee a bounded end-to-end delay to its input flows [5]. E.g., consider Figure 6, where aggregator  $s^0$  combines flows  $e$  and  $f$  to form flow  $g$ . Assume  $e$  generates packets at a rate greater than  $R_g$ ,  $f$  is generating few packets, if any, and aggregator  $s^0$  does

not delay packets. Since scheduler  $s^1$  forwards the packets of  $g$  at a rate of  $R_g$ , the queue of  $g$  may grow arbitrarily large at  $s^1$ . Therefore, the next packet of  $f$  encounters a large number of packets ahead of it in the queue of  $g$  at  $s^1$ , and suffers an excessive delay.

To prevent the above, in [5] we restricted the output channel rate of  $s$  to be no more than the rate of its aggregate  $R_g$ . Under this condition, an end-to-end delay bound similar to that of Relation (4) is obtained. However, the per-hop delay with aggregation is based on  $L_g/R_g$ , while the per-hop delay without aggregation is based on  $L_f/R_f$ . In general,  $R_g \gg R_f$  and  $L_g \approx L_f$ , and hence, aggregation provides a much smaller per-hop delay.

Aggregating schedulers, as defined in [5], are non-work-conserving. An alternative work-conserving solution, known as Coordinated Aggregate Scheduling (CAS), was presented by Sun and Shin in [23]. Aggregators are allowed to be any GR server. In Figure 6, however, the excessive delay of packets from  $f$  due to a large queue of  $g$  at  $s^1$  is avoided as follows. At  $s^0$ , the packets of  $e$  and  $f$  are tagged with their virtual finishing times, as measured at  $s^0$ <sup>4</sup>. Then, at subsequent hops, the packets of  $g$  are maintained sorted by their tags. Thus, the queue of an aggregate flow is no longer a FIFO queue, as is the case in regular aggregation. In this manner, packets from  $f$  with a low virtual finishing time can “jump” over packets of  $g$  (more precisely, of  $e$ ) with higher virtual finishing time, and hence not be delayed.

Although work-conserving, the end-to-end delay is no longer similar to that of Relation (4). In particular, the end-to-end delay of a flow depends on the burstiness of other flows. In [9] we presented an additional work-conserving flow-aggregation method where the end-to-end delay of an individual flow remains similar to Relation (4). We refer to this method as *Coordinated Aggregation with Isolation* (CAI).

In CAI, we also take advantage of intra-flow sorting to mitigate the effect of queue buildups at intermediate schedulers, as done in CAS. However, to prevent temporary denials of service to individual flows, we focus on scheduling protocols with a small Worst-Case Fair Index [1].

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

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

where  $Q_g^s(\tau)$  is the queue of flow  $g$  at scheduler  $s$  at time  $\tau$ .

That is, the scheduler is fair in the sense that it will not spend long periods of time without servicing a flow provided the flow still has packets in the queue. Using these schedulers, the end-to-end exit bound becomes as follows.

$$E_{f,i}^{s^k} \leq F_{f,i}^{s^0} + \sum_{x=1}^k W_g^{s^x} + (k-1) \frac{L_g^{max}}{R_g}$$

<sup>4</sup>This particular timestamp remains fixed, it does not change on a per-hop basis. It is simply used to determine the relative order of the packets of  $e$  and  $f$ .

There is a whole family of schedulers, called Shaped Rate Proportional (SRP) schedulers [21], [19], with a low WFI. WF<sup>2</sup>Q is its best known family member. For these protocols,

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

Hence,

$$E_{f,i}^{s^k} \leq F_{f,i}^{s^0} + \frac{(2 \cdot k - 1) \cdot L_g^{max}}{R_g} + \sum_{x=1}^k \frac{L_{max}^{s^x}}{C^{s^x}} \quad (12)$$

The introduction of work-conservation does come at a price. The per-hop delay increases from  $L_g^{max}/R_g$  in non-work-conserving aggregation to  $2 \cdot L_g^{max}/R_g$  in work-conserving aggregation. However, this is a relatively small increase that is outweighed by the advantages of a work-conserving system.

### B. Fairness in CAI

Although CAI provides a low end-to-end delay bound, it still suffers from unfairness. That is, if a flow  $f$  exceeds its rate temporarily, then the tags of its packets, i.e., their virtual finishing time at  $s^0$ , become much larger than real-time. This allows packets from another flow to “jump” over those of  $f$ , and hence temporarily deny service to  $f$ <sup>5</sup>.

To mitigate this unfairness in CAI, we propose the reuse of tags. We define the tag  $T_{f,i}$  of packet  $f.i$  as follows

$$T_{f,i} = F_{f,i}^{s^0} \quad (13)$$

where  $s^0$  is the ingress access router. Note that the tag is independent of the hop, because it is used only to sort the packets within the aggregate flow, and not for scheduling purposes.

To reuse tags, we use the same Conditions (8) and (9) that we proposed earlier. That is, a tag should not be reused if there is an outstanding packet of  $f$  that is using a tag that overlaps with the tag being reused. Also, a tag should be large enough to prevent interfering with the scheduling of other tags.

*Theorem 2:* Let  $f$  be an input flow of an aggregating scheduler  $s^0$ ,  $g$  be the output flow of  $s^0$ , and  $g$  traverses schedulers  $s^1, \dots, s^k$  with WFI index of at most  $\frac{L_g^{max}}{R_g} + \frac{L_{max}^t}{C^t}$ . Let each packet  $f.i$  be tagged at  $s^0$  according to Equation (13) above, and each scheduler  $s^1, \dots, s^k$  sorts the arriving packets of  $g$  in tag order. Let  $s^0$  reuse tags according to Conditions (8) and (9). Then,

$$E_{f,i}^{s^k} \leq F_{f,i}^{s^0} + (2 \cdot k - 1) \cdot \frac{L_g^{max}}{R_g} + \sum_{x=1}^k \frac{L_{max}^{s^x}}{C^{s^x}}. \quad (14)$$

Notice that the per-hop delay is the same as that without tag reuse, and hence, proportional to  $L_g^{max}/R_g$  per hop. As discussed earlier, this per-hop delay is lower than stateless core networks. The reason for this lower delay is because flows travel together. A delay bound of only  $L_f^{max}/R_f$  is obtained

<sup>5</sup>Nonetheless, the bound of Relation (12) still holds.

when there is no assumption about other flows joining and leaving the path of flow  $f$ .

The proof of the theorem is relegated to [25] due to space considerations. It is more involved than the proof given in Section III for stateless core networks. The reason for this difficulty comes from tags being used only to reorder packets within  $g$ , rather than being used as explicit deadlines.

### C. Guaranteed Throughput

In this section, we investigate the throughput guarantee in a CAI network with tag reuse. As mentioned earlier, without tag reuse, a flow can be denied service because it exceeded its reserved bandwidth. However, it is often the case that applications cannot accurately judge in advance the bandwidth required for their date, and hence, they should not be punished for temporarily exceeding their reserved rate. Via tag reuse, a flow is able to compete with other flows whose tags are smaller. As a result, the time period that the network can deny service to a flow becomes bounded.

Let  $W_f(\tau, \tau')$  denote the number of bytes from flow  $f$  that depart from the egress router during the time interval  $[\tau, \tau']$ . The throughput guarantee given to flow  $f$  is expressed below.

*Theorem 3:* If a flow  $f$  generates packets at a rate high enough to occupy the unallocated bandwidth, CAI with tag reuse guarantees to the flow a minimum throughput,  $W_f(\tau, \tau')$ , as follows.

$$W_f(\tau, \tau') > R_f \cdot (\tau' - t) - R_f \cdot \left( D + \max_{\forall h \in w} \left( \frac{L_h}{R_h} \right) \right) - R_f \cdot \left( (2k - 1) \cdot \frac{L_g^{max}}{R_g} + \sum_{x=1}^k \frac{L_g^{s^x}}{C^{s^x}} \right) \quad (15)$$

where  $D$  is the time for an acknowledgment to go through the network, from the egress router back to the ingress router,  $w$  is the set of flows that share any of the schedulers with  $f$ ,  $g$  is the aggregated flow, and  $k$  is the number of core routers on the path of flow  $f$ .

Note that, although in Condition (9) we use the term  $\max_{\forall h \in w} \left( \frac{L_h}{R_h} \right)$  as opposed to  $\frac{L_f}{R_f}, \max_{\forall h \in w} \left( \frac{L_g}{R_g} \right)$  is subtracted only once in Relation (15) above. The per-hop penalty is  $2 \cdot \frac{L_g}{R_g} + \frac{L_{max}}{C}$ . The throughput guarantee is therefore similar to the one achieved in stateless core networks [16]. However, a lowered end-to-end delay bound is obtained, hence a better throughput guarantee is expected as well.

## V. CONCLUDING REMARKS

In this paper, we have shown that throughput guarantees can be given in a flow aggregation network by the reuse of deadlines (tags). In addition, we have revised the conditions for deadline reuse in stateless core networks.

We have chosen the virtual finishing time of packets as the tag for packets exiting the aggregator. Other tags could have been chosen, which would yield different throughput guarantees. We plan to explore other tagging methods in [25].

## 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] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture," Internet RFC 1633.
- [4] J. Cobb, "Universal timestamp scheduling for real-time networks," *Computer Networks*, vol. 31, pp. 2341–2360, 1999, Elsevier.
- [5] —, "Preserving quality of service guarantees in spite of flow aggregation," *IEEE/ACM Transactions on Networking*, vol. 10, no. 1, pp. 43–53, Feb. 2002.
- [6] —, "Scalable quality of service across multiple domains," *Elsevier: Computer Communications*, vol. 28, no. 18, pp. 1997–2008, Nov. 2005.
- [7] J. Cobb and M. Gouda, "Flow theory," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 661–674, Oct. 1997.
- [8] 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.
- [9] J. Cobb and Z. Xu, "Maintaining flow isolation in work-conserving flow aggregation," in *Proc. of the IEEE GlobeCom Conf.*, 2005.
- [10] N. R. Figueira and J. Pasquale, "A schedulability condition for deadline-ordered service disciplines," *IEEE/ACM Transactions on Networking*, vol. 5, no. 2, pp. 232–244, 1997.
- [11] 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.
- [12] P. Goyal, S. Lam, and H. Vin, "Determining end-to-end delay bounds in heterogeneous networks," in *Proc. of the NOSSDAV Workshop*, 1995.
- [13] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured forwarding phb group," Internet RFC 2597.
- [14] V. Jacobson, K. Nichols, and K. Poduri, "An expedited forwarding phb," Internet RFC 2598.
- [15] J. Kaur and H. M. Vin, "Core-stateless guaranteed rate scheduling algorithms," in *Proc. of the IEEE INFOCOM Conf.*, 2001.
- [16] —, "Core stateless guaranteed throughput networks," in *Proc. of the IEEE INFOCOM Conf.*, 2003.
- [17] 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.
- [18] 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.
- [19] D. Stidialis and A. Varma, "Rate proportional servers: A design methodology for fair queueing algorithms," *IEEE/ACM Transactions on Networking*, Apr. 1998.
- [20] 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.
- [21] —, "A general methodology for designing efficient traffic scheduling and shaping algorithms," in *IEEE INFOCOM Conference*, 1997.
- [22] I. Stoica and H. Zhang, "Providing guaranteed services without per-flow management," in *Proc. of the ACM SIGCOMM Conference*, 1999.
- [23] 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.
- [24] G. Xie and S. Lam, "Delay guarantee of the virtual clock server," *IEEE/ACM Transactions on Networking*, pp. 683–689, Dec. 1995.
- [25] Z. Xu, "Scalable scheduling for quality of service guarantees," Ph.D. dissertation, The University of Texas at Dallas.
- [26] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 93, no. 10, Oct. 1995.
- [27] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Transaction on Communications*, vol. 42, no. 3, Mar. 1994.