End-to-End Delay Bounds for Traffic Aggregates Under Guaranteed-Rate Scheduling Algorithms

Wei Sun, *Student Member, IEEE,* and Kang G. Shin, *Fellow, IEEE*

***Abstract*—This paper evaluates, via both analysis and simulation, the end-to-end (e2e) delay performance of *aggregate scheduling* with guaranteed-rate (GR) algorithms. Deterministic e2e delay bounds for a *single* aggregation are derived under the assumption that all incoming flows at an aggregator conform to the token bucket model. An aggregator can use any of three types of GR scheduling algorithms: stand-alone GR, two-level hierarchical GR, and rate-controlled two-level hierarchical GR. E2e delay bounds are also derived for the case of *multiple* aggregations within an aggregation region when aggregators use the rate-controlled two-level hierarchical GR. By using the GR scheduling algorithms for traffic aggregates, we show not only the existence of delay bounds for each flow, but also the fact that, under certain conditions (e.g., when the aggregate traverses a long path after the aggregation point), the bounds are smaller than that of per-flow scheduling. We then compare the analytic delay bounds numerically and conduct in-depth simulation to: 1) confirm the analytic results and 2) compare the e2e delays of aggregate and per-flow scheduling. The simulation results have shown that aggregate scheduling is very robust and can exploit statistical multiplexing gains. It performs better than per-flow scheduling in most of the simulation scenarios we considered.**

Overall, aggregate scheduling is shown theoretically to provide bounded e2e delays and practically to provide excellent e2e delay performance. Moreover, it incurs lower scheduling and state-maintenance overheads at routers than per-flow scheduling. All of these salient features make aggregate scheduling very attractive for use in Internet core networks.

***Index Terms*—Aggregate scheduling, end-to-end (e2e) delay bounds, token bucket model, traffic aggregation.**

I. Introduction

REAL-TIME applications, such as voice-over-IP (VoIP) and video conferencing, require the network to provide better Quality-of-Service (QoS) in terms of delay, jitter, and loss rate. To provide such QoS support, several network architectures have been proposed. The IntServ architecture [1] supports QoS via *per-flow* resource reservation (e.g., RSVP) and packet scheduling. Numerous scheduling algorithms (e.g., see [2] for an excellent survey) have been proposed to support IntServ QoS, such as fairness, bounded per-flow (per-node or e2e) delay and backlog under a certain traffic model such as the token bucket model. Since both resource reservation and packet scheduling are per-flow-based, and hence the routers in the network must keep a large number of flow states, IntServ is not scalable for use in the core of the Internet that carries millions of flows.

To solve the IntServ's scalability problem, the DiffServ (DS) architecture [3] has been proposed by classifying traffic into a number of predefined classes, such as expedited forwarding (EF), assured forwarding (AF), and best-effort (BE) at the *edge* of each DS domain. The traffic class is identified by the marking in the DS field of each packet. Flow information is visible only at edge routers of a DS domain, and sophisticated packet classification, marking, policing, and shaping operations need only be implemented at edge routers. Within each DS domain, packets receive a particular per-hop forwarding behavior at routers on their path based on the DS field in their IP headers. In other words, flows are invisible and packet scheduling is done based on the traffic classes, *not* based on individual flows. The EF class [4] receives priority over AF and BE classes. The DiffServ architecture is more scalable than IntServ, but FIFO queueing—commonly used to schedule packets in each traffic class—is not suitable for hard QoS guarantees [5].

In this paper, we consider an extension of the IntServ architecture to support traffic aggregation. This extension is made on the premise that there are *aggregation regions* in the network, which "see" only aggregated (not individual) flows. Resource reservation and packet scheduling in an aggregation region are done on a per-aggregate basis. An example of this is the virtual paths (VPs) in ATM networks. Traffic aggregation has been studied extensively; see [6]–[9] for resource reservation, [10] and [11] for the admission control of aggregates, and [12] for flow states aggregation. We will in this paper focus on the scheduling issues associated with traffic aggregation and evaluate the deterministic e2e delay bounds of aggregate scheduling when guaranteed-rate (GR) algorithms [13] are used.

Traffic aggregates discussed in this paper are similar to the *traffic trunk*—which is defined as an aggregate of traffic flows that belong to the same class—in multiprotocol label switching (MPLS) [14]. As shown in Fig. 1, some flows enter an aggregation region from ingress router S_1 and share the same path for a number of hops inside the region. Within the region, they are treated as a *single* aggregate at the routers on the path. S_1 aggregates or "bundles" the flows by using the GR scheduling algorithms. When the aggregate leaves S_n, it is split back into individual flows. Similarly, another aggregate can be set up between S_1 and S_k. In general, a traffic aggregate can be created and terminated at any point in the network, and it can be created recursively. For example, at S_i two aggregates are bundled again into another (higher-level) aggregate which will later be split at S_l. The router (e.g., S_1) that aggregates flows (using GR algorithms) is called an *aggregator*, and the router (e.g., S_k) that splits the aggregate is called a *deaggregator*. The term "flows"

Manuscript received May 26, 2003; revised October 28, 2004; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J.-Y. Leboudec. This work was supported in part by Samsung Electronics, Inc., and the Office of Naval Research under Grant N00014-99-1-0465.

The authors are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: wsunz@eecs.umich.edu; kgshin@eecs.umich.edu).

Digital Object Identifier 10.1109/TNET.2005.857078

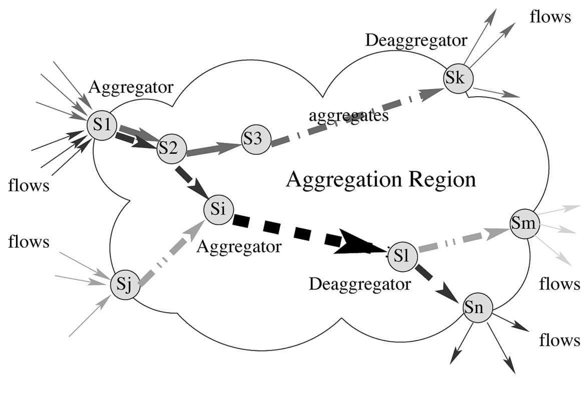


Fig. 1. Aggregate scheduling.

("traffic aggregates" or "aggregates") means the entities before (after) aggregation. We specify the aggregator and deaggregator as part of the aggregation region, although they can differentiate among the constituent flows of each aggregate.

The main contributions of the paper are twofold. First, by using the GR algorithms to schedule traffic aggregates, we show not only the existence of e2e delay bounds, but also the fact that, in many cases, the bounds are smaller than those of per-flow scheduling. Second, using in-depth simulation, we not only confirm the analytical results, but also show the advantages of aggregate scheduling.

Note that the delay bound problem of aggregate scheduling was also studied by Cobb [15], [16]. By using rate-based scheduling algorithms and *fair aggregators*, he showed that the e2e delay of an aggregate is bounded, and the bound can be smaller than the per-flow e2e delay bound.

This paper extends the results of [16] in the following *general* ways. First, the fair aggregators in [16] (both the *basic fair aggregator* and *greedy fair aggregator*) use nonwork-conserving scheduling algorithms. In contrast, our definition of fair aggregator is more general, applicable to both work-conserving and nonwork-conserving GR scheduling algorithms. Using the token-bucket traffic model, we derived the delay bounds under two types of work-conserving fair aggregators: stand-alone and hierarchical fair aggregators. Second, both the basic fair aggregator and the greedy fair aggregator implicitly assume that there is only one outgoing traffic aggregate at an output interface, and thus Cobb's derivation does not consider the interference from the packets outside the aggregate. Our delay bound results are more general: all of the three delay bounds are derived under the assumption that there could be multiple traffic aggregates through an output link. Third, we explicitly provide the delay bounds using the token-bucket traffic model and establish the relationship between a flow's delay bound and burst sizes of itself and other flows sharing the same aggregate.

The remainder of the paper is organized as follows. Section II provides the definitions of GR scheduling algorithm [17] and LR server [18], discusses their relationship, and reviews the delay bound results for per-flow scheduling in [17]. Section III introduces the concept of fair aggregator and derives the delay bound for aggregate scheduling under the token-bucket traffic model. Two delay bounds are derived: the first for stand-alone aggregator and the second for hierarchical aggregator, both of which use work-conserving GR scheduling algorithms. The hierarchical aggregator is shown to improve the delay bound.

TABLE I
SYMBOLS

S_i	the i^{th} router/server along the path of a flow or aggregate
p_f^j	the j^{th} packet of flow f
p_A^j	the j^{th} packet of aggregate flow A
ℓ_f^j	packet length of p_f^j
ℓ_A^j	packet length of p_A^j
ℓ_f^{max}	max. packet length in flow f
ℓ_A^{max}	max. packet length in aggregate flow A
L_{max}^i	max. packet length at router S_i
C^i	link capacity between S_i and S_{i+1}
σ_f	burst size of flow f under token bucket model
ρ_f	average rate of flow f under token bucket model
r_f	guaranteed rate for flow $f (r_f \geq \rho_f)$
R	sum of the guaranteed rates of all the flows in the aggregate flow, i.e., $R = \sum_{k=1}^{N} r_k$
$A_f(\tau, t)$	traffic arrived from flow f during $(\tau, t]$
$A^i(p_f^j)$	arrival time of packet p_f^j at router S_i
$GRC^i(p_f^j)$	guaranteed rate clock for p_f^j at S_i
$D^i(p_f^j)$	departure time of p_f^j at router S_i
α^i	$\alpha^i = \beta^i + \tau^{i,i+1}$
β^i	scheduling constant at router S_i
$\tau^{i,i+1}$	propagation delay between S_i and S_{i+1}
γ^i	aggregation constant at router S_i
d_f^j	e2e delay bound for p_f^j

Section IV improves the delay bound further by having the aggregators use nonwork-conserving scheduling algorithms. Section V presents numerical results, comparing the above deterministic delay bounds with that of per-flow scheduling. Simulation is also used to compare the delay performance of both aggregate and per-flow scheduling, confirming the benefits of aggregate scheduling derived from the analysis. Section VI discusses some related work on aggregate scheduling, putting our results in a comparative perspective. Finally, Section VII summarizes our contributions and discusses future directions.

II. Guaranteed-Rate Scheduling Algorithms

Before deriving the delay bound under aggregate scheduling, in this section we introduce the definitions of *guaranteed-rate* (GR) scheduling algorithm and *latency-rate* (LR) server, discuss their relationship, and review the e2e delay bound results for per-flow scheduling. For the convenience of discussion, we first give a list of symbols in Table I.

A. GR Scheduling Algorithms

The authors of [17] defined a class of GR scheduling algorithms. The delay guarantees provided by these algorithms are based on the *guaranteed rate clock* (GRC) value associated with each packet, which is defined as follows [17].

Definition 1 (GR Clock Value): Consider a flow f associated with a guaranteed rate r_f. Let p_f^j and ℓ_f^j denote the jth packet of flow f and its length, respectively. Also, let $\mathrm{GRC}^i(p_f^j)$ and $A^i(p_f^j)$ denote the GRC value and arrival time of packet p_f^j at router S_i, respectively. Then, the GRC values for packets of flow f are given by

$$\mathrm{GRC}^i\left(p_f^j\right) = \begin{cases} 0, & \text{j = 0} \\ \max\left\{A^i\left(p_f^j\right), \mathrm{GRC}^i\left(p_f^{j-1}\right)\right\} + \frac{\ell_f^j}{r_f}, & j \geq 1. \end{cases} \quad (1)$$

Definition 2 (GR Scheduling Algorithm): A scheduling algorithm at router S_i is said to belong to the GR class for flow f if it guarantees that packet p_f^j is transmitted by time $\mathrm{GRC}^i(p_f^j) + \beta^i$, where β^i is a *scheduling constant* [15] that depends on the scheduling algorithm and the router.

We will henceforth call a router equipped with the GR scheduling algorithm a *GR server*. Many scheduling algorithms are shown in [17] to belong to the GR class. For example, both packet-level generalized processor sharing (PGPS) [19] and virtual clock (VC) [20] are GR scheduling algorithms with $\beta^i = \frac{L^i_{\max}}{C^i}$, where $L^i_{\max}$ is the maximum packet length seen by router S_i and C^i is the output link capacity of S_i.

B. LR Server

The authors of [18] also defined a class of scheduling algorithms as LR servers, the definition of which is repeated below for self containment, although some notations in the definition are modified for consistency.

Definition 3 (LR Server): Let τ be the starting time of a busy period of flow f in server S_i and τ^* the time at which the last bit of traffic arrived during the busy period leaves the server. Then, server S_i is an LR server if and only if a nonnegative constant C_f^i can be found such that, at every instant t in the interval $(\tau, \tau^*]$

$$W_f^i(\tau,t) \geq \max\left\{0, r_f\left(t-\tau-C_f^i\right)\right\} \tag{2}$$

where $W_f^i(\tau,t)$ denotes the total service provided by the server S_i to flow f during the busy period until time t. The minimum nonnegative constant C_f^i satisfying the above inequality is defined as the *latency* of the server, denoted by θ_f^i.

The definition of the LR server helps establish the relationship between the burst size of an output flow and that of the corresponding input flow at a router. Before introducing Lemma 1, we define the *token-bucket traffic model* as follows: flow f conforms to the token bucket (σ_f, ρ_f) if, for any time instant τ and t satisfying $0 \leq \tau < t$, its traffic volume arrived during $(\tau, t]$, $A_f(\tau,t)$, satisfies

$$A_f(\tau,t) \leq \sigma_f + \rho_f \cdot (t-\tau) \tag{3}$$

where σ_f and ρ_f are the burst size and average rate of flow f, respectively.

Lemma 1: Suppose an incoming flow f to router S_i conforms to the token-bucket model (σ_f, ρ_f). If S_i is an LR server with parameters (θ_f^i, r_f) for flow f, where θ_f^i is the latency at S_i and r_f $(r_f \geq \rho_f)$ is the guaranteed rate for flow f, respectively, then the output traffic of flow f conforms to the token-bucket model with parameters $(\tilde{\sigma}_f, \rho_f)$, where $\tilde{\sigma}_f \leq \sigma_f + \theta_f^i \cdot \rho_f$.

For example, both PGPS and VC are LR servers with $\theta_f^i = \frac{\ell_f^{\max}}{r_f} + \frac{L^i_{\max}}{C^i}$ [18], in which case we have

$$\tilde{\sigma}_f \leq \sigma_f + \ell_f^{\max} + \frac{\rho_f \cdot L^i_{\max}}{C^i}. \tag{4}$$

The Proof of Lemma 1 is similar to that of [18, Theorem 3], which assumes $\rho_f = r_f$, but the result can be easily extended to the case of $\rho_f \leq r_f$.

C. Relationship Between GR and LR Servers

From the definition of the LR server, it is easy to show that an LR server is also a GR server.

Theorem 1: For any flow f, an LR server S_i with latency θ_f^i is also a GR server with scheduling constant θ_f^i.

Proof: Without loss of generality, we consider only one busy period of flow f. Suppose the reserved rate for flow f is r_f, and the busy period starts at time τ and ends at τ^*. From the definition of the GRC value in (1), it is easy to see that the GRC values of the packets in this busy period are

$$\mathrm{GRC}(p_f^k) = \frac{\sum_{n=1}^k \ell_f^n}{r_f} + \tau, \qquad k \geq 1.$$

Suppose t_k is the time when the kth packet leaves the server, then

$$W_f^i(\tau, t_k) = \sum_{n=1}^k \ell_f^n.$$

From the definition of LR server, we obtain

$$\sum_{n=1}^k \ell_f^n \geq \max\left\{0, r_f\left(t_k-\tau-\theta_f^i\right)\right\} \geq r_f\left(t_k-\tau-\theta_f^i\right).$$

Rearranging the terms, we obtain

$$t_k \leq \frac{\sum_{n=1}^k \ell_f^n}{r_f} + \tau + \theta_f^i = \mathrm{GRC}\left(p_f^k\right) + \theta_f^i, \qquad \forall k \geq 1. \tag{5}$$

The theorem is proven. ■

Similarly, it can be proven that a GR server is also an LR server. For more details, see the full version of this paper [21].

D. End-to-End Delay Bound Under Per-Flow Scheduling

We now review the e2e delay bound results for per-flow scheduling. Both Lemma 2 and Theorem 2 stated below were proven in [17].

Lemma 2: Suppose routers S_i and S_{i+1} are two neighboring GR servers on the path of flow f. If both routers guarantee service rate r_f for flow f, then

$$\mathrm{GRC}^{i+1}\left(p_f^j\right) \leq \mathrm{GRC}^i\left(p_f^j\right) + \frac{\ell_f^{\max}}{r_f} + \alpha^i \tag{6}$$

where $\ell_f^{\max}$ is the maximum packet size in flow f and $\alpha^i = \beta^i + \tau^{i,i+1}$. Here, $\tau^{i,i+1}$ is the propagation delay between S_i and S_{i+1}.

Lemma 2 states the relationship between the GRC values of a packet at two neighboring GR servers. Based on this relationship, the authors of [17] derived an e2e delay bound.

Theorem 2: If flow f conforms to the token-bucket model (σ_f, ρ_f) and all of the routers on its path are GR servers with guaranteed rate $r_f \geq \rho_f$, then the e2e delay for p_f^j, d_f^j, is bounded as follows:

$$d_f^j \leq \frac{\sigma_f}{r_f} + \frac{(K-1)\ell_f^{\max}}{r_f} + \sum_{n=1}^K \alpha^n \tag{7}$$

where $\alpha^i = \beta^i + \tau^{i,i+1}$ and K is the number of hops on the path of flow f.

From Theorem 2, one can see that the delay bound is inversely proportional to the flow's guaranteed rate, but is proportional to the number of hops (K), the size of packets, and the burst size of the flow. With a large number of hops and large-size packets, the delay can be substantially large. To understand the physical meaning of (7), let us consider the fluid traffic model, where packet size is infinitely small. Then, if we omit the propagation delay, the delay bound in (7) can be simplified as

$$d_f \leq \frac{\sigma_f}{r_f}. \tag{8}$$

In other words, the delay is upper bounded by the burstiness of the flow. The larger the burst size, the larger the delay bound. If we assume $\rho_f = r_f$, then the delay bound is decided by the *burst ratio* ($\frac{\sigma_f}{\rho_f}$ for flow f) of the flow.

In the next two sections, we derive e2e delay bounds under aggregate scheduling with GR scheduling algorithms. We show that, if the incoming flows at aggregators conform to the token-bucket model, the e2e delay is bounded.

Note that, by using GR scheduling algorithms, we always assume that the guaranteed rate for a flow is greater than or equal to its average rate, i.e., $r_f \geq \rho_f$. Also, in the remainder of this paper, we omit the propagation delay $\tau^{i,i+1}$. Therefore, $\alpha^i = \beta^i$.

III. Work-Conserving Aggregator

In this section, we use work-conserving GR scheduling algorithms at aggregators. Two types of GR scheduling algorithms are examined: *stand-alone* (or nonhierarchical) and *hierarchical* algorithms. To derive the e2e delay bound under aggregate scheduling, an important step is to derive the delay at the aggregator. We need to find the relationship between the GRC values at the aggregator and its next hop.

A. Fair Aggregator

Before deriving the delay at the aggregator, we first introduce an important concept, which is the *fair aggregator*.

Definition 4 (Fair Aggregator): Let router S_i be an aggregator bundling flow f and others into aggregate flow A, which, in turn, is an input to router S_{i+1}. Then, if $p_f^j = p_A^{j'}$, i.e., the jth packet of flow f corresponds to the j'th packet of aggregate flow A, S_i is said to be a *fair aggregator* if and only if S_i is a GR server and

$$\mathrm{GRC}^{i+1}\left(p_A^{j'}\right) \leq \mathrm{GRC}^i\left(p_f^j\right) + \gamma^i + \alpha^i, \qquad \forall j \geq 1 \tag{9}$$

where γ^i is a constant that depends on the scheduling algorithm, the router, and other flows in the same aggregate. We call it an *aggregation constant*.

The value $\gamma^i + \alpha^i$ can be considered as the *fairness index* of the aggregator, e.g., an aggregator is considered fairer than others if it has a smaller value of $\gamma^i + \alpha^i$. For a fair aggregator, both of its aggregation constant (γ^i) and scheduling constant (α^i) should be small. Also, our definition of *fair aggregator* is slightly different from that in [15] and [16]. It is based on the relationship between a packet's GRC values at the aggregator and its next hop.

Lemma 3: Suppose S_i and S_{i+1} are two neighboring GR servers and S_i is an aggregator. S_i aggregates flow f and other $(N-1)$ flows into aggregate flow A, which, in turn, is an input to S_{i+1}. Suppose incoming flow k at S_i conforms to the token-bucket model (σ_k, ρ_k) $(1 \leq k \leq N)$, and the guaranteed rate for flow k at S_i is r_k $(1 \leq k \leq N)$. Let $p_f^j = p_A^{j'}$, i.e., the jth packet of flow f corresponds to the j'th packet of aggregate flow A. If the outgoing flows at S_i conform to the token-bucket model $(\tilde{\sigma}_k, \rho_k)$ $(1 \leq k \leq N)$, then for packet p_f^j

$$\mathrm{GRC}^{i+1}\left(p_A^{j'}\right) \leq \mathrm{GRC}^i\left(p_f^j\right) + \frac{\sum_{k\neq f}\tilde{\sigma}_k + \ell_f^{\max}}{R} + \alpha^i, \qquad j \geq 1 \tag{10}$$

where $R = \sum_{k=1}^N r_k$, which is the guaranteed rate for aggregate flow A at S_{i+1}.

Proof: Similar to the definition of a busy period of a single flow [18], we define a *busy period of aggregate flow A* at router S_{i+1} as the maximum length of time period $(\tau_1, \tau_2]$, such that, at any time $t \in (\tau_1, \tau_2]$, the total traffic of A arrived since the beginning of the interval is $A(\tau_1, t) \geq R\cdot(t-\tau_1)$. Without loss of generality, we consider only one busy period in the proof. Suppose the busy period starts at time t_0 when the first packet p_A^1 arrives, and in that busy period time t_j is the instant when packet p_f^j arrives at S_{i+1}. Since $p_f^j = p_A^{j'}$, the total traffic arrival of aggregate A up to time t_j is

$$\begin{aligned}\sum_{k=1}^{j'}\ell_A^k &= A(t_0,t_j) = \sum_{k=1}^{N}A_k(t_0,t_j)\\ &= A_f(t_0,t_j) + \sum_{k\neq f}A_k(t_0,t_j)\\ &\leq \sum_{m=1}^{j}\ell_f^m + \sum_{k\neq f}(\tilde{\sigma}_k + r_k\cdot(t_j-t_0))\\ &= \sum_{m=1}^{j}\ell_f^m + \sum_{k\neq f}\tilde{\sigma}_k + (R-r_f)(t_j-t_0). \end{aligned} \tag{11}$$

Since all of the packets arrive in the same busy period of aggregate A, from the definition of GRC value, we have

$$\begin{aligned}\mathrm{GRC}^{i+1}\left(p_A^{j'}\right) &= \frac{\sum_{k=1}^{j'}\ell_A^k}{R} + t_0\\ &\leq \frac{\sum_{m=1}^{j}\ell_f^m + \sum_{k\neq f}\tilde{\sigma}_k + (R-r_f)(t_j-t_0)}{R} + t_0. \end{aligned} \tag{12}$$

Next, we prove that the theorem holds for both $j = 1$ and $j > 1$.

Case 1: $j = 1$.

$$\begin{aligned}\mathrm{GRC}^{i+1}\left(p_A^{j'}\right) &\leq \frac{\ell_f^1 + \sum_{k\neq f}\tilde{\sigma}_k + (R-r_f)(t_1-t_0)}{R} + t_0\\ &= \frac{\sum_{k\neq f}\tilde{\sigma}_k + \ell_f^1}{R} + t_1 - \frac{r_f\cdot(t_1-t_0)}{R}\\ &\leq \frac{\sum_{k\neq f}\tilde{\sigma}_k + \ell_f^{\max}}{R} + t_1.\end{aligned}$$

Since S_i is a GR server for flow f, we have $t_1 \leq \mathrm{GRC}^i(p_f^1) + \alpha^i$. Therefore

$$\mathrm{GRC}^{i+1}\left(p_A^{j'}\right) \leq \mathrm{GRC}^i\left(p_f^1\right) + \frac{\sum_{k\neq f}\tilde{\sigma}_k + \ell_f^{\max}}{R} + \alpha^i.$$

Case 2: $j > 1$. From the definition of GRC value, for flow f, we have

$$\mathrm{GRC}^i\left(p_f^2\right) \geq \mathrm{GRC}^i\left(p_f^1\right) + \frac{\ell_f^2}{r_f},$$

$$\vdots$$

$$\mathrm{GRC}^i\left(p_f^j\right) \geq \mathrm{GRC}^i\left(p_f^{j-1}\right) + \frac{\ell_f^j}{r_f}.$$

Adding them up, we obtain

$$\mathrm{GRC}^i\left(p_f^j\right) - \mathrm{GRC}^i\left(p_f^1\right) \geq \frac{\sum_{m=2}^{j} \ell_f^m}{r_f}$$

$$\Longrightarrow \sum_{m=1}^{j} \ell_f^m \leq \ell_f^1 + r_f \cdot \left(\mathrm{GRC}^i\left(p_f^j\right) - \mathrm{GRC}^i\left(p_f^1\right)\right). \quad (13)$$

Then, from (12) and (13), and the fact $t_j \leq \mathrm{GRC}^i(p_f^j) + \alpha^i$, we obtain (after simplification)

$$\mathrm{GRC}^{i+1}\left(p_A^{j'}\right) \leq \mathrm{GRC}^i\left(p_f^j\right) + \frac{\sum_{k\neq f} \tilde{\sigma}_k + \ell_f^{\max}}{R} + \alpha^i.$$

The lemma is proven. ∎

Next, we study the stand-alone aggregator, which uses the stand-alone GR scheduling algorithm to bundle flows into aggregates.

B. Delay Bound I: Stand-Alone Aggregator

Combining Lemmas 1 and 3, we obtain the following theorem on the relationship between the GRC values at a stand-alone aggregator and its next hop.

Theorem 3: Suppose S_i is an LR server, and both S_i and S_{i+1} are GR servers. As a stand-alone aggregator, S_i aggregates flow f and other $(N-1)$ flows into aggregate A, which, in turn, is an input to router S_{i+1}. Suppose incoming flow k at S_i conforms to the token-bucket model (σ_k, ρ_k) and the guaranteed rate for flow k at router S_i is r_k $(1 \leq k \leq N)$. Let $p_f^j = p_A^{j'}$, i.e., the jth packet of flow f corresponds to the j'th packet of aggregate A. Then, S_i is a fair aggregator with $\gamma^i = \frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k\neq f}\theta_k^i \cdot \rho_k + \ell_f^{\max}}{R}$. In other words,

$$\mathrm{GRC}^{i+1}\left(p_A^{j'}\right) \leq \mathrm{GRC}^i\left(p_f^j\right) + \alpha^i + \left[\frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k\neq f}\theta_k^i \cdot \rho_k + \ell_f^{\max}}{R}\right], \quad j \geq 1. \quad (14)$$

The proof is trivial.

Now, we are ready to derive the e2e delay bound for aggregate scheduling under the token-bucket model and the stand-alone aggregator.

Theorem 4: Suppose N flows share the same K hops of GR servers inside an aggregation region, and they are bundled into aggregate A at stand-alone aggregator S_1 and split back at S_K. Routers $S_2, \ldots, S_{K-1}$ schedule packets of aggregate A. If flow k conforms to the token bucket model (σ_k, ρ_k) and has the guaranteed rate r_k with $r_k \geq \rho_k$ $(1 \leq k \leq N)$ at S_1 and S_K, and A has the guaranteed rate $R = \sum_{k=1}^{N} r_k$ at $S_2, \ldots, S_{K-1}$, then, for any flow f $(1 \leq f \leq N)$, the e2e delay of packet p_f^j, d_f^j, is bounded as follows:

$$d_f^j \leq \frac{\sigma_f}{r_f} + \left[\frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k\neq f}\theta_k^1 \cdot r_k + \ell_f^{\max}}{R}\right] + (K-3)\frac{\ell_A^{\max}}{R} + \frac{\ell_f^{\max}}{r_f} + \sum_{i=1}^{K}\alpha^i, \quad j \geq 1. \quad (15)$$

Proof: Let $p_A^{j'}$ be the packet in the aggregate flow A corresponding to p_f^j in flow f. From Theorem 3, we have

$$\mathrm{GRC}^2\left(p_A^{j'}\right) \leq \mathrm{GRC}^1\left(p_f^j\right) + \gamma^1 + \alpha^1$$

where $\gamma^1 = \frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k\neq f}\theta_k^1 \cdot \rho_k + \ell_f^{\max}}{R}$. Since $S_2, \ldots, S_{K-1}$ are all GR servers for aggregate flow A with guaranteed rate $R = \sum_{k=1}^{N} r_k$, from Lemma 2, we have

$$\mathrm{GRC}^3\left(p_A^{j'}\right) \leq \mathrm{GRC}^2\left(p_A^{j'}\right) + \frac{\ell_A^{\max}}{R} + \alpha^2,$$

$$\vdots$$

$$\mathrm{GRC}^{K-1}\left(p_A^{j'}\right) \leq \mathrm{GRC}^{K-2}\left(p_A^{j'}\right) + \frac{\ell_A^{\max}}{R} + \alpha^{K-2}.$$

Adding them all up, we obtain

$$\mathrm{GRC}^{K-1}\left(p_A^{j'}\right) \leq \mathrm{GRC}^1\left(p_f^j\right) + (K-3)\frac{\ell_A^{\max}}{R} + \gamma^1 + \sum_{i=1}^{K-2}\alpha^i.$$

Since $p_f^j = p_A^{j'}$, for packet p_f^j, the departure time at S_{K-1} is

$$D^{K-1}\left(p_f^j\right) = D^{K-1}\left(p_A^{j'}\right) \leq \mathrm{GRC}^{K-1}\left(p_A^{j'}\right) + \alpha^{K-1}$$
$$\leq \mathrm{GRC}^1\left(p_f^j\right) + \gamma^1 + (K-3)\frac{\ell_A^{\max}}{R} + \sum_{i=1}^{K-1}\alpha^i.$$

From the definition of the GR server, the first $(K-1)$ servers for flow f can be viewed as a virtual GR server with scheduling constant $\alpha^* = \gamma^1 + (K-3)\frac{\ell_A^{\max}}{R} + \sum_{i=1}^{K-1}\alpha^i$. Since S_K is also a GR server for flow f with guaranteed rate r_f, from Lemma 2, we have

$$\mathrm{GRC}^K\left(p_f^j\right) \leq \mathrm{GRC}^1\left(p_f^j\right) + \frac{\ell_f^{\max}}{r_f} + \alpha^*.$$

Therefore, the departure time of packet p_f^j from S_K is

$$\begin{aligned} D^K\left(p_f^j\right) &\leq \mathrm{GRC}^K\left(p_f^j\right) + \alpha^K \\ &\leq \mathrm{GRC}^1\left(p_f^j\right) + \frac{\ell_f^{\max}}{r_f} + \alpha^* + \alpha^K \\ &= \mathrm{GRC}^1\left(p_f^j\right) + \left[\frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k\neq f}\theta_k^1 \cdot r_k + \ell_f^{\max}}{R}\right] \\ &\quad + (K-3)\frac{\ell_A^{\max}}{R} + \frac{\ell_f^{\max}}{r_f} + \sum_{i=1}^{K}\alpha^i. \end{aligned}$$

Then, the e2e delay d_f^j satisfies

$$\begin{aligned} d_f^j &= D^K\left(p_f^j\right) - A^1\left(p_f^j\right) \\ &\leq \left[\mathrm{GRC}^1\left(p_f^j\right) - A^1\left(p_f^j\right)\right] + (K-3)\frac{\ell_A^{\max}}{R} + \frac{\ell_f^{\max}}{r_f} \\ &\quad + \left[\frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k\neq f}\theta_k^1 \cdot r_k + \ell_f^{\max}}{R}\right] + \sum_{i=1}^{K}\alpha^i. \quad (16) \end{aligned}$$

From [17], for flow f conforming to the token-bucket model (σ_f, ρ_f) and with reserved rate $r_f (r_f \geq \rho_f)$, $\mathrm{GRC}^1(p_f^j) - A^1(p_f^j) \leq \frac{\sigma_f}{r_f}$. Thus, the theorem is proven. ∎

As in Section II, to further understand the physical meaning of (15), we consider the fluid traffic model, where packet size is infinitely small. Then, (15) can be simplified as (since in general the term θ_k^1 also relies on packet size and will become infinitely small):

$$d_f \leq \frac{\sigma_f}{r_f} + \frac{\sum_{k\neq f}\sigma_k}{R}. \tag{17}$$

Note that (17) provides the lower limit of the e2e delay bound. Comparing (8) and (17), we can see that, with aggregate scheduling, the delay bound of a flow is not only decided by the burstiness of the flow itself (term $\frac{\sigma_f}{r_f}$), but also strongly related to the burstiness of other flows that share the same aggregate with it (term $\frac{\sum_{k\neq f}\sigma_k}{R}$). Intuitively, this is easy to understand, since a packet from flow f has to wait behind not only the earlier packets from the same flow, but also those from other flows sharing the same aggregate. The result implies how aggregation should be done—a flow should not be aggregated with other flows with substantially larger burst ratios. This is similar to the conclusion in [22]. This issue will be explored further in Section V.

From (15), we can see that the delay bound is affected by the latency of the scheduling algorithm at the aggregator. To get a smaller bound, we should use a low-latency scheduling algorithm. For example, with PGPS and VC ($\theta_f^1 = \frac{\ell_f^{\max}}{r_f} + \frac{L_{\max}^1}{C^1}$) at the aggregator S_1, the e2e delay is bounded as follows:

$$d_f^j \leq \frac{\sigma_f}{r_f} + \left[\frac{\sum_{k\neq f}\sigma_k}{R} + \frac{\sum_{k=1}^{N}\ell_k^{\max}}{R}\right] + (K-3)\frac{\ell_A^{\max}}{R} + \frac{\ell_f^{\max}}{r_f} + \frac{L_{\max}^1}{C^1} + \sum_{i=1}^{K}\alpha^i. \tag{18}$$

Finally, comparing (7) and (15), we note that, depending on the burst ratios of the constituent flows and the maximum packet size in the aggregate, the delay bound under aggregate scheduling can be smaller than that under per-flow scheduling. We will compare the delay bounds numerically later in Section V.

In the next subsection, we further decrease the delay bound by using hierarchical GR scheduling algorithms at aggregators.

C. Delay Bound II: Hierarchical Aggregator

So far, the stand-alone GR scheduling algorithms have been used at aggregators. Thus, when computing the burstiness of an outgoing aggregate at aggregators, the burst size of each constituent flow in the aggregate was computed separately and then summed up. In this subsection, we use hierarchical GR algorithms at aggregators to improve the delay bound, as shown in Fig. 2. First, the flows that end up in the same aggregate are grouped together at the lower level schedulers. Then, the aggregates are scheduled at the upper level scheduler. This way, we can reduce the burstiness of the outgoing aggregate and the aggregation constant at aggregators, thus reducing the e2e delay.

How does a hierarchical scheduler reduce the burstiness of the outgoing aggregates? Fig. 3 shows the intuition behind it. Suppose there are 20 flows coming into one aggregator, and all of them have the same reserved rate. The first ten flows belong to one aggregate flow A_1, and the last ten belong to another aggregate flow A_2. Suppose 20 packets of identical size arrive at the idle aggregator at exactly the same time, one from each flow.

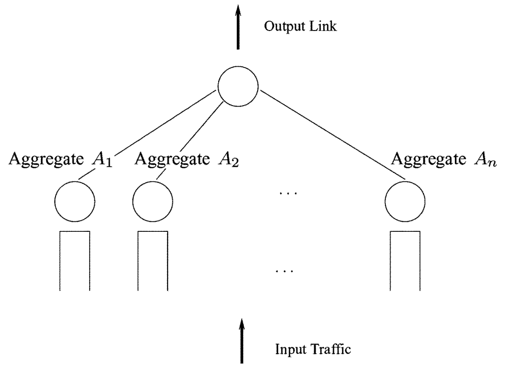


Fig. 2. Hierarchical scheduling.

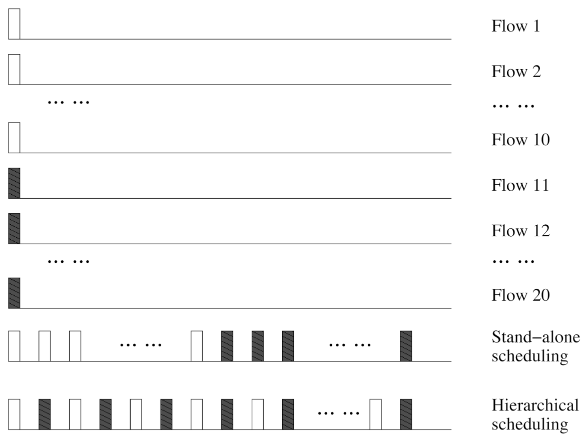


Fig. 3. Benefits of hierarchical scheduling.

Then, with a stand-alone scheduler, the 20 packets will be scheduled in a random order. Thus, it is possible that all ten packets of A_1 are scheduled before all of the packets of aggregate flow A_2. Therefore, the output of the aggregator will look like: ten packets of aggregate A_1 come out first, and then ten packets of aggregate A_2. For the next router which is handling aggregates, the burstiness of both A_1 and A_2 is very high. In contrast, with hierarchical scheduler, the upper level scheduler will make sure the packets from A_1 and A_2 are scheduled alternately, making the traffic in both aggregate flows smoother.

The hierarchical scheduling algorithms under consideration are two-level *hierarchical packet fair queueing* (H-PFQ) algorithms [23]. From [23, Theorem 2], it is easy to prove that H-PFQ also belongs to the GR class. For details of the proof, see the full version of this paper [21].

Now, we derive the burstiness of the outgoing traffic at aggregators. For any flow f, we derive $\sum_{k\neq f}\tilde{\sigma}_k$—the total outgoing burst size of all of the other flows sharing the same aggregate with f. In this derivation, we make use of the corresponding fluid generalized processor sharing (GPS) server of a packet fair queueing (PFQ) server. Let $W_k^i(\tau,t)$ ($W_{k,\mathrm{GPS}}^i(\tau,t)$) be the service received by flow k at the PFQ server S_i (the corresponding GPS fluid server) during $(\tau, t]$, and define λ_k^i and δ_k^i as

$$\lambda_k^i = \max_{t\geq 0}\left\{W_{k,\mathrm{GPS}}^i(0,t) - W_k^i(0,t)\right\} \tag{19}$$

$$\delta_k^i = \max_{t\geq 0}\left\{W_k^i(0,t) - W_{k,\mathrm{GPS}}^i(0,t)\right\}. \tag{20}$$

Intuitively, λ_k^i and δ_k^i define the maximum difference between the amount of services flow k gets from the corresponding GPS server and the PFQ server S_i in any time interval.

Theorem 5: Suppose S_i is an aggregator with a two-level H-PFQ scheduler and S_i aggregates flow f and other $(N-1)$ flows into aggregate A. Also, suppose flow k $(1 \leq k \leq N)$ at S_i conforms to the token-bucket model (σ_k, ρ_k) and has guaranteed rate r_k. Then, we have

$$\sum_{k \neq f} \tilde{\sigma}_k \leq \sum_{k \neq f} \sigma_k + \left[\lambda_f^i + \delta_f^i\right] + \left(1 - \frac{r_f}{R}\right)\left[\lambda_A^i + \delta_A^i\right]. \quad (21)$$

The details of the proof can be found in [21].

To get a small burst size, we should use PFQ with small λ_k^i and δ_k^i, especially small δ_k^i. From [24, Theorem 1], WF^{2}Q (worst-case fair weighted fair queueing) yields $\lambda_k^i = L_{\max}^i$ and a very small $\delta_k^i = (1 - \frac{r_k}{C^i})\ell_k^{\max}$ for flow k at server S_i with capacity C^i. Therefore, if H-WF^{2}Q is used by the aggregator, the burst size becomes

$$\begin{aligned}\sum_{k \neq f} \tilde{\sigma}_k &\leq \sum_{k \neq f} \sigma_k + \left[\ell_A^{\max} + \left(1 - \frac{r_f}{R}\right)\ell_f^{\max}\right] \\ &\quad + \left(1 - \frac{r_f}{R}\right)\left[L_{\max}^i + \left(1 - \frac{R}{C^i}\right)\ell_A^{\max}\right] \\ &\leq \sum_{k \neq f} \sigma_k + L_{\max}^i + \ell_f^{\max} + 2\ell_A^{\max}. \quad (22)\end{aligned}$$

Corollary 1: Using the H-WF^{2}Q algorithm at the aggregator, and under the same condition of Theorem 4, the e2e delay d_f^j is bounded by

$$d_f^j \leq \frac{\sigma_f}{r_f} + \frac{\sum_{k \neq f} \sigma_k + 2\left(\ell_f^{\max} + L_{\max}^1\right) + (K-1)\ell_A^{\max}}{R} + \frac{\ell_f^{\max}}{r_f} + \sum_{i=1}^{K} \alpha^i. \quad (23)$$

The proof is similar to that of Theorem 4; for details, see the full version of this paper [21].

In general, the bound in (23) is smaller than that in (18), especially when the number of flows (N) is large. However, as can be seen, the bound in (23) is still affected by the average burst ratio of other flows in the same aggregate.

D. Multiple Aggregations in a Region

The derivation of the two delay bounds (using work-conserving aggregators) in this section relies on the fact that the incoming traffic conforms to the token-bucket model with known parameters. Inside an aggregation region, the traffic pattern (the burst size in particular) will be distorted and become unpredictable. If more aggregations are done inside an aggregation region, this change of traffic pattern makes the derivation of delay bounds very difficult, if not impossible. In fact, we derived the two delay bounds assuming that aggregation is done only once in each aggregation region. To use our method to derive e2e delay bounds in the case of multiple aggregations in a region, the traffic has to be reshaped before every aggregator to make it follow the token-bucket model. The shaping incurs an extra delay, which is not considered here, though.

IV. Nonwork-Conserving Aggregator

In the previous section, work-conserving scheduling algorithms were used by aggregators, and the derivation of the delay bounds required knowing the incoming traffic pattern at the aggregator. This requirement limits the ability of multiple aggregations within a region, since the traffic pattern inside the region is difficult to predict. The authors of [25] proved that reshaping a flow inside the network will not change its e2e delay bound. However, their proof was done for a single flow, and it is not clear if reshaping an aggregate flow will change the e2e delay bound of a constituent flow of the aggregate. We conjecture that the delay bound will be affected.

Cobb [16] overcame this difficulty by using nonwork-conserving scheduling algorithms at aggregators. As stated in [26] and [27], nonwork-conserving scheduling algorithms have the following features: 1) the burstiness of traffic can be controlled; 2) the sum of the per-hop bounds can tightly bound the e2e delay and jitter; and 3) since the nonwork-conserving scheduler shapes the traffic at each hop, it is easy to bound the performance of heterogeneous networks. Both the *basic fair aggregator* and *greedy fair aggregator* in [16] use nonwork-conserving scheduling algorithms at aggregators to shape the outgoing traffic aggregate. This way, very few packets in the same aggregate queue up at later hops, which, in turn, makes the queueing delays in those hops very small. However, the implicit assumption in [16] is that there is only one output aggregate from each aggregator, which is not generally the case. In this section, we extend the result in [16] by allowing multiple outgoing aggregates from an aggregator and derive the e2e delay bound under the token bucket traffic model.

A. Delay Bound III: Nonwork-Conserving Aggregator

First, we define a new type of fair aggregator—*rate-controlled fair aggregator*.

Definition 5 (Rate-Controlled Fair Aggregator): Server S_i is said to be a *rate-controlled fair aggregator* if: 1) it is a two-level hierarchical GR scheduler; 2) each lower-level scheduler handles the flows belonging to one aggregate with capacity R, which is the sum of the guaranteed rates for all the constituent flows of the aggregate; and 3) the upper level is the scheduler for all the aggregates.

Note that the hierarchical scheduling algorithm defined here is different from the one used in Section III-C: it is nonwork-conserving, and the lower level consists of constant-rate servers. A packet at the lower level scheduler will not be sent to the upper level scheduler if the capacity of that scheduler does not permit it, even when the upper level scheduler is idle. In contrast, in the ordinary H-PFQ, the lower level schedulers are variable-rate servers [23]. Since the lower level schedulers are constant-rate servers, we will regard the lower level schedulers and the upper level scheduler as two virtual hops.

Lemma 4: Suppose server S_i is a rate-controlled fair aggregator, with one of the lower level schedulers $S_{i,l}$ serving N incoming flows (flow f is one of them). The output of $S_{i,l}$ is the aggregate A. $S_{i,h}$ is the upper level scheduler dealing with all of the aggregates. Suppose, at $S_{i,l}$, that these N flows have guaranteed rates $r_k, 1 \leq k \leq N$ and at $S_{i,h}$ the guaranteed rate for A is $R = \sum_{k=1}^{N} r_k$. Let the jth packet of A correspond to the j'th packet of flow f (i.e., $p_A^j = p_f^{j'}$). Then, we have

$$\mathrm{GRC}^{i,h}\left(p_A^j\right) \leq \mathrm{GRC}^{i,l}\left(p_f^{j'}\right) + \alpha^{i,l} \quad (24)$$

where $\alpha^{i,l}$ is the scheduling constant at $S_{i,l}$ (with capacity $R = \sum_{k=1}^{N} r_k$).

Proof: Without loss of generality, we consider only one busy period of aggregate A at $S_{i,h}$. Since the lower level server $S_{i,l}$ is rate-controlled with capacity R, the start times of the transmission of two consecutive packets (p_A^j and p_A^{j+1}) in the busy period are separated by an interval of $\frac{\ell_A^j}{R}$. Also, since $S_{i,l}$ and $S_{i,h}$ are two virtual nodes, the virtual link capacity between them is infinite, i.e., $C \to \infty$. Thus, the transmission time of a packet is infinitely small. Thus, we have

$$A^{i,h}\left(p_A^j\right) = \begin{cases} A^{i,l}\left(p_A^j\right), & j=1 \\ A^{i,h}\left(p_A^{j-1}\right) + \frac{\ell_A^{j-1}}{R}, & j>1. \end{cases} \tag{25}$$

Next, we prove that for all $j \geq 1$

$$\mathrm{GRC}^{i,h}\left(p_A^j\right) = A^{i,h}\left(p_A^j\right) + \frac{\ell_A^j}{R}. \tag{26}$$

We prove this by induction on j.

Base Case: $j = 1$. Since it is the first packet of the busy period, we have

$$\mathrm{GRC}^{i,h}\left(p_A^1\right) = A^{i,h}\left(p_A^1\right) + \frac{\ell_A^1}{R}.$$

Induction Hypothesis: Suppose (26) holds for $1 \leq j \leq m$.

Induction Step: $j = m+1$. From the definition of GRC value, we have

$$\mathrm{GRC}^{i,h}\left(p_A^{m+1}\right) = \max\left\{A^{i,h}\left(p_A^{m+1}\right), \mathrm{GRC}^{i,h}\left(p_A^m\right)\right\} + \frac{\ell_A^{m+1}}{R}.$$

However, from (25) and the induction hypothesis step, we know

$$A^{i,h}\left(p_A^{m+1}\right) = A^{i,h}\left(p_A^m\right) + \frac{\ell_A^m}{R} = \mathrm{GRC}^{i,h}\left(p_A^m\right),$$

$$\Longrightarrow \mathrm{GRC}^{i,h}\left(p_A^{m+1}\right) = A^{i,h}\left(p_A^{m+1}\right) + \frac{\ell_A^{m+1}}{R}.$$

Also, since $S_{i,l}$ is a GR server, by definition we have

$$A^{i,h}\left(p_A^j\right) + \frac{\ell_A^j}{R} \leq \mathrm{GRC}^{i,l}\left(p_A^j\right) + \alpha^{i,l} \quad \forall j \geq 1.$$

Since $p_A^j = p_f^{j'}$ and thus $\ell_A^j = \ell_f^{j'}$, from (26) we obtain

$$\mathrm{GRC}^{i,h}\left(p_A^j\right) \leq \mathrm{GRC}^{i,l}\left(p_f^{j'}\right) + \alpha^{i,l} \quad \forall j \geq 1.$$

■

From Lemma 4, one can easily see that the rate-controlled fair aggregator is a fair aggregator. Next, we derive the e2e delay bound and show that, by using nonwork-conserving scheduling algorithms at aggregators, the e2e delay bound of aggregate scheduling can be decreased.

Theorem 6: Suppose flow f traverses K hops of GR servers $S_1, S_2, \ldots, S_K$, and S_1 is a rate-controlled fair aggregator, which bundles f and other $(N-1)$ flows into aggregate flow A. S_K is the deaggregator for A. Suppose flow k has a guaranteed rate r_k $(1 \leq k \leq N)$ at S_1 and S_K, and the guaranteed rate for aggregate A at $S_2, S_3, \ldots, S_{K-1}$ is $R = \sum_{k=1}^{N} r_k$. Then, for packet p_f^j, the e2e delay d_f^j satisfies

$$d_f^j \leq \left[\mathrm{GRC}^{1,l}\left(p_f^j\right) - A^1\left(p_f^j\right)\right] + (K-2)\frac{\ell_A^{\max}}{R} + \alpha^{1,l} + \frac{\ell_f^{\max}}{r_f} + \sum_{i=1}^{K} \alpha^i. \tag{27}$$

The proof is similar to that of Theorem 4. See [21] for a detailed proof.

Corollary 2: If flow f conforms to the token-bucket model (σ_f, ρ_f) and $\rho_f \leq r_f$, the e2e delay result of Theorem 6 becomes

$$d_f^j \leq \frac{\sigma_f}{r_f} + (K-2)\frac{\ell_A^{\max}}{R} + \frac{\ell_f^{\max}}{r_f} + \alpha^{1,l} + \sum_{i=1}^{K} \alpha^i. \tag{28}$$

Note that, thanks to the rate-control mechanism at aggregators, the e2e bound does not depend on the burstiness of other flows in the same aggregate. Thus, the bound is smaller than those of the work-conserving cases. If scheduling algorithms, such as PGPS, VC, and WF^{2}Q, are used at the rate-controlled fair aggregator S_1, we have $\alpha^{1,l} = \frac{\ell_A^{\max}}{R}$. Then, (27) and (28) can be simplified as

$$d_f^j \leq \left[\mathrm{GRC}^{1,l}\left(p_f^j\right) - A^1\left(p_f^j\right)\right] + (K-1)\frac{\ell_A^{\max}}{R} + \frac{\ell_f^{\max}}{r_f} + \sum_{i=1}^{K} \alpha^i \tag{29}$$

and

$$d_f^j \leq \frac{\sigma_f}{r_f} + (K-1)\frac{\ell_A^{\max}}{R} + \frac{\ell_f^{\max}}{r_f} + \sum_{i=1}^{K} \alpha^i \tag{30}$$

respectively.

Moreover, the derivation of the bound in (27) does not require the knowledge of the traffic pattern of incoming flows. This enables us to derive the delay bound for the case of multiple aggregations. For simplicity, in the following discussion, we use the term $\mathrm{GRC}^i(p_f^j)$ to represent $\mathrm{GRC}^{i,l}(p_f^j)$. Also, we assume that scheduling algorithms such as PGPS, VC, and WF^{2}Q are used at the rate-controlled fair aggregators. Thus, the delay bound for single aggregation is in the form of (29).

B. Multiple Aggregations

Theorem 7: Suppose flow f traverses K hops of GR servers. Any hop can be an aggregator and all of the aggregators are rate-controlled fair aggregators. For each aggregator, there is a corresponding deaggregator at a later hop. Then, the e2e delay for packet p_f^j, d_f^j, satisfies

$$d_f^j \leq \left[\mathrm{GRC}^1\left(p_f^j\right) - A^1\left(p_f^j\right)\right] + \sum_{i=2}^{K} \frac{\ell_{\hat{A}_i}^{\max}}{\hat{R}_i} + \sum_{i=1}^{M} \frac{\ell_{A_i}^{\max}}{R_i} + \sum_{n=1}^{K} \alpha^n \tag{31}$$

where M is the number of aggregators along the path, $\hat{A}_i$ is the aggregate flow that contains flow f at S_i, $\ell_{\hat{A}_i}^{\max}$ is the maximum packet size in aggregate flow $\hat{A}_i$, $\hat{R}_i$ is the guaranteed rate for $\hat{A}_i$ at S_i, A_i is the ith aggregate along the path that contains flow f, and $\ell_{A_i}^{\max}$ and R_i are the maximum packet size in the aggregate and its guaranteed rate, respectively.

See [21] for a detailed proof. Note that there are a total of $(K-1)+M$ terms related to packet size in the above delay bound. They can be understood as follows: the $(K-1)$ terms in $\sum_{i=2}^{K} \frac{\ell_{\hat{A}_i}^{\max}}{\hat{R}_i}$ correspond to the delays at all of the hops (except the first hop). In addition, for each aggregation with guaranteed rate R_i, there is a term $\frac{\ell_{A_i}^{\max}}{R_i}$ as overhead. Compared to the delay

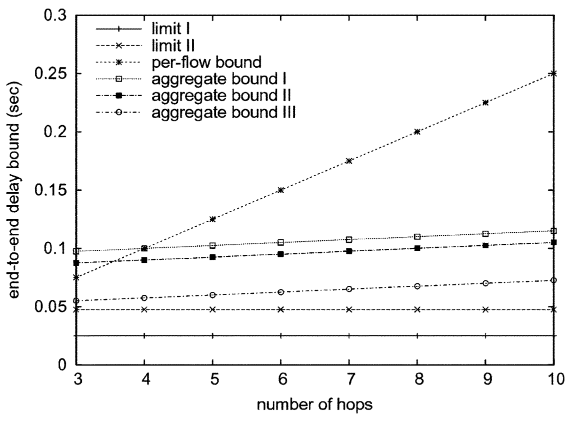


Fig. 4. Comparison of deterministic delay bounds.

bound for per-flow scheduling in (7), (31) has M more terms due to aggregation. However, since the guaranteed rates for aggregate flows are much higher at the routers in the aggregation region, the total delay bound in (31) can be smaller than that in (7). Moreover, if there is only one aggregate $(M = 1)$, and S_1 and S_K are the aggregator and deaggregator, respectively, (31) becomes just (29).

V. Evaluation

In this section, we first use some sample data to numerically calculate the deterministic delay bounds derived so far and compare them with that of per-flow scheduling. Then, we perform extensive simulations to compare the e2e delays of aggregate and per-flow scheduling.

First, the deterministic bounds were compared using the data [5, Table I]: $\sigma_f = 100$ B, $r_f = 32$ Kb/s, and $C^i = 150$ Mb/s. All packets were of the same size, 100 B, and ten identical flows made up an e2e aggregate. To see the effect of the number of hops, we varied K from 3 to 10. As shown in Fig. 4, all of the bounds increase linearly with the number of hops, but those for aggregate scheduling increase much more slowly. When the number of hops is large, the delay bounds of aggregate scheduling are smaller than that of per-flow scheduling. The *per-flow bound* is computed from (7), the *aggregate bounds* I, II, and III are computed from (18), (23), and (30), respectively, and the *limits I* and *II* are computed from $\frac{\sigma_f}{r_f}$ and $\frac{\sigma_f}{r_f} + \frac{\sum_{k \neq f} \sigma_k}{\sum_k r_k}$, respectively, which are independent of the number of hops.

From this example, one can see that, if the degree of burstiness of the flows is not very high, the delay bounds of aggregate scheduling can be smaller than that of per-flow scheduling. In addition, it shows that hierarchical aggregator and rate-controlled hierarchical aggregator can provide even smaller delay bounds. Moreover, hop count plays a significant role in the delay bound under per-flow scheduling. In contrast, it makes much less impact on the delay bounds of aggregate scheduling. The result shows that aggregation is more advantageous if the number of hops is large, which is similar to the conclusion in [28].

Next, extensive simulations were conducted to compare the actual delay performance of aggregate and per-flow scheduling. The objective of our simulation is twofold: 1) compare the deterministic delay bounds with the worst case delay from the simulation and see how tight/loose the deterministic bounds are and 2) compare the worst case delays of aggregate and per-flow scheduling. We intended to find conditions under which aggregate scheduling outperforms per-flow scheduling in terms of e2e delay.

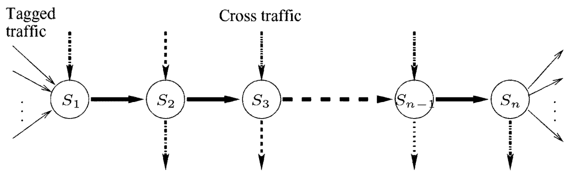


Fig. 5. Simulation topology.

A. Simulation Setup

In the simulation, we used the $ns2$ [29] simulator and the topology in Fig. 5. In this topology, a number of "tagged" flows enter the network at the ingress node S_1 and traverse all of the other nodes until they reach the egress node S_n. The "tagged" flows are those of interest to our study; their e2e delays were checked at the egress node. In order to simulate interference by cross traffic, external traffic was injected at every node on the path. The cross traffic at each node shared the path with the tagged traffic for only one hop before exiting the network at the next hop. For backbone links, we set the bandwidth to 160 Mb/s and the propagation delay to 2 ms, respectively, while for incoming and outgoing links, we set the bandwidth to 10 Mb/s and the propagation delay to 10 ms. The total number of tagged flows was fixed at 128, which was divided into multiple aggregates in the aggregate scheduling cases.

The tagged flows were generated by using a modified CBR model with varying packet and burst sizes. Each incoming tagged flow was shaped by a token bucket. The cross traffic was generated by using the Pareto ON/OFF distribution [30], which can simulate long-range dependencies and is known to be suitable for a large volume of traffic.

The $ns2$ version of weighted fair queueing (WFQ) was used as the GR scheduler at each backbone node for both per-flow and aggregate scheduling. For aggregate scheduling, two versions of aggregator were used: work-conserving stand-alone aggregator and nonwork-conserving rate-controlled (RC) aggregator. To support the nonwork-conserving aggregator, a version of rate-controlled fair queueing was implemented.

To run simulations under different scenarios, we varied several parameters, including the rate of the tagged flows and the number of flows in each aggregate. All of the parameters are summarized in Table II. The main performance metric is the worst case e2e delay. For each scenario, 36 independent runs were conducted. All of the results were plotted with the 95% confidence interval [31].

B. Simulation Results

1) Typical Result: Fig. 6 shows a typical result of e2e delays under three different scheduling algorithms: per-flow FQ, aggregate FQ (which uses the stand-alone work-conserving aggregator), and FIFO queueing. The e2e delay under aggregate FQ is found to be most stable. Aggregate FQ yields not only the smallest worst case delay but also very small delay variation. In contrast, per-flow FQ yields larger worst case delays and delay variations; the delay under FIFO has the largest fluctuation and the worst performance. The above results were obtained by using: burst size 10 000 B; packet size 1000 B; tagged flow rate

TABLE II
PARAMETERS AND THEIR VALUES IN AN ANOVA TEST

Parameters	Values
Tagged flow packet size	100B, 1000B
Cross traffic packet size	100B, 1000B
Tagged flow rate	3.2Kbps, 32Kbps
Hop count	5, 20
Burst size[a] of tagged flows	2, 10
Number of tagged flows in one aggregate	4, 128
Link utilization	5%, 50%
Shape[b] of cross traffic	1.1, 1.9

[a] *The burst size is a relative value: value k means the burst size is k times of the default packet size.*
[b] *The "shape" parameter is used by the Pareto traffic model, which affects the burstiness of the traffic. We use it to vary the burstiness of the cross traffic.*

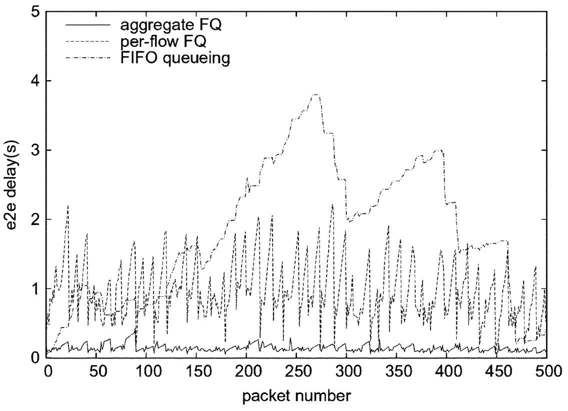


Fig. 6. End-to-end delay comparison.

32 Kb/s; hop count $K = 15$; the number of flows in the aggregate $N = 16$; and link utilization 55%. With these values, we can compute the deterministic bounds from (7) and (18). The bounds turn out to be 6 s for per-flow FQ and 5.53 s for aggregate FQ. Both of the bounds are much larger than the worst case delay found from the simulation, implying that they are rather pessimistic.

2) Homogeneous Case: To find the main factors that affect the delay performance of both aggregate and per-flow scheduling, we first used homogeneous flows in each aggregate. We used the $2^k \cdot r$ *factorial design* method [31] to evaluate the contribution of different parameters. Eight parameters $(k = 8)$ were used, each with two different values. Each scenario was run 36 times $(r = 36)$. The parameters and their values are summarized in Table II.

After collecting data, we used the statistical tool ANalysis Of VAriance (ANOVA) [31] to analyze the significance of each parameter. Our focus was on the value $d_r = \frac{d_a}{d_f}$, where d_a (d_f) is the worst case delay under aggregate (per-flow) scheduling. The intention was to find which parameters affect most the relative delay performance between aggregate and per-flow scheduling. The second parameter—packet size of cross traffic—turned out to have little effect on value d_r. This is easy to explain, since, from the delay bounds in (7), (18), and (30), the packet size of cross traffic shows up only in the term $\frac{L^i_{\max}}{C^i}$, which is usually negligible since C^i is very large. Therefore, we fixed its value at 1500 B in all of the following simulations.

To illustrate the effects of the factors on the e2e delay, we ran multiple sets of simulation. For each set of simulation, we varied only one parameter with all of the others fixed. All of the default values of the parameters are summarized in Table III; the simulation results are summarized in Fig. 7. In Figs. 7 and 8, as well as in the following discussion, "aggregate FQ" stands for aggregate scheduling using stand-alone fair queueing; "RC aggregate FQ" stands for aggregate scheduling using rate-controlled (RC) fair queueing.

Let us compare the performance of per-flow and aggregate FQ first. In Fig. 7(a), as the burst size of the flows increases, the worst case delay under per-flow FQ increases significantly faster than that under aggregate FQ. Surprisingly, the delays of aggregate FQ do not increase as fast as expected. This behavior can be attributed to the multiplexing gain of aggregate scheduling. In other words, although all flows become burstier, the probability that all flows reach their peaks (of load) at the same time is very low. Instead, the peaks and valleys are more likely to be evened out.

In Fig. 7(b), as the flow rate increases, the worst case delay under per-flow FQ decreases faster than that under aggregate FQ. Thus, aggregate FQ is shown to be more advantageous for lower rate flows. In Fig. 7(c) and (d), as the packet size (hop count) increases, the delay under per-flow FQ increases faster than that under aggregate FQ. Therefore, aggregate FQ is more advantageous when the packet size (hop count) is large. In Fig. 7(e), as network link utilization increases, the delay under per-flow FQ increases dramatically faster than that under aggregate FQ, showing that when the network utilization is high, aggregate FQ becomes more advantageous. In Fig. 7(f), as the number of flows in an aggregate increases, the e2e delay under aggregate FQ decreases, while that under per-flow FQ remains unchanged. Aggregate FQ is shown to be more advantageous when the number of flows aggregated gets larger. However, as the number increases, the pace of increase becomes smaller since the margin of multiplexing gain decreases.

The effects of flow rate, packet size, and hop count can be easily explained by the deterministic bounds. Since GR scheduling algorithms are rate-based, the delay of a flow is coupled with its reserved rate. Although aggregate scheduling has the same problem as per-flow scheduling, the reserved rate for an aggregate (R_A) is much larger than that of a single flow (r_f), so the problem is not as severe as in per-flow scheduling. As for packet size, according to (7) and (18), the delay bounds are proportional to the maximum packet size in a flow (aggregate). With aggregation, however, R_A is much larger than r_f. Thus, the delay bound increases much faster under per-flow scheduling. The same holds true for hop count. The effect of link utilization is also related to the coupling problem of GR scheduling. When the link utilization becomes higher, there is less spare bandwidth left. Therefore, the rate for a flow (aggregate) is decreased to its reserved rate. Since R_A is much larger than r_f, aggregate scheduling has less increase in delay. Finally, since per-flow scheduling is independent of the number of flows in an aggregate (N), its delay should not change with N. For aggregate scheduling, from (18), two terms are related to the number of flows N: $\frac{\sum_{k=1}^{N} \ell_k^{\max}}{R}$ and $(K-3)\frac{\ell_A^{\max}}{R}$. Since only identical flows are considered, the first term does not vary with the number of flows. However, R increases proportionally to N.

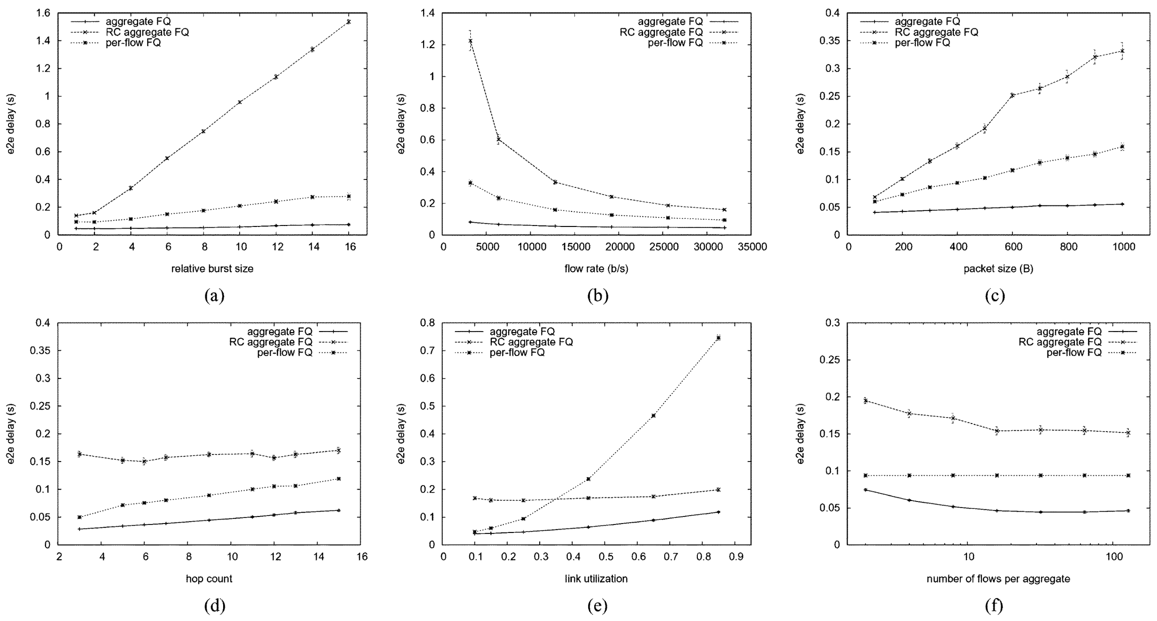


Fig. 7. Comparison of aggregate and per-flow scheduling. (a) Burst size. (b) Flow rate. (c) Packet size. (d) Hop count. (e) Link utilization. (f) Number of flows in each aggregate.

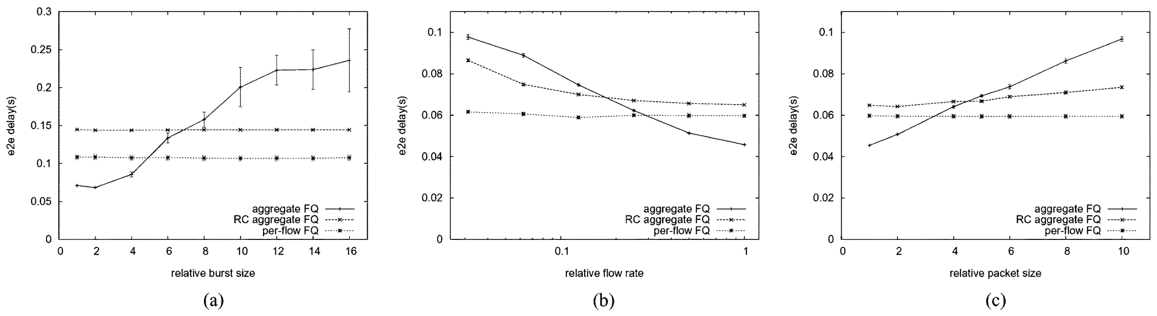


Fig. 8. Aggregation of heterogeneous flows. (a) Larger burst size. (b) Smaller flow rate. (c) Larger packet size.

TABLE III
DEFAULT PARAMETER VALUES

Parameters	Values
Tagged flow packet size	400B
Tagged flow rate	32Kbps
Hop count	10
Burst size of tagged flows	2
Number of tagged flows in each aggregate	16
Link utilization	25%
Shape of cross traffic	1.5
Cross traffic packet size	1500B
Total number of tagged flows	128

Thus, the second term decreases slightly, and the total e2e delay decreases.

Now, consider the results for RC aggregate FQ. In all of the scenarios, the worst-case delays under RC aggregate FQ follow the same trend as those under aggregate FQ. However, since the default link utilization is only 25%, RC aggregate FQ performs worse than aggregate FQ does; in most cases, its performance is even worse than that of per-flow FQ. This is mainly because RC aggregate FQ is nonwork-conserving and does not take advantage of spare bandwidth. On the other hand, as Fig. 7(e) shows, when the link utilization becomes higher, RC aggregate FQ still performs better than per-flow FQ does, and the difference between RC aggregate FQ and aggregate FQ becomes smaller.

We also ran simulations by varying the burstiness of cross traffic. As the burstiness of cross traffic increases, the worst-case delay for per-flow scheduling increases faster than those under the two aggregate FQ cases, showing that aggregate scheduling is more robust to the burstiness of cross traffic than per-flow scheduling.

3) Heterogeneous Case: After studying the effects of different parameters by using homogeneous flows in a traffic aggregate, we mixed heterogeneous flows (in terms of packet size, flow rate, and burst ratio) into an aggregate to determine the type of flows suitable for aggregation. In our simulation, 16 flows were aggregated: 15 of them were identical flows and only one flow was different from the rest. We measured the worst case delay of this flow while varying the parameters of other flows.

Fig. 8(a) shows the result when the small-burst flow (with relative burst size 1) is aggregated with larger burst flows. As the burst size of the other flows increases, the worst case delay under aggregate FQ increases quickly. As expected, mixing a flow with other larger burst flows will hurt its delay performance. Note that the performance of RC aggregate FQ is very stable since the aggregator controls the burstiness of the output aggregate.

TABLE IV
PARAMETERS OF THE VIDEO TRACES

Video Name	Frame Size (KB)		Bit Rate (Kbps)		Burst Size
	Mean	Peak	Mean	Peak	(Kb)
Soccer	5.53	17.93	1,106.6	3,585.4	140
Silence*	0.53	11.29	105.6	2,258.4	70

*Silence of the Lambs

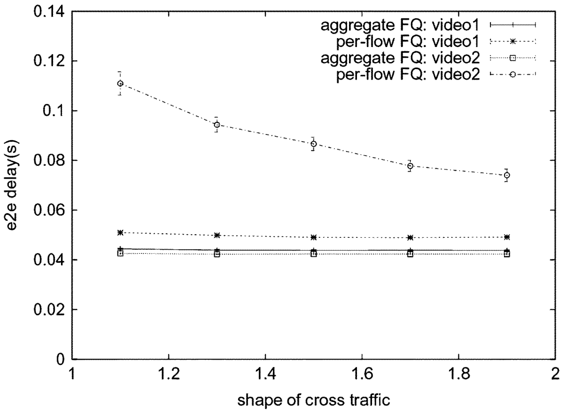


Fig. 9. Delay results for video traces.

Fig. 8(b) plots the result when a high-rate flow (with rate 128 Kb/s) is aggregated with low-rate flows. As the rate of other flows gets smaller, the delay under both aggregate FQ and RC aggregate FQ increases and that under aggregate FQ increases faster and eventually becomes larger than that under per-flow FQ. From (18), one can see that as the rates of other flows decrease, R decreases, and thus, all three terms—$\frac{\sum_{k\neq f}\sigma_k}{R}$, $\frac{\sum_{k=1}^{N}\ell_k^{\max}}{R}$ and $(K-3)\frac{\ell_A^{\max}}{R}$—increase, so the total delay increases. Due to the rate control at the aggregator, the delay increase under RC aggregate FQ is slower.

Fig. 8(c) shows the result when a small-packet flow (with default size 100 B) f_s is aggregated with large-packet flows. When the packet size of other flows becomes larger, the delay performance of f_s is shown to suffer, resulting in an even larger delay than that under per-flow FQ and RC aggregate FQ. This is because when aggregated with large-packet flows, a small packet has to wait behind other large packets in the same aggregate. This implies that flows of similar packet size should be aggregated together.

For the purpose of comparison, we also mixed a flow with other flows with smaller burst size, larger flow rate, and smaller packet size. The results show little or marginal change on the worst case delay of aggregate scheduling. Similarly, the results can be explained with the delay bound in (18) and (30).

4) Case Study With MPEG Traces: To further compare the delay performance of aggregate FQ and per-flow FQ, we also used real MPEG-4 traces [32] in the simulation. Two video traces were used: high-rate "Soccer" and low-rate "Silence of the Lambs." The parameters of these two traces are shown in Table IV. They have very different burstiness, packet rate, and packet rate variation. With the same simulation setup as before, we mixed eight tagged flows driven by the "Soccer" trace and eight other tagged flows driven by the "Silence of the Lambs" trace into an aggregate. The results are summarized in Fig. 9, which shows that, for flows driven by both traces, aggregate FQ provides smaller maximum e2e delays; also, the improvement is much larger for burstier flows driven by the "Silence of the Lambs" trace.

VI. Related Work and Discussions

Aggregate scheduling has been studied extensively in the literature. In [33] and [34], the authors proposed some grouping techniques to optimize the implementation of fair queueing. Certain flows (with similar throughput parameters) are grouped together. For example, the scheme in [34] is confined to ATM networks, in which the routers support only a fixed number of rates, and all of the flows of the same rate are placed into a single group. It takes advantage of the fact that all of the cells have the same size and all of the flows in the same group have the same rate, thus simplifying the sorting of flows. However, it still uses per-flow-based scheduling. Although these two algorithms are sometimes called *aggregate scheduling*, they are different from the aggregate scheduling considered here, because the core routers still recognize *each individual* flow. They are just efficient implementations of per-flow-based fair queueing.

The authors of [35] studied QoS guarantees under aggregation (e2e aggregation was called *grouping*). Similar to our study, fair queueing is used to handle aggregates at core routers (or in aggregation regions). Based on some given e2e delay requirement, they derived the bandwidth and buffer requirements by using the IETF guaranteed service (GS) traffic specification (TSpec) and demonstrated the advantages of aggregating flows of equal or similar delay requirements. By contrast, we derived the e2e delay bound under a given bandwidth guarantee without considering any buffer requirement.

Although the main focus of [22] is QoS routing, it discussed flow aggregation by defining the notion of "*burst ratio*." For flows conforming to the token-bucket model, the burst ratio r is the ratio of the burst size to the average rate or σ/ρ. The authors suggested aggregation of flows of same or similar burst ratio, since flows with the same burst ratio can be merged and divided without changing the burst ratio of the resulting flows. This conclusion is the same as ours in the work-conserving case in Section IV. The authors of [22] analyzed e2e delay for a fluid traffic model, without considering any nonfluid traffic model. In contrast, our delay bound analysis is based on packet traffic model.

Cobb's work [15], [16] is closest to ours. He studied the delay bound problem of aggregate scheduling by using rate-based scheduling algorithms. The core routers treat each aggregate as a single flow and handle all the aggregates by using rate-based fair queueing. He defined a class of fair aggregator and showed that, by using such a fair aggregator, the e2e delay is bounded. Such an aggregator can be used to aggregate the traffic recursively, and the e2e delay bound still exists. He also proposed two types of fair aggregators, called the *basic fair aggregator* and *greedy fair aggregator*. By using such aggregators, he showed that the e2e delay bound can be even smaller than the per-flow e2e delay bound. Our work is more general. In Section III, we showed that the delay bounds exist even for work-conserving aggregators. In Section IV, we extended the result in [16] by allowing multiple aggregates going out of the same link of an aggregator and derived the e2e delay bound under the token bucket traffic model.

A. Diffserv and Aggregate GR Scheduling

The main difference between DiffServ and the proposed aggregate GR scheduling architecture is the way packets in the same class are scheduled. Although it is not specified in the DiffServ standard documents, FIFO queueing is usually used for packet scheduling within each class, thus resulting in coarse-grained control of delay performance.

In the context of DiffServ, the authors of [5] studied the worst case delay bound of FIFO queueing, showing its unsatisfactory performance. For a certain class of traffic, such as EF, the e2e delay bound exists only if the network utilization of that class is sufficiently low. In such a case, the delay bound is a function of the utilization of any link in the network, the maximum hop count of any flow, and the shaping parameters at the network ingress nodes. However, if the utilization is high, the e2e delay could be unbounded. The main reason for this unbounded delay is that the e2e delay experienced by a packet under FIFO queueing is severely affected by the cross traffic. The authors showed that, for real-time traffic such as voice, achieving an acceptable delay bound under FIFO queueing requires the network utilization of such traffic to be rather low.

In aggregate GR scheduling architecture, the definition of traffic aggregates is more flexible and finer grained than the DiffServ traffic classes. Traffic within the same DiffServ class can be further grouped into multiple different aggregate flows that are differentiated in aggregation regions by using the fair queueing algorithms of IntServ. Moreover, at aggregators, flows are bundled into aggregates by more intelligent scheduling schemes, instead of FIFO queueing.

B. Advantages and Disadvantages of Traffic Aggregation

With traffic aggregation, the core routers need to maintain fewer states, making packet classification and scheduling simpler. Flow aggregation is also beneficial to the flows with low bandwidth requirements, because, under GR scheduling algorithms—which couples bandwidth and delay—the flows with low bandwidth requirements will suffer long delays; however, aggregation usually alleviates this problem. More importantly, as shown earlier, aggregate-based GR scheduling can provide guaranteed QoS, such as the e2e delay bound.

However, aggregation also comes with its own disadvantages: the flows in the same aggregate cannot isolate from, and protect against, other flows. Therefore, the QoS guarantee of a flow will be affected by others in the same aggregate, and all of the flows within an aggregate will receive roughly the same service. Due to the lack of isolation within an aggregate, bursty flows can "steal" the bandwidth from well-behaving flows and unduly degrade their QoS. This problem can become worse in multiservice networks [28]. However, as pointed out in [22], the problem can be controlled if flows are selectively aggregated, i.e., only those flows that have some common characteristics are aggregated (which is similar to our conclusion; see the results in Section III). Moreover, "fair" aggregation can also alleviate this problem. In this paper, we have shown this feature by using GR scheduling algorithms to aggregate flows.

VII. Conclusion and Future Work

In this paper, we first derived deterministic delay bounds for aggregates under the assumption that the incoming traffic at each aggregator conforms to the token-bucket model and GR scheduling algorithms are used in each aggregation region. We considered three types of GR scheduling algorithms at an aggregator: stand-alone, two-level hierarchical, and rate-controlled two-level hierarchical GR algorithms. The delay bounds are shown to depend on several factors, such as the scheduling constant at each hop and the latency at the aggregator. We should, therefore, use the scheduling algorithms with a small scheduling constant at each hop and those with small latency at aggregators. Among all of the rate-based scheduling algorithms, PGPS, VC, and WF^{2}Q have the smallest scheduling constant and latency. The delay bounds also indicate that it is beneficial to aggregate flows with similar burst ratios. Aggregate scheduling provides better e2e delay bounds when a large number of hops use aggregate scheduling, because the overhead at the aggregators will be offset by the larger guaranteed rate for an aggregate. If the number of hops is small, the aggregation overhead becomes "relatively" significant.

We also showed by simulation that aggregate scheduling is robust. By exploiting multiplexing gains, it can provide better worst case delay performances than per-flow scheduling, as long as those flows aggregated together are not very diverse in terms of packet size, flow rate, and burst ratio. In addition, the simulation results also showed that, in most scenarios, a nonwork-conserving aggregator performs worse than a work-conserving aggregator, since it does not take advantage of the spare bandwidth in the network. Also, the deterministic bounds are shown to be rather pessimistic. Although the simulation may not capture the worst case e2e delay, it implies that the probability for the worst case to happen be very small.

Note that resource reservation and admission control were not covered in this paper, but techniques in the literature (e.g., [11]) can be used for this purpose. We also assumed that the e2e path in an aggregation region can be set up by traffic engineering mechanisms similar to the way the label switched path (LSP) is set up in an MPLS network.

The large gap between the deterministic delay bound and the worst case delay found from the simulations suggests the need for investigation into the stochastic behavior of aggregate scheduling. We are currently exploring ways to find statistical delay bounds for traffic aggregates. With such bounds, one can admit more flows and enhance network utilization.

References

[1] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: An Overview," IETF, RFC 1633, June 1994.

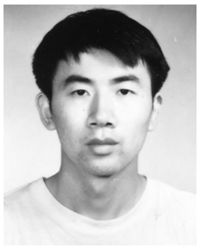
[2] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, no. 10, pp. 1374–1396, Oct. 1995.

[3] S. Blake *et al.*, "An Architecture for Differentiated Services," IETF, RFC 2475, Dec. 1998.

[4] B. Davie *et al.*, "An Expedited Forwarding PHB (Per-Hop Behavior)," IETF, RFC 3246, Mar. 2002.

[5] A. Charny and J.-Y. Le Boudec, "Delay bounds in a network with aggregate scheduling," in *Proc. QofIS*, Oct. 2000, pp. 1–13.

[6] F. Baker *et al.*, "Aggregation of RSVP for IPv4 and IPv6 Reservation,", RFC 3175, Sep. 2001.
[7] J. Ehrensberger, "Resource demand of aggregated resource reservations," in *Proc. ECUMN*, Oct. 2000, pp. 56–61.
[8] O. Schelén and S. Pink, "Aggregating resource reservation over multiple routing domains," in *Proc. IEEE IWQoS*, May 1998, pp. 29–32.
[9] A. Terzis, L. Zhang, and E. L. Hahne, "Reservations for aggregate traffic: Experiences from an RSVP tunnels implementation," in *Proc. IEEE IWQoS*, May 1998, pp. 23–25.
[10] B.-K. Choi *et al.*, "Scalable QoS guaranteed communication services for real-time applications," in *Proc. IEEE ICDCS*, Apr. 2000, pp. 180–187.
[11] H. Fu and E. W. Knightly, "Aggregation and scalable QoS: A performance study," in *Proc. IEEE IWQoS*, June 2001, pp. 39–50.
[12] S. Berson and S. Vincent, "Aggregation of internet integrated services state," in *Proc. IEEE IWQoS*, May 1998, pp. 26–28.
[13] H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," in *Proc. ACM SIGCOMM*, Aug. 1991, pp. 113–121.
[14] T. Li and Y. Rekhter, "A Provider Architecture for Differentiated Services and Traffic Engineering (PASTE)," IETF, RFC 2430, Oct. 1998.
[15] J. A. Cobb, "Preserving quality of service guarantees in spite of flow aggregation," in *Proc. IEEE ICNP*, Oct. 1998, pp. 90–97.
[16] ——, "Preserving quality of service guarantees in spite of flow aggregation," *IEEE/ACM Trans. Netw.*, vol. 10, no. 1, pp. 43–53, Feb. 2002.
[17] P. Goyal, S. S. Lam, and H. M. Vin, "Determining end-to-end delay bounds in heterogeneous networks," in *Proc. NOSSDAV*, Apr. 1995, pp. 287–298.
[18] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 4, pp. 611–624, Aug. 1998.
[19] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, Jun. 1993.
[20] L. Zhang, "Virtual clock: A new traffic control algorithm for packet-switched networks," *ACM Trans. Computer Syst.*, vol. 9, no. 2, pp. 101–124, May 1991.
[21] W. Sun and K. G. Shin, "End-to-End Delay Bounds for Traffic Aggregates Under Guaranteed Rate Scheduling Algorithms," Dept. Elect. Eng. Comput. Sci., Univ. of Michigan, Ann Arbor, Tech. Rep. CSE-TR-484-03, 2003.
[22] S. Vutukury and J. Garcia-Luna-Aceves, "A scalable architecture for providing deterministic guarantees," in *Proc. IEEE IC3N*, Oct. 1999, pp. 534–539.
[23] J. C. R. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," in *Proc. ACM SIGCOMM*, Aug. 1996, pp. 143–156.
[24] ——, "WF^{2}Q: Worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, Mar. 1996, pp. 120–128.
[25] J.-Y. Le Boudec, "Application of network calculus to guaranteed service networks," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 1087–1096, May 1998.
[26] H. Zhang and D. Ferrari, "Rate-controlled service disciplines," *J. High Speed Networks*, vol. 3, no. 4, pp. 389–412, 1994.
[27] S. Keshav, *An Engineering Approach to Computer Networking*. Reading, MA: Addison Wesley, 1997.
[28] K. Dolzer, W. Payer, and M. Eberspächer, "A simulation study on traffic aggregation in multi-service networks," in *Proc. IEEE Conf. High Performance Switching and Routing*, Jun. 2000, pp. 157–165.
[29] ns2 Simulator [Online]. Available: http://www.isi.edu/nsnam/ns/
[30] A. Popescu, "Traffic Self-Similarity," Blekinge Institute of Technology, Karlskrona, Sweden, white paper, 1999.
[31] R. Jain, *The Art of Computer Systems Performance Analysis*. New York: Wiley, 1991.
[32] F. Fitzek and M. Reisslein, "MPEG-4 and H.263 video traces for network performance evaluation," *IEEE Network*, vol. 15, no. 6, pp. 40–54, Nov./Dec. 2001.
[33] J. L. Rexford, A. G. Greenberg, and F. G. Bonomi, "Hardware-efficient fair queueing architectures for high-speed networks," in *Proc. IEEE INFOCOM*, Mar. 1996, pp. 638–646.
[34] J. C. R. Bennett, D. C. Stephens, and H. Zhang, "High speed, scalable, and accurate implementation of packet fair queueing algorithms in ATM networks," in *Proc. IEEE ICNP*, Oct. 1997, pp. 7–14.
[35] J. Schmitt *et al.*, "Aggregation of guaranteed service flows," in *Proc. IEEE IWQoS*, Jun. 1999, pp. 147–155.



Wei Sun (S'99) received the B.S. degree in mathematics from Nankai University, Tianjin, China, in 1992, the M.S. degree in computer science from Tsinghua University, Beijing, China, in 1995, and the M.S. degree in computer science from The Ohio State University, Columbus, in 1999. He is currently working toward the Ph.D. degree in computer science at the University of Michigan, Ann Arbor.

His research interests include Quality of Service, packet scheduling, and Internet routing.

Mr. Sun has been a member of the Association for Computing Machinery since 1999.

Kang G. Shin (S'75–M'78–SM'83–F'92) received the B.S. degree in Electronics Engineering from Seoul National University, Seoul, Korea in 1970, and both the M.S. and Ph.D. degrees in Electrical Engineering from Cornell University, Ithaca, New York in 1976 and 1978, respectively.

He is the Kevin and Nancy O'Connor Professor of Computer Science and Founding Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor. His current research focuses on QoS-sensitive networking and computing as well as on embedded real-time OS, middleware, and applications, all with emphasis on timeliness and dependability. He has supervised the completion of more than 50 Ph.D. dissertations and authored or co-authored approximately 600 technical papers and numerous book chapters. He has co-authored (with C. M. Krishna) the textbook *Real-Time Systems* (New York: McGraw-Hill, 1997).

Prof. Shin was the recipient of a number of best paper awards, including the IEEE Communications Society William R. Bennett Prize Paper Award in 2003, the Best Paper Award from the IWQoS'03 in 2003, and an Outstanding IEEE TRANSACTIONS OF AUTOMATIC CONTROL Paper Award in 1987. He has also coauthored papers with his students which received the Best Student Paper Awards from the 1996 IEEE Real-Time Technology and Application Symposium, and the 2000 USENIX Technical Conference. He was also the recipient of several institutional awards, including the Research Excellence Award in 1989, the Outstanding Achievement Award in 1999, the Service Excellence Award in 2000, the Distinguished Faculty Achievement Award in 2001, and the Stephen Attwood Award in 2004 from The University of Michigan, along with the Distinguished Alumni Award of the College of Engineering, Seoul National University in 2002 and the 2003 IEEE RTC Technical Achievement Award.