Coordinated Aggregate Scheduling for Improving End-to-End Delay Performance

Wei Sun and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
{*wsunz, kgshin*}*@eecs.umich.edu*

***Abstract*— We propose a novel *coordinated aggregate scheduling* (CAS) algorithm that combines both EDF (Earliest-Deadline-First) scheduling and rate-based fair queueing. CAS uses guaranteed rate (GR) scheduling [1] for traffic aggregates at the *inter*-aggregate level, but employs EDF-like scheduling at the *intra*-aggregate level. Computation of the deadline D_N of a packet at an intermediate node N is coordinated between the node N and its upstream nodes, and D_N is related to the packet's guaranteed rate clock (GRC) value at the flow-aggregation node. CAS provides tighter end-to-end (e2e) delay bounds than the "vanilla" GR aggregate scheduling that relies on FIFO queueing within an aggregate. Our in-depth simulation results demonstrate CAS's superior performance. Moreover, as an aggregate-based work-conserving scheduling algorithm, CAS incurs lower scheduling and state-maintenance overheads at routers than per-flow scheduling. These salient features make CAS very attractive for use in Internet core networks.**

I. Introduction

Real-time applications, such as voice-over-IP (VoIP) and video conferencing, require the network to provide better Quality-of-Service (QoS) than the currently dominant best-effort service, in terms of delay, jitter, and loss rate. To provide such QoS support, the IntServ architecture [2] has been proposed, which supports QoS via *per-flow* resource reservation (e.g., RSVP [3]) and packet scheduling. Numerous scheduling algorithms (e.g., see [4] for an excellent survey) have been proposed to support IntServ-like QoS, such as fairness, bounded per-flow (per-node or e2e) delay and backlog under a certain traffic model like the token bucket model. One class of such scheduling algorithms are called *guaranteed rate* (GR) scheduling algorithms [1], which include WFQ (Weighted Fair Queueing), GPS (Generalized Process Sharing) [5], VC (Virtual Clock) [6], etc. Since both its 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 does not scale well for use in the core of the Internet that carries millions of flows.

To solve the IntServ's scalability problem, an *aggregate scheduling architecture* has been proposed [7], [8], [9]. Its main idea is to extend 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 within an aggregation region are done on a per-aggregate basis. Within an aggregation region, those flows "bundled" together are treated as a *single* logical flow at each router on the path. Like a *traffic trunk*—which is defined as an aggregate of traffic that belongs to the same class—in Multiprotocol Label Switching (MPLS) [10], [11], a traffic aggregate can be created and terminated at any point in the network, and can also be created recursively (i.e., flows are aggregated/deaggregated multiple times). The ingress router that aggregates flows (using GR algorithms) is called an *aggregator*, and the egress router that splits the aggregate is called a *deaggregator*. The term "flows" ("traffic aggregates" or "aggregates") means the entities before (after) aggregation.

Similarly to the DiffServ architecture [12], this aggregate scheduling architecture pushes the overhead to the edge of the network and keeps the core network simple. The admission control and resource reservation of aggregate scheduling can be implemented by the extension of RSVP that supports traffic aggregation [13]. The idea is to use aggregate *Path* and *Resv* messages between the aggregator and the deaggregator, and hide the e2e per-flow RSVP messages from the core routers in the aggregation region. Aggregate Path messages are sent from the aggregator to the deaggregator, and aggregate Resv messages are sent from the deaggregator to the aggregator, thus establishing the aggregate reservation on behalf of the flows within the same aggregate. Per-flow RSVP messages trigger the transmission of aggregate Path or Resv messages, but these messages themselves are ignored inside the aggregation region. This results in a smaller number of (aggregate) reservations inside the aggregation region. To reduce the number of changes to aggregate reservations, advance reservation or "bulk" reservation is needed. For example, the authors of [8] examined the issue of "bulk" reservation for traffic aggregates.

This paper focuses on packet scheduling for traffic aggregates. We assume that IntServ's GR algorithms are used in this aggregate scheduling architecture. The edge routers (aggregators and deaggregators) use per-flow scheduling, while the core routers use per-aggregate scheduling.

Using aggregate GR scheduling algorithms, we [9] derived

0-7803-8277-3/04/$20.00 ©2004 IEEE.

deterministic e2e delay bounds under the assumption that all incoming flows at an aggregator conform to the token bucket model. Each aggregator uses a GR scheduling algorithm which is either work-conserving or non-work-conserving. We showed not only the existence of e2e 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 tighter than that of per-flow scheduling. The simulation results have shown that aggregate scheduling is very robust and can exploit statistical multiplexing gains, and that it performs better than per-flow scheduling in most cases.

However, when work-conserving scheduling algorithms are used at aggregators, the delay bound of a flow is dictated by the burstiness of other flows in the same aggregate. Thus, if a flow is aggregated with other bursty flows, it will suffer a long e2e delay. The main culprit of this long delay lies in the fact that packets within each aggregate are handled by the routers using FIFO scheduling.

To remedy this problem, we propose a new aggregate scheduling algorithm, which improves the e2e delay by re-ordering the packets in an aggregate using EDF scheduling based on their deadlines. The deadline of a packet at an intermediate node is related to its guaranteed rate clock (GRC) value at the aggregator. This new algorithm makes the delay of a flow independent of the burstiness of other flows in the same aggregate, yielding smaller e2e delays. Since the algorithm uses EDF inside an aggregate and computation of the deadline of a packet at each intermediate node is coordinated between the node itself and its upstream nodes, we call it *coordinated aggregate scheduling* (CAS). For the purpose of differentiation, we call the aggregate scheduling algorithms discussed in [9] that use FIFO queueing inside each aggregate *"vanilla" aggregate scheduling* (VAS).

The rest of the paper is organized as follows. For self-containment, Section II reviews the definitions of GR scheduling and the delay bound results for per-flow scheduling in [1] and aggregate scheduling in [9]. Section III introduces the CAS algorithm, and proves that under the token bucket traffic model CAS provides tighter e2e delay bound than VAS. Section IV discusses the implementation issues of CAS, and proposes a simple multi-queue structure and an adaptive queue management algorithm. Section V presents evaluation results. Simulation is used to compare the delay performance of both coordinated and vanilla aggregate scheduling, confirming the benefits of CAS derived from the analysis. Section VI discusses related work on CAS, putting our results in a comparative perspective. Finally, Section VII summarizes our contributions.

II. Guaranteed-Rate Scheduling Algorithms

Before presenting the CAS algorithm, we first review the definition of *Guaranteed-Rate* (GR) scheduling and the e2e delay bound results for per-flow and aggregate GR scheduling.

A. GR Scheduling Algorithms

The authors of [1] 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.

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 j^{th} packet of flow f and its length, respectively. Also, let $GRC^i(p_f^j)$ 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:

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

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 packet p_f^j to be transmitted by time $GRC^i(p_f^j) + \beta^i$, where β^i is a scheduling constant [7] that depends on the scheduling algorithm and the router.

Many scheduling algorithms are shown in [1] to belong to the GR class. For example, both Packet-level Generalized Processor Sharing (PGPS) [5] and Virtual Clock (VC) [6] 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 the output link capacity of S_i.

B. End-to-End Delay Bound under Per-Flow Scheduling

We now review the e2e delay bound results for per-flow scheduling. See [1] for the proofs of both Lemma 1 and Theorem 1 stated below. In the following discussion, we will call a router equipped with the GR scheduling algorithm a *GR server*.

Lemma 1: 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

$$GRC^{i+1}(p_f^j) \leq GRC^i(p_f^j) + \frac{\ell_f^{max}}{r_f} + \alpha^i, \quad (2)$$

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

Lemma 1 states the relationship between the GRC values of a packet at two neighboring GR servers. Based on this relationship, the authors of [1] derived an e2e delay bound. Before introducing that delay bound, we define the token bucket traffic model as follows: flow f is said to conform to the token bucket (σ_f, ρ_f) if for any time instant τ and t such that $0 \leq \tau < t$, its traffic volume arrived in the time interval $(\tau, t]$, denoted as $A_f(\tau, t)$, satisfies:

$$A_f(\tau, t) \leq \sigma_f + \rho_f \cdot (t - \tau), \quad (3)$$

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

Theorem 1: If flow f conforms to the token bucket model (σ_f, ρ_f) and all the routers on its path are GR servers with

rate $r_f \geq \rho_f$, then the e2e delay for p_f^j, d_f^j, is bounded as follows:

$$d_f^j \leq \frac{\sigma_f}{r_f} + \frac{(K-1)\ell_f^{max}}{r_f} + \sum_{n=1}^{K} \alpha^n, \quad (4)$$

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

From Theorem 1, 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. In case of a large number of hops and large-size packets, the delay can be substantially large. In addition, the larger the burst size, the larger the delay bound becomes.

C. End-to-End Delay Bounds under Aggregate Scheduling

In [9], we derived e2e delay bounds under aggregate GR scheduling. When incoming flows at aggregators conform to the token bucket model, the e2e delay is proven to be bounded.

Theorem 2: Suppose N flows share the same K hops of GR servers inside an aggregation region, and they are bundled into an aggregate A at a work-conserving stand-alone aggregator S_1 and split back at S_K. Routers $S_2, \ldots, S_{K-1}$ schedule the packets of aggregate A. If flow k conforms to the token bucket model (σ_k, ρ_k) and has the guaranteed rate $r_k \geq \rho_k$ $(1 \leq k \leq N)$ 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:[1]

$$d_f^j \leq \frac{\sigma_f}{r_f} + \left[\frac{\sum_{k \neq f} \sigma_k}{R} + \frac{\sum_{k \neq f} \theta_k^1 \cdot r_k + \ell_f^{max}}{R}\right] + (K-3)\frac{\ell_A^{max}}{R} + \frac{\ell_f^{max}}{r_f} + \sum_{i=1}^{K} \alpha^i, \quad j \geq 1. \quad (5)$$

Comparing Eqs. (4) and (5), we can see that under aggregate scheduling, the delay bound of a flow is not only dictated by the burstiness of the flow itself (term $\frac{\sigma_f}{r_f}$), but also strongly related to the burstiness of other flows in the same aggregate that the flow f belongs to (term $\frac{\sum_{k \neq f} \sigma_k}{R}$). This is easy to see: since FIFO queueing is used inside each aggregate, a packet of flow f has to wait behind not only its own preceding packets, but also packets of other flows in the same aggregate. This result suggests how flows should be aggregated—a flow should not be aggregated with other flows with substantially larger burst sizes.

Also, comparison of Eqs. (4) and (5) shows that, depending on the burstiness of the constituent flows and the maximum packet size in an aggregate, the delay bound under aggregate scheduling can be tighter than that under per-flow scheduling.

III. Coordinated Aggregate Scheduling with EDF inside an Aggregate

As discussed above, the delay bound under aggregate scheduling depends on the burst sizes of all the other flows in the same aggregate, mainly because at the downstream nodes of the aggregator, the packets in the same aggregate are handled with FIFO queueing, irrespective of which flows they belong to. Thus, if a large burst of packets of a flow arrive at an idle aggregator, the burst can traverse the aggregator very quickly. If packets from other flows in the same aggregate arrive immediately after the burst, they must wait behind the burst in the FIFO queue at the remaining nodes of the aggregate's path. Figs. 1 (a) and (b) illustrate this scenario. In Fig. 1 (a), a burst of flow 1 arrives at an aggregator S_1 immediately before the arrival of a packet from flow 2. Suppose S_1 is idle at that time, then the burst goes through the aggregator very quickly, as does the packet from flow 2. Since both flows share the same aggregate, the burst of flow 1 will be ahead of the flow-2 packet in the aggregate until the aggregate is split later on the path. Suppose the next node S_2 is the bottleneck, then the burst of flow 1 will be scheduled at a slower speed, and the flow-2 packet has to wait behind the burst of flow 1 in the FIFO queue, suffering a long delay. To solve this problem, one may use a rate-controlled scheduler at the aggregator S_1, thus controlling the output rate for the aggregate, and there will be no large traffic burst waiting at the downstream nodes. However, rate-controlled schedulers work in a non-work-conserving fashion, causing a longer average delay.

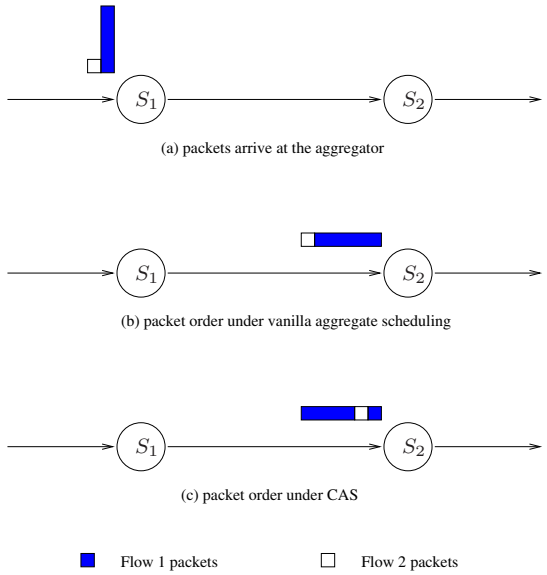


Fig. 1. Coordinated aggregate scheduling

We propose another method to solve the delay problem by changing the order of transmitting packets in the queue of an aggregate, such that higher-priority packets can be scheduled earlier. In essence, we change the FIFO queue for an aggregate used by vanilla aggregate scheduling (VAS) algorithms into an EDF queue. The rationale behind this is that, if the aggregator is relatively lightly-loaded, a large burst of packets can get through it earlier than their expected finish times according to their GRC values. If a downstream node becomes bottleneck,

[1]The term θ_k^1 represents the latency of flow k at server S_1. It is a parameter of *Latency-Rate* (LR) server defined in [14]. In [9] we proved that 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.

this burst will stay ahead of some packets with smaller GRC values in the queue of that node. By reordering the packets in the queue based on their GRC values at the aggregator, if multiple packets of an aggregate are waiting in the queue of a downstream node, a packet with the minimum GRC value will be scheduled first. In other words, the GRC value at the aggregator plays the role of a packet's deadline. Note that this algorithm only reorders the packets in the queue of the same aggregate; the scheduling of different aggregates remains the same—the GR scheduling algorithms. This new algorithm is called the *coordinated aggregate scheduling* (CAS). As shown in Fig. 1 (c), after reordering the packets in the queue at S_2, the packet from flow 2 will be scheduled earlier.

Next, we give the details of the CAS algorithm and prove that this algorithm improves the e2e delay bound of a flow. Note that when 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., $r_f \geq \rho_f$.

A. CAS Algorithm

For the downstream nodes to be able to reorder packets, the packets have to carry some information about their GRC values at the aggregator. The key idea of the CAS algorithm is to insert a *lag* field in each packet, which contains information on how much the packet was behind its "deadline" at the previous hop. Then, at the next hop the server can adjust the packet's arrival time by subtracting its lag value. The order of transmitting packets in the waiting queue of an aggregate will be adjusted according to the new "virtual" arrival times.

Now, let us consider the scheduling algorithm at both the aggregator and the downstream nodes. For simplicity, the propagation delay between neighboring nodes is omitted in the following discussion. At the aggregator (server S_i), per-flow scheduling is used. Thus, for flow f, $GRC^i(p_f^j)$ is defined based on its reserved rate r_f according to Eq. (1). Let $\delta^i(p_f^j)$ and $D^i(p_f^j)$ be the lag value and the departure time of packet p_f^j at S_i, respectively. Then,

$$\delta^i(p_f^j) = D^i(p_f^j) - GRC^i(p_f^j). \tag{6}$$

At the downstream nodes S_{i+k} ($k \geq 1$), the lag value $\delta^{i+k}(p_f^j)$ is defined differently:

$$\delta^{i+k}(p_f^j) = D^{i+k}(p_f^j) - VA^{i+k}(p_f^j), \quad k \geq 1. \tag{7}$$

The virtual arrival (VA) time $VA^{i+k}(p_f^j)$ is calculated recursively as:

$$\begin{aligned} VA^{i+k}(p_f^j) &= A^{i+k}(p_f^j) - \delta^{i+k-1}(p_f^j) \\ &= A^{i+k}(p_f^j) - (D^{i+k-1}(p_f^j) - VA^{i+k-1}(p_f^j)) \\ &= VA^{i+k-1}(p_f^j) \\ &\vdots \\ &= GRC^i(p_f^j), \quad k \geq 1. \end{aligned} \tag{8}$$

We assume $A^{i+k}(p_f^j) = D^{i+k-1}(p_f^j)$ since the propagation delay is omitted. From Eq. (8), the virtual arrival time of a packet at downstream nodes is exactly the same as the packet's GRC value at the aggregator. Thus, at every node after the aggregator, the packets are ordered in their GRC values at the aggregator. (A simple implementation is to just store the GRC value in the packet at the aggregator. However, to remove the need for clock synchronization, we store the lag value and adjust the virtual arrival time at each downstream hop, as is done in the above algorithm.) Moreover, the algorithm also preserves the order of packets in the same flow, since

$$\begin{aligned} VA^{i+k}(p_f^j) = GRC^i(p_f^j) &\geq GRC^i(p_f^{j-1}) + \frac{\ell_f^j}{r_f} \\ &= VA^{i+k}(p_f^{j-1}) + \frac{\ell_f^j}{r_f} \\ &> VA^{i+k}(p_f^{j-1}). \end{aligned} \tag{9}$$

Also, the "deadline" of a packet is defined differently at different nodes: at the aggregator, the packet's deadline is its GRC value, while at the downstream nodes, the deadline is its virtual arrival time, not its GRC value (or expected finish time) at those nodes. The reason for this is that we want to re-adjust the order of packets in the same aggregate to keep them in the order of GRC values at the aggregator. Using the virtual arrival time achieves this goal.

Based on the above analysis, we can prove that CAS provides tighter e2e delay bounds than VAS. Due to the lack of space, the proof of the following theorem is omitted.

Theorem 3: Under the same conditions of Theorem 2, the e2e delay bound of a packet under CAS is tighter than that under VAS.

IV. Implementation

Theoretically, we have shown that CAS has very good e2e delay performance. However, since CAS employs EDF scheduling at core nodes, its implementation overhead can be a concern. Therefore, we need to find an efficient algorithm for packet sorting.

The calendar queue [15] has been widely used for packet sorting. It is shown to have $O(1)$ complexity. However, depending on the number of packets in the queue, the calendar queue has to do a lot of resizing and copying to maintain the "optimal" calendar structure, i.e., not many packets in each bucket nor many empty buckets. Resizing and copying incur significant overhead. Moreover, the calendar queue does not work well over "skewed" priority distribution.

Since in the CAS algorithm, fair queueing is used at each aggregator, the virtual arrival (VA) time values of packets in each aggregate are likely to be monotonically increasing. Thus, one FIFO queue is almost enough to handle them. Only a few more queues are necessary to handle those packets that have smaller VA values (higher priority) than their predecessors.

Based on the observations above, we propose a simple multi-queue structure that consists of a main queue and multiple accessory queues. The number of accessory queues varies with the VA values and actual arrival times of packets. A newly-arrived packet is put at the end of the main queue as

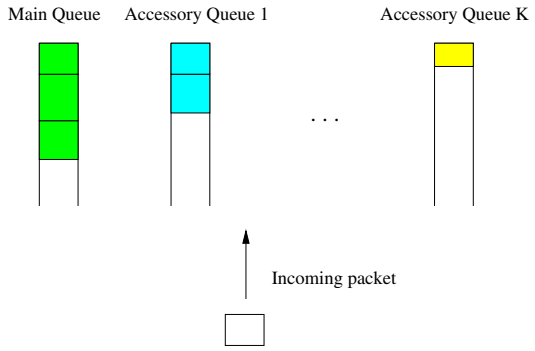


Fig. 2. The multi-queue structure

long as its VA value is larger than that of the last packet in the queue; otherwise, the VA value of the packet is compared to that of the last packet in the first accessory queue. Similarly, if its VA value is larger, the packet will be put at the end of the queue; otherwise, its VA value is compared with that of the last packet at the next accessory queue. The process continues until either the packet is put at the end of an existing queue, or (when its VA value is smaller than that of the last packet of the last accessory queue) a new accessory queue will be created where the packet will be placed. It is easy to see that the VA values of the last packets in the queues are monotonically decreasing—the one in the main queue is larger than that in the first accessory queue; the one in the first accessory queue is larger than that in the second accessory queue, etc.

When a packet from the aggregate needs to be transmitted, the packet with the minimum VA value is chosen. It can be the first packet of any of the existing queues. When an accessory queue becomes empty, it will be deleted; on the other hand, the main queue will always remain there, even if it gets empty.

Compared to the calendar queue, our simple multi-queue scheme has the following advantages: (i) it is simpler, avoiding the "copy" and "resize" operations of the calendar queue when the calendar structure is adjusted; (ii) the clustering problem (due to "skewed" distribution of packet priority) of the calendar queue is avoided; and (iii) the number of queues is independent of the total number of backlogged packets of an aggregate.

In addition, there are several interesting properties associated with the queue operations. First, since packets are scheduled according to their VA values, the accessory queues will be deleted in the reverse order of their creation. In other words, the most recently created queue will be deleted first, since its last packet has higher priority (a smaller VA value) than all the last packets in other queues. In this sense the accessory queues works as a LIFO stack. Second, the packets of a given flow will keep their order. In other words, packets in the same flow will never be reordered. Therefore, the number of queues cannot be greater than the total number of flows in the aggregate. Clearly, a new packet is put into a new queue only when it belongs to a different flow from the last packets in all the existing queues; otherwise, it will have a larger VA

TABLE I

PSEUDOCODE OF THE ADAPTIVE ALGORITHM

N_q:	The current number of queues used by the aggregate;
δ_c:	The default constant;
length[i]:	The length of queue i;
VA_{max}[i]:	The maximum value of VA in queue i;
VA(p_A^j):	The virtual arrival time of packet p_A^j;

1. *//Upon arrival of a new packet* p_A^j:
  i = 0;
  WHILE (VA(p_A^j) $<$ VA_{max}[i] - δ **AND** i $<$ N_q)
    i++;
  IF (i $\geq$ N_q) **THEN**
    N_q++;
    δ += δ_c;
  Insert p_A^j at the end of queue i;
  length[i]++;
  IF (VA(p_A^j) $>$ VA_{max}[i]) **THEN**
    VA_{max}[i] = VA(p_A^j);

2. *//Upon departure of a packet* p_A^j:
  //Suppose p_A^j is from queue i;
  Remove p_A^j from queue i;
  length[i]--;
  IF (i = N_q-1 **AND** length[N_q-1] = 0) **THEN**
    N_q--;
    δ -= δ_c;
    VA_{max}[N_q-1] = 0;

value than at least one of them. Third, the packets in the same queue are in increasing order of VA values.

To have small overhead, the number of accessory queues should be small. In fact, the number of queues required is related to several factors, such as the burstiness of flows, network utilization, etc. We expect the average number of queues to be small, since VA values of incoming packets tend to be monotonically increasing.

To further reduce the number of accessory queues, the following optimizations can be incorporated into the basic multi-queue algorithm.

Optimization 1: if a queue is short (e.g., of length one/two or shorter than the number of accessory queues), the new packet will be inserted into it.

Optimization 2: a small discrepancy is allowed, so that if a packet's VA value is only a little bit (δ) smaller than that of the last packet in the queue, it will be put behind that packet in the queue without moving to the next accessory queue.

However, this optimization can be recursively done so that the VA values of the packets in a queue may be in a reverse order. To solve this problem, the algorithm is revised by recording the maximum value of VA in each queue. A newly-arrived packet compares its VA with this maximum value, not the VA value of the last packet.

Then, the question is what value to use for δ. If it is large, the number of queues will be small, but the delay performance will suffer; if it is too small, then the number of queues will increase. We design a simple adaptive algorithm, which uses the number of accessory queues as a parameter in determining

the δ value. Suppose the current number of accessory queues is N_q, then we set $\delta = N_q \cdot \delta_c$, where δ_c is a small constant. Thus, if the number of accessory queues is large, we use a larger δ to make the number of queues smaller; when it is small, we use a smaller δ to make the ordering in each queue more accurate, improving the delay performance. The pseudocode of the adaptive algorithm is presented in Table I.

The δ_c value is directly related to the delay. When it is 0.01, for instance, the actual delay discrepancy would be in the range of 0.01 – 0.1sec per hop, if the number of queues is less than 10. In our simulation, δ_c is set to 0.01, 0.02, or 0.05.

With Optimization 2, the packets in a queue are no longer in increasing order of their VA values. This fact leads to the following optimization to reduce the overhead in choosing the next packet to transmit.

Optimization 3: Suppose a packet from queue i is chosen when a packet is to be transmitted from an aggregate, then the next time a packet is to be transmitted from the aggregate, another packet from queue i will be chosen (without searching) as long as the VA value of the first packet in queue i is smaller than that of the previous packet. This is because the first packets in other queues must have larger VA values. This can reduce the search effort during packet transmission from an aggregate. This optimization has to be used together with Optimization 2.

V. Evaluation

To demonstrate the advantages of the CAS algorithm, we conducted extensive simulations using the *ns2* [16] simulator, especially comparing the e2e delays of CAS and VAS.

A. The Simulation Setup

In the simulation, we used the topology shown in Fig. 3 where a number of "tagged" flows enter the network through the ingress node S_1, and traverse all the other nodes until they reach the egress node S_n. The "tagged" flows are the ones of interest to our study, and their e2e delays are checked by the egress node. In order to simulate interferences by cross-traffic, external traffic is injected at every node on the path. The cross-traffic at each node shares the path with the tagged traffic for only one hop before exiting the network at the next hop. For the backbone links, we set the bandwidth to 160Mbps and the propagation delay to 2ms, respectively, while for the incoming and outgoing links, we set the bandwidth to 10Mbps and the propagation delay to 10ms.

The tagged flows are generated by using a modified CBR model with varying packet and burst sizes. Each tagged incoming flow is shaped by a token bucket. The cross traffic is generated by using the Pareto On/Off distribution [17], [18], which can simulate long-range dependencies and is known to be suitable for a large volume of traffic.

To verify the performance of CAS, we used two fair queueing algorithms—WFQ (Weighted Fair Queueing) and WF^{2}Q (Worst-case Fair Weighted Fair Queueing) [19]. The

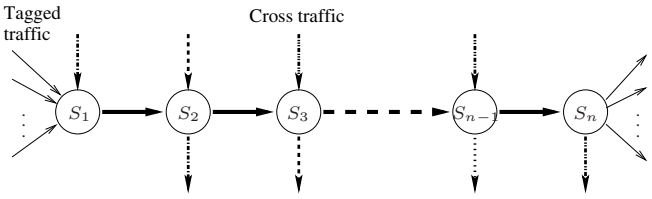


Fig. 3. Simulation topology

TABLE II
Parameters and their default values

Parameters	Values
Tagged flow packet size	400B
Cross traffic packet size	1500B
Tagged flow rate	32Kbps
Hop count	10
Burst size of tagged flows (red / blue)	2 / 8
Total number of tagged flows	128
Number of tagged flows in one aggregate	16
Link utilization	55%

ns2 versions of WFQ and WF^{2}Q were used as the GR scheduler at each backbone node. Both algorithms were modified to support VAS and CAS. For each simulation scenario, we ran simulation to obtain two independent results using WFQ and WF^{2}Q. Each simulation run lasted 50 seconds.

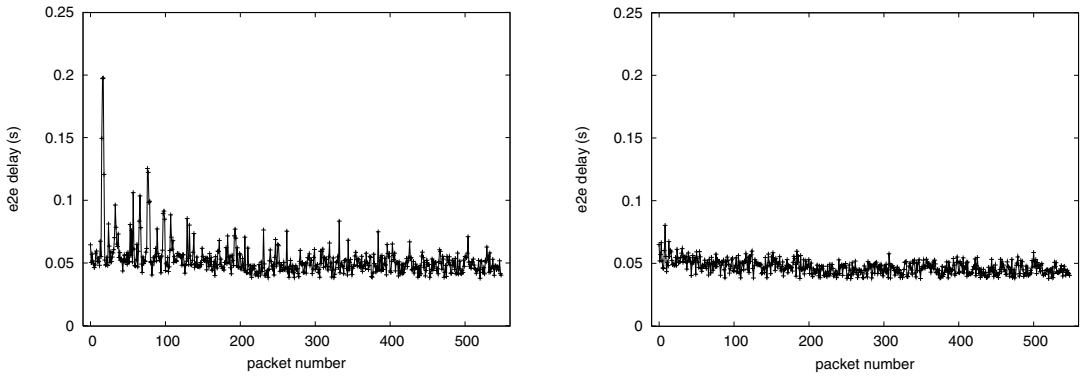
In the simulation, the tagged flows were divided into multiple groups. In each group, one flow (called *red* flow) has a fixed small burst size [2] 2, while all the other flows (called *blue* flows) have variable burst sizes (with the default value 8). Each group of flows were bundled into an aggregate flow at the aggregator. We focused on the e2e delay of the red flows to see the delay performance of flows under different scheduling schemes and network conditions. All the parameters used in the simulation and their default values are summarized in Table II.

B. Simulation Results

First, we compared the e2e delay of the red flows under VAS and CAS. Fig. 4 shows the result of one red flow when WFQ is used as the GR scheduling algorithm. As can be seen from the figure, CAS yields not only a smaller worst-case delay but also a very small delay variation. This confirms the analysis results in Section III. We repeated the simulation using the WF^{2}Q algorithm, the results of which are plotted in Fig. 5. As can be seen from this figure, the performance of CAS is consistently superior.

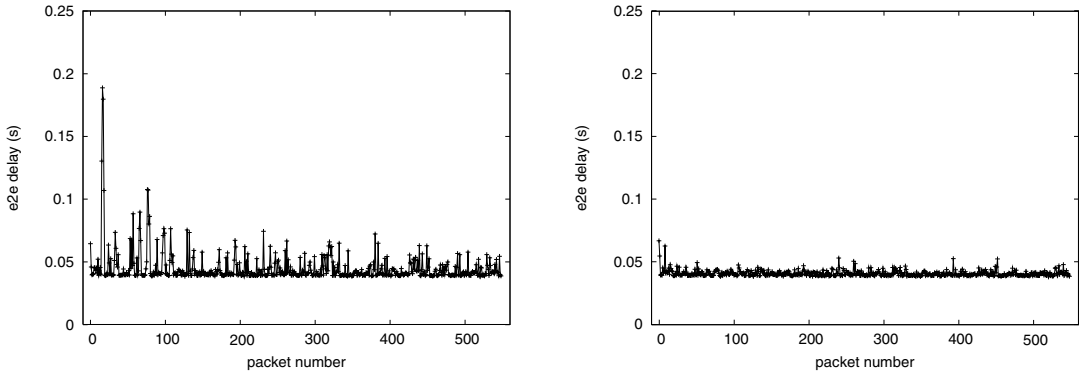
Next, we compared the performance of the red flows under different link utilizations and burst sizes of the blue flows. The main performance metric is the worst-case e2e delay. For each scenario, 36 independent runs were conducted. All the results are plotted with the 95% confidence interval [20].

[2]Note that the burst size is a relative value: value k means that the burst size is k times of the default packet size (e.g., if the packet size is 400B, then burst size 2 equals 800B).



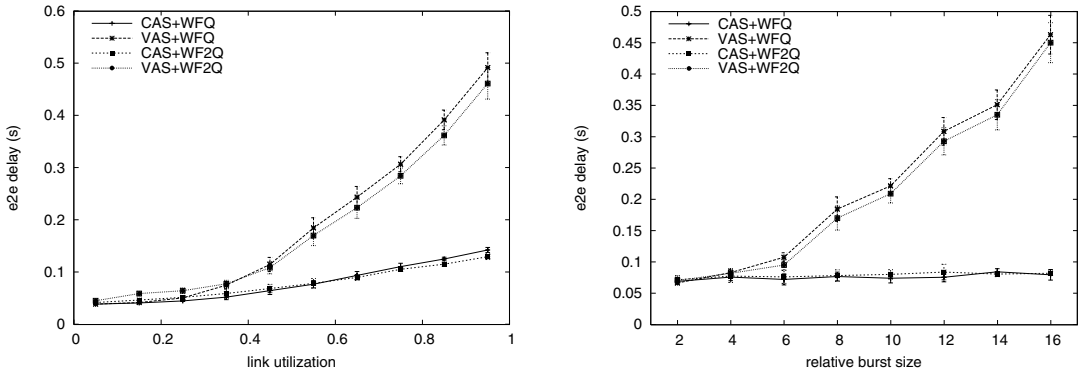
(a) VAS (b) CAS

Fig. 4. End-to-end delay comparison: WFQ



(a) VAS (b) CAS

Fig. 5. End-to-end delay comparison: WF^2Q

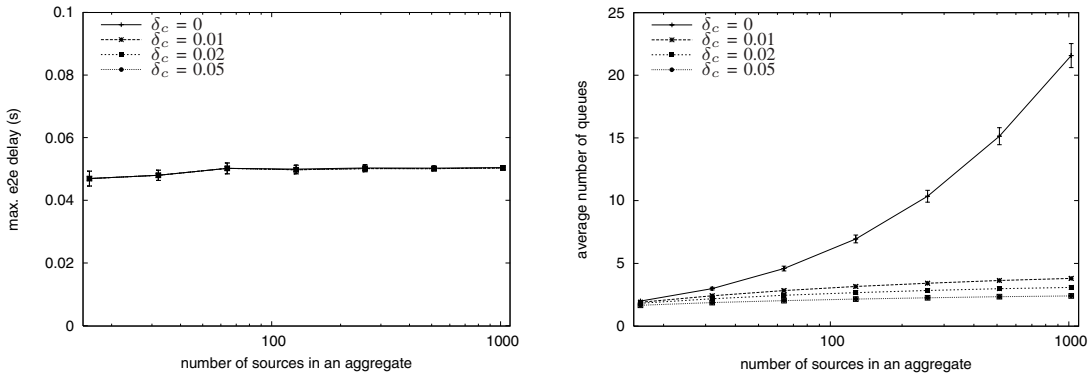


(a) Varying link utilization (b) Varying burst size

Fig. 6. End-to-end delays under different conditions

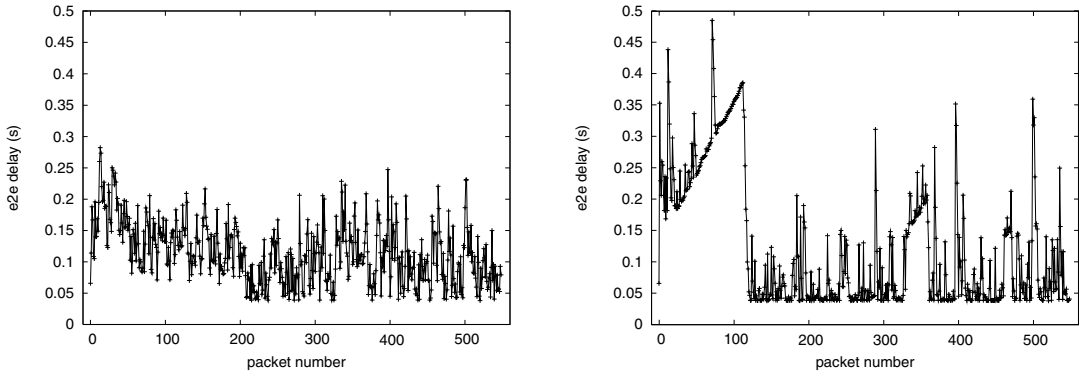
To see the robustness of the performance of CAS, we compared the performance of VAS and CAS under different link utilizations. As shown in Fig. 6(a), as the link utilization of the network links increases, the worst-case delay of the red flow under VAS increases significantly faster than that under CAS. This shows that CAS is more robust to high link utilization and congestion than VAS.

To examine the performance of CAS for large burst sizes of other flows sharing the same aggregate, we fixed the burst size of red flow at 2, and increased the burst size of the blue flows from 2 to 16. All the other parameters are set to the default values in Table II. The e2e delay of the red flow is shown in Fig. 6(b). As can be seen, with the burst size of the blue flows increasing, the e2e delay of the red flow under VAS increases very fast. By contrast, the e2e delay under CAS changes very little, confirming the delay analysis in Section III. The results



(a) maximum e2e delay (b) average number of queues

Fig. 7. The effectiveness of the adaptive algorithm



(a) Per-flow scheduling (b) Core-stateless (GRCore)

Fig. 8. End-to-end delay comparison II: WFQ

under both WFQ and WF^{2}Q algorithms are very similar and thus consistent.

To examine the effectiveness of the adaptive algorithm, we also ran simulations with hop count set to 3 and total number of tagged flows set to 1024. Other parameters use the default values in Table II. We varied the number of flows in each aggregate and monitored the maximum e2e delay and the average number of queues used by an aggregate at the second backbone router. The results are shown in Fig. 7. As can be seen from this figure, the adaptive algorithm can reduce the average number of queues significantly while almost maintaining the same maximum e2e delay (when δ_c = 0.01, 0.02, or 0.05).

VI. Comparison with Related Work

A. Delay bound of aggregate scheduling

The delay bound problem of aggregate scheduling was also studied by Cobb [7]. 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. Our CAS scheme differs from the schemes in [7] mainly in how and where the burstiness of other flows in the same aggregate is controlled. In [7], Cobb used non-work-conserving scheduling algorithms at aggregators—both the *basic fair aggregator* and *greedy fair aggregator* use rate-controlled schedulers. By contrast, our scheme is work-conserving, allowing any work-conserving GR scheduling algorithm to be used at aggregators. Burstiness is controlled in a on-demand fashion, by reordering packets in the same aggregate *only* at the congested downstream nodes of the aggregator. The aggregator simply stores the lag information in the packets. Our scheme is general since any GR scheduling algorithms can be modified to become CAS algorithms.

B. Core-stateless scheduling

Recently, in an effort to solve the scalability problem of per-flow scheduling, there have been some work on core-stateless scheduling [21]. The key idea of core-stateless scheduling is to use per-flow scheduling (rate-based or delay-based) only at edge routers. At the same time, edge routers store some key per-flow information in the packets. Therefore, the core routers need not keep per-flow information; all the needed information is carried in the packets themselves. Inside the network, core routers can restore the per-flow information from the packets and schedule them accordingly. By using this method, core-stateless scheduling can achieve the same delay bound as per-flow scheduling. The idea was first proposed by Stoica [21], and was generalized later by Kaur [22] and Zhang [23].

Our CAS scheme is similar to the core-stateless scheduling scheme in the sense that it also uses packets to carry scheduling related information. However, it differs from core-stateless schemes in the following aspects. First, core-stateless is totally stateless in the core network. All the needed information is carried in the packets themselves, from which the states of flows can be restored. By contrast, by keeping the states of traffic aggregates in core routers, CAS is still stateful, but the number of states is significantly smaller than that of per-flow fair queueing schemes (in orders of magnitude). Second, since CAS keeps states of traffic aggregates at core routers, it stores less information in the packets, with lag time δ only. In contrast, core-stateless scheduling generally needs to insert more information into packets. For example, CJVC (Core-stateless Jitter Virtual Clock) [21] needs to store four entries in each packet, with three of them relevant to scheduling (including the reserved rate for the flow). Since CAS stores less information in packets, the packet processing overhead is also relatively lower. Also, since core-stateless scheduling does not maintain any flow state at core routers, the admission control at core routers has to be based on traffic measurement or rate estimation [21], which has considerable overhead. CAS can use RSVP's extension for aggregation [13], which incurs smaller overhead at core routers. Third, core-stateless scheduling generally achieves the same delay performance as its corresponding per-flow scheduling. By taking advantage of multiplexing gains, CAS achieves tighter delay bounds than per-flow scheduling. Thus, CAS offers better delay performance. Finally, since CAS is not totally stateless at core routers, it has the advantage of isolating different traffic aggregates at core routers and confining the potential hazard problems (such as malicious traffic or denial-of-service attacks) within each aggregate, unaffecting other traffic aggregates.

To compare the performance of core-stateless scheduling with that of CAS, we repeated the first simulation in Section V using per-flow WFQ and GRCore, a work-conserving, core-stateless algorithm in [22]. The simulation was conducted using the same network setup and parameters in Section V. The results are plotted in Fig. 8. Comparison of Figs. 8 and 4 reveals a surprising result: *GRCore performs worse than both per-flow and aggregate scheduling.* The main reason for this is that under core-stateless scheduling, the packet deadline (such as the GRCore value) is computed rather conservatively, based on the flow's reserved rate. The actual arrival time of a packet at a core router is not considered in updating the deadline. At each core router, if other flows have bursts, packets from low-rate flows are pushed back in the queue and thus experience delays close to their deadlines. The fact that they may arrive earlier than other packets does not help, as long as the core router is not idle. In this sense, core-stateless scheduling is biased against low-rate flows.

We also ran the same simulation using WF^2Q and the results were similar. Interestingly, the delay results under GRCore using WFQ and WF^2Q are almost the same. This is because under GRCore, the deadlines of each packet under WFQ and WF^2Q are the same, and delays mainly occurred at the core routers. Thus, the difference in delay between WFQ and WF^2Q at the edge routers does not have much impact.

C. Coordinated scheduling

The idea of coordinating the scheduling of a packet at multiple nodes has also been explored in the literature. For example, the author of [24] proposed Coordinated EDF (CEDF) that coordinates the deadlines of a packet at multiple hops and yields a very small e2e delay. However, CEDF provides only statistical delay guarantees, and does not provide a natural way of assigning the deadline at each hop (the deadline at the first hop is randomly assigned). CAS is similar to CEDF in the sense that the scheduling inside an aggregate is based on EDF, and that it requires coordination among multiple nodes. However, by using the GRC value of a packet at the aggregator as its deadline at later hops, CAS provides a natural way of assigning deadline values to packets. In addition, CAS provides deterministic e2e delay guarantees.

In a related paper, the authors of [25] defined a general framework of *Coordinated Multihop Scheduling* (CMS), which covers many scheduling algorithms (including core-stateless algorithms [24], [21], [22]) exploring the coordination among different nodes. CAS's mechanism at the intra-aggregate level is also an example of CMS.

D. Summary of CAS's Features

The salient features of CAS are summarized as follows.

- *Scalable architecture*: by using traffic aggregation, CAS can support a large number of flows in core networks. Since CAS supports multiple aggregations, and the scale of traffic aggregates (in terms of the number of flows in each aggregate) can be flexibly set. The architecture of aggregate scheduling also fits the Internet administration architecture well.
- *Superior performance*: as our analysis and simulation results have shown, CAS provides better performance than both per-flow fair queueing and VAS.
- *Low overhead*: first, since CAS belongs to aggregate scheduling, it has lower overhead than per-flow fair queueing (e.g., lower state-maintenance overhead, simpler packet classification and scheduling). Second, with the optimization methods in Section IV, the extra overhead it has beyond VAS is marginal. Compared to VAS, CAS incurs a higher overhead at both the aggregator (computing and inserting lag values into packet headers) and core routers (packet sorting in each aggregate). For the former, the overhead is less than that in core-stateless fair queueing. For the latter, our simple multi-queue structure and adaptive algorithm have shown the extra overhead in packet scheduling to be marginal. Also, Stoica [21] has shown experimentally that the overall overhead of core-stateless fair queueing is not high—it "adds less than 5 μs overhead per enqueue operation, and about 2 μs per dequeue operation" on a 300MHz Pentium II machine. CAS has even smaller overhead than this.

TABLE III
COMPARISONS OF THE QOS ARCHITECTURES

	IntServ	DiffServ	Core-Stateless	CAS
Service guarantee	hard	soft	hard	hard
Maintaining state at core routers	per-flow state	stateless	stateless	aggregate state
Storing state in packets	no	no	yes	yes (less state information)
Rate estimation at core routers	no	no	yes	no
Admission control at core routers	per-flow RSVP	bandwidth broker	rate estimation	RSVP with aggregate support

- *Incremental deployment*: CAS facilitates incremental deployment. If some core routers do not implement CAS, then the queue at those nodes for each aggregate is FIFO. In other words, the scheduling at these routers becomes VAS. The performance will suffer, but still better than pure VAS. If the aggregator does not support CAS, as long as it supports fair queueing, it is still an aggregator of VAS. Then, the whole scheduling degrades to VAS, since the field to hold the lag value will have the default value for all packets, and the core routers will work in a FIFO fashion for each aggregate.

In addition to the features mentioned above, CAS is work-conserving and provides isolation between traffic aggregates.

Finally, we compare the four different schemes—IntServ, DiffServ, core-stateless, and CAS—in Table III. Compared to the other three schemes, CAS strikes a balance between overhead and performance.

VII. CONCLUSIONS

In this paper, we proposed a novel coordinated aggregate scheduling (CAS) algorithm, which uses EDF within each aggregate and GR scheduling algorithms among traffic aggregates. The EDF scheduling within each aggregate is coordinated among multiple nodes. Under the assumption that the incoming traffic to each aggregator conforms to the token bucket model, we proved that CAS provides tighter delay bounds for a flow than VAS. Moreover, CAS is shown to have many other salient features, e.g., it is work-conserving and incurs small packet processing overhead. We have also shown by simulation that CAS is robust, performing better in terms of worst-case e2e delay than VAS, per-flow scheduling and core-stateless scheduling algorithms.

ACKNOWLEGEMENT

The work reported in this paper was supported in part by Samsung Electronics and by the US Airforce Office of Scientific Research under Grant F49620-00-1-0327. The authors would also like to thank the anonymous reviewers for their insightful comments.

REFERENCES

[1] P. Goyal, S. S. Lam, and H. M. Vin, "Determining end-to-end delay bounds in heterogeneous networks," in *Proc. of NOSSDAV'95*, Apr. 1995, pp. 287–298.

[2] R. Braden, D. Clark, and S. Shenker, "Integrated services in the Internet architecture: an overview," RFC 1633, June 1994.

[3] B. Braden, L. Zhang, S. Berson, *et al.*, "Resource ReSerVation Protocol (RSVP) — version 1 functional specification," RFC 2205, Sept. 1997.

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

[5] 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. Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[6] L. Zhang, "Virtual Clock: A new traffic control algorithm for packet-switched networks," *ACM Trans. Computer Systems*, vol. 9, no. 2, pp. 101–124, May 1991.

[7] J. A. Cobb, "Preserving quality of service guarantees in spite of flow aggregation," *IEEE/ACM Trans. Networking*, vol. 10, no. 1, pp. 43–53, Feb. 2002.

[8] H. Fu and E. W. Knightly, "A simple model of real-time flow aggregation," *IEEE/ACM Trans. Networking*, vol. 11, no. 3, pp. 422–435, June 2003.

[9] W. Sun and K. G. Shin, "Delay bounds for end-to-end traffic aggregate under guaranteed rate scheduling algorithms," Dept. of EECS, Univ. of Michigan, Tech. Rep. CSE-TR-484-03, 2003.

[10] T. Li and Y. Rekhter, "A provider architecture for differentiated services and traffic engineering (PASTE)," RFC 2430, Oct. 1998.

[11] D. Awduche, J. Malcolm, J. Agogbua, *et al.*, "Requirements for traffic engineering over MPLS," RFC 2702, Sept. 1999.

[12] S. Blake, D. L. Black, M. A. Carlson, *et al.*, "An architecture for differentiated services," RFC 2475, Dec. 1998.

[13] F. Baker, C. Iturralde, F. L. Faucheur, *et al.*, "Aggregation of RSVP for IPv4 and IPv6 reservation," RFC 3175, Sept. 2001.

[14] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Networking*, vol. 6, no. 5, pp. 611–624, Aug. 1998.

[15] R. Brown, "Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem," *Communications of the ACM*, vol. 31, no. 10, pp. 1220–1227, Oct. 1988.

[16] "ns2 simulator." [Online]. Available: http://www.isi.edu/nsnam/ns/

[17] W. E. Leland, M. S. Taqqu, W. Willinger, *et al.*, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Trans. Networking*, vol. 2, no. 1, pp. 1–15, Feb. 1994.

[18] A. Popescu, "Traffic self-similarity," in *Proc. of IEEE Intl. Conf. on Telecommunications (ICT2001)*, June 2001.

[19] J. C. R. Bennett and H. Zhang, "WF^2Q: Worst-case fair weighted fair queueing," in *Proc. of IEEE INFOCOM'96*, Mar. 1996, pp. 120–128.

[20] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[21] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proc. of ACM SIGCOMM'99*, Sept. 1999, pp. 81–94.

[22] J. Kaur and H. M. Vin, "Core-stateless guaranteed rate scheduling algorithms," in *Proc. of IEEE INFOCOM'01*, Apr. 2001, pp. 1484–1492.

[23] Z.-L. Zhang, Z. Duan, and Y. T. Hou, "Virtual time reference system: A unifying scheduling framework for scalable support of guaranteed services," *IEEE J. Select. Areas Commun.*, vol. 18, no. 12, pp. 2684–2695, Dec. 2000.

[24] M. Andrews and L. Zhang, "Minimizing end-to-end delay in high-speed networks with a simple coordinated schedule," in *Proc. of IEEE INFOCOM'99*, vol. 1, Mar. 1999, pp. 380–388.

[25] C. Li and E. W. Knightly, "Coordinated multihop scheduling: A framework for end-to-end services," *IEEE/ACM Trans. Networking*, vol. 10, no. 6, pp. 776–789, Dec. 2002.