Maintaining Flow Isolation in Work-Conserving Flow Aggregation

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

***Abstract*— In order to improve the scalability of scheduling protocols with bounded end-to-end delay, much effort has focused on reducing the amount of per-flow state at routers. One technique to reduce this state is flow aggregation, in which multiple individual flows are aggregated into a single aggregate flow. In addition to reducing per-flow state, flow aggregation has the advantage of a per-hop delay that is inversely proportional to the rate of the aggregate flow, while in the case of no aggregation, the per-hop delay is inversely proportional to the (smaller) rate of the individual flow.**

Flow aggregation in general is non-work-conserving. Recently, a work-conserving flow aggregation technique has been proposed. However, it has the disadvantage that the end-to-end delay of an individual flow is related to the burstiness of other flows sharing its aggregate flow. Here, we show how work-conserving flow aggregation may be performed without this drawback, that is, the end-to-end delay of an individual flow is independent of the burstiness of other flows.

I. Introduction

Let *flow* denote a sequence of packets generated by a real-time application. To provide quality of service guarantees, the network must reserve resources along the path of the flow. Currently, the Internet does not provide service guarantees. However, much effort is being focused on designing Internet protocols to support these guarantees. Two general approaches have been proposed: Integrated Services (IntServ) [4] and Differentiated Services (DiffServ) [12], [13].

The seminal paper by Cruz [8] introduced the deterministic analysis of network traffic, which spawned the design of many real-time packet scheduling protocols [23]. In the IntServ approach, quality of service is provided via these scheduling protocols. Their main drawback is the need to maintain per-flow state at each router. This introduces several overheads as compared to the traditional datagram model. Due to these overheads, the scalability and robustness of IntServ has been questioned, which lead to the introduction of DiffServ.

In DiffServ, the inter-network is divided into access networks and a core network. Routers in access networks maintain per-flow state. However, routers in the core network maintain no per-flow state. Instead, a few bits in the packet header are reserved to indicate the service category, also known as per-hop behavior, that applies to the packet. Before a packet enters the core network, the access router assigns a per-hop behavior to the packet, according to the desired quality of service. At each core router, packets are classified and forwarded according to their per-hop behaviors. DiffServ is scalable, because no per-flow state is required. However, this limited amount of state provides only a coarse allocation of resources, and falls short of the quality of service level available in IntServ [21].

In an effort to reduce the amount of state at each router, but without sacrificing the quality of service provided by IntServ, we introduced flow aggregation [6]. For efficiency, multiple flows are aggregated into a single flow, and routers after the point of aggregation are aware only of the aggregate flow, and are unaware of the flows constituting the aggregate flow. Due to the reduction in the number of flows, packet scheduling is simplified, and signaling may also be aggregated [9]. At a later point in the network, the aggregate flow is separated into its constituent flows, which then continue on their own to their respective destinations. Flow aggregation has the additional advantage that end-to-end delay bounds can be proven to be lower than in the case of no aggregation [6], [5].

Another approach to reduce the per-flow state at routers has been presented in [21], [14]. Here, the quality of service level of IntServ is achieved, but without any flow state in routers. In [21], [14], the scheduling of packets is based on dynamic packet state, that is, each packet carries enough information to reproduce its deadline at each router, without per-flow state. These techniques have the disadvantage that the signaling methods with no per-flow state are in general observation methods [1], [21] that inherently lead to an inaccurate estimation of the rates of flows traversing the router. In addition, the lower end-to-end delay bound achievable under flow aggregation is not achieved using dynamic packet state.

A low amount of state, in combination with lower end-to-end delays, make flow aggregation an attractive technique for scalable packet scheduling. However, it is non-work-conserving, and thus, the output channel may be left idle even though packets remain to be transmitted.

Recently [22], a work-conserving flow-aggregation method was presented. Work-conservation is achieved by assigning a deadline to each packet of each individual flow at the moment of aggregation, and then performing intra-flow deadline sorting, i.e., the packets of each aggregate flow are sorted at intermediate routers according to the assigned deadlines.

Since the method is work-conserving, the average end-to-end delay of packets is lower. However, the guaranteed upper-

bound on delay is dependent of the burstiness of other flows (it is independent in non-work conserving flow aggregation and in dynamic packet state). Hence, all flows must be leaky-bucket constrained, preventing flows from exceeding their reserved rate significantly to take advantage of unused bandwidth.

In this paper, we present a work-conserving flow aggregation technique also based on intra-flow deadline sorting. However, the end-to-end delay bound is independent of the burstiness of other flows. This low end-to-end delay bound in combination with a work-conservation comes at a price: the per-hop delay is slightly higher than in traditional flow aggregation, and strict requirements are necessary on the scheduling protocols at intermediate routers.

II. Quality of Service Model

In this section, we define the quality of service model that the network will provide to each real-time flow. We base our service model on the models of [11], [19].

A. Virtual Finishing Times and Guaranteed-Rate Schedulers

A *flow* is a sequence of packets generated by an application. Each output channel of a computer is equipped with a scheduler, whose function is to schedule packets in an order which guarantees quality of service to each input flow. We say a packet exits/arrives from/to a scheduler when the last bit of the packet is transmitted/received by the scheduler. For simplicity, we assume the propagation delay between schedulers is zero.

Each flow is characterized by its reserved packet rate and its maximum packet size. We adopt the following notation for each flow f and each scheduler s along the path of f.

C^s	output channel bit rate of s
R_f	bit rate reserved for flow f
$f.i$	i^{th} packet of flow f
$A^s_{f.i}$	arrival time of $f.i$ at s
$E^s_{f.i}$	exit time of $f.i$ from s
$L_{f.i}$	length of packet $f.i$
$L^{max}_{f.i}$	maximum of $L_{f.j}$, where $1 \leq j \leq i$
L^s_{max}	maximum packet size at s

Consider a scheduler s and a flow f. We define the *virtual finish time*[1] $F^s_{f.i}$ of packet $f.i$ at scheduler s as follows. Assume s were to forward the packets of f at exactly R_f bits/sec.. Then, $F^s_{f.i}$ is the time at which the last bit of $f.i$ is forwarded by s. More formally, let f be an input flow of scheduler s. Then,

$$\begin{aligned} F^s_{f.1} &= A^s_{f.1} + L_{f.1}/R_f \qquad (1)\\ F^s_{f.i} &= max(A^s_{f.i},\, F^s_{f.(i-1)}) + L^s_{f.i}/R_f, \text{ for every } i,\ i > 1 \end{aligned}$$

Because scheduler s will forward the packets of f at a rate at least R_f, each packet $f.i$ exits from s close to $F^s_{f.i}$. Schedulers with this property are known as *guaranteed-rate schedulers* [11]. More formally, a scheduler s is a guaranteed-rate (GR) scheduler if and only if, for every input flow f of s and every i, $i \geq 1$,

$$E^s_{f.i} \leq F^s_{f.i} + \beta^s_f \qquad (2)$$

for some constant β^s_f. We refer to β^s_f as the scheduling constant of f at s.[2]

Since the virtual finishing time of a packet determines its exit time from a scheduler, then a bounded end-to-end delay requires a bounded per-hop increase in the virtual finishing time. This bound is well known (it was shown in [11] and also follows from the results in [7], [19]) and is as follows. Let $t^1, t^2, \ldots, t^k$ be a sequence of k GR schedulers traversed by flow f. For all i,

$$F^{t^k}_{f.i} \leq F^{t^1}_{f.i} + \sum_{x=1}^{k-1} \left(\frac{L^{max}_{f.i}}{R_f} + \beta^{t^x}_f \right) \qquad (3)$$

B. Flow Aggregation

To reduce the amount of state managed by each router, multiple flows can be combined together to form a single aggregate flow [6], [5], [9], [17].

An *aggregate* flow g is obtained by merging, at a single point in the network, the packets of multiple flows $f^1, f^2, \ldots, f^n$. In this case, $f^1, f^2, \ldots, f^n$ are said to be the *constituents* of g. The reserved rate, R_g, of aggregate flow g is at least the sum of the reserved rates of the immediate constituent flows of g. Schedulers after the aggregation point are not aware of the constituents of an aggregate flow. At a later point in the network, the aggregate flow is separated again into its constituent flows.

We consider aggregation over a core network model, shown in Figure 1. It consists of a network of core routers surrounded by access networks. Ingress/egress routers manage the input/exit of flows from/to the access networks to/from the core routers. Core routers maintain small amounts of flow state, while ingress/egress routers maintain state for each individual flow.[3]

We assume that all flows entering and exiting the network via the same ingress and egress routers are aggregated together at the ingress router. In this case, the total number of flows visible to a core router is approximately $\frac{N^2}{M}$, where N is the number of ingress/egress routers, and M is the number of core routers. For each aggregate flow g, each core router is unaware of the constituent flows contained by g (or simply chooses to ignore them). It thus schedules the packets of g as if g were a simple flow with reserved rate R_g.

[1] The virtual finishing time is also known as the guaranteed rate clock value in [11], and it is also equal to the timestamp assigned by a virtual clock scheduler [24].

[2] The value of β^s_f determines the type of delay guaranteed by s. If $\beta^s_f > 0$ (typically L^s_{max}/C^s), then it is *rate-dependent delay*, such as the delay bound provided by the Virtual-Clock and Weighted-Fair Queuing protocols [16], [10], [24]. On the other hand, if $\beta^s_f < 0$, then we have *rate-independent delay*, such as the delay bound provided by the protocols in [25]. In this paper, we will focus on the former, i.e., on rate-dependent delay, where $\beta^s_f > 0$.

[3] This network is similar to a SCORE network in [21], [14]. However, in a SCORE network, core routers maintain no per-flow state.

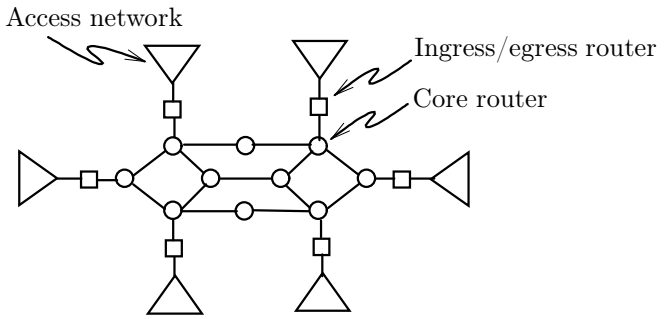

Fig. 1. Core Network

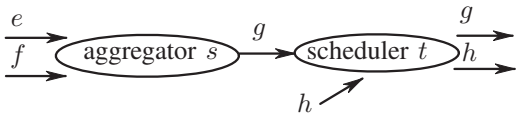

Fig. 2. Need for fair aggregation

A scheduler that receives as inputs a set of flows $f^1, f^2, \ldots, f^n$, and produces as output a single aggregate flow g, by merging the packets of the input flows, is called an *aggregator*. Thus, ingress routers contain $N-1$ aggregators, one for each egress router. A scheduler whose set of input flows is the same as its set of output flows is called a *non-aggregating scheduler*, or simply *scheduler* for terseness. Thus, core routers contain schedulers but no aggregators.

We assume all schedulers, aggregating or not, are GR schedulers. Thus, for any scheduler s and any input flow h of s (regardless of whether h is a simple or aggregate flow), every packet $p_{h.i}$ exits s no later than time $F^s_{h.i} + \beta^s_h$.

A *separator* is a process that receives as input an aggregate flow, and produces as output the set of *constituents* of the input flow. Thus, egress routers contain a separator for every ingress router. We assume a separator causes no packet delay.

Even if an aggregating scheduler is a GR scheduler, it is not sufficient to guarantee a bounded end-to-end delay bound to its input flows. E.g., consider Figure 2, where two flows, e and f, are input to an aggregating scheduler s, whose aggregate output is g, and the next scheduler after s is t. Assume e generates packets at a rate greater than R_g, i.e., greater than $R_e + R_f$, f is generating few packets, if any, and aggregator s does not delay packets. Since scheduler t forwards the packets of g at a rate of $R_e + R_f$, the queue of g may grow arbitrarily large at t. Therefore, the next packet of f encounters a large number of packets ahead of it in the queue of g at t, and suffers an excessive delay.

To prevent the above, in addition to being a GR scheduler, the aggregating scheduler should be fair [6]. That is,

$$F^t_{g.j} \leq F^s_{f.i} + \lambda^s_f \tag{4}$$

where $g.j = f.i$ and λ^s_f is the aggregating constant of s. If so, combining (3) with (4) we have the following. Let f be an input flow of an aggregating scheduler s, g be the output flow of s, and g traverses GR fair schedulers $t^1, t^2, \ldots t^k$. Then,

$$F^{t^k}_{g.j} \leq F^s_{f.i} + \lambda^s_f + \sum_{x=1}^{k-1} \left(\frac{L^{max}_{g.j}}{R_g} + \beta^{t^x}_g \right). \tag{5}$$

Notice that the bound in (5) above is similar to earlier bound (3), except that the per-hop delay is based on L_g/R_g with aggregation, and based on L_f/R_f without aggregation. In general, $R_g \gg R_f$ and $L_g \approx L_f$, and hence, aggregation provides a much smaller per-hop delay.

III. Work-Conserving Aggregation with Isolation

Fair aggregating schedulers, as defined in [6], are non-work-conserving. They solve the problem of a large queue of flow g at scheduler t in Figure 2 by ensuring that the output rate of s is restricted to R_g. Hence, since t is a GR scheduler, the queue of g at t is always kept small.

An alternative work-conserving solution, known as Coordinated Aggregate Scheduling (CAS), is presented by Sun and Shin in [22]. Aggregators are allowed to be any GR server. However, the excessive delay of packets from f due to a large queue of g at t is avoided as follows. At s, the packets of e and f are tagged with their virtual finishing times, as measured at s[4]. Then, at subsequent hops, the packets of g are maintained sorted by their tags. Thus, the queue of an aggregate flow is no longer a FIFO queue, as is the case in regular aggregation. In this manner, packets from f with a low virtual finishing time can "jump" over packets of g (more precisely, of e) with higher virtual finishing time, and hence not be delayed.

Consider again Figure 2. The packets of f are not delayed significantly because, if the queue of g has an excessive number of packets of e, these will be sorted by virtual finishing time along with the packets of f, and hence, the packets of f may overcome the packets of e.

Allowing the intermediate schedulers to be any GR scheduler provides flexibility of implementation, but significantly impacts the end-to-end delay. For example, assume intermediate nodes implement an unfair GR scheduler, such as Virtual Clock. Then, the packets of f may be delayed excessively at t, as follows.

Assume t forwards all packets of e (which are also packets of g). Then, packets of f arrive at t. Since t served flow g at a rate higher than R_g for a significant amount of time, the virtual finishing times of the packets of g at t are significantly greater than real-time. This allows t to temporarily deny service to g, and hence to f, for some time by transmitting the packets of other flows, such as h, until the virtual finishing time of h becomes equal to that of g. In consequence, all constituent flows must be leaky-bucket constrained, and furthermore, flows with different bucket sizes should not be aggregated together.

Below, we present an alternative flow aggregation method where the end-to-end delay of an individual flow is similar

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

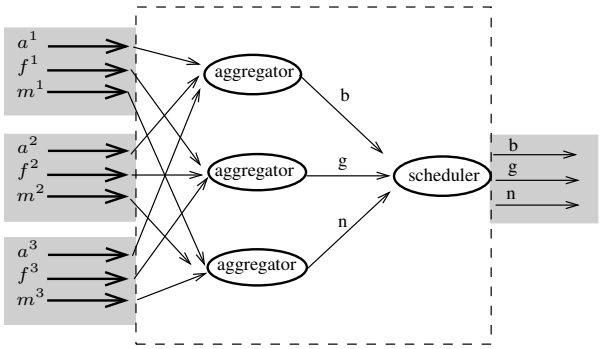

Fig. 3. Ingress router configuration

to Relation (5), and is thus independent of the leaky-bucket parameters of other flows. Since the flow is guaranted its end-to-end delay bound independently from other flows, we refer to this method as *Coordinated Aggregation with Isolation* (CAI).

In CAI, we also take advantage of intra-flow sorting to mitigate the effect of queue buildups at intermediate schedulers. However, to prevent temporary denials of service to individual flows, we focus on scheduling protocols whose Time Worst-Case Fair Index [2] is small, such as WF^{2}Q. We show that these scheduling protocols guarantee to each flow an end-to-end delay that is independent of the burstiness of other flows. In addition, we show that using the virtual finishing time as a tag value is not the only choice, and that other choices may improve fairness among flows.

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

A. Internal Aggregators and Coordinated Virtual-Finishing-Time

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

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

Consider the general case of an aggregator s whose input flows include f, its aggregate output is g, $f.i = g.j$, and g is an input to scheduler t. Aggregator s assigns a tag $T_{f.i}$ to each input packet $f.i$. We initially choose a tag equal to the virtual finishing time of the packet at s, i.e., $T_{f.i} = F^s_{f.i}$. We consider other tag values in Section IV.

If t serves g in FIFO order, as in regular flow aggregation, then $g.j$ exits t near time $F^t_{g.j}$. From the definition of virtual finishing times (Equation 1), $F^t_{g.j}$ depends only on packets $g.1 \ldots g.j$. However, if t sorts each input flow by tag value, then the exit time of $g.j$ depends not only on packets of g arriving before $g.j$, but also on packets of g arriving *after* $g.j$ whose tag is at most that of $g.j$.

To capture the above behavior, we define the *Coordinated Virtual-Finishing-Time,* Φ. Intuitively, $\Phi^t_{g.j}$ is the time at which $g.j$ would exit t if t served the packets of g at exactly the rate R_g, and, *furthermore,* t serves every packet of g whose tag is at most $T_{g.j}$ *before* it serves $g.j$.

We next provide a formal definition of Φ. We begin with some auxiliary definitions.

Function $filter(g, t, \tau)$ returns a flow that differs from g only by removing those packets of g whose tag is greater than τ. More formally, $g' = filter(g, t, \tau)$ iff the following two conditions hold.

$$\langle \forall\, k,\ g'.k \in g \wedge (g.k\ \in g' \equiv T_{g.k} \leq \tau) \rangle$$

$$\left\langle \forall\, k, k',\ (g.k = g'.k') \Rightarrow \left(A^t_{g.k} = A^t_{g'.k'} \right) \right\rangle$$

Function $advance(g', t, f.i)$ returns a flow similar to g'. The difference is that all packets in g' that arrive after $f.i$, where $f.i$ is a packet of g', are moved *ahead* of $f.i$ if their tag is at most that of $f.i$. I.e., they arrive at the same time as $f.i$. More formally, assume $f.i = g'.j$. Then, $g'' = advance(g', t, f.i)$ iff the following three conditions hold.

$$\langle \forall\, k,\ g'.k \in g'' \wedge g''.k \in g' \rangle$$

$$\left\langle \forall\, k,\ 1 \leq k < j,\ g'.k = g''.k \wedge A^t_{g'.k} = A^t_{g''.k} \right\rangle$$

$$\langle \forall\, k,\ j \leq k,\ A^t_{g''.k} = A^t_{g'.j} \rangle$$

Definition 1: Let s be an aggregator with an input flow f and with output flow g. Let $f.i = g.j$, and let t be the next scheduler after s. Then,

$$\Phi^t_{g.j} = F^t_{g''.|g''|}$$

where $g' = filter(g, t, T_{f.i})$ and $g'' = advance(g', t, f.i)$.

Given the above definition of Φ, we next provide an upper bound on Φ as the aggregate flow exits the aggregator that created it.

Theorem 1: Let s be an internal aggregator with an input flow f and with output flow g. Let $f.i = g.j$, and let t be the scheduler after s in the same router. Then,

$$\Phi^t_{g.j} \leq F^s_{f.i}$$

Below, we discuss the properties required from a scheduler to ensure a small per-hop increase in Φ that is independent of the burstiness of other flows. This, along with the above bound on the initial value of Φ , provides a bounded end-to-end delay.

B. Fair Schedulers

We argued above that if a scheduler has an aggregate flow as input, and if for extended periods of time the flow is not served, then the deadline guarantees of the constituent flows are violated. Therefore, unfair scheduling algorithms, such as

Virtual Clock [10], [24], are inadequate, even though they belong to the family of GR scheduling algorithms.

The amount of time that may ellapse without a scheduler serving a flow can be formalized by the Worst-Case Fair Index (WFI) , as defined in [3].

Definition 2: A scheduler t provides to an input flow g a Worst-Case Fair Index (WFI) of W_g^t if for any time τ, the delay of a packet arriving at τ is bounded above by

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

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

In this manner, regardless of how many packets from g have been forwarded by t, i.e., even if g has exceeded its packet rate, at all times t will serve g at a rate at least R_g, except for an additional delay of at most W_g^t. This ensures an exit bound on all packets of g that is related to their coordinated virtual-finishing time Φ, as follows.

Theorem 2: Let g be an input flow of scheduler t that sorts the packets of g by their tags, and provides a worst-case fair index to g. Then, the exit time from t of each packet of g is bounded as follows.

$$E_{g.j}^t \leq \Phi_{g.j}^t + W_g^t$$

The above bound on the exit time, along with the bound on Φ of Theorem 1, allow us to provide an end-to-end delay bound based on the virtual finishing time F at the aggregator. This bound is useful only if there is a bounded per-hop increase in Φ at each intermediate scheduler. The per-hop increase in Φ is indeed bounded, and is also related to the WFI of the intermediate schedulers, as follows.

Theorem 3: Let g be an input flow of scheduler t that sorts the packets of g by their tags, and provides a worst-case fair index to g. Let t' be the next scheduler traversed by g after t. Then,

$$\Phi_{g.j}^{t'} \leq \Phi_{g.j}^t + W_g^t + \frac{L_g^{max}}{R_g}$$

C. End-to-End Delay

The results of the previous section can be combined to form an upper bound on the end-to-end delay of a flow that was aggregated with other flows and then traversed several schedulers. Theorem 1 shows that the coordinated virtual-finishing time after the aggregator is bounded by the virtual-finishing time of the input flow. Then, Theorem 3 shows the coordinated virtual-fishing time has a bounded per-hop increase. Finally, Theorem 2 gives the exit time with respect to the coordinated virtual-finishing time. In consequence we have the following corollary.

Corollary 1: Let f be an input flow of an internal aggregator s, g be the output of s, and let g traverse schedulers t^1, t^2, ... t^k. Then, the end-to-end delay of any packet $f.i$ of flow f is as follows.

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

We thus have that the end-to-end delay has a per-hop increase proportional to $\frac{L}{R_g}$, as in the case of regular flow aggregation (see (5)). However, we have the additional WFI term W_g^t. This term should be as small as possible to ensure a low end-to-end delay.

Given that Virtual Clock has an unbounded WFI, a more suitable scheduling protocol could be Weighted Fair Queuing (WFQ) [16], since it treats its input flows fairly. However, the WFI of WFQ is actually proportional to $\frac{L}{R_{min}}$, where R_{min} is the minimum rate among the flows at the scheduler [2]. If R_{min} is allowed to be very small, this will cause a significant end-to-end delay.

Although end-to-end delay is bounded with WFQ, we desire a tighter bound in proportion to $\frac{L}{R_g}$. In [2], WF^{2}Q is proposed as an alternative to WFQ. WF^{2}Q provides a more accurate emulation of the fluid server emulated by WFQ. In particular, a packet is not considered eligible for transmission by WF^{2}Q until its first bit begins transmission in the emulated fluid server. In [2], [3], it is shown that the WFI of a WF^{2}Q scheduler t is bounded as follows.

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

WF^{2}Q is not the only protocol with the above bound of WFI. There is a whole family of schedulers, called Shaped Rate Proportional (SRP) schedulers [20], [18], whose WFI is as above. SRP schedulers are based on emulating the behavior of a fluid server, and not considering a packet eligible for transmission until the first bit of the packet is served by the fluid server.

The SRP family of protocols is broad. On one end of the spectrum is the WF^{2}Q protocol, which is work-conserving and distributes unallocated capacity among all flows in proportion to their reserved rate. On the other end of the spectrum is a non-work-conserving version of the Virtual Clock protocol, which prevents flows from making use of any unallocated capacity.

From the above bound we have the following corollary.

Corollary 2: Let f be an input flow of an internal aggregator s, g be the output of s, and let g traverse schedulers t^1, t^2, ... t^k, each of which is an SRP scheduler. Then, the end-to-end delay of any packet $f.i$ of flow f is as follows.

$$E_{f.i}^{t^k} \leq F_{f.i}^s + \frac{(2\cdot k-1)\cdot L_g^{max}}{R_g} + \sum_{x=1}^{k} \frac{L_{max}^{t^x}}{C^{t^x}} \qquad (6)$$

We thus have that the end-to-end delay has a per-hop increase proportional to $\frac{L}{R_g}$, as in the case of regular flow aggregation (see Relation (5)) plus the small per-hop term $\frac{L_{max}^t}{C^t}$. However, most GR scheduling protocols have $\beta_g^t = \frac{L_{max}^t}{C^t}$. Hence, Relation (6) differs from Relation (5) only by the additional per-hop delay of $\frac{L}{R_g}$.

IV. Fair Work-Conserving Aggregators

We have addressed thus far how to perform work-conserving aggregation with a small per-hop delay bound. However, we have not addressed fairness between the constituent flows of an

	Work Conserving	Schedulers Allowed	Burstiness Isolation
FA [6]	No	Any GR	Yes
CAS [22]	Yes	Any GR	No
CAI	Yes	SRP	Yes

TABLE I
COMPARISON OF AGGREGATION TECHNIQUES

aggregate flow. Consider again Figure 2. If flow f generates packets at a rate higher than R_f, then the tags of its latest packets, i.e., their virtual-finishing times at s, become much larger than real time. If e then begins to transmit packets, its tags will be smaller than those of f. This may cause f to be denied service at t until the tags of e reach the value of the tags of f.

We can limit this fairness by using a different tag as follows. Aggregator s tags each packet with the same tag (a.k.a. timestamp) that a WFQ scheduler with output channel capacity of R_g would use to tag the packet (note that the output channel capacity of s is infinity and not R_g). That is, $f.i$ is tagged with the time at which it exits a fluid generalized processor sharing (GPS) server [16] of capacity R_g. In this manner, f can exceed its reserved rate R_f and take advantage of any unused portion of the bandwidth of g which is currently not being used by e. If e then generates packets, its tag values will be close to those of f, and f will not be denied service.

This has its limitations, however. In particular, if f exceeds the rate R_g, then its tag values will grow beyond those of the next packets of e, and f may temporarilly be denied service. This is summarized below.[5]

Theorem 4: Let s be an internal aggregator with an input flow f and with output flow g. Let $f.i = g.j$, and let t be the scheduler after s in the same router. Let $T^s_{f.i}$ be the time at which $f.i$ exits a GPS server of capacity R_g with the same input flows of s. Then,

$$\Phi^t_{g.j} \leq E^{GPS}_{f.i}$$

where $E^{GPS}_{f.i}$ is the real-time at which $f.i$ exits the GPS server.

V. CONCLUDING REMARKS AND FUTURE WORK

Table I summarizes the main properties of all three flow aggregation methods.

In future work, we will consider applying CAI across multiple domains, similar to our work on regular flow aggregation across multiple domains [5]. In addition, as discussed above, the fairness among the flows being aggregated is limited to the aggregate rate R_g. We would like to allow a flow to exceed the aggregate rate R_g and still be given some degree of fairness. We speculate that this can be accomplished by incorporating the "timestamp reuse" technique introduced in [15].

[5]A theorem similar to Theorem 4 holds when the fluid server emulated is a fluid SRP server. However, the most practical candidate for the fluid server being emulated, due to its fairness, is the GPS server used by the WFQ protocol.

REFERENCES

[1] W. Almesberge, T. Ferrari, and J. L. Boudec, "SRP: A scalable resource reservation protocol for the internet," in *Proc. of The International Workshop on Quality of Service (IWQOS)*, 1998.

[2] J. C. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 675–689, Oct. 1997.

[3] ——, "WF^{2}Q: worst-case fair weighted fair queueing," in *IEEE INFOCOM Conference*, 1996.

[4] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture," Internet RFC 1633.

[5] J. Cobb, "Scalable quality of service across multiple domains," *Computer Communications*, Elsevier, accepted for publication, expected publication date Fall 2004.

[6] ——, "Preserving quality of service guarantees in-spite of flow aggregation," *IEEE/ACM Transactions on Networking*, vol. 10, no. 1, pp. 43–53, Feb. 2002.

[7] J. Cobb and M. Gouda, "Flow theory," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 661–674, Oct. 1997.

[8] R. L. Cruz, "A calculus for network delay, part i: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, Jan. 1991.

[9] H. Fu and E. W. Knightly, "A simple model of real-time flow aggregation," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, June 2003.

[10] X. G. and L. S., "Delay guarantee of the virtual clock server," *IEEE/ACM Transactions on Networking*, pp. 683–689, Dec. 1995.

[11] P. Goyal, S. Lam, and H. Vin, "Determining end-to-end delay bounds in heterogeneous networks," in *Proc.of the NOSSDAV Workshop*, 1995.

[12] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured forwarding phb group," Internet RFC 2597.

[13] V. Jacobson, K. Nichols, and K. Poduri, "An expedited forwarding phb," Internet RFC 2598.

[14] J. Kaur and H. M. Vin, "Core-stateless guaranteed rate scheduling algorithms," in *Proc. of the IEEE INFOCOM Conf.*, 2001.

[15] ——, "Core stateless guaranteed throughput networks," in *Proc. of the IEEE INFOCOM Conf.*, 2003.

[16] A. K. J. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, June 1993.

[17] J. Qiu and E. W. Knightly, "Measurement-based admission control with aggregate traffic envelopes," *IEEE/ACM Transactions on Networking*, vol. 9, no. 2, Apr. 2001.

[18] D. Stidialias and A. Varma, "Rate proportional servers: A design methodology for fair queuing algorithms," *IEEE/ACM Transactions on Networking*, Apr. 1998.

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

[20] ——, "A general methodology for designing efficient traffic scheduling and shaping algorithms," in *IEEE INFOCOM Conference*, 1997.

[21] I. Stoica and H. Zhang, "Providing guaranteed services without per-flow management," in *Proc. of the ACM SIGCOMM Conference*, 1999.

[22] W. Sun and K. G. Shin, "Coordinated aggregate scheduling for improving end-to-end delay performance," in *Proc. of the IEEE Workshop on Quality of Service (IWQoS)*, 2004.

[23] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 93, no. 10, Oct. 1995.

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

[25] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Transaction on Communications*, vol. 42, no. 3, Mar. 1994.