

# Scalable Quality of Service Across Multiple Domains

Jorge A. Cobb

*Department of Computer Science (EC 31), The University of Texas at Dallas,  
Richardson, TX 75083-0688*

---

## Abstract

Several state-less scheduling protocols have been proposed in the literature that achieve the same level of quality of service as state-full protocols. Thus, the scheduling accuracy of state-full protocols is combined with the scalability of state-less protocols. However, these new protocols treat each flow as an independent unit. In this paper, we present a state-less scheduling protocol that aggregates flows together in order to increase the number of flows that can be accepted into the network without violating their delay requirements. In addition, we present a state-less signaling protocol that may be used in combination with our state-less scheduling protocol. Contrary to sampling techniques, our signaling protocol is accurate yet requires only a constant amount of state at each node. Finally, our protocols can be used across multiple core networks.

*Key words:* Differentiated Services, Quality of Service, Packet Scheduling

---

## 1 Introduction

In order to support future real-time applications such as voice and video, a great deal of effort has been spent on improving the quality of service (QoS) provided by the Internet. Two approaches have been proposed for this improvement: integrated services [2][18][22] and differentiated services [12][14].

In integrated services, QoS is based on earlier packet scheduling protocols, such as virtual clock [21][24] and fair queuing [17]. To provide QoS to each flow, each router maintains per-flow state. Although integrated services can provide flows with high levels of QoS, questions about the scalability of integrated

---

*Email address:* cobb@utdallas.edu (Jorge A. Cobb).

services have been raised due to the large number of flows in the Internet. This motivated the development of differentiated services.

Differentiated services focus on core networks. Core networks consist of a set of core routers that do not maintain per-flow state, and a set of ingress/egress routers that receive individual flows from outside the core network and do maintain per-flow state.

To provide different services, a few bits in each packet indicate the desired “per-hop-behavior”. In this manner, different services may be supported, and each flow chooses the service that best fits its needs. Due to the lack of per-flow state at each router, the range of services provided is limited. Furthermore, high levels of QoS and network utilization cannot be accomplished concurrently [19].

Several new protocols have been proposed recently for core networks. These new protocols achieve the same level of QoS as virtual clock and fair queuing, but without per-flow-state at the core routers [5][7][13][19][23]. To accomplish this, both the signaling protocol and the packet scheduling protocol must be performed without per-flow state.

In [5][7], flows are aggregated together as they cross a network. This has two advantages: first, the number of flows is significantly reduced, and thus, the amount of state at each router is also significantly reduced. Second, it is possible to increase the number of flows that can be accepted into the network without violating their delay requirements. That is, the set of flows allowed by admission control increases. However, although the number of flows at core routers are reduced significantly, still per-flow state is required.

In [19][23], packet scheduling is based on “dynamic packet state”, where each packet carries enough information to reproduce its deadline at each router, without per-flow state. That is, the deadlines of each packet at each router are precomputed at the ingress router, and thus, the core router need not compute them, and no per-flow state is required. However, each flow is treated as an individual unit, and thus, the admission control advantage of flow aggregation is not present.

In this paper, we present a scheduling protocol that has the advantages of both of dynamic packet state and flow aggregation. Therefore, no per-flow state is maintained at the core routers, and flow aggregation is used to improve admission control.

In addition to the scheduling protocol requiring no per-flow state, the signaling protocol must also avoid per-flow state. Signaling protocols with no per-flow state are in general divided into two categories: observation methods and bandwidth-broker methods. Observation methods [1][19] estimate the

resource requirements by observing the traffic through the router. This inherently leads to an inaccurate estimation of the rates of flows traversing the router, since flows may temporarily generate packets at a rate lower than their normal rate. In bandwidth-broker techniques, resource reservation is managed by a bandwidth broker [16][23]. This only shifts the burden of managing a large number of flows to a centralized broker, but does not reduce the amount of state to be maintained. Furthermore, centralized brokers are vulnerable to faults, and distributed brokers have the difficulty of maintaining their state synchronized.

Below, we present a signaling protocol that maintains a constant amount of state per router. Nonetheless, the signaling protocol is accurate and resilient to process and link failures. By combining our signaling protocol with our scheduling protocol, we obtain a complete system to provide QoS without per-flow state.

Finally, rather than restrict ourselves to a network consisting of a single core network, we consider a network consisting of a set of core networks interconnected by gateway routers. In this network, only ingress/egress routers maintain per-flow state. Once a flow enters a domain, no state is maintained for this individual flow as it traverses gateway routers from one domain to the next.

The paper is organized as follows. Section 2 presents the QoS model. Flow aggregation in a core network, along with its advantages and disadvantages, is presented in Section 3. The application of flow aggregation to a single core network is given in Section 4. Dynamic packet-state, and its application to an aggregating core network is presented in Section 5. Simulation results are presented in Section 6. Extending the protocol to multiple core networks is given in Section 7. Finally, our signaling protocol is given in Section 8. Concluding remarks follow in Section 9.

## 2 Quality of Service Model

Our network model is based on the models of [5] and [11]. A *network* is a set of computers connected via point-to-point communication channels. A *flow* is a sequence of packets, all of which originate at the same source computer and are destined to the same computer. All packets of a flow must traverse the network along a fixed path.

Each flow is characterized by its packet rate and an upper bound on its end-to-end delay. Before a source introduces a new flow to the network, enough network resources are reserved from the network to ensure the flow's delay

bound is not violated. If enough resources are not available, the flow is rejected.

Each output channel of a computer is equipped with a scheduler. A scheduler receives packets from flows whose path traverses the output channel of this scheduler. Whenever its output channel becomes idle, the scheduler chooses a received packet and forwards it to the output channel.

A packet *exits* a scheduler when the last bit of the packet is transmitted by its output channel. We adopt the following notation for each flow  $f$  and each scheduler  $s$  along the path of  $f$ .

- $R_f$  bandwidth reserved for flow  $f$ .
- $p_{f,i}$   $i^{\text{th}}$  packet of  $f$ ,  $i \geq 1$ .
- $L_{f,i}$  length of packet  $p_{f,i}$ .
- $L_f^{\text{max}}$  maximum packet size of  $f$ .
- $L_s^{\text{max}}$  maximum packet length at  $s$ .
- $A_{s,f,i}$  arrival time of  $p_{f,i}$  at scheduler  $s$ .
- $E_{s,f,i}$  exit time of  $p_{f,i}$  from  $s$ .
- $C_s$  bandwidth of the output channel of  $s$ .
- $\pi_s$  propagation delay of channel  $s$ .

We define the *start-time*  $S_{s,f,i}$  of packet  $p_{f,i}$  at scheduler  $s$  as follows [5][11]. Assume  $s$  were to forward the packets of  $f$  at *exactly*  $R_f$  bits/sec.. Then,  $S_{s,f,i}$  is the time at which the first bit of  $p_{f,i}$  is forwarded by  $s$ . More formally, let  $f$  be an input flow of scheduler  $s$ . Then,

$$S_{s,f,1} = A_{s,f,1}$$

$$S_{s,f,i} = \max(A_{s,f,i}, S_{s,f,i-1} + L_{s,f,i-1}/R_f), \quad i > 1$$

Assume that scheduler  $s$  forwards the packets of an input flow  $f$  at a rate of at least  $R_f$ . Then, each packet  $p_{f,i}$  exits scheduler  $s$  not much later than its start time,  $S_{s,f,i}$ . We refer to these schedulers as *rate-guaranteed schedulers* [5][11][15]. More formally, a scheduler  $s$  is a rate-guaranteed scheduler iff, for every input flow  $f$  of  $s$  and every  $i$ ,  $i \geq 1$ ,

$$E_{s,f,i} \leq S_{s,f,i} + \delta_{s,f,i}$$

for some constant  $\delta_{s,f,i}$ . We refer to  $\delta_{s,f,i}$  as the *delay* of packet  $p_{f,i}$  at scheduler  $s$ , and we refer to  $S_{s,f,i} + \delta_{s,f,i}$  as the *deadline* of  $p_{f,i}$  at  $s$ .

For example, by choosing  $\delta_{s,f,i} = L_{f,i}/R_f + L_s^{max}/C_s$ ,  $\delta_{s,f,i}$  becomes the *rate-dependent delay* of virtual-clock protocols [17][21][24], since the delay is related to the rate of the flow. Otherwise, if  $\delta_{s,f,i}$  is chosen by flow  $f$  to be unrelated to  $R_f$ , then we say the delay is *rate-independent*.

We assume all schedulers are rate-guaranteed schedulers that provide rate-independent delay. Also, as commonly done in the literature [10][11][25], we assume the delay is fixed for all packets of the same flow, i.e., each flow  $f$  has a delay constant  $\delta_{s,f}$  at scheduler  $s$ .

The delay of a packet across a sequence of rate-guaranteed schedulers is well known [5][8][11][25]. Since the start-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 start-time of the packet. This bound is as follows.

**Theorem 1** (*End-to-End Delay*) Let  $t_1, t_2, \dots, t_k$  be a sequence of  $k$  rate-guaranteed schedulers traversed by flow  $f$ . For all  $i$ ,

$$\begin{aligned} S_{t_k,f,i} &\leq S_{t_1,f,i} + \sum_{x=1}^{k-1} \delta_{t_x,f} + \sum_{x=1}^{k-1} \pi_{t_x} \\ E_{t_k,f,i} &\leq S_{t_k,f,i} + \delta_{t_k,f} \end{aligned} \quad (1)$$

To ensure packets exit by their deadline, a scheduling test must be performed for each scheduler to determine if enough resources are available. Optimum scheduling tests have been presented in [25]. The scheduling test for constant packet sizes has too great of an overhead to be implemented in practice. Furthermore, the Internet is likely to allow variable packet sizes. The scheduling test that allows each flow to have variable packet sizes (up to a maximum) is more relaxed, and is as follows.

**Theorem 2** (*Scheduling Test [25]*) For a scheduler  $s$ , its input set of flows are schedulable (i.e. packets may be forwarded without violating their deadlines) iff, for every input flow  $f$ ,

$$\begin{aligned} \left( \sum g, \quad \delta_{s,g} \leq \delta_{s,f}, \quad L_g^{max} + (\delta_{s,f} - \delta_{s,g}) \cdot R_g \right) \\ \leq \delta_{s,f} \cdot C_s - L_s^{max} \end{aligned} \quad (2)$$

We will focus on the above scheduling test throughout the paper.

### 3 Flow Aggregation

We next describe the aggregation of multiple flows into a single flow. The purpose of flow aggregation is two-fold [5][7]: first, to reduce the number of flows that must be managed by each router, and second, to improve admission control to the network. After introducing the concepts of flow aggregation, we apply flow aggregation to the specific case of a core network.

#### 3.1 Aggregators and Aggregate Flows

A *simple flow* is a potentially infinite sequence of packets generated by an application. That is, the flows considered earlier were simple flows. A flow  $f$  is a *constituent* of flow  $g$  if the packets of  $f$  are a subset of the packets of  $g$ . The reserved rate,  $R_g$ , of aggregate flow  $g$  is simply the sum of the reserved rates of the constituent flows of  $g$ .

A scheduler that receives as inputs a set of flows  $f_1, \dots, f_n$ , and produces as output a single aggregate flow  $g$ , by merging the packets of the input flows, is called an *aggregator*. A scheduler whose set of input flows is the same as its set output flows is called a *non-aggregating scheduler*. A *separator* is simply a process that receives as input an aggregate flow, and produces as output the set of constituents of the input flow. We assume a separator causes no packet delay.

Although it is possible to aggregate several aggregate flows into a new aggregate flow, i.e., multi-level aggregation, for purposes of this paper we consider only single-level aggregation. That is, only simple flows will be aggregated together. Thus, we say that flow  $g$  is the *root* of flow  $f$  at some point in the network if  $g$  contains the packets of  $f$  at this point in the network. Thus, if at some point  $f$  is aggregated into flow  $g$ , then  $g$  is the root of  $f$ , and if  $f$  is not aggregated at this point, then  $f$  is its own root.

For each input flow  $g$  of a scheduler, the scheduler 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  $R_g$ , and the start and finishing times of each packet of  $g$  are defined accordingly

We assume all aggregators are start-time schedulers. Hence, every input packet  $p_{f,i}$  will exit an aggregator  $s$  no later than time  $S_{s,f,i} + \delta_{s,f,i}$ .

Consider as an example the computer with four input/output channels depicted in Figure 1. Here, flows  $c$  and  $e$  are the constituents of  $d$ , and they are separated from  $d$  through a separator. Input flows  $f$  and  $h$  are aggregated

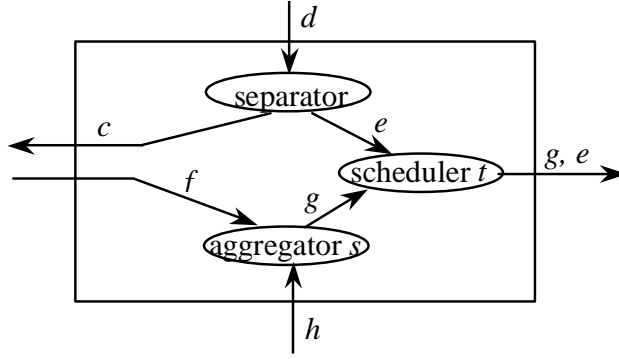


Fig. 1. Flow Aggregation Example

together to form flow  $g$ . I.e.,  $f$  and  $h$  are the constituents of  $g$ . Flows  $e$  and  $g$  are forward to the output channel by a non-aggregating scheduler, and they remain separate in the output channel.

Note that the aggregator and the scheduler are in the same computer, which allows the aggregator to forward packets to the scheduler without delay. In this case, we view the output channel of the aggregator as having infinite capacity.

### 3.2 Fair Flow Aggregation

We next investigate the end-to-end delay of a flow  $f$  as it traverses a network containing flow aggregators and separators. To guarantee a low end-to-end delay to each flow, the behavior of flow aggregators must be restricted, for the following reason.

Consider again Figure 1. Scheduler  $t$  is not aware that its input flow  $g$  is the aggregation of two flows,  $f$  and  $h$ . Hence, scheduler  $t$  only guarantees the exit time of each packet  $p_{g,j}$  to be at most  $S_{t,g,j} + \delta_{t,g}$ , and makes no guarantees about the exit times of packets from  $f$  and  $h$ .

Since the exit time at scheduler  $t$  depends on the start-time of each packet, we would like to bound the start-time of flow  $g$  at scheduler  $t$  as if no aggregation had occurred. For example, assume instead that  $s$  is a non-aggregating scheduler, and thus,  $f$ ,  $h$ , and  $e$  are individual input flows of scheduler  $t$ . Then, from Theorem (1), and  $s$  being a start-time scheduler,

$$S_{t,f,i} \leq S_{s,f,i} + \delta_{s,f}$$

Let  $p_{g,j} = p_{f,i}$ , i.e., the  $j$ th packet from  $g$  is the  $i$ th packet from  $f$ . We would like to obtain a similar relationship between  $S_{t,g,j}$  and  $S_{s,f,i}$  as the one above between  $S_{t,f,i}$  and  $S_{s,f,i}$ . This will bound the exit time from  $t$  of  $p_{f,i}$  as a

function of  $S_{s,f,i}$ . We choose the bound as follows.

$$S_{t,g,j} \leq S_{s,f,i} + \lambda_{s,f} \quad (3)$$

for some constant  $\lambda_{s,f}, \lambda_{s,f} \geq 0$ . From (3),  $t$  being a start-time scheduler, and  $p_{f,i} = p_{g,j}$ , we conclude,

$$E_{t,f,i} = E_{t,g,j} \leq S_{t,g,j} + \delta_{t,g} \leq S_{s,f,i} + \lambda_{s,f} + \delta_{t,g}$$

Therefore, the bound of Relation (3) above can be used to compute the exit time of  $p_{f,i}$  at scheduler  $t$  even though scheduler  $t$  is not aware of flow  $f$ . We formalize this definition next.

**Definition 3** (*Fair Aggregators*) *Let  $g$  be an input flow of scheduler  $s$ , and let  $g$  be the parent of  $f$ . Scheduler  $s$  is fair, iff, there exists a constant  $\lambda_{s,f}$  such that*

$$S_{t,g,j} \leq S_{s,f,i} + \lambda_{s,f}$$

where  $t$  is the next scheduler of  $g$ , and  $p_{g,j} = p_{f,i}$ .

Note that for a non-aggregating scheduler  $s$ , if  $s$  is a start-time scheduler, then from Theorem 1,  $\lambda_{s,f} = \delta_{s,f}$ . Thus, all non-aggregating start-time schedulers are fair.

As mentioned earlier, all aggregators are assumed to be start-time schedulers. However, it is easy to show that the opposite is not always true, i.e., an aggregator can be a start-time scheduler, but not a fair aggregator [5]. That is,  $S_{t,g,j}$  may diverge from  $S_{s,f,i}$ . Further below we present a simple technique to implement a fair aggregator.

### 3.3 End-to-End Delay

We next discuss the end-to-end delay of a flow in the presence of aggregation. If each aggregator is fair, then the increase in the start-time of each packet is bounded as it goes through aggregation. Therefore, we expect the end-to-end delay to be bounded in a way similar to that of Theorem 1.

**Theorem 4** (*Aggregate End-to-End Delay [5]*) *Let  $t_1, t_2, \dots, t_k$  be the path of fair schedulers traversed by flow  $f$ . Define  $\text{root}(s, f, i)$  to be the packet of the*

root flow of  $f$  at  $s$  corresponding to packet  $p_{f,i}$ . I.e.,  $\text{root}(s, f, i) = (s, g, j)$ , where the root of  $f$  at  $s$  is  $g$  and  $p_{g,j} = p_{f,i}$ . Then,

$$\begin{aligned} S_{\text{root}(t_k, f, i)} &\leq S_{t_1, f, i} + \sum_{x=0}^{k-1} \lambda_{\text{root}(t_x, f, i)} \\ E_{t_k, f, i} &\leq S_{t_1, f, i} + \sum_{x=0}^{k-1} \lambda_{\text{root}(t_x, f, i)} + \delta_{\text{root}(t_k, f, i)} \end{aligned}$$

By comparing Theorem 1 and Theorem 4, we see that the end-to-end delay obtained via flow aggregation is similar to the end-to-end delay obtained without flow aggregation. However, the following must be stressed. Consider a pair of flows  $f$  and  $h$  that are aggregated to form flow  $g$ . Since each scheduler cannot distinguish between  $f$  and  $h$ , both flows will experience the same per-hop delay throughout the entire path of  $g$ . Therefore,  $f$  and  $h$  should be aggregated together only if they have similar delay requirements.

### 3.4 Satisfiable Flow Sets

A set of flows is satisfiable at a scheduler if packets can be scheduled in a manner such that the deadline of each packet is not violated. In this section, we examine whether aggregating flows affect their satisfiability at a scheduler. As argued above, all flows that are aggregated together will receive the same per hop delay as long as they remain aggregated.

**Theorem 5** (*Scheduling Test with Aggregation [7]*) *Let  $f_1, \dots, f_n, h_1, \dots, h_m$ , be the input flows to a scheduler  $s$ . Let each flow in  $f_1, \dots, f_n$ , have the same per-hop delay at  $s$ , denoted  $\delta_{s,f}$ . Then, if flows  $f_1, \dots, f_n, h_1, \dots, h_m$  are schedulable at  $s$ , then so are flows  $g, h_1, \dots, h_m$ , where  $g$  is the aggregation of flows  $f_1, \dots, f_n$ , and  $\delta_{s,g} = \delta_{s,f}$ .*

From the above theorem, the same set of flows can be supported if a subset of these flows are aggregated together. Thus, by aggregating flows together, we reduce the total number of flows traversing a network, without sacrificing the set of schedulable flows. However, there is a subtle drawback which is not apparent from Theorem 5, which will be discussed in Section 4, where we apply flow aggregation to a core network.

### 3.5 Implementing Fair Aggregators

One final component is necessary in our discussion of flow aggregators, namely, the construction of fair aggregators. There are several techniques to do so [5], each with its own advantages and disadvantages. In this paper, we present a new fair aggregator that suits well our aggregating core network of Section 4.

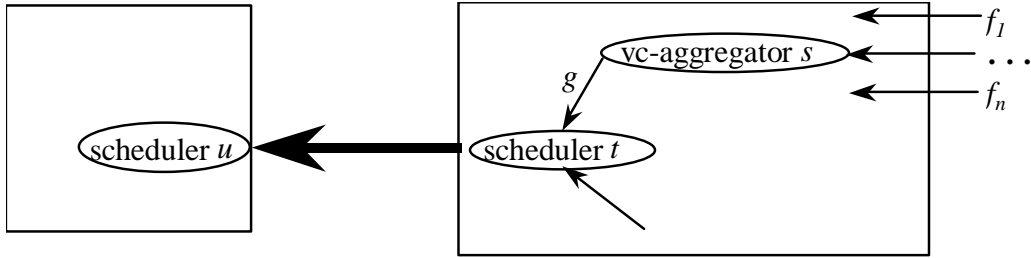


Fig. 2. Aggregator-Scheduler Pair

Throughout the paper, we assume the behavior of an aggregator  $s$  is as follows. Let  $f_1, \dots, f_n$  be the input flows of  $s$ , and let  $g$  be its output flow. Thus,  $R_g = \sum_{x=1}^n R_{f_x}$ . Let  $C_s$ , where  $C_s \geq R_g$ , be the capacity of the output channel of  $s$ . Aggregator  $s$  assigns timestamps to packets in the same way as done in the virtual clock protocol, and packets are forwarded in order of increasing timestamp. However, aggregator  $s$  is not work-conserving, as follows. If at time  $t$  it forwards a packet of size  $L$ , then it will not forward the next packet until  $\frac{L}{R_g}$  seconds later.

We refer to the above aggregator as a *vc-aggregator*. In Section 4, we make use of the configuration shown in Figure 2. Here, one computer receives a sequence of flows  $f_1, \dots, f_n$ , (among other flows), and these flows are aggregated together into a single flow  $g$ . Then, flow  $g$  is given to another scheduler  $t$  that forwards its packets to the output channel of the computer, and into scheduler  $u$  of the next computer.

**Theorem 6** (*Aggregator-Scheduler Pair*) Consider the scenario in Figure 2. For every  $m$  and  $i$ ,

$$S_{root(u,f_m,i)} \leq S_{s,f_m,i} + \frac{L_g^{max}}{R_{f_m}} + \delta_{t,g}$$

A proof of this Theorem may be found in [4].

## 4 Single Core Network

In [5][7], we introduced the theory of flow aggregation. In this section, we apply this theory to the specific case of a core network where each flow has rate-independent delay.

Consider Figure 3 which depicts a core network. The core network consists of core routers, ingress/egress routers, and access networks. Access networks are attached to the core network via the ingress/egress routers.

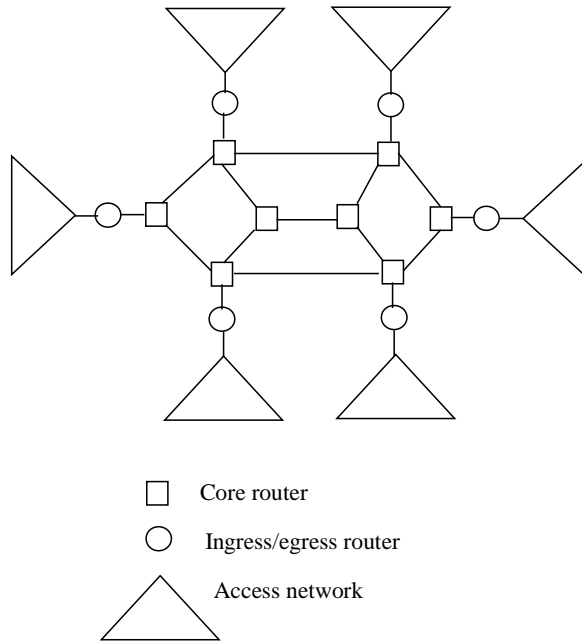


Fig. 3. Core Network

Our goal is to reduce the number of flows that traverse the core network, and thus improve scalability. As flows enter through an ingress router, they are aggregated together. Aggregate flows traverse the core routers until they arrive to their corresponding egress router. Then, each aggregate flow is separated into its constituent flows, who in turn continue into the access networks of their destination.

Recall that only flows with the same start-time delay can be aggregated together. Since the scheduling test of Theorem 2 requires a linear number of steps in the number of different start-time delays, each flow cannot choose an arbitrary value for its start-time delay. This would significantly increase the cost of admission control. However, it is likely that only a small set of start-time delays will suffice for most flows. Thus, we assume there is a standard set of start-time delays from which each flow will chose its own. We denote the cardinality of this set by  $D$ .

According to the above, each ingress router aggregates those flows which are destined to the same egress router and which have chosen the same start-time delay. Thus, if there are  $M$  egress routers, each ingress routers generates a total of  $D \cdot M$  aggregate flows. Assuming there are also  $M$  ingress routers, the total number of flows through the core routers is at most  $D \cdot M^2$ . Although not a trivial number, both  $M$  and  $D$  will be relatively small, keeping the number of flows through the core of the network manageable. Furthermore, assuming that we also have at least  $M$  core routers, the average number of flows managed per core router will be about  $D \cdot M$ , which is a sensible number.

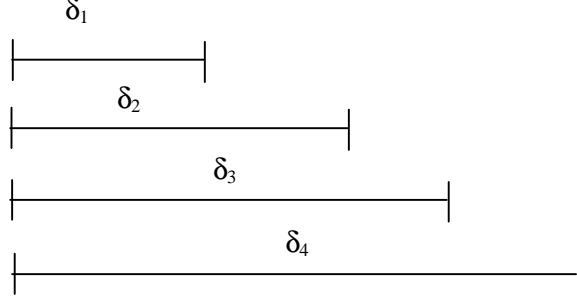


Fig. 4. Delay Values in Satisfiability Test

An alternative way to write Theorem 2 is as follows. Let  $\Delta$  be the set of delay values for scheduler  $s$ , and let  $F$  be the set of input flows of  $s$ . The set  $F$  is schedulable iff, for each value  $\delta' \in \Delta$ ,

$$\begin{aligned} \left( \sum f, \delta_{s,f} \leq \delta', L_f^{\max} + (\delta' - \delta_{s,f}) \cdot R_f \right) \\ \leq \delta' \cdot C_s - L_s^{\max} \end{aligned} \quad (4)$$

As an example, consider Figure 4. Let flows  $a_1, \dots, a_n$  have a delay value of  $\delta_1$  and a maximum packet size  $L_a^{\max}$ . Also, let flows  $f_1, \dots, f_m$  have a delay value of  $\delta_2$  and a maximum packet size  $L_f^{\max}$ . Then, the test (4) above for  $\delta' = \delta_1$  is simply

$$n \cdot L_a^{\max} \leq \delta_1 \cdot C_s - L_s^{\max}$$

and the test for  $\delta' = \delta_2$  is simply

$$\begin{aligned} m \cdot L_f^{\max} + n \cdot L_a^{\max} + \sum_{x=1}^n (\delta_2 - \delta_1) \cdot R_{a_x} \\ \leq \delta_2 \cdot C_s - L_s^{\max}. \end{aligned}$$

Consider instead flow aggregation, and assume flows  $a_1, \dots, a_n$  are aggregated into a single flow  $b$ , where  $R_b = \sum_{x=1}^n R_{a_x}$ , and flows  $f_1, \dots, f_m$  are aggregated into a single flow  $g$ , where  $R_g = \sum_{x=1}^m R_{f_x}$ . Then, the test (4) above for  $\delta' = \delta_1$  becomes

$$L_a^{\max} \leq \delta_1 \cdot C_s - L_s^{\max}.$$

and the test for  $\delta' = \delta_2$  becomes

$$\begin{aligned} L_g^{\max} + L_b^{\max} + (\delta_2 - \delta_1) \cdot R_b \\ = L_f^{\max} + L_a^{\max} + \sum_{x=1}^n (\delta_2 - \delta_1) \cdot R_{a_x} \\ \leq \delta_2 \cdot C_s - L_s^{\max}. \end{aligned}$$

We can clearly see that the left-hand side of the inequality has decreased significantly with flow aggregation. In particular, only a single packet of the flows  $a_1, \dots, a_n$  and a single packet from flows  $f_1, \dots, f_m$  is included. Thus, the scheduling test becomes more relaxed, and a larger number of flows may be supported.

There is however, a subtle issue that must be addressed. Ingress routers use the configuration of Figure 2, and from Theorem 6, the increase in the start-time of a packet  $p_{f,i}$  traversing an ingress router  $s$  is at most

$$\frac{L_f^{max}}{R_f} + \delta_{t,g}$$

In a core network without aggregation, the increase in the start-time would simply be  $\delta_{t,g} = \delta_{t,f}$ . Thus, we have an additional increase of  $\frac{L_f^{max}}{R_f}$  in the end-to-end delay. To compensate for this, flow  $f$  may have to choose a per-hop delay smaller by  $\frac{L_f^{max}}{h \cdot R_f}$ , where  $h$  is the number of hops traversed by  $f$  in the core network.

For example, without aggregation, flow  $f$  may choose a per-hop delay of  $\delta_4$  in Figure 4, but under flow aggregation it may have to choose  $\delta_2$ . In principle, the number of packets counted in Relation (4) decreases for  $\delta' = \delta_4$  if  $f$  is aggregated, and assuming there is at least one flow that chose  $\delta_2$ , the number of packets counted in Relation (4) for  $\delta' = \delta_2$  and for  $\delta' = \delta_3$  does not increase. However, there is a total increase of  $(\delta_3 - \delta_2) \cdot R_f$  to the test case of  $\delta' = \delta_3$ , and a total of  $(\delta_4 - \delta_2) \cdot R_f$  is added to the test case of  $\delta' = \delta_4$ . It can be shown that either of these is about  $\frac{L_f^{max}}{h}$ .

Thus, the test cases for  $\delta_1$  and  $\delta_2$  remain unaffected, the test case of  $\delta_3$  increases a little, and the test case for  $\delta_4$  decreases. In general, we expect an improvement in the number of flows that can be accepted by a scheduler. We will investigate this further via simulation in Section 6.

A final issue to consider is that due to the increase in delay of  $\frac{L_f^{max}}{R_f}$ , a flow may be unable to meet its end-to-end delay, even if it chooses the smallest per-hop delay  $\delta_1$ . In this case, either the flow is not aggregated (thus avoiding the increase of  $\frac{L_f^{max}}{R_f}$ ), or the flow is rejected.

To allow the largest possible number of flows, we choose not to aggregate the flow. However, if the flow is not aggregated, this has the potential of increasing significantly the number of flows managed by core routers. To avoid this increase in the state managed by core routers, we take advantage of the dynamic packet state technique [9][19][13].

## 5 Dynamic Packet State

Dynamic packet state is a technique by which the deadline of each packet is computed from information in the packet’s header, independently of other packets. In this manner, no per-flow state needs to be maintained at each router [9][19][13]. Dynamic packet state has been mostly applied to rate-dependent delay, but it is also applicable to rate-independent delay as follows.

Consider two consecutive schedulers,  $s$  and  $t$ , of flow  $f$ . From Theorem 1 and the definition of  $S$ ,

$$A_{t,f,i} \leq S_{t,f,i} \leq S_{s,f,i} + \delta_{s,f} + \pi_s.$$

Assume the above is forced to be an equality, that is,

$$A_{t,f,i} = S_{s,f,i} + \delta_{s,f} + \pi_s$$

for all  $p_{f,i}$ . Then, the following holds.

$$S_{t,f,i} = A_{t,f,i} = S_{s,f,i} + \delta_{s,f} + \pi_s \tag{5}$$

Note that  $\delta_{s,f}$  is independent of other flows, and thus,  $p_{f,i}$  could carry  $\delta_{s,f}$  in its header.

Before  $s$  forwards  $p_{f,i}$  to  $t$ ,  $s$  computes  $S_{t,f,i}$  as given in Equation (5), and stores  $S_{t,f,i}$  in  $p_{f,i}$ . Hence,  $t$  receives the value of  $S_{t,f,i}$  from  $s$ . However, Equation (5) is valid only if  $A_{t,f,i} = S_{t,f,i}$ . To ensure this, if  $p_{f,i}$  arrives earlier than  $S_{t,f,i}$ , it is kept in a buffer until time  $S_{t,f,i}$ , then it is considered “arrived” and may be scheduled for transmission. Thus, the deadline  $S_{t,f,i} + \delta_{t,f}$  of  $p_{f,i}$  at  $t$  is computed without per-flow state.

Using dynamic packet state solves the problem of allowing flows into the core network that have not been aggregated. Since each packet carries all the information to compute its deadline at each hop, no per-flow state is needed in the core routers.

Note that the  $M \cdot D$  flows maintained by each ingress router are maintained using a traditional state-full approach. This however does not introduce a problem, since it is easy to show that a router may maintain some traditional flows along with some dynamic packet-state flows without violating the deadline of any packet.

Note also that it is possible for the ingress router to use dynamic packet state with each of the  $M \cdot D$  aggregate flows it maintains, rather than using a

Routing algorithm	shortest path
Duration of simulation	30 min.
Link capacity	1 Mbit/sec.
Max. packet size	8 Kbits
Flow arrivals	Poisson, 50 to 300 calls/sec.
Flow duration	exponential, avg. 5 min.
Flow data rate	50 Kbit/sec. to 1 Mbit/sec.
Flow end-to-end delay	15 to 50 msec.
Per-hop delay $\delta$	$5 \cdot 1.2^n, 0 \leq n \leq 9$

Table 1  
Simulation parameters.

traditional state-full approach across the core network. In this manner, each core router will maintain no per-flow state, neither for aggregated flows nor for simple flows.

## 6 Simulation

In this section, we simulate the behavior of our protocols and compare the performance of a core network with and without aggregation. The topology of our core network is shown in Figure 5, which is inspired on the early NSFNET network topology.

We have chosen this topology for two reasons. First, it is based on a topology actually used in practice. More importantly, it is a topology that is not well suited for aggregation (as explained below), and thus, if aggregation is useful in this topology, it is likely to be of use in many topologies.

The parameters of the simulation are shown in Table 1.

The source and destination of each flow is chosen at random. The end-to-end delay requirements of the flow are chosen uniformly between its maximum and minimum value. Likewise, the data rate of the flow is chosen uniformly. Ten different values of  $\delta$  are supported, with a 20% increase from one to the next, starting with 5 milliseconds and ending in 25 milliseconds.

Note that the average value of  $L^{max}/R$  is 16 milliseconds. Furthermore, the maximum number of hops is three (with an average of two). Therefore, many calls will not be aggregated simply because this additional delay is too large,

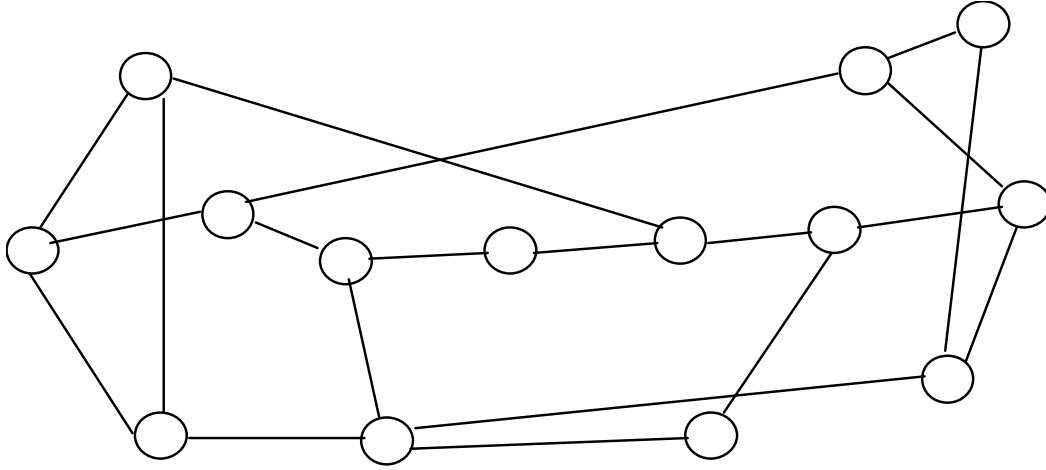


Fig. 5. Simulated Topology

and even with the smallest value of  $\delta$  it may not be possible to achieve the desired end-to-end delay, and thus the flow must be sent without aggregation. Furthermore, since there are so few hops, the value of  $L^{max}/R$  must be spread over two hops on average, for a decrease in the value of  $\delta$  chosen of about 8 milliseconds. This makes call admission more difficult. Nonetheless, even under these circumstances, the core network with aggregation outperformed the core network without aggregation.

Figure 6 shows the total number of flows dropped as a function of the flow arrival rate. Figure 7 shows the flows dropped as a percentage of the total number of flows generated. With flow aggregation, the percentage of calls dropped consistently maintained itself about 11% lower than without flow aggregation.

## 7 Multi-Core Networks

We next consider a more general network topology. Rather than considering a single core network, we consider a small set of core networks, as shown in Figure 8. Neighboring core networks are connected via gateway routers, and packets may traverse multiple core networks on their path to their ultimate destination.

Our goal is to maintain a state-less network while maintaining the advantages of flow aggregation as discussed earlier. In a multi-core network, only ingress nodes will maintain state for each flow that enters the network. Neither gateway routers nor core routers maintain per-flow state. Therefore, for each flow there is only one router that maintains detailed information about this flow, namely, the ingress router of the flow.

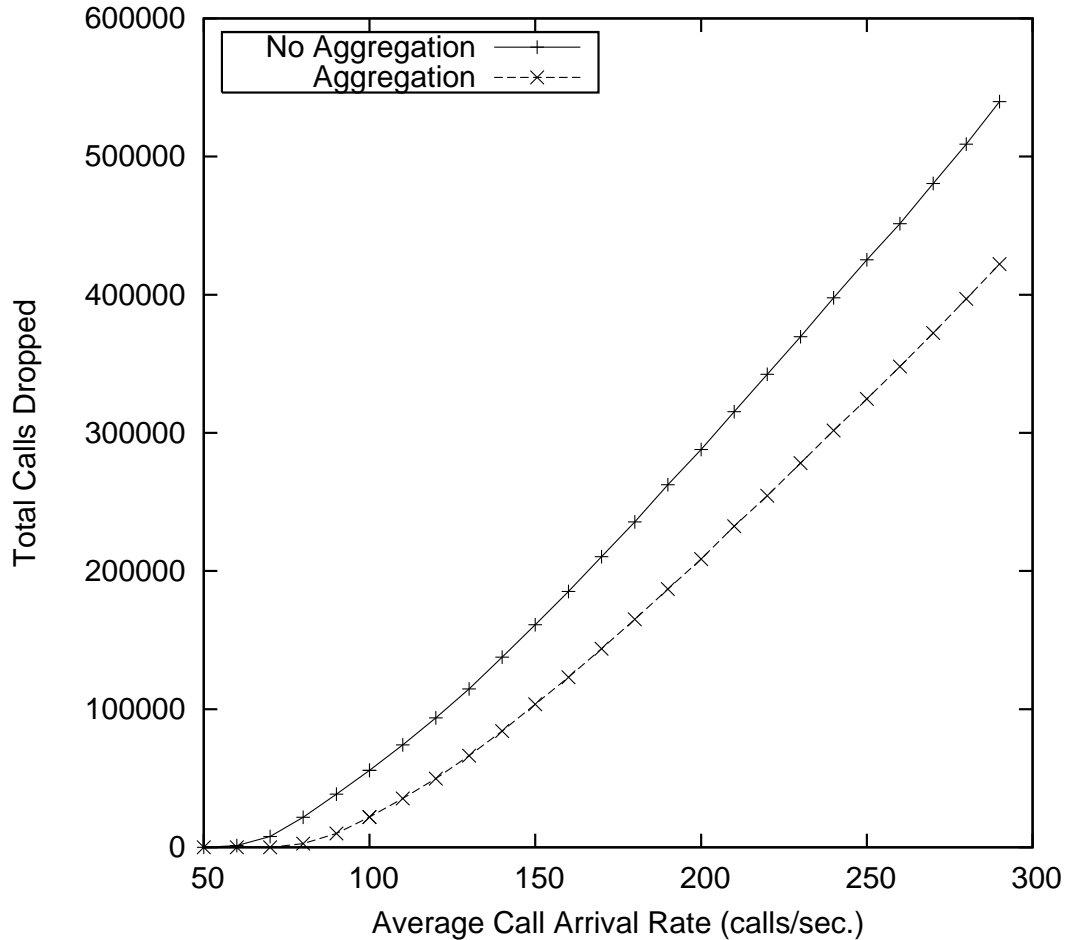


Fig. 6. Total Calls Dropped

Flows are aggregated as before. That is, all flows through a core having the same entry point, exit point, and per-hop delay are aggregated together. However, since gateways are, in addition to ingress routers, entry points into a core network, then gateways must also be involved in flow aggregation. Since gateways do not maintain per flow state, their aggregation must be done in a state-less manner.

The use of dynamic packet-state requires each packet to include its per-hop delay in its header. We assume that each flow is free to choose a different per-hop delay for each core network it traverses. Therefore, each packet carries a list of per-hop delays, one for each traversed core network. As the packet crosses into a new core network, the next per-hop delay value in the header is used.

Consider again Figure 2. Let the first computer be an ingress router, and an input flow  $f$  is being aggregated to cross the core network. Since this is an ingress router, the router maintains per-flow state for  $f$ . Before each packet of  $f$  departs the computer, its start-time at scheduler  $t$  must be included in its

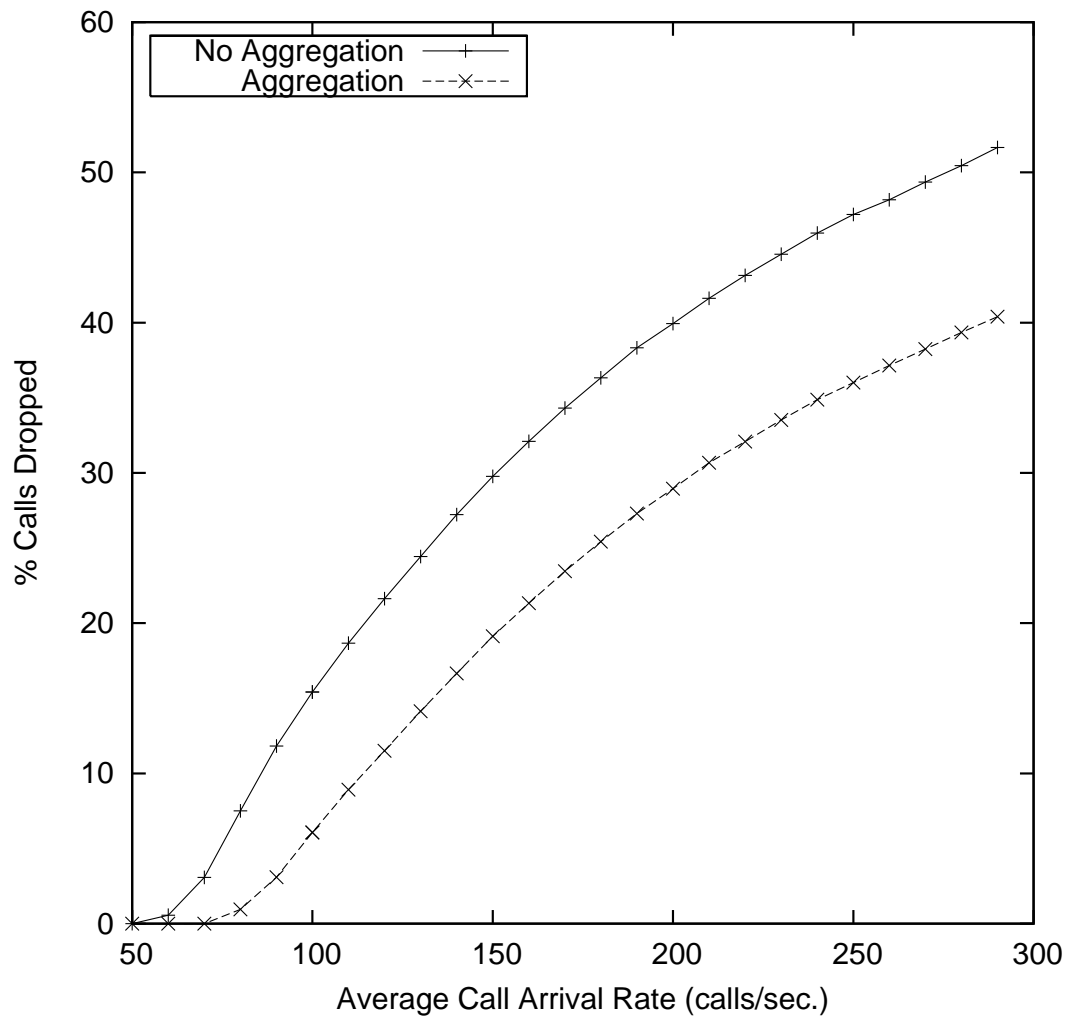


Fig. 7. Percent Calls Dropped

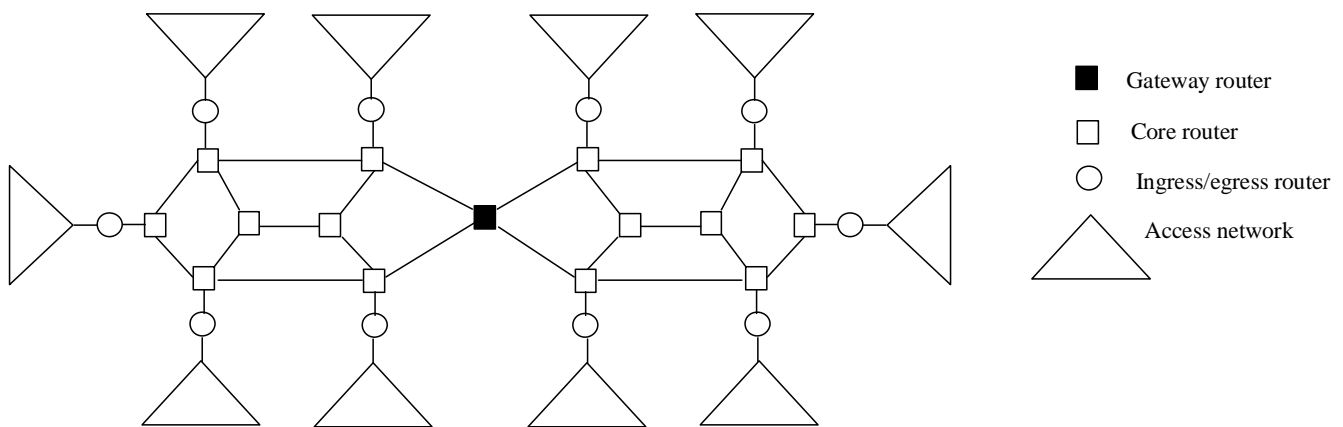


Fig. 8. Multiple Core Networks

header to support dynamic packet-state scheduling. However, in addition, we add to the header the start-time of the packet at the vc-aggregator  $s$ <sup>1</sup>. The reason for this is given below.

Assume flow  $f$  exits the core network via a gateway router into a neighboring core network. Since the gateway router does not maintain per-flow information for  $f$ , it must be able to compute its start-time from the header of each packet. Let  $\delta_f$  be the per-hop delay of flow  $f$  for this core network. Each packet of flow  $f$  incurs an increase in its start-time of  $\delta_f$  per hop. Also, from scheduler  $s$  to scheduler  $u$ , each packet has an increase in start-time of at most

$$\frac{L_f^{max}}{R_f} + \delta_f$$

(see Theorem 6). Therefore, the total increase in per-hop delay from the vc-aggregator at the ingress router to the vc-aggregator at the gateway router is at most

$$\frac{L_f^{max}}{R_f} + \delta_f \cdot h$$

where  $h$  is the number of hops to the gateway router.

Hence, in addition to the value  $\delta_f$ , each packet of  $f$  carries also the value  $\frac{L_f^{max}}{R_f}$  in its header. In this way, the gateway can compute the start-time of the packet at its vc-aggregator without per-flow state. Thus, the packets of  $f$  can be aggregated and scheduled at the gateway without per-flow state, as desired.

## 8 Signaling Protocol

We next present our signaling protocol to guarantee QoS to each flow, without maintaining per-flow state at each scheduler. We begin by examining how much information is needed to determine if a set of flows is schedulable. Then, we present the general method of our signaling protocol. Finally, we refine the method for our specific case of a multi-core network.

From Relation (4), each router requires to know, for each per-hop delay supported, how many flows have chosen each delay value, and what is the sum of their rates. Therefore, each router only needs to maintain a constant amount

<sup>1</sup> Note that both values need not be included in their entirety. One of the values plus the difference between them suffices.

of information per output channel to determine if the set of flows traversing the output channel are satisfiable.

The objective of the signaling protocol is to maintain the above information current at each router, even though new flows arrive to the scheduler and existing flows terminate. To achieve this in a fault-tolerant way, soft-state is desired. That is, information about a flow should be removed from a scheduler in the event that the flow abnormally terminates.

To maintain soft-state, each ingress router, for each flow, sends **Refresh** messages periodically along the path of the flow. These messages contain the reservation information of the flow such as its per-hop delay and reserved rate. We assume that the network will forward **Refresh** messages with sensible reliability. This can be accomplished in many ways, such as reliably transferring **Refresh** messages across each hop, or reserving buffer space and a small amount of bandwidth for **Refresh** messages. **Refresh** messages will not be a performance drain for the network, since they are sent infrequently.

For each delay value  $\delta$ , each scheduler  $s$  maintains a rate variable,  $SumRates_{s,\delta}$ , where it stores the sum of the reserved rates of its flows, and a count variable,  $SumFlows_{s,\delta}$  where it stores the number of flows with delay  $\delta$ . Every  $T$  seconds, where  $T$  is a predefined constant,  $s$  updates its state in the following way:

$$\begin{aligned} SumRates_{s,\delta} &:= ShadowSumRates_{s,\delta}; \\ SumFlows_{s,\delta} &:= ShadowSumFlows_{s,\delta}; \\ ShadowSumRates_{s,\delta} &:= 0; \\ ShadowSumFlows_{s,\delta} &:= 0; \\ SwapBits_s &:= \neg SwapBits_s; \end{aligned}$$

Each **Refresh** message from flow  $f$  contains  $R_f$ , and the delay value  $\delta$  for each core network traversed. The objective is to add  $R_f$  to  $ShadowSumRates_{s,\delta}$  and add one to  $ShadowSumFlows_{s,\delta}$  exactly once before the above sequence of assignments are done. In this way,  $R_f$  is always included in  $SumRates_{s,\delta}$ , and the flow is counted only once in  $SumFlows_{s,\delta}$ .

The purpose of  $SwapBits_s$  is to ensure flow  $f$  is added to the shadow variables only once before each state update. The **Refresh** message from flow  $f$  contains a bit,  $b_{s,f}$ , for each scheduler  $s$  in its path. Bit  $b_{s,f}$  indicates the last value of  $SwapBits_s$  encountered by a **Refresh** message from  $f$ . The values of  $f$  are added to the shadow variables only if the state has been updated since the last **Refresh** message from  $f$ . That is, the following is performed whenever  $s$  receives a **Refresh** message from  $f$ .

$$\begin{aligned} \text{if } b_{s,f} \neq SwapBits_s \text{ then} \\ \quad ShadowSumRates_{s,\delta} &:= \end{aligned}$$

```

         $ShadowSumRates_{s,\delta} + R_f;$ 
         $ShadowSumFlows_{s,\delta} :=$ 
             $ShadowSumFlows_{s,\delta} + 1;$ 
         $b_{s,f} := SwapBits_s$ 
    end if
    forward Refresh towards the destination of  $f$ 

```

When the destination receives this message, it returns a **RefreshAck** message back to the source of  $f$ , containing all bits  $b_{s,f}$  accumulated during the traversal of the **Refresh** message. A new **Refresh** message is not sent until a **RefreshAck** is received for the previous **Refresh** message.

The procedure is similar when a new flow  $f$  is created. A **Reserve** message is sent towards the destination. This message collects in  $b_{s,f}$  the value of  $SwapBits_s$  as it traverses scheduler  $s$ . However, the information of  $f$  is added to *both* the shadow variables and the regular variables of each scheduler, because this is the first time this information arrives to the scheduler.

The destination returns a **ReserveAck** to the ingress router of  $f$  including all bits  $b_{s,f}$  learned during the reservation. If not enough bandwidth is available,  $s$  returns a **Reject** message along the path to the source of  $f$  to release the resources of  $f$ . Upon receiving a **Reject** message for flow  $f$ , scheduler  $s$  performs the following.

```

         $SumRates_{s,\delta} := SumRates_{s,\delta} - R_f;$ 
         $SumFlows_{s,\delta} := SumRates_{s,\delta} - 1;$ 
    if  $SwapBits_s = b_{s,f}$  then
         $ShadowSumRates_{s,\delta}$ 
             $:= ShadowSumRates_{s,\delta} - R_f;$ 
         $ShadowSumFlows_{s,\delta}$ 
             $:= ShadowSumRates_{s,\delta} - 1;$ 
    endif
    forward Reject towards the ingress router of  $f$ 

```

As an example, consider Figure 9. In Figure 9(a), an existing flow  $f$  traverses scheduler  $s$ , and we assume its information is included in all four variables of the scheduler. In Figure 9(b), the **Reserve** message of a new flow  $g$  is received. Since  $g$  is new, the information of  $g$  is added both to the shadow variables and the regular variables. In addition, the source of  $g$  learns the value of  $SwapBits_s$ . In Figure 9(c), a state update occurs: the shadow variables are assigned to the regular variables (which does not change their value since they currently are the same), the shadow variables are reset to zero, and  $SwapBits_s$  is flipped. In Figure 9(d),  $f$  sends a **Refresh** message, and since  $SwapBits_s$  has changed, the information of  $f$  is added to the shadow variables, and  $f$  learns the new value of  $SwapBits_s$ . In Figure 9(e),  $g$  sends a **Refresh** message, and

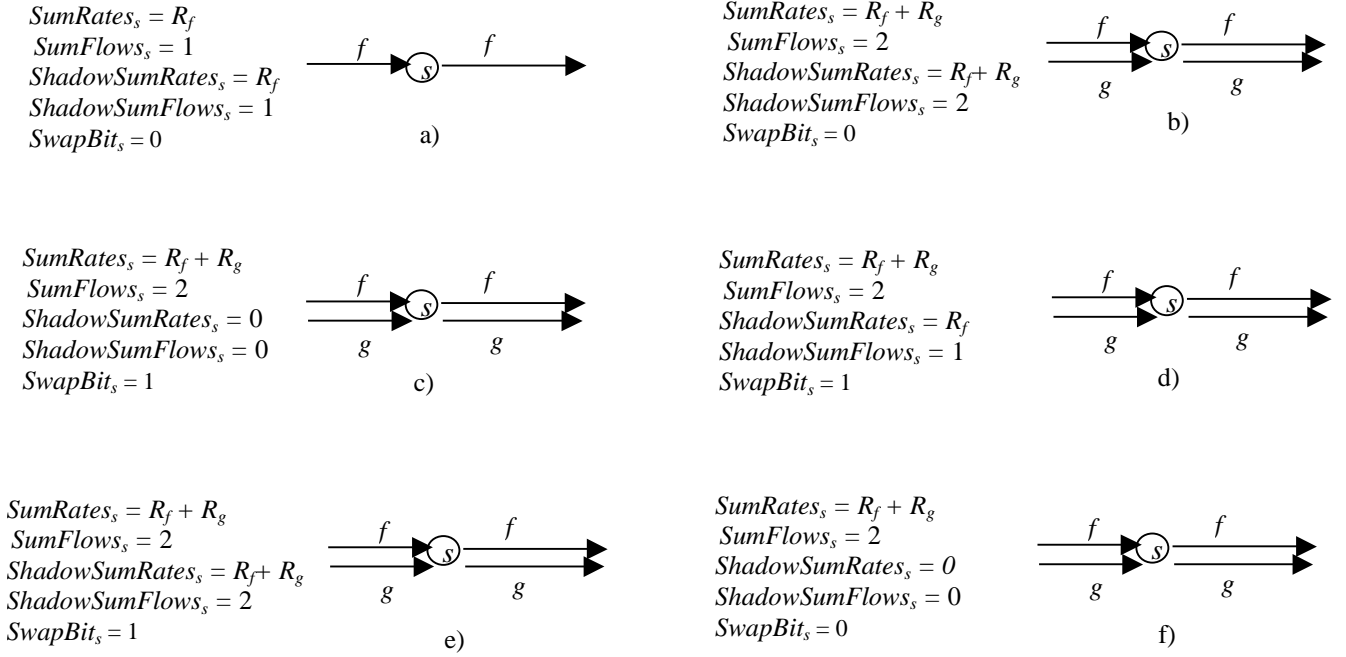


Fig. 9. Signaling example

likewise, the information of  $g$  is added to the shadow variables, and  $g$  learns the new value of  $SwapBits_s$ . Finally, in Figure 9(f), another state update occurs.

We next address how often the source of a flow should send a Refresh message. We assume signaling messages include the time they were created. Furthermore, we assume a bound,  $D$ , on the time for a signaling message to traverse the network. A signaling message created at time  $t$  is discarded by a scheduler if it is received at a time greater than  $t + D$ . State updates of different schedulers are not required to be synchronized. The only assumption is that each scheduler performs updates at least  $T$  seconds apart.

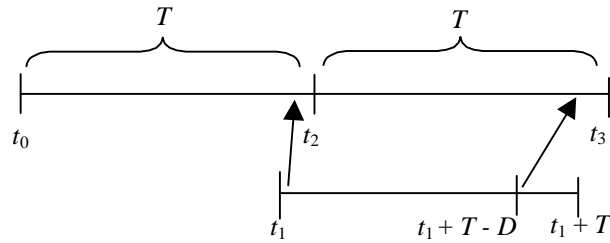


Fig. 10. Timing of Refresh messages

Consider Figure 10. Let  $s$  be a scheduler in the path of flow  $f$ . A state update occurs in  $s$  at time  $t_0$ , and another at time  $t_2$ . At time  $t_1$ , the source of  $f$  transmits a Refresh message, which arrives at  $s$  in the interval  $(t_0, t_2)$ . Thus, at least one Refresh message from  $f$  must arrive at  $s$  in the interval  $(t_2, t_3)$ . In

the worst case,  $t_1$  is almost equal to  $t_2$ , which implies that the next **Refresh** message must arrive at  $s$  no later than  $t_1 + T$ , i.e., it must be sent no later than  $t_1 + T - D$ . Furthermore, the next **Refresh** cannot be sent until a **RefreshAck** is received for the first **Refresh**, which at the latest will occur at time  $t_1 + 2 \cdot D$ . Thus, we require

$$t_1 + 2 \cdot D < t_1 + T - D.$$

That is,  $3 \cdot D < T$ , and the interval between successive transmissions of **Refresh** messages should be at most  $T - D$ .

### 8.1 Fault Tolerance

We next examine the effect of network faults on our signaling protocol. In particular, we consider the following: 1) delayed or lost signaling messages, 2) link failure, 3) process failure, 4) routing changes.

First, consider case 1). As mentioned in the previous section, a signaling message that is excessively delayed (more than  $D$  seconds) is discarded in the network. Consider then lost messages. If an ingress router does not receive a **RefreshAck** due to a lost message, then the source terminates the flow. This should occur rarely, since signaling messages are treated reliably. However, the network resources of the flow must be de-allocated. This happens automatically as follows. At any scheduler  $s$ , within  $T$  seconds, the shadow variables are assigned to the regular variables, and the shadow variables are reset to zero. This is performed again within  $T$  more seconds. Thus, within  $2 \cdot T$  seconds of the failure, the information of the flow disappears from  $s$ .

Cases 2) and 3) are similar to 1). However, due to the failure, the path from source to destination may change before the flow is terminated. For correctness, future **Refresh** messages and data packets must follow the original path where the resources are allocated.

There are multiple techniques to restrict a flow to a given path, such as [3] and [20], among others. Due to lack of space we do not discuss their application to our signaling protocol. Instead, as an example, we present the following simple technique. Let  $d$  be the destination of  $f$ . Each ingress router and gateway router  $c$  maintains a bit,  $Route_{c,d}$ , which is flipped every time the path across its core network to reach destination  $d$  changes. The ingress router of  $f$  learns the value of  $Route_{c,d}$  when it reserves resources for  $f$ , and stores the value in a bit,  $r_{c,f}$ , which is included in every subsequent message (either signaling or data) from  $f$ . Thus, each message carries a bitmap with a bit for every core network along the path of  $f$ . If gateway  $c$  receives a message with

$Route_{c,d} \neq r_{c,f}$ , then the message is dropped. Thus, if the path of  $f$  changes, its messages are dropped where the change occurred, causing the termination of  $f$ . If routing changes are rare, this should not significantly affect flow sources significantly. This technique requires that the routing table entry of each destination not be allowed to change more than once every  $T$  seconds.

## 8.2 Signaling for Aggregation

We enhance the above signaling scheme for the case of aggregation. Each signaling message carries, for each core network it traverses, a bit indicating if the flow is aggregated in this core network. If a flow is not aggregated across a core network, then it is treated as described above.

Consider a flow that is aggregated across a core network. The signaling message will traverse across core routers and gateway routers.

Consider the gateway router. The gateway must keep track of, for each per-hop delay value and each outgoing router from its core network, the number of flows being aggregated and the sum of their rates. The gateway router can maintain this information also as described above.

The difference lies in the core routers, in particular, with respect to aggregate flows. Note that each aggregate flow must be counted only *once*. Thus, if signaling messages from the constituent flows are received, then, although their rates should be added to the state maintained by the core router, the count of flows should be increased only once. To ensure this, if the bit in the header indicates that the signaling message is from a flow that has been aggregated, then *only* its rate is added to the state of the core router, and the count of flows is *not* increased.

This leaves us with the remaining task of how to ensure the core router counts the aggregate flow once. To do so, the aggregating router, either an ingress router or a gateway router, originates signaling messages *of its own*, by sending a single *Refresh* message for each of its *aggregate* flows. This ensures each core router counts the aggregate flow only once. However, these signaling messages have a rate of zero. This is because the rate of the aggregate flow is added to the state of the core router by the sum of the rates of the signaling messages of the individual flows.

## 9 Concluding Remarks

We presented an alternative approach to provide QoS guarantees without per-flow state at each router. We presented a signaling protocol that maintains a constant amount of state per router. Even though it has a constant amount of state, the signaling protocol is accurate, and it is also resilient to process and link failures. Our packet scheduling technique is a combination of the dynamic packet state technique of [19] and flow aggregation techniques of [5][7].

Our earlier work focused on rate-proportional delay, namely, where  $\delta_f = \frac{L_f}{R_f}$ . Under this assumption, it can be proven that aggregation is superior to non-aggregation.

In this work, we have focused on rate-independent delay, where the value of  $\delta_f$  is chosen by flow  $f$  and is independent of its reserved rate. This choice required modifications to the way packets are scheduled and to the signaling protocol. The additional increase of  $\frac{L_f}{R_f}$  in the delay of  $f$  when using aggregation is a significant factor to consider especially when the number of hops through the network is small, which prevents this delay from being amortized among many hops. Nonetheless, we have shown that even under these conditions flow aggregation outperforms no-aggregation.

## References

- [1] W. Almesberge, T. Ferrari, J. L. Boudec, “SRP: A Scalable Resource Reservation Protocol for the Internet”, *Proceedings of The International Workshop on Quality of Service (IWQOS)* 1998.
- [2] Braden, R., Clark, D., Shenker, S., “Integrated Services in The Internet Architecture”, Internet RFC 1633.
- [3] R. Callon, P. Doolan, N. Fieldman, A. Fredette, G. Swallow, A. Viswanathan. “A Framework for Multi-Protocol Label Switching”, Nov. 1998. Internet draft, draft-ietf-mpls-framework-02.txt.
- [4] Cobb J, “Scalable Quality of Service Across Multiple Domains”, technical report, Department of Computer Science, The University of Texas at Dallas, available from <http://www.utdallas.edu/~jcobb>.
- [5] Cobb J, “Preserving Quality of Service Guarantees In-Spite of Flow Aggregation”, *IEEE/ACM Transactions on Networking*, February 2002.
- [6] Cobb J., “Providing Quality of Service without Per-Flow State”, *Proceedings of the International Conference on Network Protocols*, 2001.

- [7] Cobb J., “An In-Depth Look at Flow Aggregation”, *Proceedings of the IEEE International Conference on Network Protocols*, 1999.
- [8] Cobb J., Gouda M., “Flow Theory”, *IEEE/ACM Transactions on Networking*, October 1997.
- [9] Cruz R. L., “SCED+: Efficient Management of Quality of Service Guarantees”, *Proceedings of the IEEE INFOCOM Conference*, 1998.
- [10] D. Ferrari, D. Verma, “A Scheme for Real-Time Channel Establishment in Wide-Area Networks”, *IEEE Journal on Selected Areas in Communications*, 8(3), April 1990.
- [11] Figueira N., Pasquale J., “Leave-in-Time: A New Service Discipline for Real-Time Communications in a Packet-Switching Data Network”, *Proceedings of the ACM SIGCOMM Conference*, 1995.
- [12] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, “Assured Forwarding PHB Group”, Internet RFC 2597.
- [13] J. Kaur, H. Vinh, “Core-Stateless Guaranteed Rate-Scheduling Algorithms”, *Proc. of the IEEE INFOCOM Conference*, 2001.
- [14] V. Jacobson, K. Nichols, K. Poduri, “An Expedited Forwarding PHB”, Internet RFC 2598.
- [15] Goyal P, Lam S., Vin H., “Det. End-to-End Delay Bounds in Heterogeneous Networks”, *Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1995.
- [16] K. Nichols, V. Jacobson, L. Zhang, “A Two-Bit Diff. Serv. Architecture for The Internet”, Internet RFC 2638.
- [17] Parekh A. K. J., Gallager R., “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case”, *IEEE/ACM Transactions on Networking*, 1(3):344-357, June 1993.
- [18] Shenker, S., Partridge, C., Guerin, R., “Specification of Guaranteed Quality of Service”, Internet RFC 2212.
- [19] Stoica I, Zhang H, “Providing Guaranteed Services without Per-Flow Management”, *Proceedings of the ACM SIGCOMM Conference*, 1999.
- [20] I. Stoica, H. Zhang, “LIRA: A Model for Service Differentiation in the Internet”, *Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* 1998.
- [21] Xie G., Lam S., “Delay Guarantee of Virtual Clock Server”, *IEEE/ACM Trans. on Networking*, Dec. 1995.
- [22] Wroclawski, J, “Specification of Controlled-load Network Element Service” Internet RFC 2211, 1997.

- [23] Z. L. Zhang, Z. Duan, L. Gao, Y. T. Hou, “Decoupling QoS Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services”, *Proceedings of the ACM SIGCOMM Conference*, 2000.
- [24] Zhang L., “Virtual Clock: A New Traffic Control Algorithm for Packet-Switched Networks”, *ACM Transactions on Computer Systems*, Vol. 9, No. 2, May 1991.
- [25] Zheng Q., Shin K.G., “On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks”, *IEEE Transactions on Communications*, Vol 42, No. 2/3/4, 1994.