Providing Quality of Service Guarantees without Per-Flow State

Jorge A. Cobb
Department of Computer Science (EC 31)
The University of Texas at Dallas
Richardson, TX 75083-0688
jcobb@utdallas.edu

Abstract

Two approaches have been proposed to improve the quality of service provided by the Internet: integrated services and differentiated services. Integrated services requires per-flow state at each router. On the other hand, differentiated services does not require per-flow state, but provides a lower level of quality of service. We present a protocol which provides quality of service similar to that provided by integrated services, but without maintaining per-flow state. To accomplish this, both the signaling protocol and the packet scheduling protocol must function without per-flow state. Our signaling protocol maintains only a constant amount of state at each router. In addition, it is accurate, and it is resilient to process and link failures. Our scheduling protocol is a combination of the dynamic packet state technique and flow aggregation techniques. This approach combines the strengths of both techniques and eliminates their weaknesses. Furthermore, our approach is applicable across multiple domains.

1. Introduction

Currently, the Internet provides only best effort service. A great deal of effort has been focused on improving the quality of service (QoS) provided by the Internet, in order to support future real-time applications such as voice and video. Two approaches have been proposed: integrated services [2][16][20] and differentiated services [10][11].

In integrated services, QoS is based on earlier packet scheduling protocols, such as virtual clock [19][22] and fair queuing [15]. To provide QoS to each flow, each router maintains per-flow state. This raises questions about the scalability of integrated services. Also, per-flow state is difficult to maintain in a distributed environment. This prompted the development of differentiated services.

In differentiated services, a few bits are reserved in each packet to indicate its per-hop-behavior. At each router, packets are classified and forwarded according to their per-hop behavior. Thus, no per-flow state is required. The disadvantage of differentiated services is that high levels of QoS and network utilization cannot be accomplished concurrently [17].

Recently, there have been attempts to provide the QoS level of integrated services, but without any per-flow state at the routers of a core network [17][21]. To accomplish this, both the signaling protocol and the packet scheduling protocol must function without per-flow state.

In [17][21], packet scheduling is based on the notion of dynamic packet state, where each packet carries enough information to reproduce its deadline at each router, without per-flow state. This technique has the disadvantage of not being able to compute the deadline accurately if a channel has variable delay. An example of such a channel could be a circuit through a packet switched network, such as ATM, with no jitter control provided to the circuit. In [4][5] the state is reduced at each router through flow aggregation. However, in practice, this can only be used inside the core of a network, and not across multiple domains.

Signaling methods with no per-flow state are in general divided into observation methods and bandwidth broker methods. Observation methods [1][17] 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 [14][21]. Centralized brokers are vulnerable to faults, and distributed brokers have the difficulty of maintaining their state synchronized.

We present an alternative approach to provide QoS guarantees without per-flow state at each router. We present 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 [17] and flow aggregation techniques of [4][5]. This approach combines the strengths of both approaches and eliminates the weaknesses of each. Furthermore, our approach is applicable across multiple domains.

The paper is organized as follows. Section 2 presents the QoS model supported by our protocols. The signaling protocol is presented in Section 3, and its fault-tolerant properties are discussed in Section 4. Dynamic packet state scheduling is reviewed in Section 5, and our approach combining dynamic packet state with flow aggregation is discussed in Section 6. Concluding remarks are given in Section 7.

2. Quality of Service Model

In this section, we define our network model, and also define the QoS that the model assigns to each flow of packets. We base our model on the models of [4] and [9]. The specific techniques that implement this QoS are presented in later sections.

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 addressed to the same destination computer. All the packets of a flow must traverse the network along a fixed path from the source to the destination. 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, it reserves enough network resources to ensure the flow's delay bound is not violated.

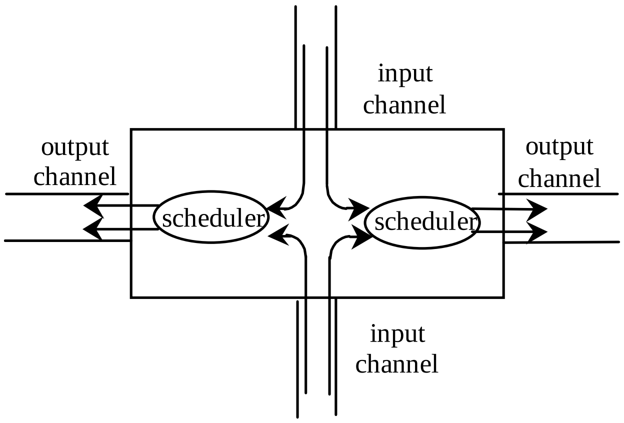

Figure 1. Output channels and their schedulers.

Each output channel of a computer is equipped with a scheduler, as shown in Figure 1. Each scheduler maintains a first-in-first-out packet queue for each flow it receives from the input channels. A scheduler receives packets from flows whose path traverses the output channel of the scheduler. Whenever its output channel becomes idle, the scheduler chooses a received packet and forwards it to the output channel.

We say a packet *exits* a scheduler when the last bit of the packet is transmitted by the output channel of the scheduler. 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^{th} packet of f, $i \geq 1$.
$L_{f,i}$	length of packet $p_{f,i}$.
$L_{f,i}^{max}$	maximum of $L_{f,j}$, where $1 \leq j \leq i$.
L_s^{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.
π_s	upper delay bound of the output channel of s.

Consider a scheduler s and a flow f. We define the *start-time* $S_{s,f,i}$ and *finish-time* $F_{s,f,i}$ of packet $p_{f,i}$ at scheduler s as follows [4][9]. 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, and $F_{s,f,i}$ is the time at which the last bit of $p_{f,i}$ is forwarded by s. More formally, let f be an input flow of scheduler s. Then,

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

Each scheduler s will forward the packets of each input flow f at a rate of at least R_f. Therefore, for each packet $p_{f,i}$, its exit time from the scheduler is close to its start time, $S_{s,f,i}$. We refer to these schedulers as rate-guaranteed schedulers [4][9][12]. More formally, a scheduler s is a rate-guaranteed scheduler if and only if, 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. Throughout the paper, we assume all schedulers are rate-guaranteed schedulers.

We next consider the delay of a packet across a sequence of schedulers. 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. Let $t1, t2, \ldots, tk$ be a sequence of k rate-guaranteed schedulers traversed by flow f. For all i,

$$S_{tk,f,i} \leq S_{t1,f,i} + \sum_{x=1}^{k-1} \Delta_{tx,f,i} + \sum_{x=1}^{k-1} \pi_{tx} \qquad (1)$$

where

$$\Delta_{s,f,i} = \max_{1\leq x\leq i}\{\delta_{s,f,x}\}$$

This bound was shown in [4][6][9], and it also follows from the results in [23].

The above definition of delay is broad enough to encompass the delay provided by many scheduling protocols. 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 [15][19][22], since the delay is related to the rate of the flow. Another example is the real-time channel protocol, [8][23] where each flow has constant packet size and constant packet delay. This is represented above by having $\delta_{s,f,i}$ and $L_{f,i}$ be constant for all i. In this protocol, the packet delay can be any value. Thus, it is known as *rate-independent delay*, since it unrelated to the rate of the flow.

To ensure packets exit by their deadline, a scheduling test must be satisfied. For rate-dependent delay, the scheduling test is simply the following.

$$\sum_f R_f \leq C_s \tag{2}$$

That is, the channel is not over allocated. For rate-independent delay [23], the scheduling test is more involved. In addition to (2), the following test is required. For all $t, t > 0$,

$$\left(\sum f,\ \delta_{s,f} \leq t,\ \left(\left\lfloor\frac{(t-\delta_{s,f})\cdot R_f}{L_f^{\max}}\right\rfloor + 1\right)\cdot L_f^{\max}\right) \leq\ t\cdot C_s \tag{3}$$

where $\delta_{s,f}$ is the delay of flow f at scheduler s. Test (3) is infeasible since it requires an infinite number of tests, one for every t. An equivalent version with only a finite number of tests is given in [23]. However, the number of tests may still be too large. Thus, the following test, which may be implemented in linear time in the number of flows, is also given in [23]. For every f,

$$\left(\sum g,\ \delta_{s,g} \leq \delta_{\mathrm{s,f}},\ L_g^{\max} + (\delta_{\mathrm{s,f}} - \delta_{\mathrm{s,g}})\cdot R_g\right) \leq\ \delta_{\mathrm{s,f}}\cdot C_s \tag{4}$$

3. Shadow Reservation State

We next present our signaling protocol to guarantee QoS to each flow, without maintaining per-flow state at each scheduler. We begin by examining each of the three scheduling tests described earlier, and determine how much information is needed to perform each test.

Consider first test (2). A scheduler must maintain only the total of the reserved rates of its flows, and the rate of its output channel. Consider next test (3). Note that the contribution of two flows, f and f' to the left-hand-side of (3) is the same, provided $\delta_{s,f} = \delta_{s,f'}$ and $R_f = R_{f'}$. Thus, each scheduler may define a set of (rate, delay) pairs which are most frequently used by applications, and allow each flow to only reserve resources equal to one of these (rate, delay) pairs. Also, the maximum packet size may be set to a common value for all flows. Although this reduces the granularity of resource reservation, it significantly reduces the amount of state required by the scheduler. In particular, the scheduler must only maintain a count of input flows in each (rate, delay) pair. Finally, case (4) is similar to case (3), except that instead of a set of (rate, delay) pairs, only a set of delays is necessary. The scheduler must only maintain, for each delay value, a count of how many flows have this delay, and the sum of their rates. We thus conclude that per-flow state is not necessary to perform scheduling tests (2) through (4).

The objective of the signaling protocol is to maintain the above information current at each node, 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 flow periodically sends Refresh messages along the path to its destination. These messages contain the reservation information of the flow. We assume that the network will forward Refresh messages with higher reliability than regular messages. This can be accomplished in many ways, such as reliably transferring Refresh messages across each hop, or reserving buffer space for Refresh messages. The reliable transfer of Refresh messages will not be a performance drain for the network, since they are sent infrequently.

For simplicity, we assume the scheduler uses rate-dependent delay and scheduling test (2). The extension to tests (3) and (4) is straightforward and discussed below.

Each scheduler s maintains a rate variable, $SumRates_s$, where it stores the sum of the reserved rates of its flows. Thus, test (2) is replaced by $SumRates_s \leq C_s$. In addition, s maintains a rate variable, $ShadowSumRates_s$, and a bit variable, $SwapBits_s$. Every T seconds, where T is a predefined constant, s updates its state in the following way:

```
$SumRates_s$ := $ShadowSumRates_s$;
$ShadowSumRates_s$ := 0;
$SwapBits_s$ := $\neg SwapBits_s$;
```

Each Refresh message from flow f contains R_f. The objective is to add R_f to $ShadowSumRates_s$ exactly once before $ShadowSumRates_s$ is assigned to $SumRates_s$. In this way, R_f is always included in $SumRates_s$.

The purpose of $SwapBits_s$ is to ensure R_f is added to

$ShadowSumRates_s$ only once before each state update. The Refresh message from flow f contains a bit, $b_{f,s}$, for each scheduler s in its path. Bit $b_{f,s}$ indicates the last value of $SwapBits_s$ encountered by a Refresh message from f. Rate R_f is added to $ShadowSumRates_s$ 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.

```
if $b_{f,s} \neq SwapBits_s$ then
    $ShadowSumRates_s$ :=
            $ShadowSumRates_s$ + $R_f$;
    $b_{f,s}$ := $SwapBits_s$
end if
forward Reserve 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 $b_{f,s}$ 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 $b_{f,s}$ the value of $SwapBits_s$ as it traverses scheduler s. Rate R_f is added to both $ShadowSumRates_s$ and $SumRates_s$, since R_f is not included in $ShadowSumRates_s$. Thus, upon receiving a Reserve message from f at s, the following is performed.

```
if $SumRates_s + R_f \leq C_s$ then
    $ShadowSumRates_s$ :=
            $ShadowSumRates_s + R_f$;
    $SumRates_s$ := $SumRates_s + R_f$;
    $b_{f,s}$ := $SwapBits_s$;
    forward Reserve towards the
            destination of $f$
else
    return a Reject message towards the
            source of $f$
end if
```

The destination returns a ReserveAck to the source of f including all bits $b_{f,s}$ 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.

```
if $SwapBits_s$ = $b_{f,s}$ then
    $ShadowSumRates_s$ :=
            $ShadowSumRates_s$ - $R_f$;
    $SumRates_s$ := $SumRates_s$ - $R_f$;
else
    $SumRates_s$ := $SumRates_s$ - $R_f$;
endif
forward Reject towards the source of $f$
```

As an example, consider Figure 2. In Figure 2(a), an existing flow f traverses scheduler s, and its rate is included in $SumRates_s$ and $ShadowSumRates_s$. In Figure 2(b), the Reserve message of a new flow g is received. Since g is new, R_g is added to both $SumRates_s$ and $ShadowSumRates_s$, and the source of g learns the value of $SwapBits_s$. In Figure 2(c), a state update occurs: $ShadowSumRates_s$ is assigned to $SumRates_s$, $ShadowSumRates_s$ is set to zero, and $SwapBits_s$ is flipped. In Figure 2(d), f sends a Refresh message, and since $SwapBits_s$ has changed, R_f is added to $ShadowSumRates_s$. In Figure 2(e), g sends a Refresh message, and likewise, R_g is added to $ShadowSumRates_s$. Finally, in Figure 2(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.

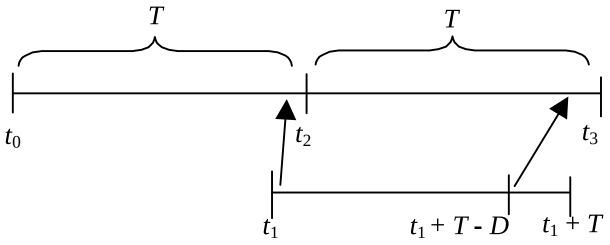

Figure 3. Timing of Refresh **messages**

Consider Figure 3. 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, which at the latest will occur at time $t_1 + 2 \cdot D$. Thus, we require $3 \cdot D < T$, and the interval between the successive transmissions of Refresh messages should be at most $T - D$.

We have assumed thus far that schedulers use rate-dependent delay and scheduling test (2). For rate-independent delay and scheduling test (3), we have two

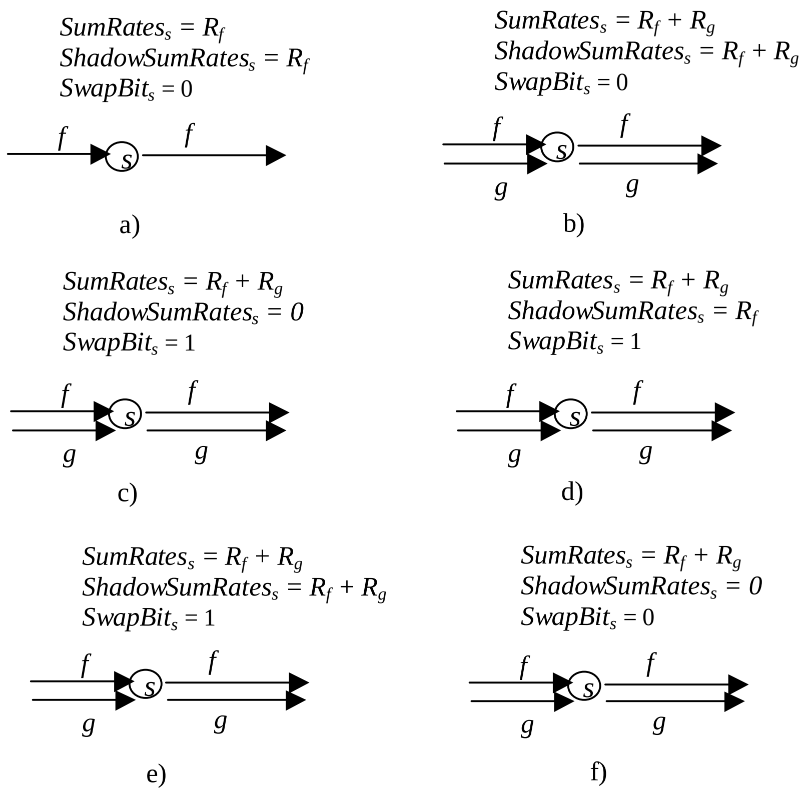

Figure 2. Signaling example

cases. If the rate-delay class of the flow is the same throughout its path, then the rate-delay class is included in the Refresh message. However, if the rate-delay class changes along its path, the Refresh message must contain a list of rate-delay classes, one for every hop in its path. The case of scheduling test (4) is similar.

4. 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. Thus, delayed messages are equivalent to lost messages. If a source does not receive a RefreshAck, then the source terminates the flow. This should occur rarely, since signaling messages are treated with higher reliability than regular messages. Therefore, in general, this is not a problem for flow sources. However, the network resources of the flow must be deallocated. This happens automatically as follows. The terminated flow ceases to send Refresh messages. At any scheduler s, within T seconds, $ShadowSumRates_s$ is assigned zero, and in T more seconds, $ShadowSumRates_s$ is assigned to $SumRates_s$. Thus, within $2 \cdot T$ seconds of the failure, the reserved rate of f disappears from s. For example, let T = 30 sec.. Thus, $D < 10$ sec., and the interval between Refresh messages is less than 20 sec.. Furthermore, within a minute, the resources of a failed flow are released.

Cases 2) and 3) are similar to 1). If a link or process along a flow fails, then the flow's source does not receive RefreshAck's, and case 1) applies. 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 [18], 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 computer c maintains a bit, $Route_{c,d}$, which is flipped every time the routing table entry to destination d changes. The source 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 computer along the path of f. If 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.

5 Dynamic Packet Scheduling

Thus far, we only addressed the signaling protocol to establish and maintain a flow. We next address how packets are scheduled without per-flow state. This can be done using a technique presented in [7][17], known as dynamic packet state. In [17], the technique was applied only for rate-dependent delay, since their signaling protocol did not allow flows to be counted. However, the technique is also applicable to rate-independent delay, as shown next.

Consider two consecutive schedulers, s and t, of flow f. From Equation (1) and the definition of S, $A_{t,f,i} \leq S_{t,f,i} \leq S_{s,f,i} + \Delta_{s,f,i} + \pi_s$. Assume $A_{t,f,i} = S_{s,f,i} + \Delta_{s,f,i} + \pi_s$ for all $p_{f,i}$. Then, the following holds.

$$S_{t,f,i} = A_{t,f,i} = S_{s,f,i} + \Delta_{s,f,i} + \pi_s \qquad (5)$$

Note that $\Delta_{s,f,i}$ is independent of other flows, and thus, $p_{f,i}$ could carry information to compute $\Delta_{s,f,i}$. E.g., in rate-dependent delay, $\Delta_{s,f,i} = L^{max}_{f,i}/R_f + L^{max}_s/C_s$. Thus, $p_{f,i}$ could include $L^{max}_{f,i}/R_f$, since L^{max}_s/C_s is known to s. In rate-independent delay, $\Delta_{s,f,i}$ is the constant delay of f at s, namely $\delta_{s,f}$. Thus, $p_{f,i}$ could include $\delta_{s,f}$ for every scheduler s along its path.

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 $S_{t,f,i} + \Delta_{t,f,i}$ of $p_{f,i}$ at t is computed without per-flow state.

In Equations (1) and (5), the start-time of each packet is measured with respect to a single reference clock. Equation (5) is valid if all schedulers have a common clock. For example, all schedulers could take part of the Network Time Protocol [13], and be part of the top levels in the clock hierarchy. This is unlikely in a large network, and each scheduler is likely to have its own independent clock.

To solve this, an alternative is proposed in [17]. Scheduler s computes the early departure of $p_{f,i}$, denoted $\varepsilon_{s,f,i}$, as follows.

$$\varepsilon_{s,f,i} = S_{s,f,i} + \Delta_{s,f,i} - E_{s,f,i}$$

Then, s includes $\varepsilon_{s,f,i}$ in $p_{f,i}$. At scheduler t, when $p_{f,i}$ is received, it is delayed $\varepsilon_{s,f,i}$ seconds in a buffer before being considered as "arrived". Immediately after these $\varepsilon_{s,f,i}$ seconds, t simply assigns the current value of its clock to $S_{t,f,i}$ and $A_{t,f,i}$. Again, $S_{t,f,i}$ is computed at t without per-flow information.

The above solution has a couple of disadvantages. First, if the output channel has variable delay, then $S_{t,f,i}$ is not computed accurately. There are several examples of channels with variable delay in large internetworks. E.g., token ring networks guarantee an upper bound on delay, but may transmit early if the token is received immediately. Another example is an ATM virtual circuit that provides bounded delay but no jitter control. Second, assume some schedulers have clocks which run fast, and forward packets to a scheduler with a normal clock. This will cause excessive delays to other flows of the normal scheduler. Note that this would not occur in a network with per-flow state, since the normal scheduler would assign to each packet a deadline which is locally computed, independent of other schedulers.

In the next section, we propose a solution which uses synchronized clocks to avoid the disadvantages mentioned above. However, by taking advantage of the hierarchical structure of internetworks, only a few computers need to be involved in clock synchronization.

6. Region Aggregation

Our signaling and packet scheduling protocols require messages to carry items for each hop along their path. For each hop, signaling messages must carry a swap bit, a routing bit, and the delay class (if it varies per hop). Data packets must carry the routing bit, and the delay class (if it varies per hop). Also, to deal with variable delay channels, schedulers must synchronize clocks. To reduce this overhead, we take advantage of the hierarchical structure of internetworks. We assume the internetwork is divided into regions, and that gateway routers interconnect regions, as shown in Figure 4. In the Internet, a region would correspond to an autonomous system.

Using packet switching techniques, such as [3], each gateway in a region establishes a circuit to other gateways in its region. Thus, with G neighboring regions, a maximum of G^2 circuits cross an interior router. If rate-independent delay is used, let M be the number of delay classes. Each gateway establishes M circuits to each other gateway, and thus at most $M \cdot G^2$ circuits cross an interior router. Note that G and M are likely to be small, so only a small number of circuits may cross an interior router.

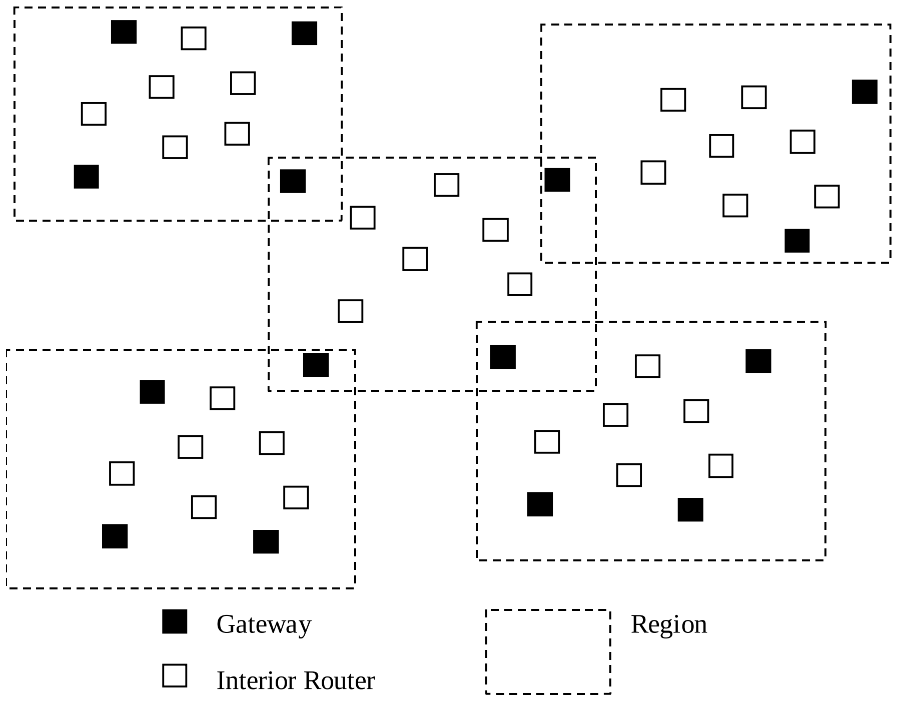

Figure 4. Regions in an inter-network

Using these circuits, from the perspective of our network model of the previous sections, the gateways are nodes in the network, and the circuits between gateways are output channels with variable delay. We assume gateways have synchronized clocks using the NTP protocol [13]. Each packet $p_{f,i}$ arriving at a gateway s contains the value of $S_{s,f,i}$ computed at the previous gateway. Gateway s will hold $p_{f,i}$ in a buffer until time $S_{s,f,i}$. After time $S_{s,f,i}$, $p_{f,i}$ is considered arrived at s. Before forwarding $p_{f,i}$, gateway s computes the start time of $S_{t,f,i}$ at the next gateway t, and includes it in $p_{f,i}$.

The packets of all the flows sharing the same circuit are aggregated together to become a single flow g. The aggregation should be done in a fair manner, as described in [4][5], to ensure each individual flow has a fair share of the aggregate flow. This aggregation may be done using the start time $S_{s,f,i}$ included in the packet. At a gateway s, a packet $p_{f,i}$ of an individual flow f encounters a delay of $L_{f,i}/R_f + L_g^{max}/R_g + L_s^{max}/C_s$ seconds [5], where R_g is the sum of the rates of the individual flows comprised by g. After s, the circuit is treated as a single flow g, and the per-hop delay of its packets depends on the setup of the circuit. Packets of the circuit are transmitted using traditional state-full techniques.

We have shown in [4][5] that by aggregating flows together, a lower per-hop delay is possible for the aggregate flow than for the individual flows. Thus, by reducing the amount of state at the schedulers we increase the ability to satisfy the delay requirements of flows, at the expense of only adding L_g^{max}/R_g of delay at the entrance to the region.

Since each circuit is handled using traditional state-full protocols, only the gateway is involved in handling Refresh messages from individual flows, and maintaining shadow state, e.g., $SumRates_s$ and $ShadowSumRates_s$, for each of the $G \cdot M$ circuits it originates. Therefore, each signaling message carries only one swap bit per region. Also, for each region, each packet and signaling message carries the label indicating the circuit used to cross the region. In the event that at all times each gateway has only a single circuit to each destination, then only a single routing bit per region is necessary, as described in Section 4. When a gateway receives a Reserve message for a new flow, its state ($SumRates_s$ and $ShadowSumRates_s$) is updated as usual, and the resources for the circuit are upgraded using a traditional state-full approach.

7. 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 [17] and flow aggregation techniques of [4][5]. This combines the strengths of both approaches and eliminates the weaknesses of each.

If rate-dependent delay is chosen, then the scheduling test of the proposed approach will be the same as that of the state-full approach. Thus, both approaches support the same set of flows. However, for the rate-independent delay, the granularity of the rate and delay allocation must be reduced. Otherwise, the each flow could request a different (rate, delay) pair, which would require per-flow state. We plan to investigate through simulation the tradeoffs between increasing the granularity of resource reservation in our protocol versus the call blocking probability.

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, "Preserving Quality of Service Guarantees In-Spite of Flow Aggregation", *Proceedings of the IEEE International Conference on Network Protocols*, 1998.

[5] Cobb J., "An In-Depth Look at Flow Aggregation", *Proceedings of the IEEE International Conference on Network Protocols*, 1999.

[6] Cobb J., Gouda M., "Flow Theory", *IEEE/ACM Transactions on Networking,* October 1997.

[7] Cruz R. L., "SCED+: Efficient Management of Quality of Service Guarantees", *Proceedings of the IEEE INFOCOM Conference*, 1998.

[8] 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.

[9] 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.

[10] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, "Assured Forwarding PHB Group", Internet RFC 2597.

[11] V. Jacobson, K. Nichols, K. Poduri, "An Expedited Forwarding PHB", Internet RFC 2598.

[12] 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.

[13] Mills, D.L. "Improved algorithms for synch. computer network clocks", *IEEE/ACM Transactions on Networking* 3 (3) (June 1995).

[14] K. Nichols, V. Jacobson, L. Zhang, "A Two-Bit Diff. Serv. Architecture for The Internet", Internet RFC 2638.

[15] 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.

[16] Shenker, S., Partridge, C., Guerin, R, "Specification of Guaranteed Quality of Service", Internet RFC 2212.

[17] Stoica I, Zhang H, "Providing Guaranteed Services without Per-Flow Management", *Proceedings of the ACM SIGCOMM Conference*, 1999.

[18] 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.

[19] Xie G., Lam S., "Delay Guarantee of Virtual Clock Server", *IEEE/ACM Trans. on Networking*, Dec. 1995.

[20] Wroclawski, J, "Specification of Controlled-load Network Element Service" Internet RFC 2211, 1997.

[21] 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.

[22] Zhang L., "Virtual Clock: A New Traffic Control Algorithm for Packet-Switched Networks", *ACM Transactions on Computer Systems,* Vol. 9, No. 2, May 1991.

[23] 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.