Delay Differentiation and Adaptation in Core Stateless networks

Thyagarajan Nandagopal Narayanan Venkitaraman Raghupathy Sivakumar Vaduvur Bharghavan
Coordinated Sciences Laboratory
University of Illinois at Urbana-Champaign
Email:{thyagu, murali, sivakumr, bharghav}@timely.crhc.uiuc.edu

Abstract—**We present a core-stateless quality of service architecture for achieving delay differentiation between flows. There are two key components in our approach:**
1. *Per-class per-hop relative average delay:* **the average queueing delay perceived by the packets in a delay class at a link is inversely proportional to the delay weight of the class.**
2. *Per-flow end-to-end delay class adaptation:* **the delay class of a flow is dynamically adjusted based on its perceived end-to-end delay in order to maintain the desired end-to-end average delay requirement of the flow.**
We show through simulations and analysis that these two components can, in concert, support end-to-end delay differentiation between flows using only simple mechanisms at the core routers in the network.

I. Introduction

The current Internet provides a single service class – best effort – that requires no pre-specified quality of service (QoS) contracts and provides no minimum QoS measures for packet flows. In order to enhance this basic service, two broad paradigms for quality of service in the future Internet have emerged: Integrated services (Intserv), and Differentiated services (Diffserv). The Intserv approach [1], [2] provides end-to-end guaranteed or predictive service on a per-flow basis for rate and delay, but requires each router of the network to maintain per-flow state in order to process per-flow signalling messages on the control plane and to perform per-flow classification, scheduling and buffer management on the data plane. Since it may be infeasible for core routers to perform all of the above actions efficiently when there are millions of flows traversing through the network simultaneously, the Diffserv [3], [4] approach proposes a coarser notion of quality of service, focusing primarily on aggregates of flows in the core routers, and intending to differentiate between service *classes* rather than provide absolute per-flow QoS measures. In particular, while the access routers process packets on the basis of finer traffic granularity such as per-flow or per-organization, core routers do not maintain fine grain state, and process traffic based on a small number of Per Hop Behaviors (PHB) encoded in the packet header [5]. Diffserv is gaining popularity as the quality of service paradigm of the future Internet, primarily because it moves the complexity of providing quality of service out of the core and into the edges of the network, where it may be feasible to maintain a restricted amount of per-flow state.

Different approaches exist to realize the Diffserv philosophy. At one end of the spectrum, *absolute differentiated services* seek to provide Intserv-type end-to-end absolute performance measures without per-flow state in the network core [6], [7]. At the other end of the spectrum, *relative differentiated services* seek to provide *per-hop, per-class relative services*, wherein each router has N service classes, and *Class i is better (or at-least no worse) than Class (i-1) for* $2 \leq i \leq N$, in terms of service metrics for that hop [8]. In the latter approach, there are no absolute guarantees due to the lack of admission control or resource reservations. Consequently, the network cannot provide worst case bounds for a service metric. Instead, each router only guarantees that the service invariant is locally maintained, even though the absolute service might vary with network conditions.

We believe that most existing and emerging real-time applications require *minimum end-to-end absolute rate contracts* in either guaranteed or predictive modes for effective operation [9], [10]. On the other hand, since most of these applications are soft real-time in nature, they are tolerant to occasional delay violations and hence do not require worst case delay bounds. Consequently, we perceive the need for a quality of service architecture that provides end-to-end per-flow absolute rate contracts but we believe that per-hop relative delay classes may be sufficient to achieve effective delay service differentiation so long as they are augmented with end-to-end delay class adaptation.

In order to achieve this service model, we have proposed the *Corelite* QoS architecture [7] which supports the following services: (a) per-flow end-to-end absolute rate contracts, (b) per-flow end-to-end weighted rate fairness, (c) per-hop relative delay classes and (d) per-flow end-to-end delay class adaptation, without maintaining per-flow state in the core of the network. In this paper, we focus only on the delay service model of Corelite. The following are the two key aspects of our approach:

• *A simple packet forwarding mechanism at a router that achieves* average packet delay *for a delay class inversely proportional to the* delay weight *of the class.*

• *An end-to-end delay adaptation mechanism that dynamically adjusts the delay class of a flow in order to match the end-to-end delay requirements of the flow.*

Our goal is to show that it is possible to meet the average delay requirement of all flows (so long as there exists a feasible solution), using a very simple forwarding mechanism at the core router, in tandem with delay adaptation at the access routers of the connection endpoints.

The rest of the paper is organized as follows. In Section II, we briefly describe the high-level QoS architecture. In Section III, we present the router packet forwarding mechanism. In Section IV, we describe the delay class adaptation algorithm. In Section V, we evaluate our mechanisms through simulation. In Section VI, we discuss some of the issues with our approach. In Section VII, we compare our approach to related work, and we summarize our work in Section VIII.

II. High-Level Architecture

For simplicity, we consider a single-cloud network with core routers at the center and access routers at the fringes. The core routers do not maintain any per-flow state, while the access routers maintain some per-flow state. The QoS requirements of a flow are provided to the access router, which provides the end-to-end QoS management functions including delay class adaptation. With this simple network model in place, we now provide a quick overview of the mechanisms for providing delay and rate QoS in Corelite. Figure 1 provides a high level overview of the delay component of the architecture.

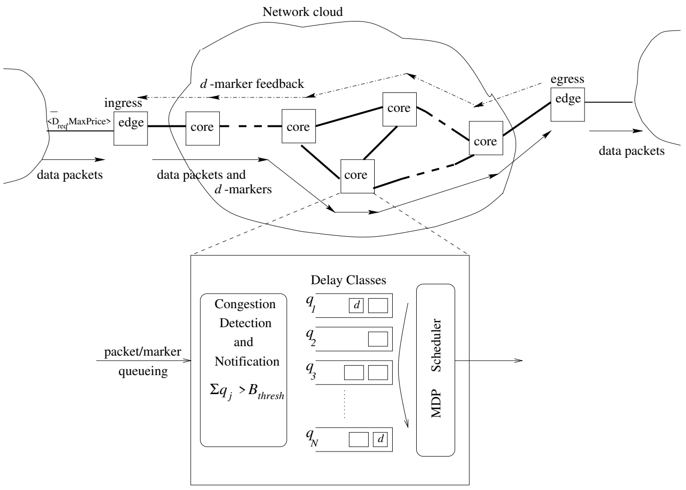


Fig. 1. Overview of the architecture

A. Delay Differentiation at a Router

Each delay class has an associated *delay weight*, and all routers support the same set of delay classes. The goal of the forwarding behavior at a router[1] is to ensure that *the average delays perceived by packets in any two delay classes are in the inverse ratios of the corresponding delay weights*, i.e. $|\bar{d}_1(t)\Delta_1 - \bar{d}_2(t)\Delta_2| \rightarrow 0$, where $\bar{d}_i(t)$ is the average delay of packets served by a delay class i with delay weight Δ_i over a time window of $[0, t]$. Since it is impossible to achieve this goal over infinitesimal time periods in a packetized service regime, the averages are computed over some minimum time window t_{min} (the choice of t_{min} turns out to be non-trivial for reasons that are briefly hinted at, but not fully explored, in this paper due to space constraints). When many flows share the same delay class, the expected average delay of a flow is equal to the average delay of the delay class to which it belongs. Section 3 describes the forwarding mechanism in detail.

B. End-to-end Delay Class Adaptation

While delay differentiation at a router does not say much about the absolute delay experienced by packets of a flow, this can be augmented with end-to-end delay class adaptation. The goal of delay class adaptation is to allocate to a flow the lowest delay class that can satisfy its delay requirement.

Since per-hop per-class relative delay differentiation translates to consistent end-to-end flow-based relative differentiation [8], the access router periodically sends special packets to find out the end-to-end delay experienced by a flow. As there is no bound on the absolute value of end-to-end expected average delay of a flow in a particular class, the access router tries to jump to a higher delay class whenever it observes a violation of the delay requirement, since a higher delay class implies a lower delay. On the other hand, if the access router sees that the delay observed is less than the required delay, and if the next lower class can sustain the delay requirement of this flow, then it jumps to the lower class.

End-to-end delay class adaptation assumes that a solution (i.e. one that leads to all flows meeting their delay requirements) is feasible. If such a solution is not possible, then all flows will end up at the highest delay class. In order to alleviate this problem, we require each flow to specify a *price* that it is willing to pay for the *end-to-end delay requirement*, which then gets mapped to a *maximum delay class* (we do not investigate the mapping policies in this paper). Once the mapping has taken place, the maximum delay class upper bounds the delay class to which the flow can adapt. Section 4 describes the delay adaptation mechanism in detail.

The advantages of our approach are that the packet forwarding behavior is quite simple, core routers do not maintain per-flow state, and that applications can renegotiate their delay requirements without involving end-to-end signalling and resource allocation/deallocation. The disadvantages of our approach are that there is no guarantee on average per-flow delay, and there is no guarantee of a feasible delay class allocation.

C. Rate Mechanisms in Corelite

It turns out that one of the key requirements for the delay differentiation mechanism to work is that the aggregate packet queues over all the delay classes need to be bounded. In Corelite, the rate allocation mechanism provides this functionality. Since a discussion of our rate allocation mechanisms is beyond the scope of this paper [7], we only present a brief overview of the relevant portion of the mechanisms in this section.

The key idea is to decouple the rate and delay mechanisms. This means that upon detection of incipient congestion, a router must send fair feedback based on the normalized transmission rate of the flows in the recent past, rather than on which packets are currently in the packet queues (otherwise flows belonging to lower delay classes will consistently get lower rates). To this end, the access router periodically generates special "marker packets" for a flow, such that the *rate of the marker packets reflects the normalized rate of the packet flow*. Markers are transparent to the end hosts since they are injected by ingress access routers and not forwarded by egress access routers. When a core router sees a marker, in addition to forwarding the marker, it also *caches* the marker in a cache. Upon detection incipient congestion (when the aggregate queue sizes over all the delay classes exceed a threshold), a core router randomly selects markers from its cache and sends these markers back to the access router that generated them. This causes the access routers to throttle the corresponding flows for which the markers were generated. While the details and the properties of the approach are presented in [7], it is important to note that (a) the aggregate queue size does not exceed the threshold significantly (because markers are generated progressively more aggressively as the queue size exceeds the threshold), and (b) flows are throttled in the ratio of their normalized rates, and not depending on which packets are actually in the packet queues (or which packets arrive) after incipient congestion is detected. It turns out that these are critical requirements for ensuring rate-delay decoupling as well as providing relative delay differentiation at a router.

III. Forwarding Behavior at a Router

Each router has a pre-specified number of delay classes (N) and each delay class i is assigned a delay weight Δ_i, where

[1] Throughout this paper, we say "forwarding at a router" when we mean "forwarding at the output link of a router".

$\Delta_1 < \Delta_2 < ... < \Delta_N$. Queues of different delay classes are served such that the average delay experienced by packets in a delay class is inversely proportional to the delay weight of the class. In other words, if $\bar{d}_i(t)$ is the average delay for class i over a time window of $[0, t]$, then the goal is to achieve $|\bar{d}_i(t)\Delta_i - \bar{d}_j(t)\Delta_j| \to 0$.

A simple heuristic to achieve this is to serve the delay class with the maximum value of $\bar{d}_i(t)\Delta_i$ at any time t. This heuristic points towards asymptotic convergence such that $|\bar{d}_i(t)\Delta_i - \bar{d}_j(t)\Delta_j| \le \epsilon$, for $t > t_{min}$, where t_{min} is a minimum time window that is a function of ϵ and the Δ_is, assuming that there are no further arrivals in the time window under consideration.

A. Mean-Delay Proportional (MDP) Scheduler

Each delay class is served by a single first-in-first-out (FIFO) packet queue[2]. Packets of a flow belonging to a delay class i are queued in the corresponding queue along each router that the flow passes through. All flows with the same delay class specification share the same FIFO queue at the router. For each queue with delay weight Δ_i, the router maintains three variables: $d_i^*(t)$ – the aggregate delay experienced by all packets that have either been served or are currently in the queue at time t, $s_i(t)$ – the number of packets served from delay class i till time t, and $q_i(t)$ – the number of packets queued in i at time t. Considering the simple case where time is divided into slots and the link capacity is 1 packet/slot, the variables are updated in each time slot as follows:
(a) $q_i(t+1) \leftarrow q_i(t) + a_i(t) - I_i(t)$, where $a_i(t)$ is the number of arrivals at time t, and $I_i(t) = 1$ if the queue i is selected for transmission at time t, or 0 otherwise,
(b) $s_i(t+1) \leftarrow s_i(t) + I_i(t)$, and
(c) $d_i^*(t+1) \leftarrow d_i^*(t) + q_i(t)$.

Hence, if at time t we assume that no other packet will arrive at a queue i in the future, then the *minimum average delay* $\bar{d}_i(t)$ *for all the packets that have already arrived* can be achieved if all the packets in the queue i are transmitted back-to-back starting at time t. It can be calculated as

$$\bar{d}_i(t) \leftarrow \frac{(d_i^*(t) + \frac{1}{2}q_i(t)(1 + q_i(t)))}{(s_i(t) + q_i(t))}$$

where $\frac{1}{2}q_i(t)(1 + q_i(t))$ is the lower bound on the cumulative delay experienced by all packets in the queue, served from time t.

When a router needs to select a packet for transmission at time t, it selects a queue j for transmission such that

$$j \leftarrow arg\ max_i\{\bar{d}_i(t) \times \Delta_i\}$$

. In other words, *the router selects the head-of-line packet of the queue for which the minimum possible normalized average delay at time t, based on the arrivals till the current time, is maximum among all backlogged queues*. Since the router always serves the backlogged queue for which the normalized average delay is maximum, asymptotically all queues will observe the same normalized average delay, i.e. the average delay perceived by a delay class is inversely proportional to its delay weight.

$$\frac{\bar{d}_i(t)}{\bar{d}_j(t)} = \frac{\Delta_j}{\Delta_i}, t \to \infty$$

[2]In the rest of the paper, we use the terms class and queue interchangeably.

Example 1: Consider a simple example with two delay classes C1 and C2 that have delay weights of 1 and 2 respectively. The top two rows of Figure 2 show the arrival patterns for each of these queues. The progression of the values of $\bar{d}_i(t)$ and other state variables with time is shown in the subsequent rows. The order of packet transmission is shown in the last row of the figure. Consider, for instance, the system at time t = 4. The normalized average delay $\bar{d}\Delta$ of class 1 is higher than that of class 2. Hence class 1 is selected for transmission. Thus, the average delay of class 2 increases, but its normalized average is still less than that of class 1, which is served again at time t = 5. Now, $\bar{d}\Delta$ of class 2 increases to exceed that of class 1. Thus, at time t = 6, class 2 is served. As can be seen, even though the rates obtained by both classes are same, the packets are served in such a way that the ratio of the average delays experienced by C1 and C2 is maintained at around 2 (the ratio of the inverse of their delay weights).

Time (in slots) →	1		2		3		4		5		6		7		8		9		10	
C1 $\Delta = 1$	↓↓		↓		↓↓															
C2 $\Delta = 2$	↓↓		↓		↓↓															
Variables	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1	C2
s	0	0	0	1	0	2	0	3	1	3	2	3	2	4	2	5	3	5	4	5
q	2	2	3	2	5	3	5	2	4	2	3	2	3	1	3	0	2	0	1	0
d*	0	0	2	2	5	4	10	7	15	9	19	11	22	13	25	14	28	14	30	14
$\bar{d}$	1.5	1.5	2.7	1.7	4.0	2.0	5.0	2.0	5.0	2.4	5.0	2.8	5.6	2.8	6.2	2.8	6.2	2.8	6.2	2.8
$\bar{d}\Delta$	1.5	3.0	2.7	3.4	4.0	4.0	5.0	4.0	5.0	4.8	5.0	5.6	5.6	5.6	6.2	5.6	6.2	5.6	6.2	5.6
Class served	C2		C2		C2		C1		C1		C2		C2		C1		C1		C1	

Fig. 2. Example of Delay Classes

B. Properties

B.1 Upper Bound on Queue Size

In Corelite, flows increase their rate by $\alpha = 1$ packet/second in the absence of congestion feedback, and decrease their rates by $\beta = 1$ packet/second for every congestion marker received. At the core router, when the aggregate queue size q exceeds a certain threshold q_{th}, the core router sends back

$$M = C[\frac{q}{q+1} - \frac{q_{th}}{q_{th}+1}] + \kappa(q - q_{th})^3$$

markers, where C is the link capacity in packets/second and κ is a constant [7].

Consider the situation when N flows pass through the core router. When the aggregate queue length q exceeds the threshold q_{th}, the rate of change of aggregate queue size is given by

$$\frac{dq}{dt} = R + N - 2M - C$$

where $R = C\frac{q}{q+1}$ is the aggregate rate of N flows (under Poisson assumptions).

$$\frac{dq}{dt} = N + C(\frac{q_{th}-1}{q_{th}+1}) - C\eta - 2\kappa(q - q_{th})^3$$

where $\eta = \frac{q}{q+1}$ and $\frac{q_{th}}{q_{th}+1} \le \eta < 1$. Thus,
the maximum queue size is given by

$$q_{max} = q_{th} + (\frac{N - \frac{C}{q_{th}+1}}{2\kappa})^{1/3}.$$

This puts a bound on the aggregate queue size over all delay classes in our delay calculations, and we will assume a maximum bound of B in the following discussions.

B.2 Average Queueing Delay

Consider a single router with X delay classes. Absolute values of the average delay are a function of the aggregate rate of flows through each delay class, and the delay weights of each class. Let $r_i(t)$ be the aggregate rate of flows belonging to delay class i with delay weight Δ_i, and let the expected buffer occupancy of the delay class i be $B_i(t)$ at time t. $B = \Sigma B_i(t)$ is the aggregate buffer size. The average queueing delay $\bar{d}_i(t)$ of class i at time t is:

$$\bar{d}_i(t) = B_i(t)/r_i(t) = k/\Delta_i$$

$$\bar{d}_i(t) = \frac{1}{\Delta_i} \frac{B}{\sum_{j=1}^{X} \frac{r_j(t)}{\Delta_j}} \qquad (i = 1...N) \tag{1}$$

From Equation 1 we can infer the following properties about the dynamics of the relative delay model:

Property 1: The average delay of class i increases with the arrival rate of each class j.

Property 2: Increasing the load of a higher class causes a larger increase in the average delay of a class than increasing the load of a lower class.

Property 3: When a flow leaves the system, we would expect the average delays for all classes to decrease. However, if it turns out that flows with higher rate weights exist in higher delay classes, then most of this excess rate could be absorbed by the higher rate weight flows[3] in a higher delay class. This results in an increase in the average delays of all classes and could offset the initial decrease in delay. This non-intuitive property is illustrated using simulations in Section V.

C. Implementation

In the implementation of the mechanism described above, we try to address three practical concerns:

1. *Simplification of the computation of $\bar{d}_i(t)$*

In order to reduce the computation during packet forwarding, we approximate the calculations as shown in Figure 3. In this algorithm (Lines 5 - 8), $\bar{d}_i(t)$ is updated according to $\bar{d}_i(t) \leftarrow (d_i^*(t) + q_i(t))/s_i(t)$ rather than $\bar{d}_i(t) \leftarrow (d_i^*(t) + 0.5q_i(t)^2 + 0.5q_i(t))/(s_i(t) + q_i(t))$. This approximation turns out to be acceptable in most of our simulations.

2. *Impact of non-backlogged queues*

When delay classes become non-backlogged, their $\bar{d}_i(t)$ does not change over time, since the queue length $q_i(t) = 0$. However, the average delay $\bar{d}_i(t)$ for backlogged classes changes according to the instantaneous system load. When the empty delay classes become freshly backlogged, then the service order is distorted because of the newly backlogged queue. In order to limit the impact of this problem, we consider periodically reinitializing the state of the classes and restarting the average delay computations. While this is a crude first-cut solution, we are investigating a more sophisticated approach.

3. *Time window of computation of $\bar{d}_i(t)$*

The reinitializing of the state variables can also help address another issue. It can help eliminate past history (from before the current computation period) in the computation of $\bar{d}_i(t)$ and $s_i(t)$. In absence of periodic re-initialization of the variables, as the number of packets that are served increases, current queue sizes start to have minimal impact on the service order. This is an important practical issue to consider.

receive(p)
1 $q_i \leftarrow q_i + 1$;
2 **enqueue** in corresponding delay class;

select_packet_to_transmit()
3 $j = arg\ max_i\{\Delta_i {d_i}^*(t)/s_i(t)\}$;
4 **transmit** from delay class j;
5 $q_j(t) \leftarrow q_j(t) - 1$;
6 $s_j(t) \leftarrow s_j(t) + 1$;
7 **foreach** delay class i
8 $\quad d_i^*(t) \leftarrow d_i^*(t) + q_i(t)$;

Fig. 3. Implementation of delay mechanism

In order to address the inaccuracies caused by non-backlogged queues and accumulated history, we maintain the values for $d_i^*(t)$ and $s_i(t)$ over a moving time window. The width of the time window is proportional to the past history maintained by the system, which determines how closely the computed $\bar{d}_i(t)$ value follows the short term variations in delay. In our simulations, we use averages taken over a moving time window of 150 packets, thus making the forwarding behavior more responsive to current queue conditions.

D. Decoupling of Delay Classes from Rate

The delay mechanism described here can be *decoupled* from the mechanism to provide rate, subject to certain conditions. By decoupling we mean that (a) the end-to-end rate allocated to a flow is independent of the delay class to which it belongs, and (b) the ratio of average delays in different delay classes at a router is independent of the rates of flows.

This implies that if two flows with the same rate belong to two different delay classes, their expected average delays will be in the inverse ratio of the delay weights of their respective classes. This can be seen in Example 1.

To ensure this decoupling, the requirements on the rate mechanisms are

1. The core router should detect (incipient) congestion by checking if the *aggregate* number of packets in all packet queues exceeds a threshold.

2. Congestion feedback is not based on what packets are currently queued in the router, but only on the normalized transmission rates of the flows.

The first requirement implies that the size of packet queues corresponding to each delay class should not be limited, the aggregate queue size alone has to be bounded. By doing so, we eliminate the dependence of the average delay ratios of different delay classes on the rates of individual flows. The second requirement ensures that a flow transmitting more than its fair share will be selected for throttling even if it belongs to a delay class with a large delay weight and gets served preferentially. This effectively makes the rate of a flow independent of the delay class in which it is queued in.

The Corelite rate mechanisms that we briefly described in Section II have the above properties. It is important to note that the absolute value of the average delay experienced by a delay

[3] In Corelite, any excess bandwidth is distributed to flows according to their rate weights.

class is a function of the aggregate average rate of the flows passing through each delay class, as well as the delay weights of each delay class. However, the ratio of the average delays is independent of rate.

IV. Delay Class Adaptation

In the previous section, we described the forwarding mechanism used to achieve relative delay differentiation at a router. As the expected average delay of a flow in a class is equal to the average delay of the class when a large number of flows share a class, the expected end-to-end average delay of a flow is the sum of the average delays of the flow's delay class at each hop. As shown in the previous section, the expected delay of a class is a function of the rates of all the flows passing through the router. Hence, if a flow is assigned a fixed delay class, the end-to-end delays are subject to significant variations due to network dynamics (as the rates of each delay class change). Thus, in order to meet the expected delay requirements, a flow should have the ability to dynamically adjust its delay class.

In Corelite, in addition to specifying the average end-to-end delay requirement $\bar{D}_f$, an application also specifies the maximum price it is willing to pay to get the required delay performance. The access router maps this price to a delay class and fixes it as the maximum delay class that this flow can attain. The exact mapping functions are beyond the scope of discussion of this paper. The key idea of the adaptation algorithm is to adjust the delay class of a flow such that the current delay class of the flow is the lowest possible class that satisfies the flow's requirements (subject to the maximum delay class bound). In Corelite, the responsibility of delay class adaptation is placed on the access routers in order to enforce the adaptation [4]. The following properties of the delay model have to be considered in the design of the delay class adaptation algorithm.

Property 4: If a flow moves from class i to class j with the aggregate load remaining constant, then the average delay of each class increases if $\Delta_i < \Delta_j$, and decreases if $\Delta_i > \Delta_j$. On the other hand, the expected average delay for the flow decreases if $\Delta_i < \Delta_j$ and increases if $\Delta_i > \Delta_j$.

Proof: Recall from the previous section, that the average delay of a flow i can be written as

$$\bar{d}_i(t) = \frac{1}{\Delta_i} \frac{B}{\sum_j \frac{r_j(t)}{\Delta_j}}$$

The variation in delay of flow i assuming that it is the only flow that is adapting its delay class, and no other flow is performing adaptation is given by the partial derivative of $\bar{d}_i(t)$ with respect to $\Delta_i(t)$, i.e.

$$\frac{\partial \bar{d}_i}{\partial \Delta_i} = -\frac{1}{\Delta_i^2(t)} \frac{B}{\sum_j \frac{r_j}{\Delta_j(t)}} \left(1 - \frac{\frac{r_i}{\Delta_i(t)}}{\sum_j \frac{r_j}{\Delta_j(t)}}\right)$$

Note that $\partial \bar{d}_i / \partial \Delta_i$ is always negative. This shows that increasing the delay class of a flow always causes its delay to decrease.

Property 5: When a flow moves to a lower delay class, with all other flows remaining in their delay classes, the expected average delay for other flows will decrease.

Proof: The variation in expected average delay of flow i due to delay class adaptation of flow k, when flow i is not adapting is given by the partial derivative of $\bar{d}_i(t)$ with respect to $\Delta_k(t)$, i.e.

$$\frac{\partial \bar{d}_i}{\partial \Delta_k} = \frac{1}{\Delta_i(t)} \frac{B \frac{r_k}{\Delta_k^2(t)}}{(\sum_j \frac{r_j}{\Delta_j(t)})^2}$$

Note that $\partial \bar{d}_i / \partial \Delta_k$ is always positive. This shows that if a flow decreases its delay class, it can only improve the delay expectations of other flows.

Property 6: The marginal increase in the expected average delay with a decrease in the delay class is decreasing.

Proof: It is easy to see that $\partial^2 \bar{d}_i / \partial \Delta_i^2$ is always positive. Since $\partial^2 \bar{d}_i / \partial \Delta_i^2$ is always positive and $\partial \bar{d}_i / \partial \Delta_i$ is always negative, the above property holds.

A. Adaptation algorithm

The adaptation algorithm consists of two steps. The first step involves obtaining estimates of the end-to-end expected average delay for a flow. The second step is to change the delay class of the flow such that the end-to-end expected average delay is in tune with the average delay required by the flow.

```
  process_d_marker()
1 select delay class i for sending next packet
2 if ( head-of-line packet p of class i is d_marker )
3     p.estimated_delay += d̄_i;
4     if ( egress access router)
5         dest = p → ingress_access;
6         send_packet(p, dest);
```

Fig. 4. Delay Measurement Algorithm

The ingress access router periodically introduces special packets called d_markers along the path of the flow, to probe the end-to-end delay offered by the network for the delay class of that flow. The contents of the marker packet identify the access router that generated it and the packet flow uniquely within the ingress access router. The d_markers contain an $estimated\_delay$ field that is set to zero initially. When the d_markers are processed by the core routers, the $estimated\_delay$ field is updated by adding the average per-hop delay of the flow's delay class as shown in Figure 4. When an egress access router receives a d_marker, it sends it back to the access router that generated the d_marker (Lines 4-5). The core routers or the egress access router need not maintain state about the flows pertaining to the d_markers they process.

Figure 5 shows the adaptation algorithm at the access router. When the ingress access router receives the d_marker, it will have an estimate of the end-to-end expected average delay for packets of a flow along that path (Lines 2-4). This is the path-specific end-to-end average delay experienced by packets belonging to the delay class of the flow (Lines 5-6). If this delay is greater than the required average delay, the access router increases the delay class of the flow to the next higher class, subject to the maximum delay class constraint (Lines 7-9).

However, if the estimated delay is less than the required delay, the access router tries to shift the flow to a lower delay class, if possible. To do this, the access router calculates the estimated delay for the next lower delay class. If it is less than the required delay, then the access router decreases the delay class of the flow to the next lower class. However, when a number of flows perceive improved delay performance, if all flows move to a lower

[4] As a result, we only enforce edge-to-edge adaptation as opposed to true end-to-end adaptation.

adapt_delay()

1 **on** receiving d_marker p from egress access
/* get pointer to flow */
2 $f = get\_flow(p)$;
3 $f \rightarrow est\_delay = p \rightarrow estimated\_delay$;
4 $i = f \rightarrow delay\_class$;
5 $\bar{d}_i = f \rightarrow est\_delay$;
6 **if** ($\bar{d}_i > \bar{D}_f$) /* True: delay violation */
7 **if** ($i < f \rightarrow max\_delay\_class$)
8 $f \rightarrow delay\_class$ ++;
9 **else** /* Move to lower delay class */
10 $\bar{d}_{i-1} = \bar{d}_i \Delta_i / \Delta_{i-1}$;
/* True: lower delay class is sufficient */
11 **if** ($\bar{d}_{i-1} \leq \bar{D}_f$)
/* probabilistic jump */
12 $prob = random(0, 1)$;
13 **if** ($prob < (\bar{D}_f - \bar{d}_{i-1})/\bar{d}_i$)
14 $f \rightarrow delay\_class$ - - ;

Fig. 5. Delay Class Adaptation Algorithm

delay class, they all might end up exceeding their required average delay. To compensate this they would increase again to the next higher class. To reduce such oscillations in adaptation, when the access router jumps to a lower delay class $i-1$ from a delay class i, it does so with a probability $(\bar{D}_f - \bar{d}_{i-1})/\bar{d}_i$ (Lines 10-15), thereby allowing only a subset of flows whose $\bar{D}_f - \bar{d}_{i-1}$ is higher than the others.

Initially, a flow starts in its maximum delay class. At steady state, all flows will settle down to the lowest delay class possible that satisfies their delay constraints. However, if the network state changes, unsatisfied flows will move up to the maximum possible delay class j that gives the required average delay, from their current class i, where $i \leq j$. In the event that a flow is unable to get the required average delay from the network, it will remain in its maximum possible delay class and that is a function of the price it is willing to pay.

B. Delay Class Convergence

We show through a simple analysis that the delay class of a flow converges during adaptation.

Since a flow's expected average delay at a hop is the same as the average delay of the class it belongs to, equation 1 can be restated to give the expected average delay $\bar{d}_i(t)$ of flow i at time t.

$$\bar{d}_i(t) = \frac{1}{\Delta_i(t)} \frac{B}{\sum_j r_j/\Delta_j(t)}$$

where r_j is the rate of flow j. Let $\bar{D}_i$ be the average delay requirement of flow i at each hop.

Property 7: Consider the case when flow i is adapting, while the other flows do not adapt. The change in delay class of flow i at time t is given by [5]

$$\frac{d\Delta_i}{dt} = -\alpha \Delta_i^2 (\bar{D}_i - \bar{d}_i(t)). \tag{2}$$

This differential equation can be solved as

$$\frac{1}{\Delta_i(t)} - \frac{B\,Y_i(t)}{r_i(\bar{D}_i r_i - B)} \log(\frac{1}{\Delta_i(t)} + \frac{Y_i(t)}{r_i} + \frac{BY_i(t)}{r_i(\bar{D}_i r_i - B)}) = \frac{\alpha}{r_i}(\bar{D}_i r_i - B)t \tag{3}$$

where $Y_i(t) = \Sigma_{i \neq j} r_j / \Delta_j(t)$.

Consider two cases:
Case (a) $\bar{D}_i > B/r_i$

$$t \rightarrow \infty, \Delta_i \rightarrow 0$$

In this case, the required average delay is greater than the worst case average delay that will be experienced by the flow. Hence, the flow will tend to the class with the lowest possible delay weight.

Case (b) $\bar{D}_i < B/r_i$

$$t \rightarrow \infty, \Delta_i = \frac{r_i}{Y_i(t)}(\frac{B - \bar{D}_i r_i}{\bar{D}_i r_i})$$

Here, adaptation occurs and the flow oscillates around the point where $\bar{D}_i = B/r_i$. The oscillations dampen exponentially and hence the system converges.

□

When the rates of flows do not change, then Equation 2 can be written as

$$\frac{d(\sum_j \frac{r_j}{\Delta_j(t)})}{dt} = \alpha(\sum_j \bar{D}_j r_j - B) \tag{4}$$

since we know that $\sum_j r_j \bar{d}_j(t) = B$.

This results in a very interesting observation. The derivative is constant (assuming the rates do not change with time). In other words, for a given scenario, the set of requirements is either feasible or not. This results in the following stability/feasibility condition:

$$\Sigma_j \bar{D}_i r_j \geq B.$$

□

The feasibility condition described above is simplistic and a detailed treatment of feasibility is not presented due to space limitations. If the set of delay requirements is not feasible, then each flow will converge to its maximum delay class that is mapped from the price that the flow is willing to pay.

Property 8: From Equation 4, we can get the delay class of a flow at equilibrium, when all the flows are simultaneously adapting.

$$\Delta_i(t) = \frac{\sum_j \bar{D}_i r_j}{\alpha \bar{D}_i (\sum_j \bar{D}_i r_j - B)} \frac{(t+b)^a - b^a}{(t+b)^{a+1} - b^{a+1}} \tag{5}$$

where $a = \frac{B}{(\sum_j \bar{D}_j r_j - B)}$ and $b = \frac{\sum_j r_j/\Delta_j(0)}{(\sum_j \bar{D}_i r_j - B)}$.

As $t \rightarrow \infty, \Delta_i(t) \rightarrow \Delta_i^{'}$. Thus, the delay class at which the flow i converges is given by

$$\Delta_i^{''} \leftarrow \min\{\max(\Delta_i^{'}, \Delta_{min}), \Delta_{max}\}$$

[5] For simplicity of analysis, we assume the change in delay class is proportional to the slack in delay. However, since this might lead to oscillations around the stable point, in our implementation, we always change only to the next delay class.

where Δ_{min} and Δ_{max} are the lowest and highest delay classes of the system respectively.

□

V. SIMULATIONS

In this section, we use simulations to evaluate the algorithms used to achieve proportional delay differentiation with a stateless core and delay adaptation at the source (rather than the access router). The simulations as on the *ns*-2.1b4a simulator [11]. The sources are rate adaptive, decreasing their rates upon congestion feedback from the routers and incrementing their rates periodically otherwise. The topology used is shown in Figure 6. The three links C1-C2, C2-C3 and C3-C4 are shared by the different sources and observe periods of low to heavy congestion. All the links have a bandwidth of 4Mbps and a latency of 5ms. Also, in all the simulations, we have used a fixed packet size of 1KB, a moving window of 150 packets to update d/s, and a period of 200ms(roughly 3 round trip times) to send d_markers. Though there are a total of 20 flows traversing different sets of links, for clarity, only the three flows (8, 9 and 10) that traverse all the three shared links are used to show results that depict per-flow values.

In all our examples, the marker generation algorithm is not aggressive by design. Consequently, it allows queue sizes to grow sharply before throttling flows aggressively. This causes oscillations in queue sizes and observed delays; the interesting point that we illustrate is that even under this scenario, i.e. even if the absolute delays oscillate sharply, the relative delay invariant is maintained.

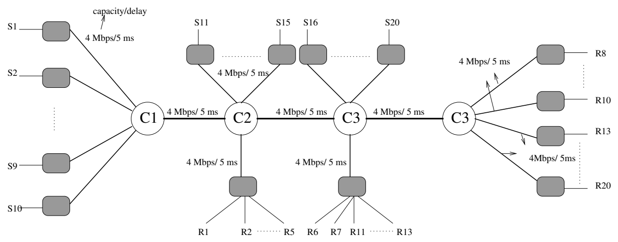


Fig. 6. Network topology

A. Relative Delay Classes

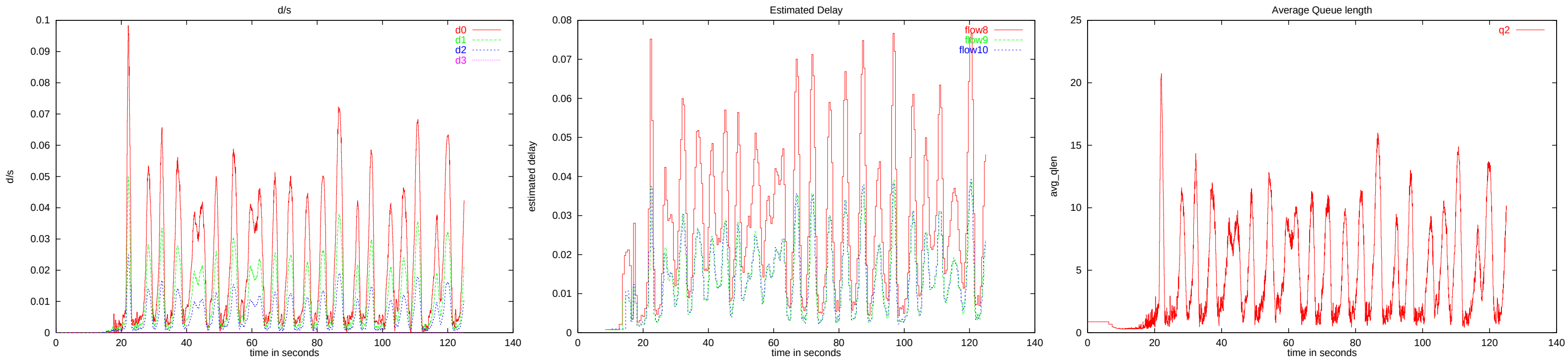
The first set of results demonstrates the efficacy of the MDP scheduler to achieve proportional delay classes at any given router. There are a total of 3 delay classes 0, 1 and 2 with delay weights 1, 2 and 4 respectively. Flow 8 belongs to delay class 1, while flows 9 and 10 belong to delay class 2. The graphs in Figure 7 (a), (b) and (c) show respectively the average delay experienced by each of the 3 delay classes at the core router C3, the end-to-end queuing delay estimated by flows 8-10 by sending d_marked packets and the average queue length at the core router C3 during the course of the simulation. As explained before, the variations in average queue size shown in Figure 7 (c) are a result of the specific congestion detection and notification mechanisms at the core router and the rate adaptation at the access router. The size of the window used to compute d/s determines how closely this value follows the current queue size. Figures 8 (a) and (b) are the zoomed in versions($10sec$) of Figures 7 (a) and (b), and Figure 8 (c) shows the actual delay experienced by individual packets from the 3 flows during the corresponding 10seconds. From Figure 8 (a) it is clear that the ratio of 1:2:4 is maintained at all times. The graph in Figure 8 (b) corresponds to $\sum \frac{d_i}{s_i}$ obtained by flows that belong to class i in a given path which is the average end-to-end queuing delay experienced by its packets. As both flows 9 and 10 have the same delay weights they both have approximately overlapping estimated delay values. Also, for flow 8, the estimated delay is twice the value perceived by the other 2 flows which conforms to the ratio of their delay weights. Figure 8(c) shows that the average delay experienced by packets of a flow, does not deviate significantly from the average delay of the class the flow belongs to.

B. Delay Increase Anomaly

Figure 9 illustrates the anomaly that we observe when a few flows leave the network. Intuitively, when some flows leave the network the delay experienced by the remaining flows that had shared the path with the flows that left, should either decrease or at least remain the same. But we show that the average delay may increase when some select flows leave the network. In this scenario, we consider two sets of flows, say $s1$ and $s2$, with delay weights of 1 and 8 respectively. They also share the available bandwidth in the ratio 1:5. When a bunch of flows from $s1$ leave the network, the average delay experienced by all the remaining flows increases. In the simulation scenario given here, flows 8 and 10 belong to the $s1$ and flow 9 belongs to $s2$. Figure 9 (a) shows the instantaneous sending rates of these flows. At time 60 sec, a few flows in $s1$(flow8) leave the network. The network reaches the steady state by time 70 sec. From Figures 9 (b) and (c), it is clearly seen that the average delay experienced by the flows is higher than before. This is because, when flows with a lower delay class leave the system, this available bandwidth is absorbed by the remaining flows and if these flows belong to a higher delay class, then by equation (1), the average delay will increase.

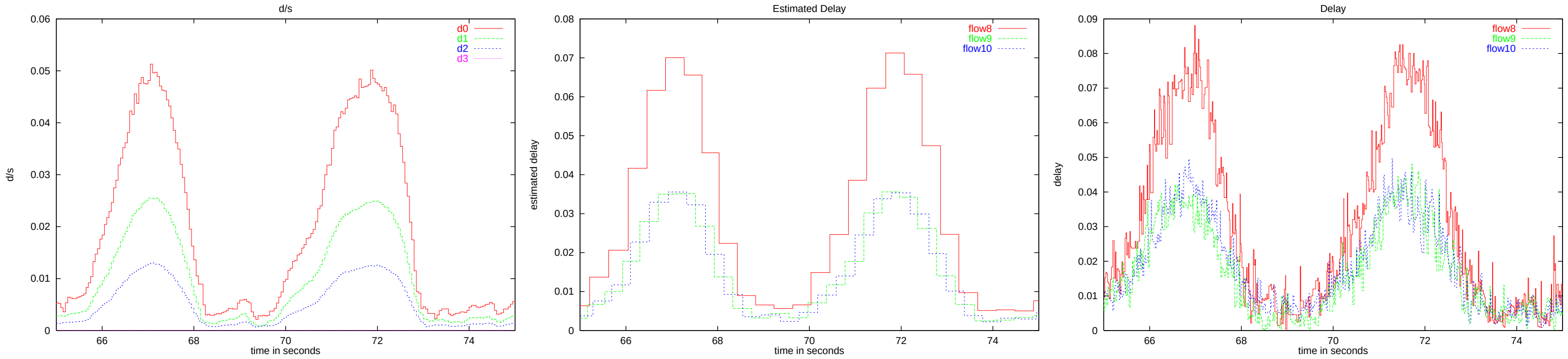
C. Delay Adaptation

This scenario illustrates the effect of delay adaptation. In this case, each router has 5 delay classes. Flows 8, 9 and 10, can all move up to the maximum delay class of 4 and have a delay requirement of 0.01 sec, 0.02 sec and 0.05 sec respectively. The graphs in Figure 10 (a), (b) and (c) show respectively the current delay class of each flow, the estimated average end-to-end queuing delay for the class the flow is in and the actual delay experienced by the packets of the flow, during a 10 sec interval in the simulation. The variation in queue size was similar to that observed in the simulations in section V-A. At time 35 sec, all flows have their delay requirements satisfied. Flow 8, 9 and 10 are in delay classes 4, 3 and 2 respectively. Between time 35 and 37 sec, there is a fall in the average queue sizes, resulting in lower average delays even for the lower delay classes. As flows are always kept in the lowest possible delay class that provides the desired delay, flows 9 and 10 shift to delay class 0, while flow 8 moves to delay class 1. If the delay requirement for a flow cannot be satisfied then it remains at the maximum delay class possible. As the estimated delay increases beyond the desired limit, the flow is moved to a higher delay class if possible. For example, by the $39th$ sec, the delay experienced by all of the flows force them to be shifted to higher delay classes. Though flow 10 can move up to delay class 4, it remains at 2, because its delay requirement of 0.05 sec is satisfied. From Figure 10 (b), it is clear that most of the time, the flows do manage to maintain a delay average that is close to their desired value. Also the actual



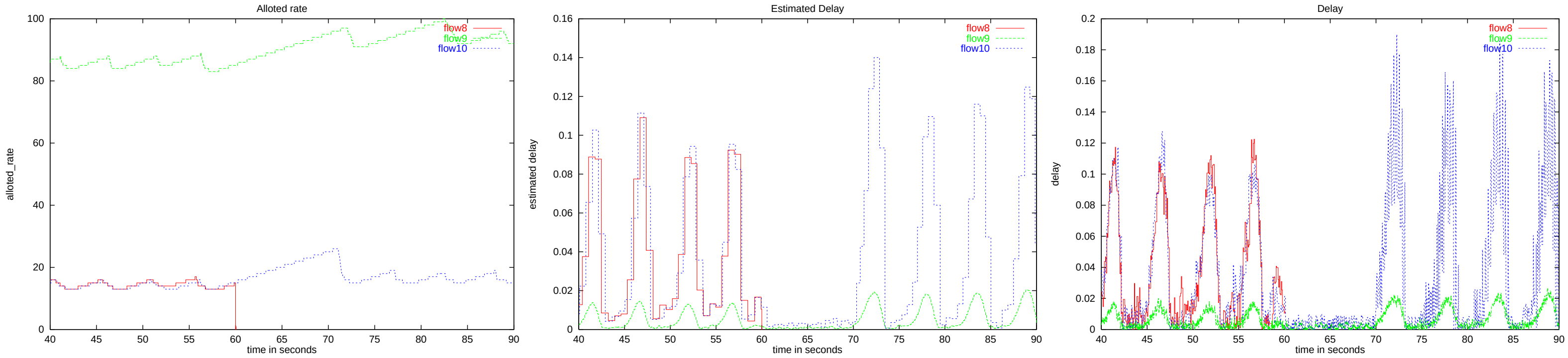
(a) Average Delay for Delay Class (b) Estimated Delay for Flows (c) Queue length at router C3

Fig. 7. Relative Delay



(a) Average Delay for Delay Class (b) Estimated Delay for flows (c) Actual Delays for flows in (b)

Fig. 8. Relative Delay - over a period of 10 seconds



(a) Rate of flows (b) Estimated Average Delay for flows (c) Actual Average Delay for flows

Fig. 9. Delay Adaptation

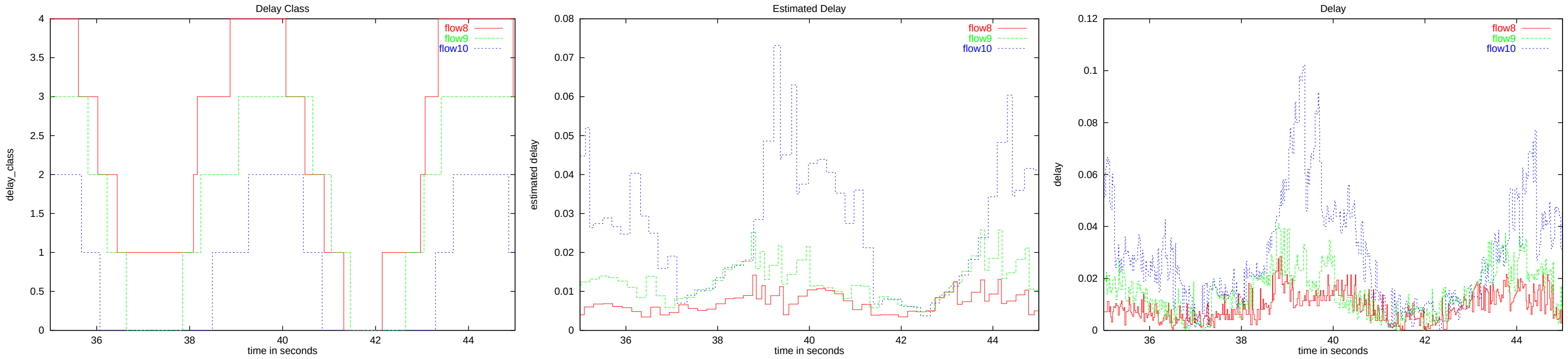
delay experienced by the flows, shown in Figure 10 (c) indicates that the delay estimated using d_markers does provide a correct estimate of the average delay experienced.

D. Summary of Simulations

Though we have done additional simulations using different traffic patterns and with different topologies, we have not presented them due to space constraints. In summary, our simulations indicate that the MDP algorithm maintains the proportion between the delay classes effectively for the small simulations under consideration. We believe that the approach will scale well with increase in the number of flows, but we need to verify the behavior under more bursty scenarios. The delay adaptation algorithm gracefully adapts to the dynamics of the network and converges to a stable state fairly responsively though it will still be too slow for very short-lived flows. Further, in all the simulations shown in this section, flows obtained their fair-share of bandwidth as determined by the rate adaptation and congestion indication mechanisms, independent of the current delay class to which they belong.

VI. Discussion

In this section, we present some of the issues and limitations of our work. Some of these are inherent to our architecture, while the others are due to the status of our current work.



(a) Delay classes of flows (b) Estimated Average Delay of flows (c) Actual Average Delay for flows

Fig. 10. Delay Adaptation - over a period of 10 seconds

- *Architectural issues:* We have considered a single-cloud network in this paper. If all clouds in a multi-cloud network support the same set of delay classes (with the same delay weight), then our approach is just as applicable. Otherwise, we need a translation mechanism at the ingress router of each cloud that maps the desired delay class to a corresponding delay class for the cloud. While the basic ideas of relative delay differentiation and delay adaptation will still work in this case, the analysis will need to change to accommodate the delay class translation along the path of a flow. The interaction between the clouds and a comparison between delay adaptation on a per-cloud basis versus end-to-end delay adaptation by the access router are ongoing work.
- *Implementation issues:* At the core router, the variables of each delay class are updated every time a packet is served. We can reduce the computation overhead in two ways: first, as we showed in Section 3, we can simplify the state update computation; and second, we can invoke the computation less frequently (e.g. once every 10 packets). Neither simplification will significantly change the overall performance of the approach, though both will increase the minimum time window over which the delay invariant can be maintained.

Let us now investigate the packet overheads. The access router sends d_markers periodically. The d_marker contains the id of the ingress access router, and the $estimated\_delay$ field in addition to the regular flow identification information contained in a data packet. The frequency of markers determines the granularity of delay estimation. The marker overhead itself is quite small (less than 0.2% for a marker sent every 10 packets). From an implementation perspective, this approach seems quite feasible.
- *Adaptation issues:* There are two important issues to be addressed while doing delay class adaptation at the access router. First is the interaction between adaptive applications and the access router adaptation. Adaptive applications adapt their performance according to the delay of the received packets. This adaptation could happen on a finer granularity than the access router adaptation, and hence lead to undesirable interactions. To prevent these, the application could refrain from adaptation till the maximum delay class is reached and the access router cannot help sustain the required delay for the application any longer.

The second, and by far more important, issue is that of packet reordering during adaptation. Specifically, when a flow is moved to a higher delay class i from a lower delay class j, the packets of that flow in class i might reach earlier than the packets of the flow in class j. This will result in out-of-order delivery of packets. Transport protocols like TCP might consider them as lost packets and react adversely in such scenarios. There are three possible solutions to this problem. First, the ingress access router can hold packets (proportional to the difference in average delay between the two classes) prior to a delay class transition in order to "flush out" preceding packets in a lower delay class. Second, the egress access router can buffer packets for some time (proportional to the difference in average delay between the two classes) and resequence them. Third, most soft real-time applications inherently buffer data packets for a lookahead period, thereby smoothing the effect of out-of-order delivery. All these approaches are feasible, though they all have drawbacks. We are working on this problem at this time.

VII. Related Work

There exists a large body of work on providing absolute end-to-end delay bounds and relative per-hop delay differentiation. We however know of no other approach that provides delay class adaptation. GPS based schedulers and EDF schedulers provide absolute guarantees using per-flow information and with complex admission control mechanisms[12], [13], [14]. Non-work conserving disciplines also provide similar guarantees with per-flow information [15], [16]. However, the requirement of per-flow state and complex admission control requirements make these approaches unviable in a Diffserv architecture.

More recently, interest has turned to providing guaranteed per-flow delay bounds based on traffic aggregates, in tune with the Diffserv paradigm. Jacobson et.al. [3] have proposed a premium service model that provides the equivalent of a dedicated link between two access routers. However, to achieve the guarantees it seeks to provide, the fraction of bandwidth that can be allocated to premium service has to be very low. The SCED+ algorithm proposed by Cruz [17] provides both guaranteed and statistical rate and delay bounds, and addresses scalability through traffic aggregation and statistical multiplexing. In addition, the bounds for rate and delay are decoupled. The SCED+ algorithm has a complex admission control policy, which may not be feasible as a practical implementation at high-speed core routers. It also requires per-session traffic shaping at each hop, where a session is either an individual flow or a traffic aggregate. Stoica et.al. describe an architecture to provide guaranteed services without per-flow state management using a technique called Dynamic Packet State (DPS) [6]. DPS assures service differentiation and performance guarantees on a per-flow basis, through a distributed implementation of the Virtual Clock algorithm. The coupling between the rate and delay bounds exists in DPS, though. Finally, both SCED+ and DPS require significant processing overhead at the core router, as they have to maintain a sorted queue of packets, sorted on the basis of deadlines. This complexity increases with the number of flows in the network core. It is interesting to note that most of the above algorithms assume the presence of best-effort traffic to improve network efficiency.

If absolute delay differentiation lies at one end of the differentiated services spectrum, relative delay differentiation is at the other end. The former guarantees worst-case bounds on a per-flow basis, while the latter provides service differentiation on a per-hop basis, and only in the average case. Relative delay differentiation is discussed in detail by Dovrolis et.al. [8]. They present two packet schedulers that try to achieve proportional delay differentiation at each individual hop. However, the schedulers are not ideal, in the sense that, the average delays experienced by different classes tend to deviate from the proportional model under light traffic loads, and in certain short time-scales. Another algorithm that seeks to provide relative delays is the Time-Dependent Priorities algorithm [18], where the service priority of a packet in queue i at time t is given by $p_i(t) = w_i(t)\Delta_i$, where $w_i(t)$ is the waiting-time of the packet at time t and Δ_i is the weight of the delay class. It differs from our algorithm in the fact that the head-of-line packet reflects the system load only up to the moment it arrived in the queue. In particular, when the delay weights are equal for all classes, it degenerates to an FCFS server. In our algorithm, the head-of-line packet reflects the system load till the time it is served, thus leading to better convergence to the averages under light and heavy traffic loads, and under short time-scales too. On the contrary, the Time-Dependent Priorities algorithm has been shown to fail under moderate or light loads, and with widely ranging Δ_i ratios [8].

Contemporary literature has focussed on the large class of soft real time applications and their requirements in terms of statistical bounds rather than worst-case bounds for soft real-time applications. Given such bounds, applications have three choices regarding the service they receive: (a) they might choose to use what they receive from the network and tolerate network-induced service disruptions, (b) applications might adapt their performance to suit the delay they perceive from the network, as in a playback application [9], [10], and (c) they might want the adaptation to be performed in the network, and obtain consistent performance, which we think is better than the other two since the application can be less complex by off-loading the burden of adaptation onto the network.

VIII. Conclusion

This paper contains two contributions: a mechanism to provide per-hop per-class relative average delay differentiation, and a mechanism to perform end-to-end delay adaptation. In concert, these mechanisms seek to achieve the average delay requirements of soft real-time applications in a core-stateless network. Through simulations and analysis, we have shown that the adaptation algorithm is stable, scalable and feasible. Several important issues need to be addressed: (a) we need to discuss the feasibility constraints in more detail, (b) we need to consider the interaction between multiple network clouds, and between the end host adaptation and the access router adaptation, (c) we need to understand the relationship between our work and the larger issue of network pricing, and (d) we need to evaluate possible approaches for eliminating packet reordering during delay adaptation. These are some of the areas of ongoing work.

We conclude with a final observation: absolute differentiation and relative differentiation cater to applications with contrasting requirements. The former offers absolute assurances to 'unelastic' applications, while the latter is tailored towards 'elastic' applications. Since both types of applications are likely to co-exist in the future Internet, absolute and relative differentiation mechanisms are equally important for the commercial viability of the future inter-networks. We believe that our architecture will increase the flexibility and capabilities of services that can be built using a Diffserv-like architecture.

References

[1] S. Shenker, and C. Patridge. Specification of Guaranteed Quality of Service. *RFC 2212*, September 1997.

[2] S. Shenker, and J. Wroclawski. General Characterization Parameters for Integrated Service Network Elements. *RFC 2215*, September 1997.

[3] K. Nichols, V. Jacobson and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet," *draft-nichols-dsopdef-00.txt, Internet Draft*, November 1997.

[4] V. Jacobson, "An Architecture for Differential Services," *Talk in the Intserv WG at the 39th IETF*, Munich, Germany, August 1997.

[5] S. Brim, B. Carpenter and F. Le Faucheur, "Per Hop Behavior Identification Codes," *draft-ietf-diffserv-phbid-00.txt, Internet Draft*, October 1999.

[6] I. Stoica and H. Zhang, "Providing Guaranteed Services Without Per Flow Management," *ACM SIGCOMM '99*, Cambridge, MA, September 1999.

[7] R. Sivakumar, T. Kim, N. Venkitaraman and V. Bharghavan, "Achieving Per-Flow Weighted Rate Fairness in a Core Stateless Network," *IEEE Conference on Distributed Computing Systems 2000*, Taipei, Taiwan, March 2000.

[8] C. Dovrolis, D. Stiliadis and P. Ramanathan, "Proportional Differentiated Services: Delay Differentation and Packet Scheduling," *ACM SIGCOMM '99*, Cambridge, MA, September 1999.

[9] D. Clark, S. Shenker and L. Zhang, "Supporting real-time applications in an Integrated Services Packet Network: architecture and mechanism," *ACM SIGCOMM '92*, Baltimore, MD, August 1992.

[10] S. B. Moon, J. Kurose, D. Towsley, ""Packet Audio Playout Delay Adjustment: Performance Bounds and Algorithms," *ACM/Springer Multimedia Systems*, vol.5, pp. 17-28, January 1998.

[11] UCB/LBNL/VINT Network Simulator - ns (version 2), *http://www-mash.CS.Berkeley.EDU/ns/*.

[12] A. Parekh, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks," *PhD Thesis*, MIT Laboratory for Information and Decision Systems, Technical Report LIDS-TR-2089, 1992.

[13] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms,"*ACM SIGCOMM '96*, Stanford, CA, September 1996.

[14] L. Georgiadis, R. Guerin and A. Parekh, "Optimal multiplexing on a single link: delay and buffer requirements," *IEEE Transactions on Information Theory*, vol. 43, no. 5, pp. 1518-1535, September 1997.

[15] N. R. Figueira and J. Pasquale, "Leave-in-time: A new service decipline for real-time communications in a packet-switching network," *ACM SIGCOMM '95*, Cambridge, MA, September 1995.

[16] D. Verma, H. Zhang and D. Ferrari, "Delay Jitter Control for Real-Time Communications in a Packet-Switching Network," In *Tricom '91*, Chapel Hill, NC, March 1991.

[17] R. L. Cruz, "SCED+: Efficient Management of Quality of Service Guarantees," *IEEE INFOCOM '98*, San Francisco, CA, March 1998.

[18] L. Kleinrock, "A Delay Dependent Queueing Discipline," *Naval Research Logistics Quarterly*, vol. 11, no. 4, December 1964.