Dynamic Class Selection: from Relative Differentiation to Absolute QoS

Constantinos Dovrolis
University of Delaware
dovrolis@cis.udel.edu

Parameswaran Ramanathan
University of Wisconsin
parmesh@ece.wisc.edu

Abstract— *The relative differentiation architecture does not require per-flow state at the network core or edges, nor admission control, but it can only provide higher classes with better service than lower classes. A central premise in this context is that users with absolute QoS requirements should dynamically search for an appropriate class. We investigate this Dynamic Class Selection (DCS) framework, and illustrate that, under certain conditions, DCS-capable users can meet absolute QoS requirements, even though the network only offers relative differentiation. For a single link model, we can examine whether it is feasible to satisfy all users, and when this is the case, compute the minimum acceptable class selection for each user. Users converge in a distributed manner to this minimum acceptable class, if the DCS equilibrium is unique. However, suboptimal DCS equilibria may also exist. Simulations of a delay-based DCS algorithm show the relation between class differentiation and DCS, and demonstrate how to control the trade-off between the performance and cost of a flow.*

I. Introduction

Providing some kind of service differentiation in packet networks, and especially in the Internet, has been a problem of considerable commercial and research importance in at least the last ten years. The Differentiated Services (DiffServ) architecture [1] was proposed as a more scalable solution to this problem, compared to previous approaches such as the Integrated Services (IntServ) architecture. The DiffServ vision is that stateless priority mechanisms at the network core can be combined with stateful mechanisms at the network edges, in order to construct versatile end-to-end services. Among the first DiffServ proposals was the Virtual Leased Line (VLL) service [2]. VLL offers negligible queueing delays and no losses, when the VLL traffic is shaped at the network ingress to a certain contracted rate. Recent findings, however, have raised concerns about the effect of aggregation on the VLL service [3], [4]. Another early DiffServ proposal was the Assured service [5]. This model offers a certain bandwidth profile to a user through selective marking at the edges and priority dropping at the core, when the network is appropriately provisioned. It has been shown, however, that it is difficult to design good provisioning algorithms for the Assured service [6]. Also, in the context of TCP transfers, [7] showed that it may be impossible to provide a certain throughput to a TCP connection. Several other models for scalable differentiated services have been proposed. The *SCORE* architecture can provide per-flow service guarantees without per-flow state in the core routers [8]; instead, the QoS-related information is carried in the packet headers. [9] proposes a core-stateless scheme in which resource management and admission control are performed only at egress routers. The Asymmetric Best Effort [10] provides two service classes: one for delay-sensitive applications and another for throughput-sensitive applications.

A simpler proposal is to provide *relative differentiation* [11]. In this architecture, the network offers N classes of service which are ordered so that *Class i is better than Class i-1 for $1 < i \leq N$, in terms of local (per-hop) metrics for the queueing delays and packet losses.* There is no admission control or reservation of resources, and so the network cannot provide absolute Quality-of-Service (QoS) guarantees, such as a maximum end-to-end delay or loss rate. Instead, it is up to the end-points (applications, end-host protocols, users) to select the class that best meets their QoS requirements, cost, and policy constraints. A central premise in the relative differentiation architecture is that *if the end-points have absolute QoS requirements, they have to dynamically search for a class that satisfies these requirements.* Since higher classes would obviously cost more, users that also care to minimize the cost of their flows, would have to search for the *minimum acceptable class*. This model of Dynamic Class Selection (DCS), however, has not been investigated. It also raises some critical questions that need to be answered before accepting the relative differentiation architecture as feasible and robust. What happens when a population of users search for an acceptable class in a distributed manner? Does each user find an acceptable class when the network is well-provisioned? Which factors determine the well-provisioning of a network? Which are the important parameters in a practical DCS algorithm?

Our objective in this paper is to investigate these questions

This work was supported by a research fund from Cisco Systems Inc.

in the context of the Proportional Differentiation (PD) model [11]. We first argue that the PD model provides an appropriate 'per-hop behavior' for supporting DCS, because it maintains the relative class differentiation even when DCS causes significant variations in the class load distribution. Also, the PD model can control the QoS spacing between classes, allowing the network provider to provision the class differentiation based on the absolute user QoS requirements. We then consider a simplified analytical model of DCS on a link that offers proportional delay differentiation. We give an algorithm that checks whether it is feasible to satisfy all users, and if this is the case, computes the minimum acceptable class selection for each user. Users converge in a distributed manner to this minimum acceptable class, if the DCS equilibrium is unique. However, other suboptimal DCS equilibria may also exist. When the link is under-provisioned, an equilibrium still exists but some users, namely those with the most stringent requirements, stay unsatisfied in the highest class. Finally, we present simulation results of an end-to-end DCS algorithm in a multihop network. The simulations provide additional insight in several DCS aspects, such as the importance of selecting appropriate class differentiation parameters, the trade-off between the performance and cost of a flow, and the factors that determine the well-provisioning of a network.

The structure of the paper is as follows. Section II discusses DCS in more detail, and shows the relation between DCS and proportional differentiation. A single link DCS model and the related analytical study are given in Section III. Section IV summarizes the DCS simulation study. A discussion of related work follows in Section V. We summarize in Section VI.

II. DCS AND PROPORTIONAL DIFFERENTIATION

In the context of relative differentiated services, the network only offers per-hop relative delay and loss differentiation between a number of classes. A user that does not have absolute QoS requirements, can choose a class based on the maximum tariff that she is willing to pay. That class will provide the best possible QoS, given her cost constraints. A user (or application) that has absolute QoS requirements, on the other hand, has to dynamically choose a class in which the observed QoS is acceptable. If the user is also interested in minimizing the cost of the session, she would choose the least expensive, or minimum, class that is acceptable. Suppose that it is the sender that makes the class selections, and the receiver that monitors the end-to-end QoS. The receiver can notify the sender about the observed QoS through a *feedback channel*, such as the one provided by the Real-Time Control Protocol (RTCP) [12]. Based on this feedback, the sender decides whether to stay in the same class, or switch to the higher or lower class. We refer to this user-adaptation framework, as *Dynamic Class Selection (DCS)*. In DCS, users may experience transient performance degradations as they search for an acceptable class. Such transient performance degradations would be unacceptable for *unelastic* applications that require strict (or deterministic) QoS guarantees. We consider such applications outside the scope of relative differentiation. If the application is *adaptive*, or if it only has *statistical* QoS requirements, the transient performance degradations can often be masked using a variety of techniques based on playback-adaptation, retransmissions, FEC, or loss concealment.

The proportional differentiation model of [11] provides an appropriate per-hop behavior for DCS due to the following reasons. First, the proportional differentiation model is *predictable*, meaning that *higher classes provide better performance than lower classes, independent of the aggregate load or the class load distribution*. This is particularly important in DCS, because the class load distribution can be strongly nonstationary as users move from one class to another. Relative differentiation mechanisms that are based on static resource partitioning between classes are not predictable when the class load distribution varies. This was shown in [11] for the case of Weighted-Fair-Queueing (WFQ) scheduling, and in [13] for the case of Complete-Buffer-Partitioning (CBP) dropping.

The second reason is that the proportional differentiation model is *controllable*, meaning that *the network provider can adjust the performance spacing between classes based on certain class differentiation parameters*. These differentiation parameters allow the network provider to provision the performance spacing between classes based on the corresponding performance spacing in the QoS requirements of different traffic types or users [14]. As shown in §IV, this type of 'class provisioning' can increase the number of DCS users that can find an acceptable class, leading to a more efficient network operation. Other relative differentiation mechanisms, such as the *strict prioritization* model [11], cannot adjust the performance spacing between classes, and so the network provider has no means to provision the class differentiation.

Finally, the proportional differentiation model can be applied in both queueing delays [15] and packet losses [13], and so it can support a number of DCS performance requirements, such as a maximum end-to-end or round-trip delay, a maximum loss rate, or combinations of these metrics, such as a minimum TCP throughput.

III. A SINGLE LINK DCS MODEL

In this section, we study an analytical model of DCS in the simplified case of a single link. The link provides proportional delay differentiation, while the users search for the minimum class that provides them with an average queueing delay that is less than a certain threshold.

Consider a network link $\mathcal{L}$. The offered rate in $\mathcal{L}$ is λ, the

capacity is C, and the utilization is $u = \lambda/C < 1$. Assume that the link has adequate buffers to avoid any packet losses. $\mathcal{L}$ offers N classes of service, which are relatively differentiated based on the *Proportional Delay Differentiation (PDD) model* of [15]. Specifically, if $\bar{d}_i$ is the average queueing delay in class i, the PDD model requires that

$$\frac{\bar{d}_i}{\bar{d}_j} = \frac{\delta_i}{\delta_j} \qquad 1 \le i, j \le N \tag{1}$$

where $\delta_1 = 1 > \delta_2 > \ldots > \delta_N > 0$ are the *Delay Differentiation Parameters (DDPs)*. Note that according to the PDD model, higher classes have lower average delays, *independent of the class loads*. As shown in [15], when the class load distribution $\{\lambda_n\}$ is given, the average queueing delay in class i under the PDD constraints is

$$\bar{d}_i = \frac{\delta_i \, \bar{q}_{ag}}{\sum_{n=1}^{N} \delta_n \lambda_n} \tag{2}$$

where $\bar{q}_{ag} = \sum_{n=1}^{N} \lambda_n \bar{d}_n$ is the *average aggregate backlog* in $\mathcal{L}$. From the conservation law [16], the aggregate backlog $\bar{q}_{ag}$ is independent of the class load distribution or the scheduling algorithm, when the latter is work-conserving and indifferent to packet sizes. $\bar{q}_{ag}$ only depends on the link utilization and on the statistical properties (burstiness) of the traffic.

Suppose now that a population of users $\mathcal{U} = \{1, \ldots, U\}$ create the traffic of link $\mathcal{L}$. Each user j generates a stationary flow with an average rate r_j, and can tolerate a maximum average queueing delay ϕ_j in $\mathcal{L}$. The total offered rate to $\mathcal{L}$ is $\lambda = \sum_j^U r_j$. Without loss of generality, $\mathcal{U}$ is ordered so that $\phi_1 \ge \phi_2 \ge \ldots \ge \phi_U > 0$. We also assume that all users generate traffic with the same packet size distribution. Suppose that each user j in $\mathcal{U}$ selects a class $c_j \in \{1, \ldots N\}$. The corresponding *Class Selection Vector (CSV)* $\mathbf{c}$ is defined as $\mathbf{c} = (c_1, c_2, \ldots, c_U)$. Given a CSV $\mathbf{c}$, the offered rate in each class i is

$$\lambda_i(\mathbf{c}) = \sum_{j:c_j=i}^{U} r_j \tag{3}$$

From (2) and (3), we see that the CSV $\mathbf{c}$ determines the average delay $\bar{d}_i(\mathbf{c})$ in each class i, when the DDPs and the average aggregate backlog are given. Also note that the average aggregate backlog does not depend on the CSV $\mathbf{c}$. *The objective of each user is to select the minimum class that meets the user's delay requirement.* A user j is said to be *satisfied with* $\mathbf{c}$ if $\bar{d}_{c_j}(\mathbf{c}) \le \phi_j$; otherwise, the user is said to be *unsatisfied with* $\mathbf{c}$. The CSV $\mathbf{c}$ is called *acceptable*, if all users in $\mathcal{U}$ are satisfied with $\mathbf{c}$. When $\mathbf{c}$ is acceptable, we also say that class c_j is acceptable for user j. CSVs can be compared in the following sense: $\mathbf{c}' \ge \mathbf{c}$ when $c'_j \ge c_j$ for all $j = 1 \ldots U$.

The set of acceptable CSVs is denoted by C_A. Note that C_A may be the null set; this occurs when the user requirements cannot be met with the given link capacity and DDPs. If $C_A \neq \emptyset$ and $\mathbf{c} \in C_A$, we say that the user population $\mathcal{U}$ is *satisfied with* $\mathbf{c}$, or simply *satisfiable*. Equivalently, the link $\mathcal{L}$ is said to be *well-provisioned* for $\mathcal{U}$. If $C_A = \emptyset$, the user population is said to be *unsatisfiable*, and the link is said to be *under-provisioned* for $\mathcal{U}$.

The following result expresses an important property of the PDD model. When one ore more users move to a higher class, the delay of all classes increases. The class delays decrease, on the other hand, when one or more users move to a lower class[1].

Proposition 1: If $\mathbf{c}$ and $\mathbf{c}'$ are two different CSVs such that $\mathbf{c} \ge \mathbf{c}'$, the average delay in each class i is $\bar{d}_i(\mathbf{c}') \le \bar{d}_i(\mathbf{c})$.

A second important property of the PDD model is that when a user moves from one class to another, while the rest of the users stay in the same class, the user observes a consistent class ordering, i.e., the higher class provides a lower delay. In the following, the notation $\mathbf{c}' = \mathbf{c})_j^i$ means that the CSV $\mathbf{c}'$ is identical to $\mathbf{c}$, except that the j'th entry of $\mathbf{c}$ is replaced with class i ($j \in \{1, \ldots, U\}$ and $i \in \{1, \ldots, N\}$).

Proposition 2: Suppose that $\mathbf{c}' = \mathbf{c})_j^k$ with $k \in \{1, \ldots, N\}$. If $k > c_j$ then $\bar{d}_k(\mathbf{c}') \le \bar{d}_{c_j}(\mathbf{c})$. Similarly, if $k < c_j$ then $\bar{d}_k(\mathbf{c}') \ge \bar{d}_{c_j}(\mathbf{c})$.

A. The well-provisioned case

First consider the case when $\mathcal{U}$ is *satisfiable*. We start with some important properties for the set of acceptable CSVs C_A. *A CSV is said to be ordered when users with more stringent delay requirements select higher classes.* The following property shows that an acceptable CSV $\mathbf{c}$ can be always replaced with a lower acceptable CSV $\mathbf{c}' \le \mathbf{c}$ that is *ordered*. As shown in the proof of this property, the CSV $\mathbf{c}'$ is such that $c'_j = \min_{k=j,\ldots,U}\{c_k\}$ for $j = 1, \ldots, U$.

Proposition 3: Given an acceptable CSV $\mathbf{c}$ we can always construct another acceptable CSV $\mathbf{c}' \le \mathbf{c}$, such that $c'_i \le c'_j$ for any $i < j$ ($i, j \in \{1, \ldots U\}$).

The following property shows that given two acceptable CSVs, we can construct another acceptable CSV in which each user selects the minimum between the two acceptable classes for that user. For example, if (1,2,2,3,4) and (1,1,3,4,4) are two acceptable CSVs, then the CSV (1,1,2,3,4) is also acceptable. To express this 'per-user minimum class' operation, we write $\mathbf{c}^m = \min\{\mathbf{c}^1, \mathbf{c}^2\}$, or more generally, $\mathbf{c}^m = \min\{\mathbf{c}^1, \ldots, \mathbf{c}^k\}$.

Proposition 4: Suppose that $\mathbf{c}^1$ and $\mathbf{c}^2$ are two acceptable CSVs. The CSV $\mathbf{c}^m$, with $c_j^m = \min(c_j^1, c_j^2)$ ($j = 1, \ldots, U$), is also acceptable.

A.0.a Feasibility test and minimum acceptable CSV:. We can now examine whether there exists a CSV in which all users are satisfied, i.e., whether the link $\mathcal{L}$ is *well-provisioned* for $\mathcal{U}$ ($C_A \neq \emptyset$). If the link is well-provisioned, what is the

[1] The proofs of the results in this section can be found in [17].

```
min_CSV (c_1, c_2, ..., c_U)
{
// c = (c_1, c_2, ..., c_U).
// Initially, call min_CSV (1, 1, ..., 1).

    compute_class_rates λ(c); // From (3).
    compute_class_delays d̄(c); // From (2).

    if (c ∈ C_A) // (i.e., d̄_{c_j}(c) ≤ φ_j for all j ∈ U)
        return (ĉ = c);
    else {
        k = max_{j=1...U} j such that c_j < N;
        if (k == 1 and c_1==N-1)
            return (U: Unsatisfiable);
        else if (k == 1)
        min_CSV (c_1 + 1, c_1 + 1, ..., c_1 + 1, c_1 + 2);
        else // (c_1==N-1)
        min_CSV (c_1, c_2, ..., c_{k-1}, c_k + 1, ..., c_k + 1);
    }
}
```

Fig. 1. Algorithm to compute the minimum acceptable CSV $\hat{\mathbf{c}}$.

CSV with the minimum acceptable class for each user? Such a CSV would be optimal for the user population, because all users would be satisfied, and with the minimum cost for each user. Let $\hat{\mathbf{c}}$ be such an optimal CSV. Formally, $\hat{\mathbf{c}}$ is defined as

$$\hat{\mathbf{c}} = \min_{\mathbf{c} \in C_A} \{\mathbf{c}\} \tag{4}$$

Based on Proposition 4, $\hat{\mathbf{c}}$ is also acceptable, and by definition, unique. We refer to $\hat{\mathbf{c}}$ as the *minimum acceptable CSV*.

A bruteforce approach to compute $\hat{\mathbf{c}}$ is to search through all CSVs. A more efficient algorithm is shown in Figure 1. The algorithm determines the minimum acceptable CSV $\hat{\mathbf{c}}$, given the rates and delay requirements of the users in $\mathcal{U}$, the average aggregate backlog $\bar{q}_{ag}$ (which is an invariant given $\mathcal{U}$ and $\mathcal{L}$), and the DDPs. The algorithm generates *the sequence of ordered CSVs in increasing order*[2]. For each CSV, it is examined whether it is acceptable. If this is the case, we can show (based on Proposition 3) that this is the minimum acceptable CSV $\hat{\mathbf{c}}$, and the algorithm terminates. If there is no acceptable CSV in the sequence of ordered CSVs, then the user population $\mathcal{U}$ is unsatisfiable, and the link $\mathcal{L}$ is *under-provisioned* for $\mathcal{U}$ ($C_A = \emptyset$). CSVs of the form $(c_i, c_i, \ldots, c_i)$ with $c_i \neq 1$ are not examined, because if they are acceptable, then the lower CSV $(1, 1, \ldots, 1)$ is also acceptable.

Note that whether link $\mathcal{L}$ is well-provisioned for the user population $\mathcal{U}$ depends on the user rates $\{r_i\}$, the user delay requirements $\{\phi_i\}$, the specified DDPs $\{\delta_i\}$, and the average aggregate backlog $\bar{q}_{ag}$. The latter depends on the traffic burstiness and the link utilization u. The 'inverse' problem is to determine the optimal DDPs and the minimum link capacity (or the maximum utilization) that can satisfy a user population $\mathcal{U}$; we study this *class provisioning problem* in [14].

A.0.b Distributed DCS model:. The algorithm of Figure 1 computes the minimum acceptable CSV in a centralized manner. In practice, users would act independently to select the minimum class that satisfies their delay requirement, without knowing the class selections and delay requirements of other users. What is the resulting CSV in this *distributed DCS model*?

In the distributed DCS model, users perform the following *class transitions*. A user j is supposed to only know the queueing delay $\bar{d}_{c_j} = \bar{d}_{c_j}(\mathbf{c})$ in the class c_j that she uses. If $\bar{d}_{c_j} > \phi_j$ and $c_j < N$ the user moves to the higher class $c_j + 1$, expecting to get a lower average delay (Proposition 2). If the user is already in the highest class N and $\bar{d}_{c_j} > \phi_j$, the user stays unsatisfied in class N. Also, if $\bar{d}_{c_j} \leq \phi_j$ and $c_j > 1$, the user moves to the lower class $c_j - 1$, in order to examine whether the higher delay of that class is also acceptable (Proposition 2). If class $c_j - 1$ is not acceptable, the user returns to class c_j. Note that the occasional transitions to a lower class are necessary in order for users to search for the *minimum* acceptable class.

It is important to note that in this distributed DCS model *a user does not stay in an acceptable class indefinitely*. So, strictly speaking, *the user population does not converge to a certain CSV*, even when $\mathcal{U}$ is satisfiable. We can define, though, a *distributed DCS equilibrium* $\tilde{\mathbf{c}}$ as a CSV such that, first, all unsatisfied users are in the highest class, and second, when a satisfied user moves to the lower class, while all other users remain in the same class, that user becomes unsatisfied. Formally, let $\mathcal{U}_s(\mathbf{c})$ and $\mathcal{U}_u(\mathbf{c})$, respectively, be the set of satisfied and unsatisfied users for a CSV $\mathbf{c}$. We say that a CSV $\tilde{\mathbf{c}}$ is a distributed DCS equilibrium if it meets the following two conditions:

$$\tilde{c}_j = N \quad \text{for all } j \in \mathcal{U}_u(\tilde{\mathbf{c}}) \tag{5}$$

$$\forall j \in \mathcal{U}_s(\tilde{\mathbf{c}}) \ (\text{with } \tilde{c}_j > 1), j \notin \mathcal{U}_s(\mathbf{c}') \text{ where } \mathbf{c}' = \tilde{\mathbf{c}}\rangle_j^{\tilde{c}_j - 1} \tag{6}$$

If $\tilde{\mathbf{c}} \in C_A$, we say that it is an *acceptable DCS equilibrium*; otherwise, it is an *unacceptable DCS equilibrium*.

Ren and Park considered a game theoretic model of DCS in [18]. Specifically, [18] showed that *the users converge to a distributed DCS equilibrium (Nash equilibrium), under either sequential or concurrent class transitions*, when three 'per-hop control' properties are met. The PDD model satisfies these properties because of Equation 1, Proposition 1,

[2] By increasing order, we mean that if $\mathbf{c}^k$ is generated before $\mathbf{c}^l$, $\mathbf{c}^k \leq \mathbf{c}^l$.

and Proposition 2. So, the results of [18] regarding the existence and stability of the DCS equilibria are applicable to our PDD-based link model. The major result of [18] is that there can be no *persistent cycles* in a sequence of CSVs that results from DCS class transitions. Cycles can occur with concurrent class transitions, but they are only transient, in the sense that from any CSV on the cycle there exist class transitions (sequential or concurrent) that lead to a distributed DCS equilibrium.

It is easy to see that the minimum acceptable CSV $\hat{\mathbf{c}}$ is an acceptable DCS equilibrium. So, if there is only one DCS equilibrium, it has to be $\hat{\mathbf{c}}$. Additionally, it can be shown that if all users start from the lowest class, i.e., if the initial CSV is $\mathbf{c} = (1, 1, \ldots, 1)$, then the resulting DCS equilibrium is $\hat{\mathbf{c}}$. Other DCS equilibria can also exist however. For example, consider the case of two users and two classes in a well-provisioned link. Suppose that the minimum acceptable CSV is $\hat{\mathbf{c}} = (1, 1)$, and that the CSVs $(1, 2)$ and $(2, 1)$ are unacceptable. Since $(1, 1)$ is acceptable, the CSV $(2, 2)$ is also acceptable (the users encounter the same average delays in both CSVs). Consequently, the CSV $(2, 2)$ is also an acceptable DCS equilibrium. Such acceptable equilibria are suboptimal for the users, because the users need to pay for a higher class than the minimum acceptable class that exists for them. On the other hand, such equilibria may be more preferable for the network provider, since they can lead to higher network revenues. The resulting DCS equilibria can also be unacceptable, even when the link is well-provisioned. For example, consider a well-provisioned link with three users and three classes, and let $\hat{\mathbf{c}} = (1, 2, 2)$. It is possible that the CSVs $(1, 2, 3)$, $(1, 3, 2)$, and $(1, 3, 3)$ are unacceptable. In that case, the CSV $(1, 3, 3)$ is an unacceptable DCS equilibrium, even though the link is well-provisioned. Unacceptable DCS equilibria are of course suboptimal both for the users and the network provider.

B. The under-provisioned case

Suppose now that $\mathcal{U}$ is unsatisfiable ($C_A = \emptyset$). By definition, there are some users that cannot meet their delay requirement in any class. Based on the distributed DCS model of the previous paragraph, these users remain unsatisfied in the highest class. As in the well-provisioned case, it can be shown that the population of users converges to a distributed DCS equilibrium, that is always unacceptable in this case. The following result shows that, in the under-provisioned case, a distributed DCS equilibrium $\tilde{\mathbf{c}}$ is such that the unsatisfied users have the most stringent (i.e., smallest) delay requirements.

Proposition 5: If $\mathcal{U}$ is unsatisfiable, a distributed DCS equilibrium $\tilde{\mathbf{c}}$ is always of the following form, with S being the number of satisfied users ($0 \leq S < U$) in $\tilde{\mathbf{c}}$

$$\tilde{\mathbf{c}} = (c_1, \ldots, c_S, N, \ldots, N) \tag{7}$$

So, when $\mathcal{U}$ is unsatisfiable, S users with the largest delay requirements are satisfied, while the rest $U - S$ users with the smallest delay requirements are unsatisfied in the highest class. The maximum value of S and the minimum acceptable class for those S satisfied users can be determined using a centralized algorithm that is similar to the algorithm of Figure 1 (see [19]).

In practice, the unsatisfied users may leave the network, or change their rates and/or delays requirement. Such user behavior leads to a network load relaxation, and to a new user population $\mathcal{U}'$ that may be satisfiable. Unsatisfied users, in practice, also introduce some cost for the network provider (in the form of loosing customers for example).

IV. Simulation study of an end-to-end DCS algorithm

The model of Section III considers only a single link, and assumes that users accurately know the average queueing delay in the class that they use. Also, the model does not specify the defining timescales for the class average delays and the period in which users adjust their class selections. In this section, we focus on an end-to-end DCS algorithm that takes into account all these issues. The behavior of this DCS algorithm in a multi-hop network that is loaded with heavy tailed cross-traffic is studied with simulations.

A complete DCS algorithm: In the following DCS algorithm, the user specifies a requirement D_{max} on the maximum Round-Trip Delay (RTD) in the flow's path. The sender timestamps each packet k before transmitting it, while the receiver returns the timestamps back to the sender in the same class that they are received in, so that the sender can measure the RTD D_k of each packet k. In a bi-directional flow, such as a telephony session, the flow of timestamps back to the sender can be piggybacked with the reverse data flow. The minimum RTD D_{min} is also measured, in order to estimate the total propagation and transmission delays in the path; obviously it must be that $D_{max} > D_{min}$. After the k'th timestamp is received, the DCS sender estimates a *short-term average RTD* $\tilde{D}$ using an exponential running-average of the form $\tilde{D}_{k+1} = (1 - w)\tilde{D}_k + wD_k$ where $0 < w \leq 1$ is the averaging weight. $\tilde{D}$ expresses the performance timescales that the user is interested in. For instance, if $w = 1$ the user cares about *per-packet RTDs*, which may be the case in a highly interactive application. If $w = 0.01$, the last 500 RTDs contribute 99% to $\tilde{D}_k$, while the last 100 RTDs contribute 63% to $\tilde{D}_k$. In the following experiments w is set to 0.01, except one experiment which considers per-packet RTD constraints (w=1).

The Dynamic Class Selection (DCS) algorithm is shown in Figure 2. The class that the user selects is denoted by c, with $c \in \{1, \ldots, N\}$. When the measured RTD D_k is larger than D_{max}, the class selection is increased. This decision is based on the last RTD D_k, instead of the average

DCS algorithm:
{
D_k= k'th RTD measurement;
if $(D_k > D_{max})$ {
$c = \min\{c + \lceil \log_2 \frac{D_k - D_{min}}{D_{max} - D_{min}} \rceil, N\}$;
Do not increase class in next $f_I D_k$ time units;
}
if $(D_k < \kappa D_{max})$ {
$c = \max\{c - 1, 0\}$;
Do not decrease class in next $f_D D_k$ time units;
}
}

Fig. 2. Simulated DCS algorithm.

$\tilde{D}_k$, in order to react faster to excessive delays in the current class. The class increase is based on the deviation of the measured queueing delay $D_k - D_{min}$ from the maximum allowed queueing delay $D_{max} - D_{min}$. In this particular case, the DCS algorithm assumes that the DDPs in each hop are ratioed as $\delta_1/\delta_i = 2^{i-1}$, and so the appropriate class increase is given by the logarithmic formula shown. The DDPs can be estimated measuring the queueing delay ratios between successive classes. A DDP-based class increase can move the user to the appropriate class faster. We emphasize, though, that *it is not required that the sender knows the actual DDPs in the network*, and that a simpler class increase formula can be used instead. After the class increase, the sender waits for some time $f_I D_k$ before a further class increase. This *adaptation delay* is needed in order to measure the resulting RTD in the new class selection. A typical value for the adaptation delay factor would be f_I=1 (i.e., wait for one RTD), but a more delay-sensitive user can set f_I to less than one assuming that the class increase will cause a lower RTD. Unless stated otherwise, f_I=1.

The class is decreased, on the other hand, when the RTD is lower than κD_{max} ($\kappa < 1$). κ is a *tolerance factor* for the maximum RTD that triggers a class decrease. A user that is not so interested to minimize the cost of her flow can reduce κ. The class is only decremented by one, since the reaction latency is not as critical in the class decrease as in the class increase case. A class decrease is also followed by an adaptation delay $f_D D_k$. Cost-sensitive users can set f_D close to one, while delay-sensitive users can set f_D to a higher value. Unless stated otherwise, $f_D = 4$ and κ=0.9. Note that when $\kappa < 1$, it is possible that a user settles in a class that is not the minimum acceptable class. This is a trade-off for the user, since a lower κ reduces the 'annoyance' of lower class transitions, but it can also lead to a suboptimal class.

Simulation topology and parameters: Figure 3 shows the

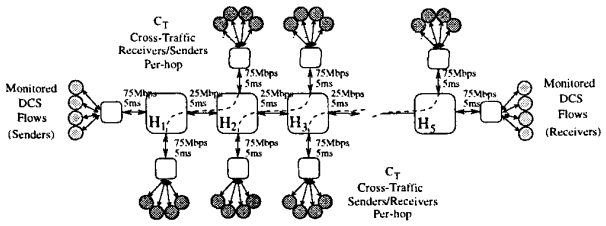


Fig. 3. Simulation topology.

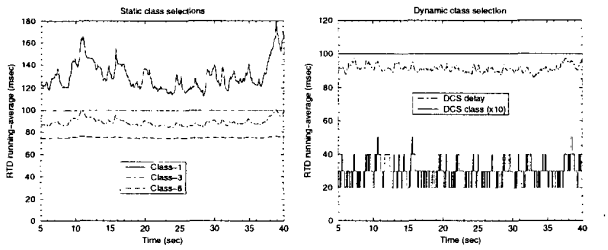
multi-hop simulation topology. The *monitored* DCS flows go through five backbone hops, while Cross-Traffic (CT) is generated from bi-directional flows that go through the topology in the vertical direction. C_T=50 such flows are created in each hop. The link capacities and propagation delays are given in Figure 3. Note that the monitored DCS flows have a minimum RTD D_{min}=70 msec due to propagation delays. The class adjustments are performed at the DCS nodes labeled as 'Senders'. The CT sources are either DCS flows themselves, or they generate packets with a given, average class load distribution (non-DCS flows). Both directions in the backbone path are equally loaded from CT sources. All traffic sources generate 500-byte packets, based on a Pareto distribution with infinite variance (α=1.5). The monitored DCS flows have an average rate of 400kbps, or 100 packets per second. The rate of the CT sources is adjusted to cause a certain utilization u in the backbone links; u is set to 90% in the following experiments.

The backbone links use the WTP scheduler [15], [20], to provide proportional delay differentiation[3]. The network offers N=8 classes with DDPs $\delta_1/\delta_i = \{1, 2, 4, 8, 12, 16, 24, 32\}$ ($i = 1 \ldots 8$), unless stated otherwise. Notice that these DDPs are not entirely consistent with the 'power-of-two' DDPs that the algorithm of Figure 2 assumes. When the CT is created from non-DCS flows, the average class load distribution is (10,20,20,15,15,10,5,5)[4]. When the CT is created from DCS flows, the class load distribution is determined from the delay requirements of those flows. The number of buffers in the links is adequately large to avoid losses.

Performance metrics: The performance of a DCS flow is measured with the fraction $\mathcal{P}$ of RTD values $\tilde{D}_k$ that are lower than the specified maximum RTD D_{max}. For K RTD values, $\mathcal{P} = \frac{\sum_k^K I(D_{max} - \tilde{D}_k)}{K}$ where $I(x)$ is zero if $x < 0$ and one otherwise. We refer to $\mathcal{P}$ as the *acceptable delay ratio*. Note that the acceptable delay ratio is based on the running average $\tilde{D}_k$, and not on the individual RTD measurements, because $\tilde{D}_k$ expresses the performance timescales that the user cares about. The cost of a DCS flow is mea-

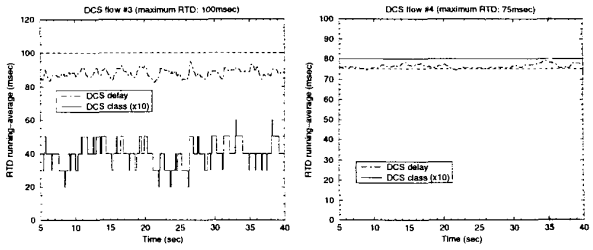
[3] In the high utilization range that we simulate here, the WTP scheduler can achieve the PDD model quite accurately (see [15]).

[4] Each CT source generates traffic with this class load distribution.



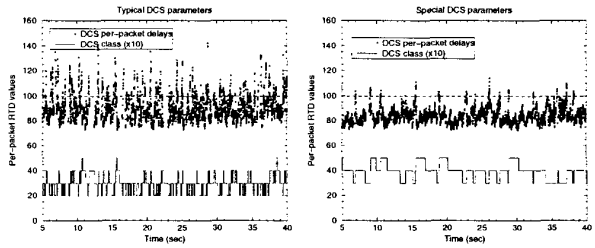
(a) Three static class selections (b) Dynamic class selection

Fig. 4. Static versus dynamic class selection for a flow with D_{max}=100msec.



(a) Flow-3: D_{max}=100msec (b) Flow-4: D_{max}=75msec

Fig. 5. A satisfied and an unsatisfied DCS flow.



(a) $f_I = 1, f_D = 4, \kappa = 0.9$ (b) $f_I = 0.9, f_D = 20, \kappa = 0.8$

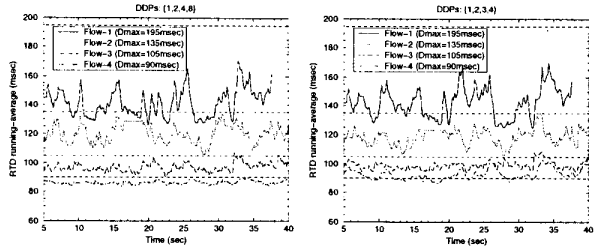
Fig. 6. Controlling the DCS parameters to meet a per-packet RTD requirement.

sured with the *average class* metric $\mathcal{C}$. If the k'th packet was sent in class c_k, the average class of a DCS flow that transferred K packets is $\mathcal{C} = \frac{\sum_k c_k}{K}$ A lower value of $\mathcal{C}$ causes a lower cost for the user, since higher classes are assumed to be more expensive (in a monetary or other sense). The DCS framework introduces a trade-off between the performance and cost of a flow. Higher classes lead to lower delays, but they also cost more. The objective of the DCS algorithm is to achieve a high acceptable delay ratio $\mathcal{P}$ and a low average class $\mathcal{C}$. The exact point in this trade-off can be chosen from the user, based on her performance/cost sensitivity. The algorithm of Figure 2 allows the control of this trade-off, through the parameters f_I, f_D, and κ.

Static versus dynamic class selection: We first compare the DCS algorithm with a simple static class selection scheme. A monitored flow has a maximum RTD requirement D_{max}=100msec. Figure 4-a shows a sample path of the RTD average $\bar{D}$ in the case of three static class selections for that flow. Class-1 provides excessive delays, and the acceptable delay ratio is only $\mathcal{P}$=0.12. Class-8, on the other extreme, leads to much lower delays than needed, and $\mathcal{P}$=1.00. Class-3 turns out to be the minimum class that leads to an acceptable RTD for almost all packets ($\mathcal{P}$=0.99). So, Class-3 is the minimum acceptable class for this flow. Figure 4-b shows what happens when the monitored flow uses DCS instead of a static class selection. The class selection variations are also shown (scaled by a factor of ten). Note that DCS meets the specified RTD constraint, and the acceptable delay ratio is $\mathcal{P}$=1.00. The average class metric is $\mathcal{C}$=2.92, and so the flow uses mainly Class-3, at least in an average sense. This is also shown from the class selection variations in the graph. In other words, *in the resulting DCS equilibrium of this experiment, the DCS flow selects (in an average sense) the minimum acceptable class for that flow*, namely Class-3. The oscillations around Class-3 are caused by attempts to find a lower acceptable class, and by some excessive RTDs, due to traffic bursts, that cause temporary class increases.

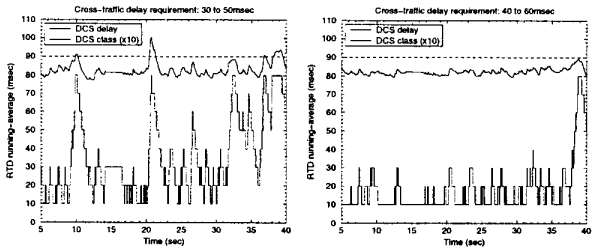
Satisfied and unsatisfied DCS flows: In this experiment, we focus on four monitored DCS flows with diverse maximum RTD requirements. The value of D_{max} for these flows is 300, 150, 100, and 75msec. We only show $\bar{D}$ for flows 3 and 4. The first three flows find a class in which the acceptable delay ratio is $\mathcal{P}$=1.00, and so they are *satisfied*. The average class for these flows is $\mathcal{C}$=1.06, 2.13, and 4.17, respectively. The fourth flow, on the other hand, requires that D_{max}=75msec, which is only 5msec more than the propagation delays. The flow moves to Class-8, but still cannot meet its RTD requirement. The acceptable delay ratio for this flow is only $\mathcal{C}$=0.13, which implies that it is *unsatisfied*. In other words, *the unsatisfied users, if they exist, are flows with the most stringent requirements, they move to the highest class, and they remain unsatisfied there. The satisfied users, on the other hand, converge to an acceptable class, oscillating around that DCS equilibrium.*

The performance versus cost tradeoff in DCS: To illustrate the performance versus cost trade-off, consider a flow that has a stringent requirement D_{max}=100msec on the *individual RTDs of each packet*. Such a constraint means that



(a) Well provisioned DDPs (b) Poorly provisioned DDPs

Fig. 7. The effect of the DDPs on DCS flows.



(a) More demanding CT flows (b) Less demanding CT flows

Fig. 8. The effect of the CT delay requirements on a DCS flow.

w=1 in the RTD estimator. Figure 6-a shows a sample path of the per-packet RTDs with the 'typical' DCS parameters that we mentioned earlier (f_I=1, f_D=4, κ=0.9). The acceptable delay ratio is only $\mathcal{P}$=0.81, and the average class is $\mathcal{C}$=2.92. Violating the RTD bound in about 20% of the packets would probably be unacceptable for several interactive applications. Figure 6-b shows the resulting sample path of per-packet RTDs, when the DCS parameters are selected as f_I=0.9, f_D=20, and κ=0.8. The acceptable delay ratio now is significantly improved to $\mathcal{P}$=0.97, and only 3% of the packets miss their deadlines. The average class $\mathcal{C}$ is increased from 2.92 to 4.21, though, which shows the price that has to be paid for the more stringent QoS. This experiment illustrates two important points. First, *the DCS framework can be used to meet absolute RTD requirements not only in terms of short-term averages, but also for individual packets.* Second, *meeting a more strict performance requirement requires the use of higher classes, and thus a larger cost.* A practical DCS algorithm has to provide the flexibility to control this trade-off.

Class provisioning and the selection of DDPs: How is the selection of DDPs important in satisfying DCS flows? Suppose that four DCS monitored flows have a D_{max} requirement of 195, 135, 105, and 90msec, respectively, and the network offers four classes. Consider, first, the case that $\delta_1/\delta_i = \{1, 2, 4, 8\}$ $(i = 1 \ldots 4)$. Figure 7-a shows that with these DDPs each DCS flow gets an almost perfect acceptable delay ratio $\mathcal{P} \approx 1.00$. The average class $\mathcal{C}$ for the four flows is 1.28, 1.97, 2,87, and 3.59, respectively. The given DDPs in this experiment *were chosen based on the queueing delay requirements of the DCS flows.* To see how, note that the minimum RTD in the path (due to propagation and transmission delays) is about D_{min}=75msec. So, the four flows can tolerate up to 120, 60, 30, and 15msec of queueing delays in the round-trip path. These maximum queueing delays are ratioed as: 120/15=8, 60/15=4, and 30/15=2. So, the DDPs in this case are ratioed based on the corresponding ratios of the queueing delay requirements of the DCS flows.

Figure 7-b, on the other hand, shows what happens if the network operator chooses the DDPs $\delta_1/\delta_i = \{1, 2, 3, 4\}$ $(i = 1 \ldots 4)$. These DDPs do not match well the queueing delay requirements of the DCS flows. Specifically, these DDPs only provide a maximum delay ratio of four between the highest and the lowest class, while the DCS flows require a maximum ratio of eight. The result of this mismatch is that the flow with the tightest delay requirement (Flow-4) remains unsatisfied in the highest class with an acceptable delay ratio $\mathcal{P}$=0.29. It is important to note that the network utilization is the same in both cases; the only difference is the specified DDPs. This experiment illustrates the importance of selecting DDPs that are appropriate for the delay requirements of DCS flows. The problem of computing the optimal DDPs and the minimum link capacity that are required for a certain user population is studied in [14].

The effect of the Cross-Traffic (CT) performance requirements: In the experiment of this paragraph, *all sources, both monitored and CT, generate DCS flows.* In this scenario, an important parameter is the delay requirements of the CT flows. Since they also perform DCS, the more stringent delay requirements they have, the higher classes they use, and so the harder it becomes for any DCS flow to find an acceptable class. Figure 8 shows what happens to a monitored DCS flow with a maximum RTD requirement D_{max}=90msec, in two different cases of CT delay requirements. In Figure 8-a, the CT flows ask for a maximum RTD that is uniformly distributed in the range 30-50msec. Note that the minimum RTD in the CT path is about 30msec. So, some of the CT flows in this case ask for practically zero queueing delays. Such a demanding CT load causes a relatively low acceptable delay ratio to the monitored flow ($\mathcal{P}$=0.86), while its average class is $\mathcal{C}$=3.69. In Figure 8-b, on the other hand, the CT flows are less demanding, asking for a maximum RTD in the range 40-70msec. Even though the aggregate load and the DDPs remain the same, the monitored DCS flow is able now to get a perfect acceptable delay ratio ($\mathcal{P}$=1.00) with a lower cost ($\mathcal{C}$=1.94).

V. Related work on dynamic class selection

A review of some recent results in the broad area of differentiated services was given in the introduction. The discussion here is limited to previous works in the context of dynamic class selection. A brief investigation of DCS in the context of proportional delay differentiation appeared in [21]. That work considered a class selection algorithm that routers can use in order to provide a maximum delay to each flow. With a continuous-time model, and assuming a continuous range of class choices, [21] showed that when the set of user requirements is feasible, each flow converges to a class that provides the requested delay bound.

Chen and Park considered individual flows with absolute performance requirements in a stateless network [22]. The class selection is formulated as an optimization problem in which the overall resource usage cost is to be minimized, subject to the constraint that the performance requirement of each flow has to be met. Interestingly, there is a distributed algorithm that solves this problem.

Ren and Park considered the DiffServ model of per-hop priority mechanisms and traffic aggregation from individual flows to service classes in [18]. They derived an optimal classifier (minimizing the resource usage in a mean-square sense) for mapping a flow to a class, subject to each flow's performance requirements. The model and results of [18] are based on game theory. Even though the DCS problem can be formulated as a non-cooperative game, with users that act independently and selfishly to meet their requirements given a finite resource, we chose in this work to take a different approach that uses basic queueing concepts. The underlying ideas, though, are common in both approaches. For instance, the DCS equilibria are called *Nash equilibria* in game theoretic terms, while the minimum acceptable CSV $\hat{c}$ can be shown to be Pareto and system optimal for the user population.

VI. Conclusions

The strongest point about relative differentiation is its simplicity. Its major drawback, on the other hand, is that it does not provide users with absolute QoS. This work has demonstrated that if the end-points dynamically search for an acceptable class, the per-hop relative differentiation that the network offers can lead to absolute and end-to-end services under certain conditions. DCS should be also evaluated in the context of other performance requirements, such as a maximum loss rate or a minimum TCP throughput. The dynamic properties of the proportional differentiation model (stated in Propositions 1 and 2 for average class delays) also hold for those performance measures, and so we expect a similar DCS behavior. Finally, an important extension would be to combine DCS with some kind of end-point admission control, such as the scheme proposed in [23], so that the network is always well-provisioned for the population of admitted users.

References

[1] S. Blake, D.Black, M.Carlson, E.Davies, Z.Wang, and W.Weiss, *An Architecture for Differentiated Services*, Dec. 1998. IETF RFC 2475.

[2] V. Jacobson, K. Nichols, and K.Poduri, *An Expedited Forwarding PHB*, June 1999. RFC 2598.

[3] A. Charny and J. L. Boudec, "Delay Bounds in a Network with Aggregate Scheduling," in *Proceedings QOFIS*, Oct. 2000.

[4] R. Guerin and V.Pla, "Aggregation and Conformance in Differentiated Service Networks: A Case Study," in *Proceedings ITC Specialist Seminar on IP Traffic Modeling, Measurement, and Management*, Sept. 2000.

[5] D. D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delivery Service," *IEEE/ACM Transactions on Networking*, vol. 6, pp. 362–373, Aug. 1998.

[6] I. Stoika and H.Zhang, "LIRA: An Approach for Service Differentiation in the Internet," in *Proceedings NOSSDAV*, 1998.

[7] S. Sahu, P.Nain, D.Towsley, C.Diot, and V.Firoiu, "On Achievable Service Differentiation with Token Bucket Marking for TCP," in *Proceedings of ACM SIGMETRICS*, June 2000.

[8] I. Stoika and H.Zhang, "Providing Guaranteed Services Without Per Flow Management," in *Proceedings of ACM SIGCOMM*, Sept. 1999.

[9] C. Cetinkaya and E.W.Knightly, "Egress Admission Control," in *Proceedings of IEEE INFOCOM*, Mar. 2000.

[10] J. L. Boudec, M. Hamdi, L. Blazevic, and P.Thiran, "Asymmetric Best Effort Service for Packet Networks," in *Proceedings Global Internet Symposium*, Dec. 1999.

[11] C. Dovrolis and P.Ramanathan, "A Case for Relative Differentiated Services and the Proportional Differentiation Model," *IEEE Network*, Oct. 1999.

[12] H. Schulzrinne, S.Casner, R.Frederick, and V.Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, Jan. 1996. RFC 1889.

[13] C. Dovrolis and P.Ramanathan, "Proportional Differentiated Services, Part II: Loss Rate Differentiation and Packet Dropping," in *IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, June 2000.

[14] C. Dovrolis and P.Ramanathan, "Class Provisioning in Proportional Differentiated Services Networks," in *Scalability and Traffic Control in IP Networks (SPIE ITCOM304)*, Aug. 2001.

[15] C. Dovrolis, D.Stiliadis, and P.Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," in *Proceedings of ACM SIGCOMM*, Sept. 1999.

[16] G. Bolch, S.Greiner, H.Meer, and K.S.Trivedi, *Queueing Networks and Markov Chains*. John Wiley and Sons, 1999.

[17] C. Dovrolis and P. Ramanathan, "Dynamic Class Selection: From Relative Differentiation to Absolute QoS (extended version)," tech. rep., University of Delaware, Apr. 2001. http://www.cis.udel.edu/~dovrolis/Papers/DCS-01.ps.

[18] H. Ren and K. Park, "Toward a Theory of Differentiated Services," in *Proceedings IWQoS*, June 2000.

[19] C. Dovrolis, *Proportional Differentiated Services for the Internet*. PhD thesis, University of Wisconsin-Madison, Dec. 2000.

[20] M. K. H. Leung, J. Lui, and D. Yau, "Characterization and Performance Evaluation for Proportional Delay Differentiated Services," in *Proceedings International Conference on Network Protocols (ICNP)*, Oct. 2000.

[21] T. Nandagopal, N.Venkitaraman, R.Sivakumar, and V.Bharghavan, "Delay Differentiation and Adaptation in Core Stateless Networks," in *Proceedings of IEEE INFOCOM*, Mar. 2000.

[22] S. Chen and K. Park, "An Architecture for Noncooperative QoS Provision in Many-Switch Systems," in *Proceedings of IEEE INFOCOM*, 1999.

[23] L. Breslau, E. Knightly, S. Shenker, I. Stoika, and H. Zhang, "Endpoint Admission Control: Architectural Issues and Performance," in *Proceedings of ACM SIGCOMM*, 2000.