Dynamic Class Selection and Class Provisioning in Proportional Differentiated Services

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 the relative differentiation architecture is that *users with an absolute QoS requirement can dynamically search for a class which provides the desired QoS level.* In the first part of this paper, we investigate this Dynamic Class Selection (DCS) framework in the context of Proportional Delay Differentiation (PDD). We illustrate that, under certain conditions, DCS-capable users can meet absolute QoS requirements, even though the network only offers relative differentiation. For a simple link model, 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, suboptimal and even unacceptable DCS equilibria may also exist. Simulations of an end-to-end DCS algorithm provide further insight in the dynamic behavior of DCS, show the relation between DCS and the network Delay Differentiation Parameters, and demonstrate how to control the trade-off between a flow's performance and cost using DCS.

In the second part of the paper, we consider the related problem of *class provisioning.* At the provisioning phase, the network manager configures the link to support the QoS requirements of all traffic types. Each traffic type is specified by an expected arrival rate and a delay requirement. The objective of the provisioning phase is to jointly determine: the *minimum link capacity* needed to support the given traffic types, the *nominal class of service* for each traffic type, and the *appropriate resource allocation* between classes. Our class provisioning methodology is also based on Proportional Delay Differentiation (PDD). The major advantage of PDD is that it avoids the computation of an explicit bandwidth share for each class. The class provisioning methodology is illustrated with examples.

This work was supported by a research fund from Cisco Systems Inc. Parts of this paper have previously appeared at IEEE ICNP 2001 and at the SPIE "Scalability and Traffic Control in IP Networks" 2001 conference.

1 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 the first part of this paper is to investigate these questions in the context of the Proportional Delay Differentiation (PDD) model [11]. We first argue that the PDD 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 PDD 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 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 ('well-provisioned' case), and if so, it 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. Unfortunately, suboptimal and even unacceptable 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, are unsatisfied in the highest class. We also 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.

Since the relative differentiation architecture does not use admission control, however, the offered load at a link cannot be controlled. Consequently, it is possible that an application with absolute QoS requirements will not find an acceptable class through DCS. This depends on the amount of forwarding resources at the link (transmission capacity and number of buffers), on the allocation of forwarding resources between classes, and on the volume and performance requirements of the rest of the traffic. As shown in the first part of this paper, if the link is *well-provisioned* then there should be an acceptable class for each traffic type. But what does it exactly mean for the link to be well-provisioned? And how can a network manager perform such provisioning? This is the question that we address in the second part of this paper.

Specifically, we investigate the following instance of the provisioning problem: *how can a network manager provision a link to meet an average delay requirement for each traffic type, requiring the minimum link capacity?* In order to provision a link, the network manager needs a *workload profile*. The workload profile is a specification of the anticipated traffic types in the link, in terms of their arrival rate and average delay requirement. Given this profile, a *class provisioning methodology* has the following outcomes:

1. The nominal service class for each traffic type.

2. The minimum link capacity for the given workload profile.

3. The required capacity allocation between classes.

The class provisioning methodology that we propose is based on Proportional Delay Differentiation (PDD) scheduling [12]. The major advantage of PDD scheduling is that it avoids the computation of an explicit bandwidth share for each class. Instead, PDD scheduling determines the order of packet transmissions in order to meet the $N-1$ ratios of the N target class delays. Having fixed the delay ratios with PDD, we then set the N class delays to their target values adjusting a single knob, which is the link capacity.

The structure of the paper is as follows. Section 2 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 3, while Section 4 summarizes an end-to-end DCS simulation study. Section 5 presents the class provisioning methodology and gives examples of its use. A discussion of related work follows in Section 6. We summarize in Section 7.

2 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) [13]. 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 [14] 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 [15]. As shown in §4, 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 [12] and packet losses [14], 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.

3 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 [12]. Specifically, if $\bar{d_i}$ is the average queueing delay in class i, the PDD model requires that

$$\frac{\bar{d_i}}{\bar{d_j}} = \frac{\delta_i}{\delta_j} \qquad 1 \leq i,j \leq N \tag{1}$$

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.* For the purposes of this section, we assume that all packets have the same size (normalized to one 'data unit'). As shown in [12], 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. $\bar{q}_{ag}$ 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 $\phi_1 \geq \phi_2 \geq \ldots \geq \phi_U > 0$. 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 $\bar{d}_{c_j}(\mathbf{c}) \leq \phi_j$; 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: $\mathbf{c}' \geq \mathbf{c}$ when $c'_j \geq c_j$ 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 $\mathbf{c} \geq \mathbf{c}'$, the average delay in each class i is $\bar{d}_i(\mathbf{c}') \leq \bar{d}_i(\mathbf{c})$.

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 $\mathbf{c}' = \mathbf{c}\rangle_j^i$ 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 $\mathbf{c}' = \mathbf{c}\rangle_j^k$ with $k \in \{1, \ldots, N\}$. If $k > c_j$ then $\bar{d}_k(\mathbf{c}') \leq \bar{d}_{c_j}(\mathbf{c})$. Similarly, if $k < c_j$ then $\bar{d}_k(\mathbf{c}') \geq \bar{d}_{c_j}(\mathbf{c})$.

3.1 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 $\mathbf{c}' \leq \mathbf{c}$ 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 $\mathbf{c}' \leq \mathbf{c}$, such that $c'_i \leq c'_j$ 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.

3.1.1 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 CSV with the minimum acceptable class for each user? Such a CSV would be optimal for the user

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

```
min_CSV $(c_1, c_2, \ldots, c_U)$
{
// $\mathbf{c} = (c_1, c_2, \ldots, c_U)$.
// Initially, call min_CSV $(1, 1, \ldots, 1)$.

    compute_class_rates $\lambda(\mathbf{c})$; // From (3).
    compute_class_delays $\bar{\mathbf{d}}(\mathbf{c})$; // From (2).

    if ($\mathbf{c} \in C_A$) // (i.e., $\bar{d}_{c_j}(\mathbf{c}) \leq \phi_j$ for all $j \in \mathcal{U}$)
        return ($\hat{\mathbf{c}} = \mathbf{c}$);
    else {
        $k = \max_{j=1 \ldots U} j$ such that $c_j < N$;
        if ($k$ == 1 and $c_1$==N-1)
            return ($\mathcal{U}$: Unsatisfiable);
        else if ($k$ == 1)
        min_CSV $(c_1 + 1, c_1 + 1, \ldots, c_1 + 1, c_1 + 2)$;
        else // ($c_1$==N-1)
        min_CSV $(c_1, c_2, \ldots, c_{k-1}, c_k + 1, \ldots, c_k + 1)$;
    }
}
```

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

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.

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

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 [15].

3.1.2 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 [17]. Specifically, [17] 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, and Proposition 2. So, the results of [17] regarding the existence and stability of the DCS equilibria are applicable to our PDD-based link model. The major result of [17] 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.

3.2 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 [18]).

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).

4 Simulation study of an end-to-end DCS algorithm

The model of §3 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

```
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;
    }
}
```

Figure 2: Simulated DCS algorithm.

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 $\tilde{D}_k$, 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

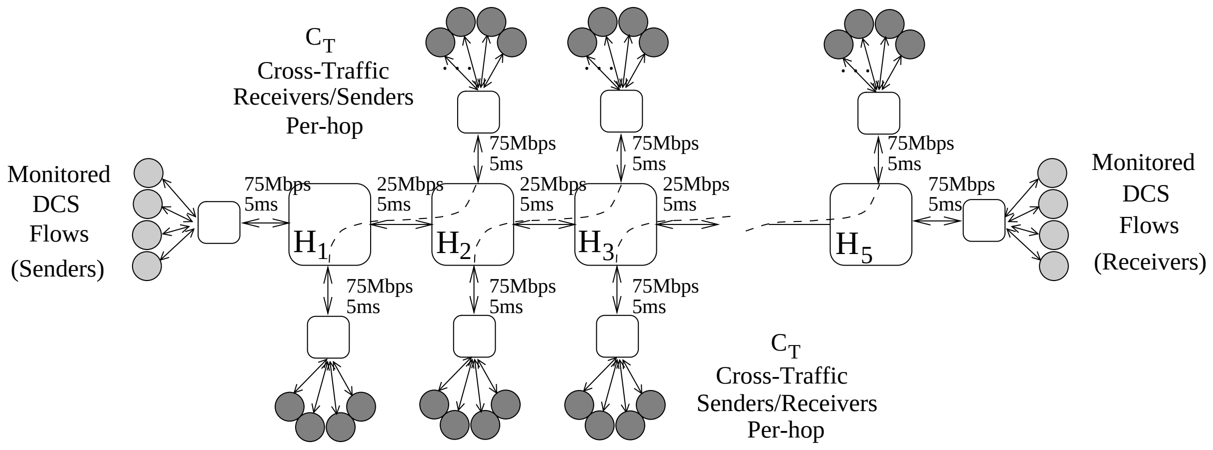


Figure 3: Simulation topology.

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 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.

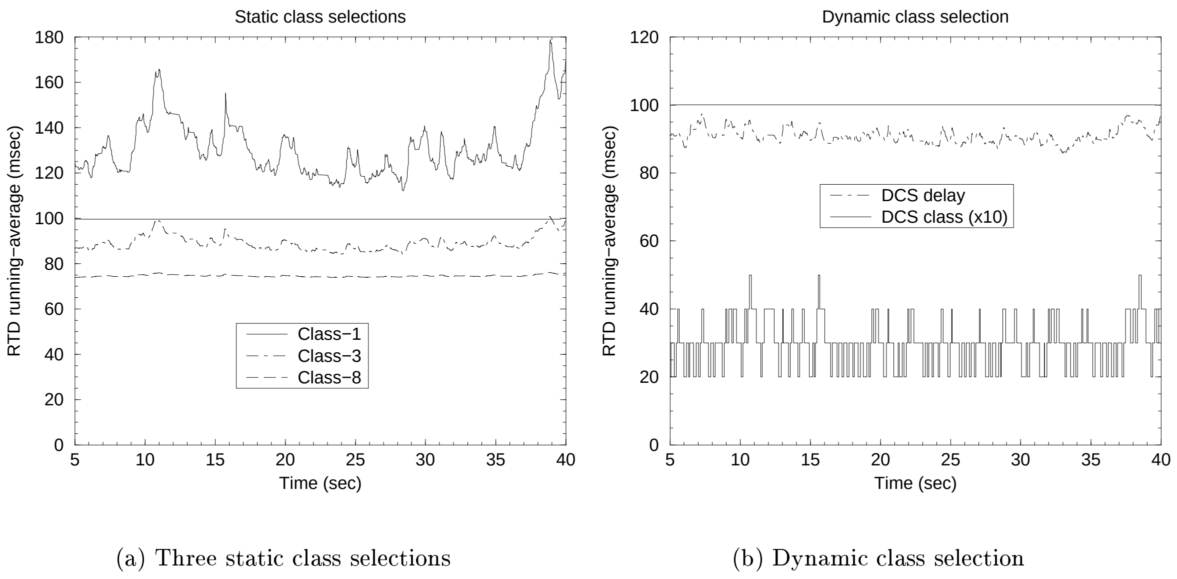
The backbone links use the WTP scheduler [12, 19], to provide proportional delay differentiation[3]. The network offers N=8 classes with DDPs $\delta_1/\delta_i = \{1, 2, 4, 8, 12, 16, 24, 32\}$ $(i = 1 \dots 8)$, unless stated otherwise. Notice that these DDPs are not entirely consistent with the 'power-of-two' 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 measured 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 $\tilde{D}$ 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.

[3]In the high utilization range that we simulate here, the WTP scheduler can achieve the PDD model quite accurately (see [12]).

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



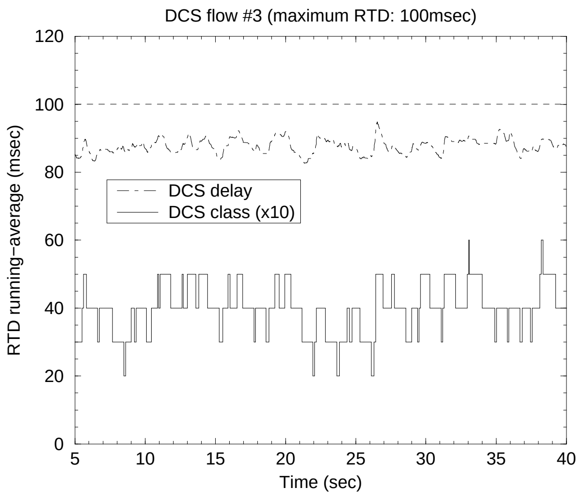
(a) Three static class selections

(b) Dynamic class selection

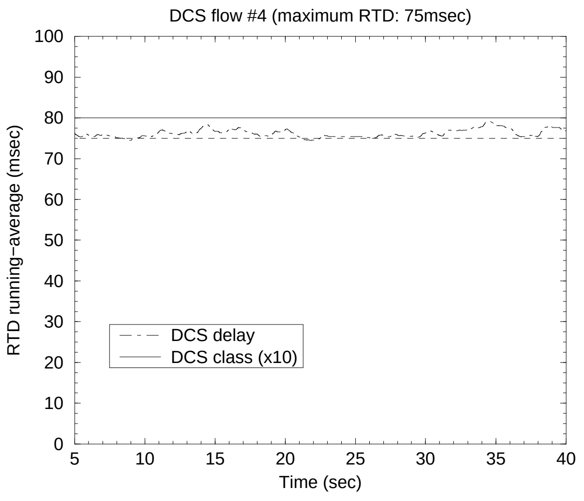
Figure 4: Static versus dynamic class selection for a flow with D_{max}=100msec.

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 $\tilde{D}$ 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 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*

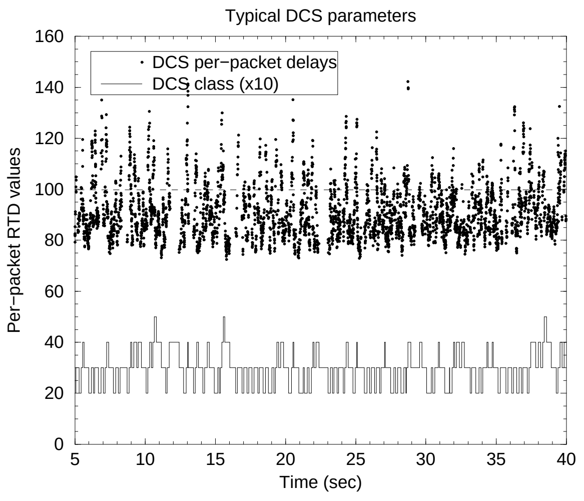


(a) Flow-3: D_{max}=100msec

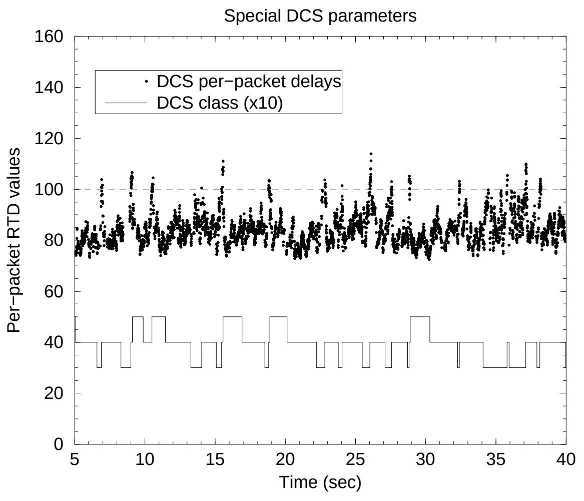


(b) Flow-4: D_{max}=75msec

Figure 5: A satisfied and an unsatisfied DCS flow.

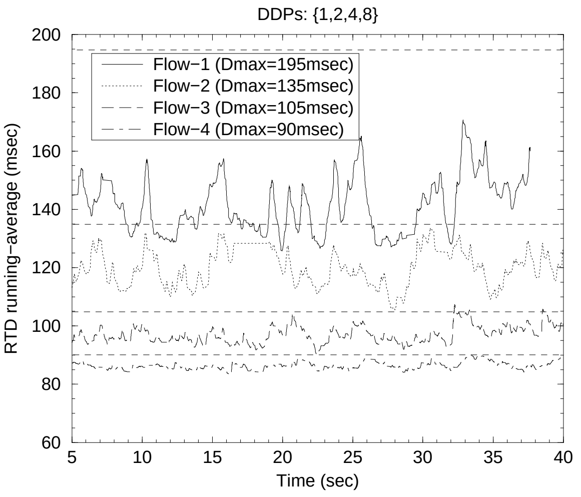


(a) $f_I = 1, f_D = 4, \kappa = 0.9$

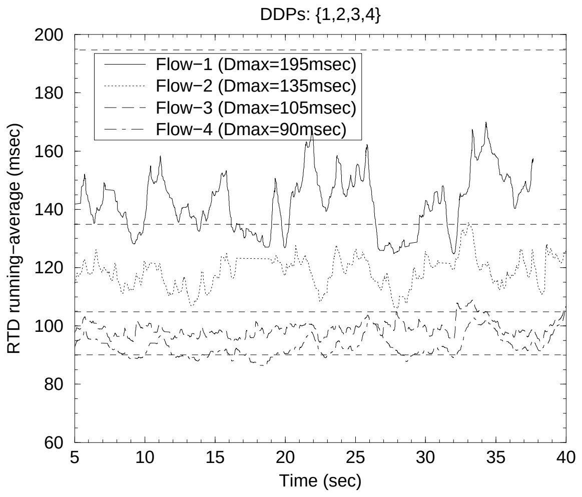


(b) $f_I = 0.9, f_D = 20, \kappa = 0.8$

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



(a) Well provisioned DDPs



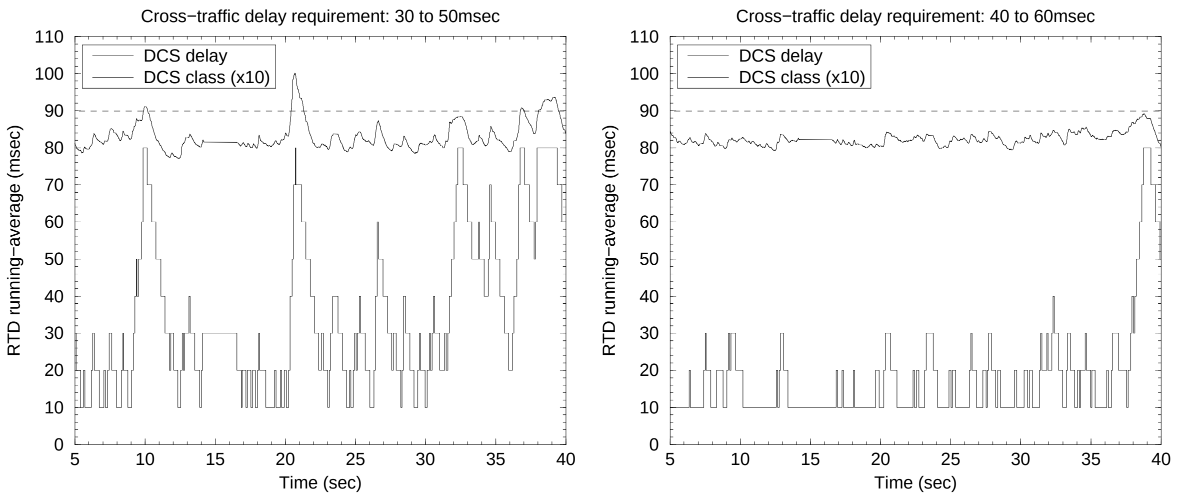
(b) Poorly provisioned DDPs

Figure 7: The effect of the DDPs on DCS flows.

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



(a) More demanding CT flows

(b) Less demanding CT flows

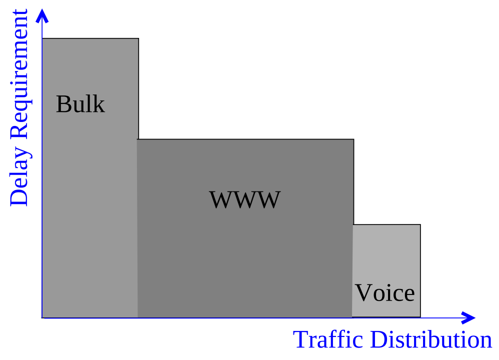
Figure 8: The effect of the CT delay requirements on a DCS flow.

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 the second part of this paper, in §5.

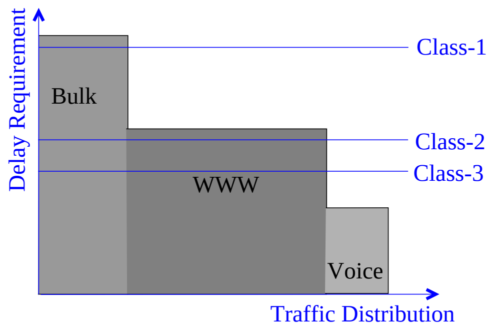
The effect of the Cross-Traffic (CT) performance requirements: In the experiment of this paragraph, *all sources, both monitored and CT, generate DCS flows.* In this scenario, an important parameter is the delay requirements of the CT flows. Since they also perform DCS, the more stringent delay requirements 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).

5 Class provisioning

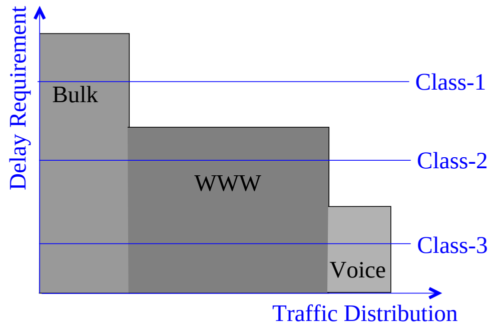
In the provisioning phase, the objective of the network manager is to configure a network link at a *desired operating point.* The ”knobs” that the manager can control are the link forwarding



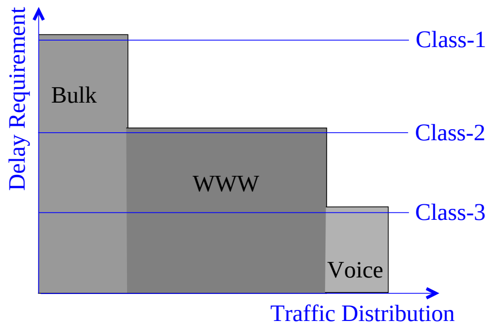
(a) Delay requirement for three traffic types



(b) Under-provisioned differentiation



(c) Over-provisioned differentiation



(d) Well-provisioned differentiation

Figure 9: An under-provisioned, over-provisioned, and well-provisioned link with three classes and three traffic types.

resources, as well as the allocation of these resources between classes. An important issue in the provisioning phase is to use the minimum capacity, especially if the cost of the link increases with its capacity. Even when this is not strictly the case (say in optical networks), the network manager would still be interested to *at least know* the minimum link capacity required.

The desired operating point is determined by the link's workload, i.e., by the traffic types that the link carries. By 'traffic type', we mean an aggregation of flows that have the same performance requirements. In order to perform provisioning, the network manager needs a profile for the link workload. This *workload profile* consists of the offered load and the QoS requirement for each traffic type. Such a profile is often available, based on operational data and statistics, in stable networks that are well monitored.

The exact form of link provisioning that we consider here can meet an average queueing delay requirement for each traffic type. For example, a network provider can provision an average delay of 50msec for the E-mail, Network-News (NNTP), and other 'Bulk' traffic, 20msec for the WWW traffic, and 10 msec for the IP telephony and video conferencing traffic.

To illustrate the importance of class provisioning, let us consider the following simple example. Figure 9-a shows a *delay requirement curve* for three traffic types (Bulk, WWW, and Voice) at a certain link. In this example, about 25% of the link's traffic is Bulk transfers that can tolerate large delays, 60% is WWW flows with moderate delay requirements, and 15% is Voice, having low delay requirements. The link, in this example, offers three classes of service: Class-1, Class-2, and Class-3. It is noted that the number of classes may be different than the number of traffic types, and in practice it is likely that the traffic types will be more than the offered classes.

When the link is *under-provisioned*, one or more traffic types cannot meet their delay requirements even in the highest class of service. In Figure 9-b, Voice does not get an acceptable delay even in Class-3. A link can be under-provisioned either because it does not have an adequate amount of forwarding resources (capacity), or because the differentiation between classes (i.e., the delay spacing in this case) is not appropriately configured.

When the link is *over-provisioned*, an acceptable class exists for each traffic type, but the link may operate with more than the minimum required capacity. In Figure 9-c, Bulk meets its requirement in Class-1, WWW in Class-2, and Voice in Class-3. Notice, however, that each class offers a much lower delay that what the corresponding traffic type needs.

Finally, when the link is *well-provisioned*, an acceptable class exists for each traffic type, and additionally, the link operates with the minimum required capacity. In Figure 9-d, the link is well-provisioned when Bulk uses Class-1, WWW uses Class-2, and Voice uses Class-3. Such a *nominal class allocation* can be enforced by a network provider using an ingress classifier that operates based on the packet port numbers. Notice that each class provides (almost) the delay requirement of the corresponding traffic type. Also, none of the traffic types would be able to meet their delay requirement in a lower class.

5.1 Model

Suppose that we provision a network link $\mathcal{L}$ that offers N *service classes* and carries M *traffic types*. A traffic type j is characterized by an average queueing delay requirement χ_j, and an average input rate ζ_j. Without loss of generality, the traffic types are ordered based on their delay requirements, so that $\chi_1 > \chi_2 > \ldots > \chi_M > 0$. For simplicity, we assume that all traffic types have the same packet size L, normalized as $L = 1$. The set $\{(\chi_j, \zeta_j), j = 1 \ldots M\}$ is the input of the class provisioning methodology.

We assume that the network manager provisions $\mathcal{L}$ for lossless operation. This is a reasonable assumption, as most backbone providers today provision their links for lossless operation. The link capacity, which is an outcome of the class provisioning methodology, is denoted by C. The offered rate in class i is λ_i, while the aggregate offered rate is $\lambda = \sum_i^N \lambda_i = \sum_j^M \zeta_j$. The utilization is denoted by $u = \lambda/C$. Note that λ depends on the traffic type rates, and is constant for a given workload profile. The capacity, and thus the utilization, are variables, however, that are to be computed by the provisioning methodology.

The delay differentiation between classes in $\mathcal{L}$ follows the Proportional Delay Differentiation (PDD) model [12], as in §3. For completeness, we remind the reader that according to the PDD model, if $\bar{d}_i$ is the average queueing delay in class i, the ratios between class delays are fixed to:

$$\frac{\bar{d}_i}{\bar{d}_j} = \frac{\delta_i}{\delta_j} \qquad 1 \leq i,j \leq N \tag{8}$$

where δ_i are the Delay Differentiation Parameters (DDPs) ($\delta_1 = 1 > \delta_2 > \ldots > \delta_N > 0$). Notice that the PDD model consists of $N-1$ ratios of N class delays.

The packet scheduler in $\mathcal{L}$ is a work-conserving, non-preemptive, Proportional Delay Differentiation scheduler that can meet the PDD model, when the specified DDPs are feasible. Such schedulers have been the subject of recent research [12, 20, 19, 21, 22, 23].

The proposed class provisioning methodology consists of two parts. First, we determine the *target average delay* $\hat{v}_i$ and the corresponding *target offered rate* $\hat{h}_i$ for each class $i = 1 \ldots N$. The objective in the selection of the N pairs $\{(\hat{v}_i, \hat{h}_i), i = 1 \ldots N\}$ is that $\mathcal{L}$ *meets the average delay requirement* $\{\chi_j, j = 1, \ldots, M\}$ *of the* M *traffic types, with the minimum link capacity.* Second, we compute this minimum link capacity $\hat{C}$, and the required DDPs $\{\hat{\delta}_i, i = 2 \ldots N\}$.

5.2 Class Operating Point (COP) selection

We define a *Class Operating Point (COP)* as a vector $\mathbf{v} = \{v_1, \ldots, v_N\}$, such that $v_1 \geq v_2 \geq \ldots \geq v_N > 0$, where v_i is the desired (target) average delay in class i. A COP $\mathbf{v}$ is *acceptable* when for each traffic type j there exists at least one class i such that $v_i \leq \chi_j$. Let V be the set of acceptable COPs. If $\mathbf{v} \in V$, then for each traffic type j there exists a class $n(j) \in \{1 \ldots N\}$ such that $v_{n(j)} \leq \chi_j < v_{n(j)-1}$ ($v_0 = \infty$).

Given an acceptable COP $\mathbf{v}$, each traffic type j is assigned to class $n(j)$, since that is the *minimum class* that satisfies the delay requirement of j. We say that traffic type j is *mapped* to class $n(j)$, or that $n(j)$ is the *nominal class* for traffic type j. Note that when $M > N$ (which is probably the more practical case), there will be more than one traffic types mapped to some classes. Some classes, that are referred to as *void*, may not be nominal for any traffic type. To denote the inverse mapping, from classes to traffic types, $t(i)$ is the *maximum* traffic type that maps to class i; if class i is void, then $t(i)$=0.

The *expected rate* h_i in class i is the aggregate rate of all traffic types that map to class i. Since an acceptable COP $\mathbf{v}$ determines the nominal class for each traffic type, the expected rates are a function of $\mathbf{v}$,

$$\mathbf{h}(\mathbf{v}) = \{h_1(\mathbf{v}), h_2(\mathbf{v}), \ldots, h_N(\mathbf{v})\} \quad \text{with} \quad h_i(\mathbf{v}) = \sum_{j:n(j)=i} \zeta_j \geq 0 \tag{9}$$

When the particular $\mathbf{v}$ that we consider is obvious, we write $\mathbf{h}$ or h_i, instead of $\mathbf{h}(\mathbf{v})$ or $h_i(\mathbf{v})$, respectively. The *total expected rate* in the link is

$$h = \sum_{i=1}^{N} h_i = \sum_{j=1}^{M} \zeta_j \tag{10}$$

that is independent of $\mathbf{v}$.

We say that an acceptable COP is *realized* if the average delay in each class i becomes $\bar{d}_i$=v_i, when the class rates are λ_k=h_k for $k = 1 \ldots N$. The link capacity that is required for realizing $\mathbf{v}$ is called the *capacity requirement* of $\mathbf{v}$ and is denoted by $C(\mathbf{v})$. When $\mathbf{v}$ is realized, the aggregate backlog in $\mathcal{L}$ becomes

$$\bar{q}_{ag}(\mathbf{v}) = \sum_{i=1}^{N} \bar{d}_i \lambda_i = \sum_{i=1}^{N} v_i h_i \tag{11}$$

Note that the average backlog $\bar{q}_{ag}$ depends on the link utilization and the statistical characteristics of the traffic, and not on the scheduler or the class load distribution [16].

If we want to compute the capacity requirement $C(\mathbf{v})$ of a COP $\mathbf{v}$, we need to know how the average backlog $\bar{q}_{ag}$ varies with the link utilization u. We refer to this relation as the *average backlog function* $\bar{q}_{ag} = \beta(u)$. $\beta(u)$ is assumed to be strictly increasing and convex when $u \in (0,1)$, and it is unbounded as $u \to 1$. $\beta(u)$ is thus invertible, meaning that the link utilization u can be computed from the average backlog through the *inverse backlog function* $\beta^{-1}(\bar{q}_{ag})$. The problem of estimating the average backlog function is discussed in §5.4. Given the inverse backlog function, we can determine the capacity requirement of $\mathbf{v}$ from

$$C(\mathbf{v}) = \frac{h}{u(\mathbf{v})} = \frac{h}{\beta^{-1}(\bar{q}_{ag}(\mathbf{v}))} \tag{12}$$

where $u(\mathbf{v})$ is the link utilization that creates an average backlog $\bar{q}_{ag}(\mathbf{v})$.

An important part of the class provisioning methodology is to select the *optimal COP* $\hat{\mathbf{v}}$ among all acceptable COPs. The optimality constraint in the selection of $\hat{\mathbf{v}}$ is that *it has to be the acceptable COP with the minimum capacity requirement*,

$$\hat{\mathbf{v}} = \arg\min_{\mathbf{v} \in V} C(\mathbf{v}) \tag{13}$$

Since the average backlog function $\bar{q}_{ag} = \beta(u) = \beta(\lambda/C)$ is strictly increasing though, *the COP with the minimum capacity requirement is the COP with the maximum average backlog.* So, the optimal COP is the acceptable COP with the maximum average aggregate backlog,

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v} \in V} \bar{q}_{ag}(\mathbf{v}) \tag{14}$$

To determine $\hat{\mathbf{v}}$ in practice, we only need to consider a finite set of acceptable COPs. To see why, consider the example of Figure 10. The example refers to a link with N=2 classes and M=4 traffic types, and it shows two acceptable COPs. In the COP of Figure 10-a, the first traffic type and a large part of the second traffic type are mapped to Class-1; the rest of the traffic is mapped to Class-2. Notice that the four traffic types meet their delay requirements with this class mapping, but there is some 'waste' of resources since the two classes provide lower delays than what the traffic types need.

In the COP of Figure 10-b, on the other hand, the average delay in each class is equal to the delay requirement of one of the traffic types. Specifically, the first two traffic types map to Class-1, which offers the delay requirement χ_2 of the second traffic type, while the two higher traffic types map to Class-4, which offers the delay requirement χ_4 of the fourth traffic type. Note that the shaded area in each COP represents the average backlog $\bar{q}_{ag}(\mathbf{v}) = \sum_{i=1}^{N} v_i h_i$. The optimal COP has to maximize the average backlog, and thus, to maximize the shaded area in Figure 10. In the previous example, the COP of Figure 10-b can be shown to be optimal.

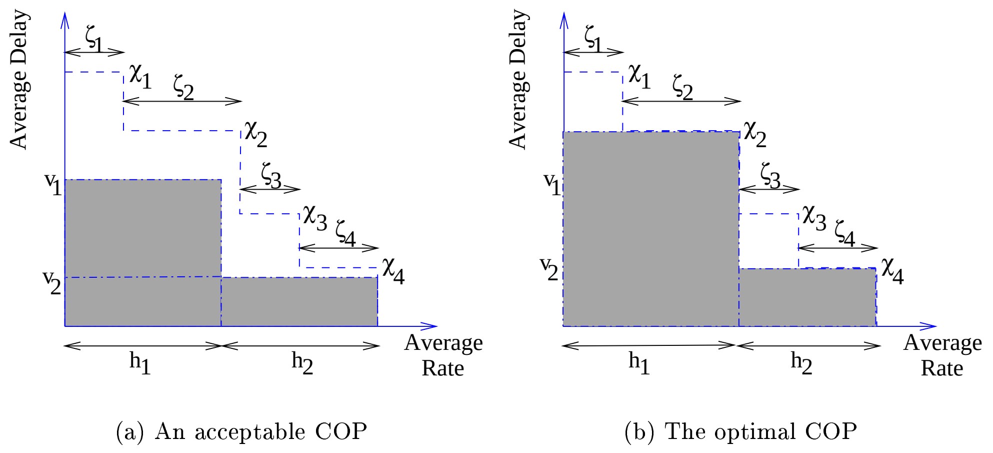


Figure 10: An acceptable COP and the optimal COP for a link with N=2 classes and M=4 traffic types.

Based on the graphical insight from the previous example, we can see that the optimal COP $\hat{\mathbf{v}}$ satisfies the following properties. First, *each optimal class delay $\hat{v}_i$ should be equal to the delay requirement of a traffic type*, i.e., for each $i = 1 \dots N$ there is a $j \in \{1 \dots M\}$ such that $\hat{v}_i = \chi_j$. Second, *the optimal COP should not have void classes*, because void classes always lead to an average backlog that is less than maximum. So, if $\hat{v}_i = \chi_j$, then there should be no other class k with $\hat{v}_k = \chi_j$. Third, following from the previous two properties, the target delay for the highest class should be the most stringent traffic type delay requirement, i.e., $\hat{v}_N = \chi_M$.

Putting the previous three properties together, we see that the finite set of acceptable COPs that should be examined in order to determine the optimal COP $\hat{\mathbf{v}}$ is

$$\{\mathbf{v} \in V : v_1 > v_2 > \dots v_N, \ \forall i = 1 \dots N, \exists j \in \{1 \dots M\} \text{ such that } v_i = \chi_j \quad (v_N = \chi_M)\} \tag{15}$$

Note that the strict inequalities between the v_i's prevent the existence of void classes.

A recursive algorithm for selecting the optimal COP is shown in Figure 11. The run-time complexity of the algorithm is $O\Big((M-N)^{N-1}\Big)$. For instance, in the case of N=3 classes and $M \geq 3$ traffic types, the algorithm examines $(M-N+1)(M-N+2)/2$ COPs. Since the provisioning methodology is performed off-line, and the number of classes and traffic types is expected to be relatively small (e.g., N=8, M=16), the run-time complexity of the algorithm is not prohibitive.

Example of optimal COP selection:
Suppose that a certain link supports N=2 classes and M=3 traffic types. We need to consider two COPs, depending on whether the maximum traffic type that maps to Class-1 is traffic type 1 or 2. Specifically, the two COPs are:

$$\mathbf{v_1} = (\chi_1, \chi_3) \text{ with } \mathbf{h}_1 = (\zeta_1, \zeta_2 + \zeta_3) \quad \text{and} \quad \mathbf{v_2} = (\chi_2, \chi_3) \text{ with } \mathbf{h}_2 = (\zeta_1 + \zeta_2, \zeta_3)$$

The average backlog in the two COPs is:

$$\bar{q}_{ag}(\mathbf{v_1}) = \chi_1\zeta_1 + \chi_3(\zeta_2 + \zeta_3) \quad \text{and} \quad \bar{q}_{ag}(\mathbf{v_2}) = \chi_2(\zeta_1 + \zeta_2) + \chi_3\zeta_3$$

```
optimal_cop ($t_1, t_2, \ldots, t_{N-1},\ i, max\_q, best\_cop$)
{
// $t_i$: maximum traffic type ($\in \{1 \ldots M\}$) that maps to class $i$.
// $max\_q$ and $best\_cop$ are call-by-reference arguments.
// Initially, call optimal_cop ($0, 0, \ldots, 0,\ 1,\ 0,\ \emptyset$).
// The optimal COP $\hat{\mathbf{v}}$ is returned in the $best\_cop$ argument.
// avg_backlog() computes the backlog of a COP as in (11).
// Note: $t_N = M$ and $t_0 = 0$.

    if ($i \leq N-1$) {
        for $t_i = (t_{i-1}+1)$  to  $(M-N+i)$
            optimal_cop ($t_1, t_2, \ldots, t_{N-1},\ i+1, max\_q, best\_cop$);
    }
    else {
        $q$ = avg_backlog ($\chi_{t_1}, \chi_{t_2}, \ldots, \chi_{t_{N-1}}, \chi_{t_N}$);
        if ($q > max\_q$) {
            $max\_q = q$;
            $best\_cop = (\chi_{t_1}, \chi_{t_2}, \ldots, \chi_{t_{N-1}}, \chi_{t_N})$;
        }
    }
}
```

Figure 11: Algorithm to determine the optimal COP $\hat{\mathbf{v}}$.

Which COP has the maximum average backlog depends on the relation between the traffic type rates and average delay requirements. If $\zeta_1(\chi_1 - \chi_2) > \zeta_2(\chi_2 - \chi_3)$, then $\bar{q}_{ag}(\mathbf{v_1}) \geq \bar{q}_{ag}(\mathbf{v_2})$ and the optimal COP is $\mathbf{v_1}$; otherwise, the optimal COP is $\mathbf{v_2}$.

5.3 Minimum link capacity and Delay Differentiation Parameters

In the first part of the class provisioning methodology, the goal was to determine the mapping from traffic types to classes that leads to the minimum capacity requirement. Given this optimal COP $\hat{\mathbf{v}}$ and the corresponding expected rate vector $\mathbf{h}(\hat{\mathbf{v}})$, the second part of the provisioning methodology determines the minimum capacity requirement and the required DDPs.

The minimum capacity requirement $\hat{C}$ can be computed using the inverse backlog function, as

$$\hat{C} = C(\hat{\mathbf{v}}) = \frac{h}{\beta^{-1}\left(\bar{q}_{ag}(\hat{\mathbf{v}})\right)} = \frac{h}{\beta^{-1}\left(\sum_{i=1}^{N} \hat{v}_i \hat{h}_i\right)} \tag{16}$$

where h is the total expected rate given by (10). The required DDPs, on the other hand, are simply the ratios of the corresponding optimal average class delays, i.e.,

$$\frac{\hat{\delta}_i}{\hat{\delta}_1} = \frac{\hat{v}_i}{\hat{v}_1} \qquad i = 2 \ldots N \tag{17}$$

with $\hat{\delta}_1$=1.

Notice that these particular DDPs provide the $N-1$ delay ratios of the N class delays in the optimal COP. Without the appropriate link capacity, however, the absolute class delays will not be as in the optimal COP. Specifically, if the capacity is $C > \hat{C}$, it is easy to see that *all* class delays will be lower, i.e., $\bar{d}_i < \hat{v}_i$ for all i. This would be an instance of *over-provisioning.* On the other hand, if the capacity is $C < \hat{C}$, *all* class delays will be larger, i.e., $\bar{d}_i > \hat{v}_i$ for all i. That would be an instance of *under-provisioning.* In practice, there is also a *well-provisioned* operating region in which the capacity is $C \in (\hat{C}_-, \hat{C}_+)$, where $\hat{C}_+ = \hat{C}$ and $\hat{C}_- = f\hat{C}$, with f being a *tolerance factor* ($f < 1$). Such a tolerance factor is necessary because of uncertainties in the workload profile and in the estimation of the average backlog function.

Also note that, throughout the provisioning methodology, we did not have to compute explicit capacity shares for each class. That would be the case if we had used a *link sharing* scheduler, such as WFQ [24], Class Based Queueing (CBQ) [25], or Hierarchical Packet Fair Queueing (H-PFQ) [26]. Unfortunately, there is no straightforward approach to compute the $N-1$ weights of such schedulers in order to meet a certain average delay in each class. Additionally, a 'trial-and-error' approach would require searching in an $(N-1)$-dimensional space, making the approach impractical even for a small number of classes.

With PDD scheduling, on the other hand, we avoid the explicit computation of a capacity allocation between classes. A PDD scheduler services backlogged packets in an appropriate order for the given delay ratios to be met. Having fixed the $N-1$ delay ratios, the absolute delay in each of the N classes depends only on the link capacity. If the average backlog function is known, the calculation of $\hat{C}$ is straightforward. If the average backlog function is unknown, we can adjust the link capacity until the class delays become as in the optimal COP. Such a trial-and-error approach is simpler, because there is only one 'knob' to vary, and the relation between the link capacity and the average class delays is monotonic.

Example of DDP and capacity calculations:
Suppose that a certain link offers N=4 classes, and that we need to meet the following COP:

$$\mathbf{v} = (40, 20, 10, 5)\text{msec} \quad \text{and} \quad \mathbf{h} = (0.5, 0.5, 2.0, 1.0)\text{kpps}$$

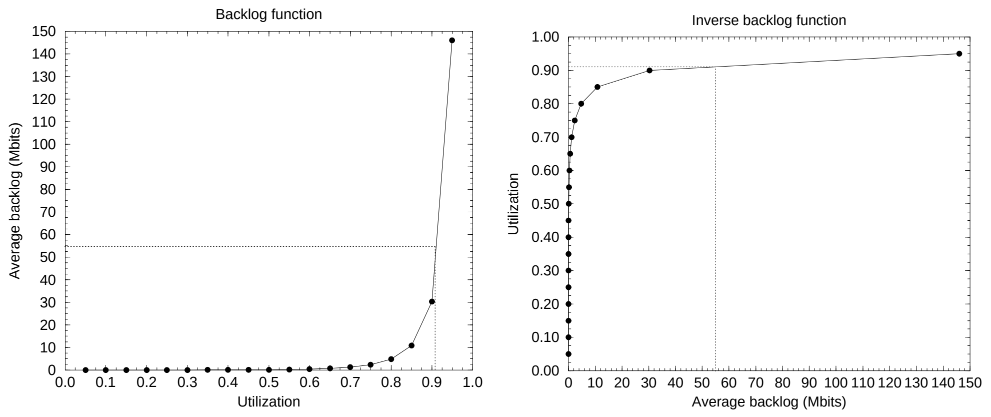
where *kpps* stands for 'kilo-packets-per-second'. If the average packet size is 1000 bytes, the total expected rate is $h = \sum_i h_i$=4kpps, or about 32Mbps. The problem is to determine the DDPs and the minimum link capacity required to realize this COP.

From (17), the required DDPs are

$$\delta_2 = \frac{20}{40} = 0.5 \quad \delta_3 = \frac{10}{40} = 0.25 \quad \delta_4 = \frac{5}{40} = 0.125$$

With this optimal COP, the average backlog is

$$\bar{q}_{ag} = \sum_i v_i h_i = 40 \times 0.5 + 20 \times 0.5 + 10 \times 2.0 + 5 \times 1.0 = 55 \text{ packets}$$



(a) $q_{ag} = \beta(u)$ (b) $u = \beta^{-1}(q_{ag})$

Figure 12: The backlog and the inverse backlog functions for Pareto traffic with α=1.5.

We can now use the inverse backlog function $u = \beta^{-1}(\bar{q}_{ag})$ to compute the required utilization u. In this example, suppose that the average backlog function (and its inverse) are as in Figure 12 (these curves are generated from simulating Pareto interarrivals with α=1.5). For $\bar{q}_{ag}$=55 packets we find that the utilization is $u = \beta^{-1}(55) \approx$92.0%, and so the capacity requirement is C=h/u=4444pps, or about 35.6Mbps.

Simulating the link with the previous DDPs, with u=92.0%, and with a WTP scheduler [12], we get that the class average delays are $(\bar{d}_1, \bar{d}_2, \bar{d}_3, \bar{d}_4) = (35.1, 17.5, 8.8, 4.4)$msec, that are slighly less than the given maximum average delays specified in the given COP. Because the backlog curve is quite steep in the heavy load range, however, slight variations in the utilization or in the expected class rates can violate the average delay requirements. For example, if the utilization is increased to u=94.0%, the average class delays become $(\bar{d}_1, \bar{d}_2, \bar{d}_3, \bar{d}_4) = (68.2, 34.1, 17.1, 8.6)$msec, that violate the average delay requirements. The large sensitivity of the capacity requirement in the heavy load range implies that the network operator *should* use some tolerance in the computation of C. Even if the network provider provisions the link with a higher capacity than $\hat{C}$, it is still useful to know $\hat{C}$ as a *lower bound* on the required link capacity.

The class provisioning methodology can be performed over relatively long timescales (say weeks or months), depending on how simple it is to adjust the link capacity. It is noted though that it gradually becomes simpler to adjust the capacity of a link even in a few minutes or seconds, through the use of Wavelength-Division-Multiplexing (WDM). Using such technologies, an ISP can lease the capacity of an additional 'wavelength' from the backbone provider that owns the network fibers, when a larger traffic demand is anticipated or encountered. Also, if the characterization of traffic types on a certain link follows different patterns through the day (e.g., many IP-telephony sessions through the day and mostly WWW sessions in the evening), the network operator can perform class provisioning for the different traffic patterns, and operate the link with a schedule of different capacities and DDPs during the day.

5.4 The average Backlog Function

The calculation of the capacity requirement of a COP $\mathbf{v}$ can be performed if we know the average backlog $\bar{q}_{ag}$ as a function of the link utilization u. For simple queueing models, the function $\beta(u)$ is analytically known. For instance, in the $M|M|1$ system $\beta(u) = \frac{u^2}{1-u}$ packets, while in the $M|G|1$ system $\beta(u) = \frac{u^2}{1-u}\frac{1+c_L^2}{2}$, where c_L is the coefficient of variation of the packet size distribution [16]. In the very general $G|G|1$ system, the average backlog can be approximated by the Allen-Cunneen formula $\beta(u) \approx \frac{u^2}{1-u}\frac{c_A^2+c_L^2}{2}$ [27], where c_A is the coefficient of variation of the distribution of interarrivals.

A practical alternative, instead of relying on queueing models, is to *measure* the function $\beta(u)$ directly on the router, by monitoring the actual backlog in the link. The network operator, in that case, would need to record the average backlog in different link utilizations. If the underlying traffic dynamics are stationary, it would be possible to extract an empirical curve for the average backlog function.

It is noted that the backlog function may not only depend on the utilization u, but also on the capacity C. This can occur if the statistical properties of the traffic (traffic burstiness) depend on C. For instance, with the same utilization, an OC-3 link (155Mbps) may have a larger backlog than a T-1 link (1.5Mbps), because higher capacity links attract more bursty traffic in general. In a relatively narrow range of C though, it is reasonable to assume that the traffic burstiness remains invariant, and that the backlog function depends on u, but not on C.

6 Related work

A brief investigation of DCS in the context of proportional delay differentiation appeared in [28]. That work considered a class selection algorithm that routers can perform in order to provide a maximum delay to each flow. With a continuous-time model, and assuming a continuous range of class choices, [28] showed that when the set of user requirements is feasible, each flow converges to a class that provides the requested delay.

Orda and Shimkin considered multi-class networks with users that dynamically choose a class based on performance and pricing constraints [29]. The users select, in a distributed and selfish manner, the class that provides them with the maximum difference between utility and cost. The problem then is to compute the optimal class prices that will cause users to select the *nominal service class* that the network has provisioned for them. Even though there are some similarities between such an 'incentive pricing' framework and DCS, the problem formulation, the proposed mechanisms, and the final results are quire different. It is also likely that in practice the class prices would depend on marketing and competition, rather than on network provisioning mechanisms.

Chen and Park considered individual flows with absolute performance requirements in a stateless network [30]. 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 [17]. 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 [17] 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 $\hat{\mathbf{c}}$ can be shown to be Pareto and system optimal for the user population.

[17] assumed that the underlying per-hop mechanism is GPS scheduling with *periodic GPS weight adjustments.* Such adjustments create a feedback loop that modifies the GPS weights based on the measured class loads. Changing the GPS weights, however, affects the per-class delays and losses, and in the DCS context, this leads to new class transitions and modified class loads. So, there is another feedback loop in the system that modifies the class loads based on the GPS weights. The previous two feedback loops can interact, causing races or instabilities, if they operate in about the same timescales. For this reason, we argue that the proportional differentiation model is a more appropriate per-hop behavior for DCS, as compared to GPS-like schedulers (or other static resource partitioning schemes) that perform periodic weight adjustments.

Our class provisioning model follows the framework of [29]. In that framework, a number of traffic types with diverse QoS requirements are mapped to a normally smaller number of classes, while the network provider provisions a nominal service class for each traffic type. As mentioned earlier though, [29] uses this framework in the problem of computing of optimal class prices, while we focus on computing the optimal class differentiation parameters and the minimum link capacity.

[31] considers the problem of sharing the bandwidth and buffers of a link between a number of classes of service. The objective is to find the optimal resource partitioning between classes so that the *utility* of each class is maximized, given the 'wealth' of that class, and the average offered rate in that class. One of the utility functions considered is an upper bound on the loss rate in each class. A difference between that work and our class provisioning methodology is that [31] is based on static resource partitioning mechanisms (Weighted-Round-Robin scheduling and Complete-Buffer-Partitioning dropping), while we focus on proportional differentiation mechanisms.

[32] studied the provisioning problem in the context of providing a certain loss rate to each class. [32] showed that the PLR(∞) dropper, jointly with a FCFS scheduler, is *optimal* because it requires *the minimum capacity for the given loss rates*, compared to any other work-conserving dropping scheme. It is interesting to examine if a proportional delay scheduler has the same property, i.e., whether it can provide a certain average delay in each class with the minimum capacity among all work-conserving schedulers.

7 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 first demonstrated that if the end-points dynamically search for an acceptable class as in DCS, then the per-hop relative differentiation that the network offers can lead to absolute and end-to-end QoS under certain conditions. The proportional differentiation model provides an appropriate per-hop behavior for DCS for two reasons. First, it maintains a predictable class ordering even when the class load distribution is constantly changing due to DCS traffic. Second, it allows the network provider to adjust the class spacing based on the corresponding spacing in the QoS requirements of the DCS workload. For a single link model, we gave an algorithm that computes the minimum acceptable class selection for each user, when it is feasible to satisfy all users (well-provisioned case). Users converge to this minimum acceptable class when the distributed DCS equilibrium is unique. Other DCS equilibria can also exist, however, that are suboptimal for either only the users, or for both the users and the network provider. In the under-provisioned case, some users (with the most stringent requirements) converge to the highest class and remain unsatisfied there. The simulation study of an end-to-end DCS algorithm provided further insight into the dynamic behavior of DCS.

A central issue in DCS is whether the network is well-provisioning for a certain user population and traffic mix. In the second part of this paper, we proposed a class provisioning methodology that determines the minimum link capacity needed to support the given traffic types and volume, the nominal class of service for each traffic type, and the appropriate resource allocation between classes. The class provisioning methodology is effective as long as the workload profile is valid. If the traffic types have larger arrival rates, or if the average backlog function is not accurately estimated, some traffic types may not be adequately supported. Similar problems can arise due to dynamic routing changes, link or router failures, or unexpected increases in the traffic demand. In those cases, the link may not operate in its well-provisioned operating point. The PDD differentiation, however, will still provide a controllable and predictable relative differentiation between classes, even though the absolute QoS of each class will not be known.

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

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

[14] 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.

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

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

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

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

[19] 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.

[20] C. C. Li, S.-L. Tsao, M. C. Chen, Y. Sun, and Y.-M. Huang, "Proportional Delay Differentiation Service Based on Weighted Fair Queueing," in *Proceedings IEEE International Conference on Computer Communications and Networks (ICCCN)*, Oct. 2000.

[21] Y. Moret and S.Fdida, "A Proportional Queue Control Mechanism to Provide Differentiated Services," in *International Symposium on Computer and Information Systems (ISCIS)*, Oct. 1998.

[22] J. Liebeherr and N.Christin, "JoBS: Joint Buffer Management and Scheduling for Differentiated Services," in *Proceedings IWQoS*, June 2001.

[23] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," tech. rep., University of Delaware, Apr. 2001. Conditionally accepted for publication to the IEEE/ACM Transactions on Networking.

[24] A. Demers, S.Keshav, and S.Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," in *Internetworking: Research and Experience*, pp. 3–26, 1990.

[25] S. Floyd and V. Jacobson, "Link-Sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 365–386, Aug. 1995.

[26] J. Bennett and H.Zhang, "Hierarchical Packet Fair Queueing Algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 675–689, Oct. 1997.

[27] O. Allen, *Probability, Statistics, and Queueing Theory with Computer Science Applications.* Academic Press, 2nd edition, 1990.

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

[29] A. Orda and N. Shimkin, "Incentive Pricing in Multi-Class Communication Networks," in *Proceedings of IEEE INFOCOM*, 1997.

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

[31] J. Sairamesh, D. F. Ferguson, and Y. Yemini, "An Approach to Pricing, Optimal Allocation and Quality of Service Provisioning in High-Speed Packet Networks," in *Proceedings of IEEE INFOCOM*, pp. 1111–1119, 1995.

[32] T. Yang and J.Pan, "A Measurement-Based Loss Scheduling Scheme," in *Proceedings of IEEE INFOCOM*, 1996.

Appendix

Proof of Proposition 1:
Since $c_j \geq c'_j$ for all j (and $\mathbf{c} \neq \mathbf{c}'$), there exists a class $k \in \{1, \ldots N-1\}$ such that $\sum_{i=1}^{k} \lambda_i < \sum_{i=1}^{k} \lambda'_i = \sum_{i=1}^{k} \lambda_i + \epsilon$ with $\epsilon > 0$, and $\sum_{i=k+1}^{N} \lambda_i > \sum_{i=k+1}^{N} \lambda'_i = \sum_{i=k+1}^{N} \lambda_i - \epsilon$. Since $\delta_1 = 1 > \delta_2 > \ldots > \delta_N > 0$, we have that $\sum_{i=1}^{N} \delta_i \lambda'_i > \sum_{i=1}^{N} \delta_i \lambda_i + \epsilon(\delta_k - \delta_{k+1}) > \sum_{i=1}^{N} \delta_i \lambda_i$. It follows that the average delay in each class with the CSV $\mathbf{c}'$ is lower[5] than with $\mathbf{c}$, because

$$\bar{d}_i(\mathbf{c}') = \frac{\delta_i \, \bar{q}_{ag}}{\sum_{n=1}^{N} \delta_n \lambda'_n} \leq \frac{\delta_i \, \bar{q}_{ag}}{\sum_{n=1}^{N} \delta_n \lambda_n} = \bar{d}_i(\mathbf{c}) \tag{18}$$

[5]Lower or equal, because it may be that $\bar{q}_{ag}$=0.

Proof of Proposition 2:
The proof of this result follows directly from the following property of the proportional delay differentiation model. Suppose that the class load distribution changes from $\{\lambda_n\}$ to $\{\lambda'_n\}$, with $\lambda'_i = \lambda_i - \epsilon$, $\lambda'_j = \lambda_j + \epsilon$, and $\lambda'_k = \lambda_k$ for all $k \neq i, j$ $(0 < \epsilon \leq \lambda_i)$. Let $\bar{d}'_k$ be the average delay in class k when the class load distribution is $\{\lambda'_n\}$. *If $i > j$ then $\bar{d}'_j \geq \bar{d}_i$. Similarly, if $i < j$ then $\bar{d}'_j \leq \bar{d}_i$.* In the following, we show this property.

Say that $S = \sum_{n=1}^{N} \delta_n \lambda_n$ and $S' = \sum_{n=1}^{N} \delta_n \lambda'_n$. Then, $S' = S + \epsilon(\delta_j - \delta_i)$. In the case $i > j$, we have that $\delta_j > \delta_i$. It is then easy to see that $\delta_j S \geq \delta_i S'$, because $S \geq \epsilon\delta_i$. Consequently, $\bar{d}'_j = \delta_j \bar{q}_{ag}/S' \geq \delta_i \bar{q}_{ag}/S = \bar{d}_i$. Similarly, when $i < j$, it follows that $\bar{d}'_j \leq \bar{d}_i$. Note that both load distributions cause the same average aggregate backlog $\bar{q}_{ag}$. The equality holds if $\epsilon = \lambda_i = \lambda$.

Proof of Proposition 3:
Suppose that $\mathbf{c}$ is such that $c_i > c_j$ for two users i and j with $i < j$ (and thus $\phi_j \leq \phi_i$). Let us construct the CSV $\mathbf{c}' = \mathbf{c}\rangle_i^{c_j}$, which results if user i moves to the same class c_j as user j. Since $\mathbf{c} \geq \mathbf{c}'$, from Proposition 1 we have that $\bar{d}_{c'_i}(\mathbf{c}') = \bar{d}_{c_j}(\mathbf{c}') \leq \bar{d}_{c_j}(\mathbf{c}) \leq \phi_j \leq \phi_i$, which means that user i is satisfied with $\mathbf{c}'$. The rest of the users are also satisfied with $\mathbf{c}'$ because the average delay in each class is lower with $\mathbf{c}'$ than with $\mathbf{c}$. So, $\mathbf{c}'$ is also acceptable.

Applying the above procedure iteratively, we can construct an acceptable CSV $\mathbf{c}'$ that is ordered. The order of applying the above iteration does not affect the resulting CSV. From the way $\mathbf{c}'$ is constructed, we have that $c'_j = \min_{k=j,\dots,U}\{c_k\}, \qquad j = 1, \dots, U$.

Proof of Proposition 4:
From the definition of $\mathbf{c^m}$, we have that $\mathbf{c}^1 \geq \mathbf{c}^2$. If these two CSVs are equal, the proof is complete. Otherwise, from Proposition 1, we have that $\bar{d}_i(\mathbf{c}^m) \leq \bar{d}_i(\mathbf{c}^1)$ for each class i. Similarly, $\bar{d}_i(\mathbf{c}^m) \leq \bar{d}_i(\mathbf{c}^2)$. For each user $j \in \mathcal{U}$, however, $\bar{d}_{c_j^1}(\mathbf{c}^1) \leq \phi_j$ and $\bar{d}_{c_j^2}(\mathbf{c}^2) \leq \phi_j$. So, $\bar{d}_{c_j^m}(\mathbf{c}^m) \leq \phi_j$. Since this is true for all users, $\mathbf{c}^m$ is an acceptable CSV.

Proof of Proposition 5:
Based on the DCS algorithm, all unsatisfied users move to the N'th class, while the rest of the users are satisfied in a class, which may also be class N. Let $\tilde{\mathbf{c}}$ be the corresponding distributed equilibrium CSV. We prove that *any satisfied user must have a larger average delay requirement than any unsatisfied user.*

For any unsatisfied user, say j, we have that $\tilde{c}_j = N$ and that $\bar{d}_{\tilde{c}_j}(\tilde{\mathbf{c}}) > \phi_j$. If there are no satisfied users, there is nothing to prove. Let i be a satisfied user in a class $\tilde{c}_i \leq N$. Suppose that user i has a lower or equal average delay requirement than user j, i.e., $\phi_i \leq \phi_j$. User i is satisfied, and so $\bar{d}_{\tilde{c}_i}(\tilde{\mathbf{c}}) \leq \phi_i \leq \phi_j$. So, $\bar{d}_{\tilde{c}_i}(\tilde{\mathbf{c}}) < \bar{d}_{\tilde{c}_j}(\tilde{\mathbf{c}})$. Since $\tilde{c}_i \leq \tilde{c}_j$, however, the PDD model requires that $\bar{d}_{\tilde{c}_i}(\tilde{\mathbf{c}}) \geq \bar{d}_{\tilde{c}_j}(\tilde{\mathbf{c}})$ which leads to a contradiction. So, it must be that $\phi_i > \phi_j$ for any satisfied user i and unsatisfied user j. Since the satisfied users have larger delay requirements than the unsatisfied users, the resulting DCS equilibrium must be of the form $\tilde{\mathbf{c}} = (c_1, \dots, c_S, N, \dots, N)$ where S is the number of satisfied users $(0 \leq S < U)$.