A Distributed Protocol for Multi-Class QoS Provision in Noncooperative Many-Switch Systems

Shaogang Chen* Kihong Park†
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
{chensg,park}@cs.purdue.edu

Abstract

This paper presents an architecture for multi-class quality of service (QoS) provision in wide area networks. Users or applications are assumed to be selfish and end-to-end QoS is determined by the service levels received by an application traffic flow at each router or switch along a path.

In previous work, we have given a comprehensive analysis of the noncooperative multi-class QoS provision game for single-switch systems showing when Nash equilibria exist and under what conditions they are Pareto and/or system optimal. In this paper, we propose a specific network architecture for facilitating noncooperative QoS provision in many-switch systems such as the Internet with emphasis on realizability. We shield the user from having to choose the service classes on the switches along a route—a hard combinatorial optimization problem even assuming perfect knowledge about network state—while preserving the basic premise of selfishness.

This is achieved by employing a set of QoS agents installed at routers which act on behalf of an user's traffic flow. The QoS agent intercepts packets entering a switch implementing generalized processor sharing (GPS) packet scheduling—and using only constant space packet header overhead and zero per-connection state at the routers—determines which service class to assign the packet to to satisfy the user's end-to-end QoS requirement at minimum cost. We present simulation results which show that our architecture is able to provide stable, stratified services to application traffic with diverse QoS requirements.

*Supported in part by NSF grant ANI-9714707.
†Contact author. Supported in part by NSF grants ANI-9714707, ESS-9806741, and grants from PRF and Sprint.

1. Introduction

Motivation With the advent of global network infrastructures and its emerging role as an important enabling platform for services spanning commerce, entertainment, education, and a compendium of other everyday activities, building an efficient network infrastructure capable of supporting diverse user needs has become a critical problem. The current Internet provides a single best-effort service class wherein resolution of resource contention conflicts and facilitation of differentiated services cannot be easily affected. Unless application traffic is transported over private or leased communication channels, messages corresponding to an on-line banking transaction are treated no differently from packets constituting bulk file transfers stemming from casual browsing on the World Wide Web. Moreover, the current Internet provides no unified criteria to the "who should get what" question, and technically more important (and the subject matter of this paper), the mechanisms to effectively deal with the "how" problem. A consensus is emerging that the "who should get what" question needs to be addressed in the framework of microeconomics and game theory where *fairness* and *optimality* can be given well-founded, precise formulations.

Significant work has been carried out in formulating resource allocation problems spanning a number of different domains in the context of microeconomics and game theory [2, 6, 9, 10, 12, 16, 20, 21]. The models and approaches proposed in the literature differ along several dimensions, some of the important ones being whether applications or users are assumed to be cooperative or selfish, whether pricing is used or not, and how much computing responsibility is delegated to the user. Several papers have addressed the issue of multi-class QoS provision in high-speed networks [2, 8, 13, 21, 20]. Some of the works employ a cooperative framework or place significant computing responsibilities on the part of the user [13, 20], some investigate the effect of pricing incentives [2], and others represent flow/congestion control and routing models that only partially address the quality of service problem [8, 16, 21].

One dimension that has received relatively little attention is the realization of multi-class QoS provision schemes in *many-switch* systems such as the Internet. In the latter, the scale of the system renders impractical certain user behavior (e.g., assumption of knowledge of global network state) and the coupling between routers running control algorithms introduces a slew of additional complexities that are difficult to analyze using current tools of game theory. This paper complements previous works, in particular, our work on multi-class QoS provision in *single-switch* environments with heterogeneous user QoS requirements where game theoretic techniques were used to show when Nash equilibria exist and under what conditions they are Pareto and/or system optimal [19]. In this paper, we advance a specific network architecture for provisioning multi-class QoS in many-switch systems where desirable abstractions and features are crystallized, given practical realizations, and implementation and efficiency issues are emphasized.

Stratified Best-Effort Service The traditional approach to QoS provision uses resource reservations and admission control such that a traffic stream's mean data rate and burstiness can be suitably accommodated by a network. Although research abounds [3, 5, 7, 15, 17], analytic tools for computing QoS guarantees rely on shaping of input traffic to preserve well-behavedness across switches which implement some form of packet scheduling discipline such as generalized processor sharing (GPS), also known as weighted fair queueing [4]. Real-time constraints of multimedia traffic and the scale-invariant burstiness associated with self-similar network traffic [11, 18] limit the shapability of input traffic while at the same time reserving bandwidth that is significantly smaller than the peak transmission rate. Thus QoS and utilization stand in a trade-off relationship with each other [18] and transporting application traffic over reserved channels, in general, incurs a high cost.

This makes it important to organize today's best-effort bandwidth, as examplified by the Internet, into *stratified* services with graded QoS properties such that the QoS requirements of a compendium of applications can be effectively met. This is particularly useful for applications that possess diverse but—to varying degrees—flexible QoS requirements. It would be overkill to transport such traffic over reserved channels. On the other hand, relying on homogenous best-effort service, characteristic of today's Internet, would be equally unsatisfactory. A dual architecture capable of supporting reserved and stratified best-effort service is needed which, in turn, helps amortize the cost of inefficiencies stemming from overprovisioned resources for guaranteed traffic.

Many-Switch Architecture Extending the game-theoretic analysis [19] to many-switch systems is a daunting task due to the nonlinear coupling introduced by switch interactions. Whereas in the single-switch system it was reasonable to assume that users had responsive access to the switch's internal state, in many-switch systems this assumption becomes infeasible. Moreover, even if users had instantaneous, perfect knowledge of all switch states along a connection, finding an optimal service class assignment along a (fixed) route is an NP-hard combinatorial optimization problem and the user is straddled with significant computing responsibility.

A practical solution designed for everyday use needs to shield the average user[1] from having to make complex computations and decisions. Since computing minimal cost service class assignments (even for a single user) is a hard problem, an alternative criterion for "goodness" would be for the network to find a solution which would be as good as the ones that a user would discover when acting selfishly with similar information. This allows us to preserve *fairness* in the noncooperative game theory sense which, in turn, also acts as the network's "contract" to the user in terms of its resource contention resolution policy. We will use this *selfishness emulation* idea as a design principle for QoS provision and distributed control in our many-switch QoS architecture.

There are four additional design features that characterize our system. One, for scalability, routers should be *stateless* in the sense that they need not maintain per-connection or per-flow information at switches. Similarly, the packet header should be simple and small, its size being independent of the hop count of a route. Two, to deal with stationary and nonstationary variations in network state, the service class assignment should be *adaptive* to these changes. Moreover, the adaptive mechanism should be implemented as a *distributed control* for scalability. Three, for the system to be usable, the services delivered by the network should be stable and *predictable*. Four, the system should be able to deliver *stratified* service with a range of QoS levels that match—and adapt to—the QoS needs of the current application pool.

[1] Some users may want the option to exercise intimate control of service class assignments.

2. Single-Switch Model: Overview

Following is a summary of the single-switch QoS provision model which is necessary in understanding the architecture of the many-switch model described in Section 3. For a more complete description of the single-switch system including game theoretic results, we refer the reader to [19].

2.1. Switch Model

We assume a switch model implementing generalized processor sharing (GPS) packet scheduling. The service rate of an output link is given by μ and we assume that the switch implements a form of GPS with service weights $\alpha_1, \alpha_2, \ldots, \alpha_m$ where $\alpha_j \geq 0$, $j \in [1, m]$, and $\sum_{j=1}^{m} \alpha_j = 1$. Here, m denotes the number of service classes. We omit the link index for notational clarity. The total traffic entering each service class is denoted by q_j. Other things being equal, the larger the service weight α_j or the smaller the traffic q_j flowing into service class j, the better the QoS rendered by that class. In the case of hierarchical GPS scheduling, one of the service classes, say class 1, can be refined into several subclasses that are then used for per-connection bandwidth reservation to deliver guaranteed QoS service in conjunction with admission control.

2.2. User Model

We are given n users or applications where each user $i \in [1, n]$ has a traffic demand given by its mean data rate λ_i. Each user can choose *where* (i.e., which service class) and *how much* of its traffic to apportion to the m service classes given by its allocation vector $\Lambda_i = (\lambda_{i1}, \lambda_{i2}, \ldots, \lambda_{im})^T$ where $\lambda_{ij} \geq 0$ and $\sum_j \lambda_{ij} = \lambda_i$. Thus, $q_j = \sum_i \lambda_{ij}$. Each user is endowed with a utility function U_i which indicates the satisfaction experienced by user i when receiving a certain QoS.

Selfishness, in our context, will mean that each application $i \in [1, n]$ will try to take actions so as to maximize its individual utility U_i. The forms for U_i as well as user i's decision variables for the multi-class QoS provision problem are defined next.

2.3. Multi-Class QoS Provision Game

We arrive at a resource allocation problem in the following way. $\lambda_{ij} \geq 0$, $i \in [1, n]$, $j \in [1, m]$, denotes the traffic volume of the i'th application assigned to service class j. Hence application i is given the freedom to choose which service classes to assign her traffic to and how much. For the many-switch system, we will be particularly interested in the special case when traffic assignments are restricted to be *unsplittable*: $\lambda_{ij} \in \{\lambda_i, 0\}$, for all $j \in [1, m]$.

Let $\Lambda = (\lambda_{ij} : i, j)$ denote the resource assignment matrix, and let $c_1, c_2, \ldots, c_m$ be the packet loss rates of the m service classes. Each packet loss rate is a function of Λ, $c_j = c_j(\Lambda)$, $\quad j \in [1, m]$. The model can be extended to the case when application QoS requirements are represented by multi-dimensional QoS vectors $\mathbf{x} \in \mathbb{R}^s$, $s \geq 1$. For example, in addition to packet loss rate, $\mathbf{x}$ may specify delay requirements as well as restrictions on their fluctuations such as jitter. It turns out that the analysis of the multi-dimensional case reduces to the scalar case under certain conditions [19], and we will proceed with packet loss rate c as the sole QoS indicator for illustrative purposes.

The *weighted utility* of application i, given assignment Λ, is defined as $\bar{U}_i(\Lambda) = \sum_{j=1}^{m} \lambda_{ij} U_i(c_j)$. Subject to the above constraints, the static optimization problem can be formulated as $\max_\Lambda \quad \bar{U}(\Lambda) = \sum_{i=1}^{n} \bar{U}_i(\Lambda)$. This is a nonlinear programming problem with equality constraints.

2.4. Relevant Properties

In the following, we list some properties of single-switch QoS provision systems that are relevant to the construction of the many-switch architecture.

In [19], we show that Nash equilibria need not exist for the single-switch multi-class QoS provision game, and even if they do, they need not be Pareto nor system optimal. On the positive side, we show that for certain "resource-plentiful" systems, Nash equilibria always exist and they coincide with Pareto and system optima. Of particular interest to the many-switch system is the unsplittable case $\lambda_{ij} \in \{\lambda_i, 0\}$ for which we can show that Nash equilibria always exist.

In spite of the mixed mathematical results, our simulation study of *dynamic* single-switch games in *stationary* and *nonstationary* environments showed that desirable allocations can be achieved in the noncooperative context with selfish users. Even when oscillations result without convergence to Nash equilibria, the resulting overall QoS allocation achieved was, in many instances, still desirable from the user QoS requirement point-of-view [19].

The switch state, at any given time, is characterized by a *service class-QoS association table* which tracks the QoS rendered at each of the service classes over a local time window. This table is continuously updated by the router and made accessible—i.e., readable—by the user. Without loss of generality, we can assume there is a total ordering on the service classes induced by the rendered QoS levels

$$1 \succeq 2 \succeq \cdots \succeq m \iff c_1 \leq c_2 \leq \cdots \leq c_m \qquad (2.1)$$

where "$a \succeq b$" means service class a is "superior" to service class b. If c_j denotes packet loss rate or queueing delay, this may be due to the packet loss or queueing delay associated with service class a being smaller than that of service class b.

The service class-QoS association table also contains a *price column* associating a current price p_j for each service class $j \in [1, m]$. Each packet entering a service class j is charged a cost of p_j. Pricing is introduced to induce users with different QoS requirements to settle into different service classes such that stringent QoS applications end up receiving better service than less stringent QoS applications, albeit, at higher cost, and vice versa. The *pricing policy* necessary for this to hold must satisfy

$$c_a \leq c_b \implies p_a \geq p_b, \qquad a, b \in [1, m]. \qquad (2.2)$$

We model users with *step utility* functions

$$U_i(c) = \begin{cases} 1, & \text{if } c \leq \theta_i, \\ 0, & \text{otherwise,} \end{cases}$$

where $\theta_i \geq 0$ is a threshold that represents the i'th application's preference. Threshold or step utility functions have been applied in practical and analytical settings to represent and convey QoS preferences. For example, *hard* real-time systems, as defined in the real-time systems literature, have this "all or nothing" property. Furthermore, irrespective of whether the user of an application possesses a step utility preference or not, when interacting with a network system through an application, the user must ultimately code and convey her preference to the underlying system. *Bounds* on packet loss rate, delay, jitter, and other QoS measures have been used to encode application traffic QoS requirements in different contexts including ones where they are used to compute resource reservations and in some commercial applications [14].

A specific form of selfish user behavior is modeled by the following rule: Given QoS requirement θ_i, user i inspects the current service class-QoS association table and chooses a service class j such that

(i) $c_j \leq \theta_i$,

(ii) p_j is minimal among all service classes satisfying condition (i).

That is, user i selects the service class that satisfies its QoS requirement at least cost. Of course, this is just one among a multitude of specific selfish behavioral modes. However, it is of special interest due to the fact that for *unsplittable games* with *threshold utilities* it is an optimal (pure) user strategy in the Nash sense. That is, it is an optimal *selfish* strategy which maximizes individual utility. Here we assume that infusion of λ_i does not impact the given service class and price ordering. It is straightforward to generalize conditions (i), (ii) to handle the complement case.

Theorem 2.3 (Self-Optimization) *For unsplittable games with threshold utilities, the strategy captured by conditions* (i) & (ii) *is user optimal.*

The proof is not difficult and we omit it for brevity. We remark that the theorem is more interesting for utility functions where price is included, in particular, with U_i remaining a threshold function of c_j and being monotone (decreasing) in p_j. An instance of such a function would be $U_i(c_j)/p_j$. The existence of a simple user optimal strategy is used as a building block for the QoS control mechanism in the many-switch system.

3. Many-Switch QoS Provision Architecture

3.1. Architectural Features

Shield User from Complex Computation An important goal is to facilitate a "simple user" model where the user is shielded from making complex resource allocation decisions. The user's interface with the network system is narrow and well-defined. Its primary component is a *QoS requirement vector* $\boldsymbol{\theta}^i \in \mathbb{R}^s$ which represents user i's bounds on desired target QoS. The network system takes $\boldsymbol{\theta}^i$ as input and tries to deliver end-to-end QoS $\mathbf{x}^i$ such that $\mathbf{x}^i \leq \boldsymbol{\theta}^i$, and, at least cost to the user. The user is charged a usage fee by the network, and if at any time the cost is too prohibitive, the user can terminate her traffic. The latter can be automated by incorporating a termination condition at start-up that specifies a price bound.

Selfishness Emulation We seek to shield the user from complex resource allocation computations while at the same time adhering to the framework of non-cooperative game theory and microeconomics where fairness and efficiency are well-defined. On the surface, these two objectives seem to be at odds with each other since without some form of active user participation, *selfishness* seems to lose its meaning.

We deal with this problem by *emulating* selfish user behavior inside the switch. In fact, in the single-switch system, we are able to emulate *optimal* selfish user behavior by Theorem 2.3. The emulation is accomplished by installing a *QoS agent* inside a switch which executes the user optimal decision rule (conditions (i) & (ii) in Section 2.4) *on behalf* of the "selfish" user application.

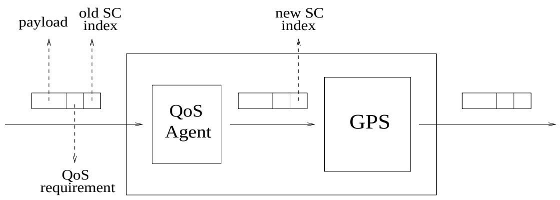

Figure 3.1. GPS switch augmented by QoS agent.

The structure of the *augmented* GPS switch containing the QoS agent is shown in Figure 3.1. As a packet enters a switch, the QoS agent intercepts it, assigns it a service class by writing a *service class index* or *key* onto the header, then forwards the packet to the GPS switch proper. The only information needed by the QoS agent to faithfully emulate selfish user behavior locally is the user QoS requirement. This is, essentially, the only piece of information that the user is asked to convey to the network, and since there are only a finite number of QoS indicators, this information can be enscribed in the packet header with constant overhead. Hence, per-flow information need not be maintained at the QoS agent and switch. This will continue to hold even after the distributed control aspect of the QoS agent has been specified.

Adaptive Service Class Assignment using Distributed Control An important difference between the single-switch and many-switch QoS provision problem is that, in the latter, the *end-to-end QoS* of a connection is determined by the QoS received by the connection at *each* of the switches or hops along a route. For example, the end-to-end delay is an *additive* function of all the per-hop queueing delays and link latencies whereas end-to-end packet loss rate is *multiplicative*. This implies that the same end-to-end QoS can be achieved by a multitude of service class assignments along a given route, some "better" than others. In other words, there is a *many-to-one* mapping between service class assignments and end-to-end QoS. The distributed control for solving the service class assignment problem is described in Section 3.3.

Per-Connection QoS Control with Stateless Routers Per-connection adaptive QoS control using distributed service class assignment is achieved without the switch maintaining any per-flow information, i.e., it is *stateless*. This is important for scalability. We achieve this by maintaining all relevant per-connection control information on small, *fixed-size* packet headers whose length is independent of the hop count of a route. The adaptive service class assignment scheme uses *end-to-end feedback* computed at the receiver—a synopsis of a connection's end-to-end QoS experienced thus far—which is used locally at switches in making service class assignments.

The principals header fields are as follows: The `Required QoS` field contains the QoS requirement vector indicated by the user. The `Achieved QoS` field contains the short-term and long-term QoS experienced by the connection which is updated at the receiver. The `Agg Factor` field constains a scalar that is used by the adaptive control. The `Charge` field contains the charge accrued by a packet on its journey thus far. The `QA Index` (**Q**oS **A**gent Index) field is optional and can be used to select other QoS Agents if multiple QoS Agent support is provided by the switch[2]. Finally, the `SC Index` (**S**ervice **C**lass Index) field is used by the GPS switch as a key to its service classes.

3.2. Many-Switch QoS Provision Game

Network Game Extension Fix user $i \in [1, n]$ and assume its traffic is assigned a route with $r \geq 1$ hops or switches. We will use $k \in [1, r]$ as the switch index. The user has a choice of r selection variables $\xi_i^k \in [1, m]$ where $\xi_i^k = j$ indicates that user i has selected to channel his traffic through service class j at switch k. Let $\boldsymbol{\xi}_i$ denote user i's service class assignment vector and let $\boldsymbol{\xi}$ denote the service class assignment of all users.

The *end-to-end QoS* received by user i, $\mathbf{x}^i \in \mathbb{R}^s$ (s denotes the number of QoS indicators), is clearly a function of $\boldsymbol{\xi}$, $\mathbf{x}^i = \mathbf{x}^i(\boldsymbol{\xi})$, and given i's traffic demand λ_i, we arrive at the individual utility $\lambda_i U_i(\mathbf{x}^i)$, consistent with the single-switch formulation. Nash equilibria, Pareto optima, and system optima can be defined with respect to $\lambda_i U_i(\mathbf{x}^i)$.

Note that the many-switch network satisfies certain *conservation laws* that are relevant to distributed QoS control. Let $\lambda_{ij}^k \in [0, \lambda_i]$ denote the volume of i's traffic flow entering into service class j at switch k. Letting c_j^k denote the packet loss rate in service class j at switch k, we have the recursive relation

$$\lambda_{ij}^{k+1} = (1 - c_j^k)\lambda_{ij}^k, \qquad k \in [1, k-1], \tag{3.1}$$

with $\lambda_{ij}^1 \equiv \lambda_{ij}$. Hence, $\lambda_{ij}^{k+1} = \prod_{\ell=1}^{k}(1 - c_j^\ell)\lambda_{ij}$, and the *end-to-end packet loss rate* for user i along a fixed

[2]For example, one possible context is the active network framework [22].

path of length r with service assignment $\boldsymbol{\xi}_i$ is, thus,

$$1 - \prod_{k=1}^{r} (1 - c^k_{\xi^k_i}). \tag{3.2}$$

Letting d^k_j denote the queueing delay in service class j at switch k, the *end-to-end delay* is given by

$$\sum_{k=1}^{r} (d_{\xi^k_i} + L_k + T_k) \tag{3.3}$$

where L_k is the link latency at hop k and T_k the transmission time.

Network Game with Pricing Pricing can be introduced by imposing relation (2.2) (see Section 2.4),

$$\mathbf{c}^k_a \succeq \mathbf{c}^k_b \implies p^k_a \geq p^k_b, \qquad a, b \in [1, m], \tag{3.4}$$

where $\mathbf{c}^k_j$ is the QoS rendered in service class j at switch k and p^k_j is its price. Thus the amount charged at switch k to user i's traffic flow is $p^k_{\xi^k_i} \lambda^k_{i\xi^k_i}$. For *threshold* utilities, one version of the *noncooperative* network game with pricing is given by

$$\min_{\boldsymbol{\xi}_i} \sum_{k=1}^{r} p^k_{\xi^k_i} \lambda^k_{i\xi^k_i} \tag{3.5}$$

subject to $\mathbf{x}^i(\boldsymbol{\xi}_i) \leq \boldsymbol{\theta}^i$. That is, the user seeks a minimum cost assignment that satisfies the user's QoS requirement. This problem turns out to be NP-hard—it can be reduced to a version of multiple choice knapsack—and approximate solutions need to be sought.

Theorem 3.6 (Many-Switch QoS Assignment) *The per-user constrained many-switch QoS assignment problem given by* (3.5) *is NP-hard.*

The reduction to multiple choice knapsack is simple and elegant and a proof can be found in [1]; we omit it here due to space constraints. For a single switch system (i.e., $r = 1$), however, the optimal selfish user strategy of Section 2.4 yields an optimal solution to the constrained minimization problem. The latter, in turn, is equivalent to maximizing utilities that are monotone (increasing) in QoS as well as being threshold, and monotone (decreasing) in price.

3.3. Distributed QoS Control

The distributed QoS control—i.e., service class assignment algorithm across a path—seeks to solve (3.5) in a decentralized fashion while implementing the design features set forth in Sections 1 and 3.1. The algorithm is implemented using two types of modules, the QoS agent (QA) running at the routers, and the sender interface (SI) and receiver interface (RI) running at the end stations. The latter are primarily responsible for monitoring per-connection end-to-end QoS and making this information available (`Achieved QoS` field in the packet header) to the QoS agents at the routers. The overall structure of a path configuration for a flow is shown in Figure 3.2. The *end-to-end feedback loop* is, again, realized using a similar path configuration; it is omitted in the diagram for clarity. The logical structure of the algorithm is composed of two parts; this is described next.

3.3.1 Single-Switch Reduction

Uniform Responsibility For flow (i.e., user) i, the network must find a service class assignment

$$\xi^1_i = j_1,\ \xi^2_i = j_2,\ \ldots,\ \xi^r_i = j_r$$

along a given route such that cost (3.5) is minimized while satisfying the flow's QoS requirement.

Let $j^* = (j^*_1, j^*_2, \ldots, j^*_r)$ be an optimal assignment and let $c^* = (c^*_1, c^*_2, \ldots, c^*_r)$ be the corresponding packet loss rates (or any other QoS indicator) experienced by i's traffic stream at the r switches. If c^* were known—an optimal *local QoS responsibility split* to satisfy the flow's end-to-end QoS requirement—it would reduce the many-switch service class assignment problem to a set of r single-switch service class assignment problems which, in turn, can be solved by the self-optimization procedure for the single-switch system.

Of course, the problem is that c^* is not known. In the absence of further information and as a first approximation, our distributed control makes the assumption that c^* is *uniform*. That is, given the user's end-to-end QoS requirement—e.g., end-to-end delay θ^i_d—we set $c^*_k = \theta^i_d / r$ for all $k \in [1, r]$. θ^i_d and r are known—they are enscribed in the packet header—and c^*_k is computed at each switch by the *QoS Agent* without per-connection state. Recall that we are using threshold utilities wherein a QoS *upper bound* is specified. For packet loss rate requirement θ^i_c, *uniform packet loss delegation* is computed at switch k by the QoS Agent by solving

$$1 - (1 - x)^r = \theta^i_c \tag{3.7}$$

for x and setting $c^*_k = x$. The LHS of equation (3.7) represents end-to-end packet loss rate and it follows from formula (3.2) by substituting x for $c^k_{\xi^k_i}$. For de-

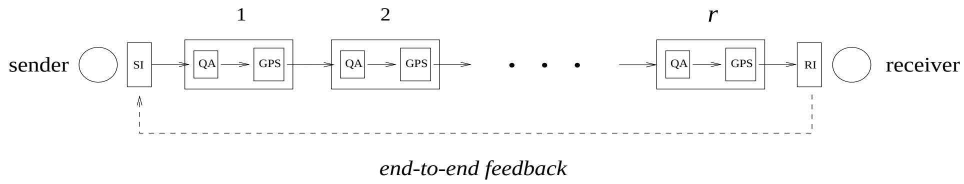

Figure 3.2. Distributed QoS control over an r-hop path.

lay variance or packet loss variance (i.e., jitter) requirement θ_v^i, uniform QoS delegation degenerates to $c_k^* = \theta_v^i$.

QoS Agent Algorithm To complete the description of the QoS Agent, after computing the *local* QoS responsibility c_k^* (in general, one for each QoS indicator), c_k^* is input to the self-optimization procedure which looks up the switch's QoS-service class association table and outputs a service class index j_k which is then used as the demux key at the GPS packet scheduler. When multiple QoS indicators (e.g., delay, packet loss, jitter) are present, then the self-optimization procedure selects the least costly service class that satisfies the QoS requirements of *all* QoS indicators. Figure 3.3 shows the basic protocol—also called *Algorithm* 0—executed by a QoS Agent at switch $k \in [1, r]$ upon seeing a packet belonging to user i.

QoS Agent Protocol at Switch k:

1. Compute c_k^* for each QoS indicator using
   a. $c_k^* \leftarrow 1 - (1 - \theta_c^i)^{1/r}$
   b. $c_k^* \leftarrow \theta_d^i / r$
   c. $c_k^* \leftarrow \theta_v^i$
2. Find service class index $j_k \in [1, m]$ such that
   a. $c_k^* \le \theta_a^i$ for each QoS indicator $a \in [1, s]$,
   b. p_{j_k} is minimal
   Else choose j_k such that p_{j_k} is maximal

Figure 3.3. Basic QoS Agent Protocol.

Nonuniform Responsibility In many instances, uniform "division of labor" with respect to achieving a target end-to-end QoS will not yield optimal solutions due to the fact that a path will consist of hot spots—i.e., bottleneck links—as well as underutilized switches.

If the QoS assignments at switches are *stable*—i.e., the QoS rendered to a flow at a switch stays invariant over time—and at hop k the QoS assignment over the *subpath* $1 \to 2 \to \cdots \to k-1$ is available, then, in the case of packet loss, instead of solving equation (3.7) we may instead exploit the given information and solve

$$1 - u(1-x)^{r-k+1} = \theta_c^i \tag{3.8}$$

for x where u is the packet loss experienced by flow i over the path leading to hop k. This reduces the scope of the uniformity assumption thereby also reducing the "uncertainty" incurred by it. The subpath QoS u can be made available to switch k by letting each QoS Agent update a field in the header based on equation (3.7). A similar, albeit additive, update can be performed for *delay*. This modification to computing step 1 of the basic QoS Agent protocol (Algorithm 0) we call *Algorithm* 1.

3.3.2 Adaptive Refinement

The nonuniform QoS assignment equation (3.8) allows an existing, nonuniform service class assignment to be preserved without pulling it toward a uniform assignment as solving the completely "memoryless" equation (3.7) would. However, this is insuffient to drive an initial uniform QoS assignment to a (possibly) more efficient nonuniform one.

As mentioned in Section 3.3, the sender interface (SI) and receiver interface (RI) form a feedback loop whereby the end-to-end QoS measured at RI is fed back to SI for subsequent propagation with the connection's payload to the switches servicing the flow. The `Agg Factor` field of the packet header contains a value $\delta_i \ge 0$ that is updated at SI based on the feedback values contained in the `Achieved QoS` field—e.g., the (short-term) end-to-end packet loss rate x_c^i experienced by connection i—and the past value of δ_i. First, to the usage of δ_i. After step 1 of Algorithm 0 or 1—but before step 2—we perform the additional update step

$$c_k^* \leftarrow \min\{c_k^* \delta_i, 1\} \tag{3.9}$$

in the case of packet loss rate. Thus the smaller δ_i the more stringent the local QoS assignment responsibility. A similar update is applied to delay and jitter.

Now to the update of δ_i. Let $\epsilon = (\theta_c^i - x_c^i)/\theta_c^i$. The update rule for δ_i is given by

$$\delta_i \leftarrow \begin{cases} \delta_i, & \text{if } \delta_* \le \epsilon \le \delta^*, \\ \max\{\delta_i - \beta, 0\}, & \text{if } \epsilon < \delta_*, \\ \delta_i + \beta, & \text{otherwise,} \end{cases}$$

where $0 < \delta_* \leq \delta^*$ and $\beta > 0$ are fixed parameters. That is, δ_i is symmetrically adjusted downwards or upwards pending on whether user i's end-to-end QoS requirement is violated or "redundantly" satisfied. δ_i imparts the service class assignment algorithm with memory such that if an initial uniform QoS assignment does not lead to user i's end-to-end QoS requirement being met—setting a local target QoS responsibility does not imply that it can be realized at a local switch (e.g., a overutilized hop will have more difficulty meeting a fixed QoS responsibility than an underutilized one)—then by decreasing δ_i the relative QoS responsibility is shifted to underutilized switches.

3.3.3 Pricing and Accounting

The GPS switch proper—in addition to executing GPS packet scheduling—does a small amount of bookkeeping. First, it monitors the instantaneous QoS rendered at each of its service classes and updates the `Measured QoS` column in its service class-QoS association table. The `Price` column is maintained such that relation (3.4) is satisfied *proportionally*—i.e., the QoS gap is linearly reflected in the price differential[3]. Lastly, to enable accounting of per-connection usage fees, the `Charge` field in the packet header holds the running sum of $p_{\xi_i^1}, p_{\xi_i^2}, \ldots, p_{\xi_i^k}$—i.e., the per-packet cost charged at each switch k for entering service class $\xi_i^k \in [1, m]$.

4. Performance Measurement

In this section, we present a synopsis of performance results based on simulations. We refer the reader to [1] for a more complete exposition.

4.1. Simulation Set-Up

We use the LBNL Network Simulator *ns* (version 2) as the basis of our simulation environment. We have modified *ns* in order to model our multi-class QoS provision architecture. This entailed, among other things, implementing the QoS Agents as separate modules invoked by the routing agent, and implementing a GPS packet scheduler module with extended functionalities.

We show results for a vBNS-based network topology which is depicted in Figure 4.1. In the current vBNS—an NSF-sponsored backbone network designated for networking and high-performance computing research—the backbone runs at OC-12 (~622Mbps) with link latency ranging from 3ms to 15ms. Some

[3] A nonlinear scheme suggested by [14] may have certain desirable properties and is currently being investigated.

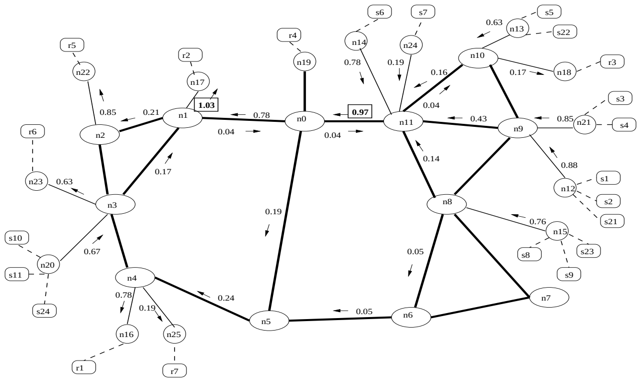

Figure 4.1. vBNS-like WAN topology.

of the links that connect member institutions to the backbone are OC-3 (~155Mbps) with link latency of about 1ms, whereas others are DS-3 (e.g., Purdue University). We make two simplifications: one, we only consider OC-3 drops from the backbone to institution nodes, and two, we scale down *all* bandwidths uniformly by a factor of 10 to reduce simulation overhead.

There are in total 15 individual flows of which 11 are application traffic with various QoS constraints (shown in separate QoS requirement configuration tables), and 4 are background traffic which are used to inject further resource contention and system variability. This particular traffic configuration has been engineered to create a number of localized *hot spots* where several connections are multiplexed onto the same output link thereby potentially causing severe contention. There are two main bottlenecks. The first one is from n_{11} to n_0 and has a load factor or utilization of 0.97. The second one is from n_1 to n_{17} and has a load factor of 1.03. Both are shown highlighted in Figure 4.1.

4.2. Performance Synopsis

4.2.1 Algorithms 1, 2, and Reservation

Table 1 shows the performance of Algorithm 1, 2, and a reservation based service class assignment scheme at delivering end-to-end QoS given the flows' QoS requirements. All switches possess three service classes of which one is set aside for background traffic. The user population consists of 0.05-, 0.01-, and 0.001-packet loss rate applications (column 2), and the end-to-end packet loss rate they actually received is shown in the column marked *pls*. *vio* denotes the fraction of instances—*over time*—that a QoS requirement is violated, and *charge* denotes the total cost accrued by all packets belonging to the same connection.

Comparing Algorithm 2 with Algorithm 1, we ob-

Flow	Pls. Req.	Algorithm 1				Algorithm 2				Reservation			
		pls.	vio.	charge	U/P	pls.	vio.	charge	U/P	pls.	vio.	charge	U/P
1	.05	.0460	.30	49.81	.0141	.0365	.03	49.64	.0195	.0487	**.43**	49.51	.0115
2	.05	.0172	.00	39.92	.0251	.0341	.00	39.66	.0252	.0321	.00	39.68	.0252
3	.05	.0275	.00	60.00	.0167	.0338	.00	59.66	.0168	.0316	.00	59.68	.0168
4	.01	**.0265**	.99	60.00	.0002	.0002	.01	60.00	.0165	.0000	.00	60.00	.0167
5	.01	**.0270**	.99	60.00	.0002	.0003	.01	60.00	.0165	.0000	.00	60.00	.0167
6	.05	.0277	.00	50.00	.0200	.0341	.00	49.66	.0201	.0320	.00	49.68	.0201
7	.01	**.0276**	.99	30.00	.0003	.0004	.01	30.00	.0330	.0000	.00	30.00	.0330
8	.05	.0464	.22	49.81	.0157	.0363	.11	49.65	.0179	.0490	**.43**	49.51	.0115
9	.01	.0000	.00	50.00	.0200	.0000	.00	50.00	.0200	.0000	.00	50.00	.0200
10	.05	.0189	.00	29.81	.0335	.0264	.00	29.74	.0336	.0175	.00	29.83	.0335
11	.001	.0000	.00	60.00	.0167	.0000	.00	60.00	.0167	.0000	.00	60.00	.0167
		$\sum U/P$ = .1625				$\sum U/P$ = **.2358**				$\sum U/P$ = .2217			

Table 1. Algorithms 1, 2, and reservation scheme.

serve that in the latter the end-to-end QoS requirements of flows 4, 5, and 7 are not being satisfied (highlighted in bold) whereas for Algorithm 2 they are. This is indicative of the added power of adaptive refinement. The *Reservation* algorithm is a static, fixed service class allocation scheme where the service class assignment, once picked, is held invariant during a connection's lifetime. It is an off-line, centralized QoS allocation scheme that assigns a strict priority ordering based on a flow's QoS requirement, giving precedence to stringent-QoS applications over less stringent ones. A service class assignment is then chosen such that this precedence condition is satisfied at every switch for every flow. A more detailed description can be found in [1]. The reservation algorithm is used as a reference point in evaluating the relative performance of Algorithm 2. We observe that the former also satisfies the end-to-end packet loss rate requirements for all flows. However, one noticeable difference with Algorithm 2 is the high violation penalty that flows 1 and 8 incur under the reservation scheme.

4.2.2 Dynamical Properties of Algorithm 2

Figure 4.2 (top) shows the end-to-end packet loss trace of flow 1 which travels through both the bottleneck links (n_{11}, n_0) and (n_1, n_{17}). We observe that there are three brief instances when the QoS requirement is violated. Figure 4.2 (middle) shows that on switch n_{11} flow 1 is assigned to service class 2 except for two brief moments when it is assigned to class 1. It turns out that class 2 is the inferior service class incurring a packet loss rate of about 0.04. Figure 4.2 (bottom) shows the service class assignment trace for flow 1 on switch n_1. We observe that although flow 1 is most of the time assigned to the superior service class—here it is class 1—occasionally there are excursions to service class 2 which, in turn, are correlated to the elevation in the end-to-end packet loss rate.

The reason for the temporary excursions is that when the QoS Agent at n_1 notices that the end-to-end QoS delivered is overly good for flow 1, then service class 2 becomes the locally optimal choice for flow 1, but this reassignment is not sustainable in the long run without violating flow 1's QoS requirement. Thus the switch back to service class 1 momentarily thereafter.

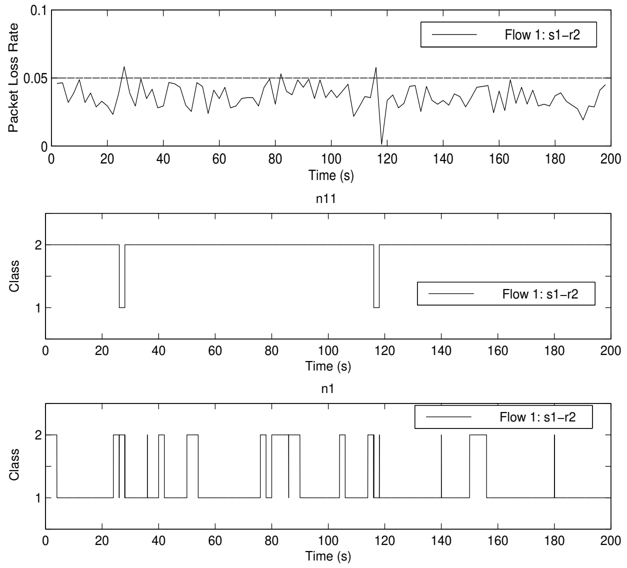

Figure 4.2. Dynamics of flow 1. Top: Packet loss trace. Middle: Service class assignment trace on (n_{11}, n_0). Bottom: Service class assignment trace on (n_1, n_{17}).

4.2.3 Impact of the Number of Service Classes

The number of service classes and the values of their service weights are important design parameters in our QoS provision architecture. If there are "too few" service classes and the QoS requirements of the user population are diverse, then, in spite of the resources being sufficient, it is possible that QoS violations may arise due to stringent and non-stringent QoS applications being assigned to the same service class (a form of the *pigeon hole principle*). Table 2 shows the QoS requirement make-up of a subset of connections—flows 1, 8, and 10—whose stringency and diversity has been increased from what they were before. We observe that in the 2 service classe case, our architecture is unable to find a service class assignment across the switches that satisfies the QoS requirements of all three flows. Indeed, it is not difficult to show that unless the service weights are allowed to be changed, satisfying all three QoS requirements is impossible. Table 2 also shows that increasing the number of service classes from 2 to 3/4 helps resolve this problem. Having 4 service classes does not further improve system performance but neither does it deteriorate it.

Flow	Pls. Req.	Four Classes		Three Classes		Two Classes	
		pls.	vio.	pls.	vio.	pls.	vio.
1	.005	.0044	.18	.0046	.14	.0371	.98
8	.025	.0116	.26	.0119	.28	.0390	.97
10	.060	.0541	.38	.0540	.37	.0238	.00

Table 2. Effect of number of service classes on QoS.

There are a number of performance results including adaptability to nonstationary changes, pricing and cost efficiency, user population diversity, multi-dimensional QoS vectors (e.g., delay and jitter), problem instance generation and robustness which have been omitted due to space constraints. We refer the reader to [1] for a more detailed exposition.

References

[1] S. Chen and K. Park. An architecture for noncooperative QoS provision in many-switch systems. Technical Report CSD-TR-98-013, Department of Computer Sciences, Purdue University, 1998.

[2] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang. Pricing in computer networks: motivation, formulation, and example. *IEEE/ACM Trans. Networking*, 1(6):614–627, 1993.

[3] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE J. Select. Areas Commun.*, 13(6):1048–1056, 1995.

[4] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking: Res. Exper.*, 1:3–26, 1990.

[5] A. Elwalid and D. Mitra. Effective bandwidth of general Markovian traffic sources and admission control in high speed networks. *IEEE/ACM Trans. Networking*, 1(3), 1993.

[6] D. Ferguson, C. Nikolaou, and Y. Yemini. An economy for flow control in computer networks. In *Proc. IEEE INFOCOM '89*, pages 110–118, 1989.

[7] L. Georgiadis, R. Guérin, V. Peris, and K. Sivarajan. Efficient network QoS provisioning based on per node traffic shaping. *IEEE/ACM Transactions on Networking*, 4(4):482–501, 1996.

[8] Y. Korilis and A. Lazar. On the existence of equilibria in noncooperative optimal flow control. *Journal of the ACM*, 42(3):584–613, 1995.

[9] Y. Korilis, A. Lazar, and A. Orda. Achieving network optima using Stackelberg routing strategies. *IEEE/ACM Trans. Networking*, 5(1):161–173, 1997.

[10] J. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Trans. on Computers*, 38(5):705–717, 1989.

[11] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2:1–15, 1994.

[12] S. Low and P. Varaiya. A new approach to service provisioning in ATM networks. *IEEE/ACM Trans. Networking*, 1(5):547–553, 1993.

[13] S. Low and P. Varaiya. An algorithm for optimal service provisioning using resource pricing. In *Proc. IEEE INFOCOM '94*, pages 368–373, 1994.

[14] J.-F. Mergen. Personal communication.

[15] R. Nagarajan and J. Kurose. On defining, computing and guaranteeing quality-of-service in high-speed networks. In *Proc. IEEE INFOCOM '90*, pages 2016–2025, 1992.

[16] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Trans. Networking*, 1(5):510–521, 1993.

[17] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM Trans. Networking*, 2(2):137–150, 1994.

[18] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proc. IEEE International Conference on Network Protocols*, pages 171–180, October 1996.

[19] K. Park, M. Sitharam, and S. Chen. Quality of service provision in noncooperative networks: heterogeneous preferences, multi-dimensional QoS vectors, and burstiness. To appear in *Proc. 1st International Conference on Information and Computation Economies*, 1998.

[20] J. Sairamesh, D. Ferguson, and Y. Yemini. An approach to pricing, optimal allocation and quality of service provisioning in high-speed networks. In *Proc. IEEE INFOCOM '95*, pages 1111–1119, 1995.

[21] S. Shenker. Making greed work in networks: a game-theoretic analysis of switch service disciplines. In *Proc. ACM SIGCOMM '94*, pages 47–57, 1994.

[22] D. Tennenhouse, J. Smith, D. Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. *IEEE Communications Magazine*, 35:80–86, 1997.