Providing Absolute Differentiated Services for Real-Time Applications in Static-Priority Scheduling Networks

Shengquan Wang, Dong Xuan, Riccardo Bettati, and Wei Zhao

Abstract— **In this paper, we propose and analyze a methodology for providing absolute differentiated services for real-time applications in networks that use static-priority schedulers. We extend previous work on worst-case delay analysis and develop a method that can be used to derive delay bounds without specific information on flow population. With this new method, we are able to successfully employ a utilization-based admission control approach for flow admission. This approach does not require explicit delay computation at admission time and hence is scalable to large systems. We assume the underlying network to use static-priority schedulers. We design and analyze several priority assignment algorithms, and investigate their ability to achieve higher utilization bounds. Traditionally, schedulers in differentiated services networks assign priorities on a class-by-class basis, with the same priority for each class on each router. In this paper, we show that relaxing this requirement, that is, allowing different routers to assign different priorities to classes, achieves significantly higher utilization bounds.**

Keywords— **absolute differentiated services, static-priority scheduling, utilization-based admission control, priority assignment, delay bound.**

I. Introduction

The differentiated service (`DiffServ`) Internet model is aimed at supporting service differentiation for aggregated traffic in a scalable manner. Many approaches have been proposed to realize this model. At one end of the spectrum, *absolute* differentiated services [16], [17], [19] seek to provide `IntServ`-type end-to-end absolute performance guarantees without per-flow state in the network core. The user receives an absolute service profile (e.g., guarantees on bandwidth, or end-to-end delay). For example, assuming that no dynamic routing occurs, the *premium service* can offer the user a performance level that is similar to that of a leased line, as long as the user's traffic is limited to a given bandwidth [16]. At the other end of spectrum, *relative* differentiated services seek to provide per-hop, per-class relative services [9]. Consequently, the network cannot provide end-to-end guarantees. Instead, each router only guarantees that the service invariant is *locally* maintained, even though the absolute *end-to-end* service might vary with networking conditions.

Many real-time applications, such as, Voice over IP, DoD's C4I, or industrial control systems, demand efficient and effective communication services. In this context, by *real-time* we mean that a packet is delivered from its source to the destination within a predefined end-to-end deadline. Packets delivered beyond these end-to-end deadlines are considered useless. Within a differentiated-services framework, real-time applications must rely on *absolute* differentiated services in order to have a guarantee on the end-to-end delay. Consequently, in this paper, we will focus on a quality-of-service (QoS) architecture that provides end-to-end absolute differentiated services.

Progress has been made to provide absolute differentiated services for real-time applications in networks with rate-based scheduling algorithms [19]. In this paper, we consider networks that use *static-priority* schedulers. This type of scheduler is supported in many current routers, and our approaches can therefore be easily realized within existing networks.

In order to provide service guarantees, an admission control mechanism has to be in place, which makes sure that enough sources are available to satisfy the requirements of both the new and the existing connections after the new connection has been admitted. In order to keep steps with the scalability requirements for differentiated services networks, any admission control mechanism must be light-weight so that it can be realized in a scalable fashion. We show how, through appropriate system (re-)configuration steps, the delay guarantee test at run time is reduced to a simple utilization-based test: As long as the utilization of links along the path of a flow is not beyond a given bound, the performance guarantee can be met. The value of the utilization bound is verified at system (re-)configuration time. Once verified, the use of this utilization bound is relatively simple at flow admission time: Upon the arrival of a flow establishment request, the admission control admits the flow if the utilization values of links along the path of the new flow are no more than the bound. Thus, this approach (called Utilization-Based Admission Control – UBAC – in the following) eliminates explicit delay analysis at admission time, and renders the system scalable.

Utilization-based admission control is not new to networks. The fluid-flow model in the `IntServ` framework, for example, allows various forms of utilization based admission control [18]. Such approaches cannot be used in a `DiffServ` framework, however, because they rely on guaranteed-rate schedulers, which need to maintain flow information. The challenge of using the UBAC method is how to verify the correctness of a utilization bound at the configuration time. Obviously, the verification will have to rely on a delay analysis method. We will follow the approach proposed by Cruz [6] for analyzing delays. Cruz's approach must be adapted to be applicable in a flow-unaware environment. The delay analysis proposed in [6] depends on the information about flow population, i.e., the number of flows at input links and the traffic characteristics (e.g., the average rate and burst size) of flows. In our case the delay analysis is done at system configuration time when the information about flow population is not available. We will develop a method that allows us to analyze delays without depending on the dynamic

The authors are with the Department of Computer Science, Texas A&M University, College Station, TX 77843. E-mail: {swang, dxuan, bettati, zhao}@cs.tamu.edu .

information about flows.

Priority Assignment is an inherent issue in the networks with static priority scheduling. As priority assignment has direct impact on the delay performance of individual packets, it must be carefully addressed. In the DiffServ domain, applications are differentiated by their *classes*. Accordingly, many previous studies assume that priorities are assigned on a class basis only, where all the flows in a class are assigned the same priority [4]. We study more generalized priority assignment algorithms, where the flows in a class may be assigned different priorities and flows from different classes may share a same priority. While the proposed algorithms are still relatively simple and efficient, we find that they are effective in achieving higher utilization bounds.

The rest of the paper is organized as follows. In Section II we describe previous related work. The underlying network and traffic models for this study are introduced in Section III. In Section IV, we introduce our proposed architecture for providing absolute differentiated services in networks with static-priority scheduling. In Section V, we derive a delay computation formula that is insensitive to the flow population. In Section VI, we discuss our heuristic algorithms for priority assignment. In Section VII, we illustrate with extensive experimental data that the utilization achieved by our new algorithms is much higher than traditional methods. A summary of this paper and motivation of future work are given in Section VIII.

II. Previous Works

A good survey on recent work in absolute differentiated services and relative differentiated services has been done in [17]. Here, we compare our work with others from the view point of providing absolute differentiated services. Nichols et. al. [16] propose the *premium service* model, which provides the equivalent of a dedicated link between two access routers. It provides absolute differentiated services in priority-driven scheduling networks with two priorities, in which the high priority is reserved for premium service. The algorithm in [7] provides both guaranteed and statistical rate and delay bounds, and addresses scalability through traffic aggregation and statistical multiplexing. Stoica and Zhang describe an architecture to provide guaranteed service without per-flow state management by using a technique called *dynamic packet state* (DPS) [19]. Our work is based on static priority scheduling algorithm, which is relatively simple and widely supported.

Admission control has been investigated widely [8], [10], [15]. The various approaches differ from each other in that they may require different scheduling schemes and so can be of vastly different complexity. For example, traditional admission control in networks with static priority scheduling is very complicated. Due to absence of flow separation, for any new flow request, admission control needs to explicitly compute and verify delays for the new and existing flows. This procedure is very expensive with increasing numbers of flows. The utilization-based admission control adopted dramatically reduces this complexity.

In its basic form, UBAC was first proposed in [14] for preemptive scheduling of periodic tasks on a simple processor. A number of utilization-based tests are known for centralized systems (e.g., 69% and 100% utilization bounds for periodic tasks on a single server using rate-monotonic and earliest-deadline-first scheduling, respectively[14]), or distributed systems (such as 33% for synchronous traffic over FDDI networks [23]). In this paper, we adopt utilization-based tests in providing differentiated services in static priority scheduling networks.

Flow-population-insensitive delay analysis has been recently studied in [3] for the case of aggregate scheduling. Lower bounds on the worst-case delay are derived. These bounds are a function of network utilization, maximum hop count of any flow, and the shaping parameters at the entrance to the network. The work in [3] only considers FIFO scheduling. Also, delay bounds are not tight, although almost independent of the network topology. In this paper, we will derive a better delay bound in static-priority scheduling networks.

This paper focuses on priority assignment in static priority scheduling networks for real-time communication applications within DiffServ domains. In [6], Cruz proposed a two-priority assignment scheme for a ring network. The work in [13] described and examined various priority assignment methods for ATM networks. Our work is the very first on priority assignment for absolute differentiated services.

III. Network and Traffic Models

In this section, we describe the model and define the terminology that will be used in the rest of this paper.

A. Network Model

The DiffServ architecture distinguishes two types of routers: *Edge routers* are located at the boundary of the network, and provide support for traffic policing. *Core routers* are inside the network. A router is connected to other routers or hosts through its input and output links. For the purpose of delay computation, we follow standard practice and model a router as a set of *servers*, one for each router component, where packets can experience delays. Packets are typically queued at the output buffers, where they compete for the output link. We therefore model a router as a set of output *link servers*. All other servers (input buffers, non-blocking switch fabric, wires, etc.) can be eliminated from the delay analysis by appropriately subtracting constant delays incurred on them from the deadline requirements of the traffic. We assume there are L_k input links for Server k, and all output links are of capacity C, in bits per second. Consequently, the network can be modeled as a graph with V connected link servers. The link servers in the server graph are connected through either links in the network or paths within routers, which both make up the set of edges in the graph.

B. Traffic Model

We call a stream of packets between a sender and receiver a *flow*. Packets of a flow are transmitted along a single path, which we model as a sequence of link servers. Following the DiffServ architecture, flows are partitioned into classes. QoS requirements and traffic specifications of flows are defined on a class-by-class basis. We use M to denote the total number of classes in a network. We assume that at each link server, a certain portion of bandwidth is reserved for each traffic class separately. Let α_k^i denote the portion of bandwidth reserved for Class i at Server k. We assume static-priority schedulers with support for N distinct priorities in the routers. The bandwidth assigned to Class i at Server k is further partitioned into portions $\alpha^i_{p,k}$, one for each priority p traffic of Class i at that server. We note that $\alpha^i_k = \sum_{q=1}^{N} \alpha^i_{q,k}$ (the question of how much bandwidth to assign to each priority will be discussed in Section VI.). We aggregate flows into *group of flows*. All flows of Class i with priority p going through Server k from input link j form a group of flows $G^i_{p,j,k}$ and all flows of all classes with priority p going through Server k from input link j form a group of flows $G_{p,j,k}$.

In order to appropriately characterize traffic both at the ingress router and within the network, we use a general traffic descriptor in form of traffic functions and their time independent counterpart, constraint traffic functions [6].

Definition 1: The *traffic function* $f(t)$ is defined as the amount of the traffic in a group of flows during time interval $[0, t)$. The function $F(I)$ is called the *traffic constraint function* of $f(t)$ if

$$f(t + I) - f(t) \leq F(I), \tag{1}$$

for any $t > 0$ and $I > 0$. In this paper, we use $F^i_{p,j,k}$ and $F_{p,j,k}$ to express the traffic constraint function for group of flows $G^i_{p,j,k}$ and group of flows $G_{p,j,k}$ respectively.

We assume that the source traffic of a flow in Class i is controlled by a leaky bucket with burst size σ^i and average rate ρ^i. Define $H^i(I)$ as the *source traffic function* for any class-i traffic flow, which is constrained at the entrance to the network by

$$H^i(I) = \min\{CI, \sigma^i + \rho^i I\}. \tag{2}$$

Since the QoS requirements of flows (in our case, end-to-end delay requirements) are specified on a class-by-class basis as well, we can, where we define the end-to-end deadline requirement of class-i traffic to be D^i and use a triple $\langle \sigma^i, \rho^i, D^i \rangle$ to represent class-i traffic. As no distinction is made between flows belonging to the same class, all flows in the same class are guaranteed the same delay. In the following, we use $d_{p,k}$ to denote the local worst-case delay suffered by flows with priority p at Server k.

IV. A QoS Architecture for Absolute Differentiated Services

In this section, we propose an architecture to provide absolute differentiated services in static priority scheduling networks. This architecture consists of three major modules:

- Utilization bound verification: In order to allow for a utilization-based admission control to be used at run time, safe utilization levels on all links must be determined during system configuration. Using a flow-population-insensitive delay computation method, a delay upper bound is determined for each priority traffic at each router. This module then verifies whether the end-to-end delay bound in each feasible path of the network satisfies the deadline requirement, as long as the bandwidth usage on the path is within a pre-defined limit – the utilization bound. This is also the point when priorities are assigned within classes and when bandwidth is assigned to classes and to priorities. We will discuss bandwidth and priority assignment algorithm later.
- Utilization-based admission control: Once safe utilization levels have been verified at configuration time, the admission control only needs to check if the necessary bandwidth is available along the path of the new flow.
- Packet forwarding: In a router, packets are transmitted according to their priorities, which can be derived from the (possibly extended) class identifier in the header. Within the same priority, packets are served in FIFO order.

While utilization-based admission control significantly reduces the admission control overhead compared to traditional approaches that require explicit delay computation, excessive connection establishment activity can still add substantial strain to the admission control components. In [5] we describe ways to distribute the load for admission control by appropriately pre-allocating resources and give the control to ingress nodes to the domain.

In the rest of this paper, we will focus on flow-population-insensitive delay computation analysis and on priority assignment.

V. Flow-Population-Insensitive Delay Computation

In this section, we will present a new delay computation formula, which is insensitive to flow population. We then discuss the approach with which this delay formula is derived.

A. Main Result

Since static priority scheduling does not provide flow separation, the local delay at a server depends on detailed information (number and traffic characteristics) of other flows both at the server under consideration and at servers upstream. Therefore, all the flows *currently* established in the network must be known in order to compute delays. Delay formulas for this type of systems have been derived for a variety of scheduling algorithms [13]. While such formulas could be used (at quite some expense) for flow establishment at system run time, they are not applicable for delay computation during configuration time, as they rely on information about flow population.

As this information is not available at configuration time, the worst-case delays must be determined assuming a worst-case combination of flows. Fortunately, the following theorem gives an upper bound on this worst-case delay without having to exhaustively enumerate all the flow combinations.

Theorem 1: The worst-case queuing delay $d_{p,k}$ suffered by traffic with priority p at Server k is bounded by [1]

$$d_{p,k} \leq \frac{1}{\|\hat{\boldsymbol{\alpha}}_{p,k}\|} \sum_{q=1}^{p} \omega_{q,k}(\boldsymbol{\alpha}_{q,k} \cdot \boldsymbol{Z}_{q,k}), \tag{3}$$

[1] In the following discussion, we will rely heavily on vector notation, which is written in bold style. If the symbol a^i denotes some value specific to class-i traffic, then the notation $\mathbf{a}$ denotes the $M - dimensional$ vector $(a^1, a^2, \ldots, a^M)$. We will use the operator "$\cdot$" for the inner product and the operator "$\| * \|$" for the vector norm, i.e.,

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{M} a^i b^i, \quad \|\mathbf{a}\| = \sum_{i=1}^{M} |a^i|.$$

where

$$\omega_{q,k} = \begin{cases} 1, & q < p \\ \frac{L_k - \|\hat{\boldsymbol{\alpha}}_{p,k}\|}{L_k - \|\boldsymbol{\alpha}_{p,k}\|}, & q = p \end{cases}, \quad (4)$$

$$Z^i_{q,k} = \frac{\sigma^i}{\rho^i} + Y^i_{q,k}, \quad (5)$$

$$Y^i_{q,k} = \max_{\mathcal{R} \in S^i_{q,k}} \sum_{s \in \mathcal{R}} d_{q,s}, \quad (6)$$

for $q = 1, \ldots, p$, and

$$\|\hat{\boldsymbol{\alpha}}_{p,k}\| = 1 - \sum_{q=1}^{p-1} \|\boldsymbol{\alpha}_{q,k}\|. \quad (7)$$

$S^i_{q,k}$ is the set of all paths passed by the packets of Class i with priority q *before arriving at Server k*, then $Y^i_{q,k}$ is the maximum of the worst-case delays experienced by all flows of Class i with priority q before arriving at Server k. $\|\hat{\boldsymbol{\alpha}}_{p,k}\|$ is the available bandwidth for traffic with priority no higher than p.

Derivation of Inequality (3) will be discussed in Subsection V-B. At this point we would like to make the following observations on Theorem 1:

- In a previously derived delay computation formula in [4], there was an implicit limitation on the relationship of traffic classes and priorities: One traffic class can only have a single priority, and one priority can only be assigned to a single class traffic. The new delay formula removes this limitation, and gives more flexibility when differentiating service. Our priority assignment algorithms will take advantage of this flexibility to better utilize available resource.
- Usually a delay computation formula for a server would depend on the state of the server, i.e., the number of flows that are admitted and pass through the server. We note that Inequality (3) is independent from this kind of information and just depends on $\boldsymbol{\sigma}$, $\boldsymbol{\rho}$, $\boldsymbol{\alpha}_{q,k}$, and L_k. The values of these parameters are available at the time when the system is (re-)configured. Hence, the delay computation formula is insensitive to the flow population information.
- We define $\frac{\sigma^i}{\rho^i}$ as the *burst delay* for class-i traffic, which is the time for class-i traffic to get to burst size σ^i at the average rate ρ^i. The delay formula depends on the burst delay $\frac{\sigma^i}{\rho^i}$.
- We note that $d_{p,k}$ in Inequality (3) depends on $Y^i_{q,k}$. The value of $Y^i_{q,k}$, in turn, depends on the delays experienced at servers other than Server k. In general, we have a circular dependency. Hence, the delay values depend on each other and must be computed simultaneously. We use the $(N \times V)$-dimensional vector $\mathbf{d}$ to denote the upper bounds of the delays suffered by the traffic with all priorities at all servers:

$$\mathbf{d} = (d_{1,1}, d_{1,2}, \cdots, d_{1,V}, d_{2,1}, d_{2,2}, \cdots, d_{2,V}, \cdots, d_{N,1}, d_{N,2}, \cdots, d_{N,V}). \quad (8)$$

Define the right hand side of (3) as $\Phi_{p,k}(\mathbf{d})$, and then define

$$\boldsymbol{\Phi}(\mathbf{d}) = (\Phi_{1,1}(\mathbf{d}), \Phi_{1,2}(\mathbf{d}), \cdots, \Phi_{1,V}(\mathbf{d}), \Phi_{2,1}(\mathbf{d}), \Phi_{2,2}(\mathbf{d}), \cdots, \Phi_{2,V}(\mathbf{d}), \cdots, \Phi_{N,1}(\mathbf{d}), \Phi_{N,2}(\mathbf{d}), \cdots, \Phi_{N,V}(\mathbf{d})). \quad (9)$$

The queuing delay bound vector $\mathbf{d}$ can then be determined by iteratively solving the following vector equation:

$$\mathbf{d} = \boldsymbol{\Phi}(\mathbf{d}). \quad (10)$$

In some special cases, closed-form solutions for the delay can be derived. This is the case, for example, in a network with a single real-time traffic class that is assigned a single priority in a network of identical servers and identical allocations of bandwidth to the class on all servers (in this case, we simplify the notation to let $\sigma = \sigma^1, \rho = \rho^1, Y_k = Y^1_{1,k}, L = L_k$, and $\alpha = \alpha^1_{1,k}$). The following corollary shows how a delay bound can be computed if we loose the bound on Y_k.

Corollary 1: Suppose d is the worst-case delay bound across any node in the network, and the path of any flow in the network traverses at most h nodes, then we have $Y_k \leq (h-1)d$. If $\alpha < \frac{1}{1+(h-2)(1-\frac{1}{L})}$, then

$$d \leq \frac{1}{\frac{1}{\alpha}\frac{L-\alpha}{L-1} - (h-1)} \frac{\sigma}{\rho}. \quad (11)$$

Therefore the end-to-end delay d^{e2e} can be bounded by

$$d^{e2e} \leq \frac{h}{\frac{1}{\alpha}\frac{L-\alpha}{L-1} - (h-1)} \frac{\sigma}{\rho}. \quad (12)$$

This delay formula does not depend on topology of the network except for the length h of the longest flow path. We note that a very similar result was derived using a different approach in [3].

B. Deriving the Delay Formula

In this subsection, we discuss how to derive the delay formula given in (3). We will start with a formula for delay computation that depends on flow population, which we call *the general delay formula*. We will describe how to remove its dependency on information of flow population.

For Server k, suppose that the group of flows $G^i_{p,j,k}$, at some time moment, has $n^i_{p,j,k}$ traffic flows. The constraint function $F^i_{p,j,k}(I)$ can be formulated as the summation of the constraint functions of individual flows, that is,

$$F^i_{p,j,k}(I) = \sum\nolimits_{x \in G^i_{p,j,k}} F_x(I), \quad (13)$$

where $F_x(I)$ is the constraint function for flow x in $G^i_{p,j,k}$. Further, the aggregate traffic of group of flows $G_{p,j,k}$ is constrained by

$$F_{p,j,k}(I) = \min\{CI, \sum_{i=1}^{M} F^i_{p,j,k}\}. \quad (14)$$

The worst-case delay $d_{p,k}$ of priority-p flows at Server k can then easily be formulated in terms of the aggregated traffic constraint functions and the service rate C of the server as follows [13]:

$$d_{p,k} = \frac{1}{C} \max_{I>0} (\sum_{q=1}^{p-1} \sum_{j=1}^{L_k} F_{q,j,k}(I + d_{p,k}) + \sum_{j=1}^{L_k} F_{p,j,k}(I) - CI). \quad (15)$$

Substituting (13) and (14) into (15), we observe that the above delay formula depends on flow population. In fact, (15) depends on $n^i_{p,j,k}$, the number of flows in $G^i_{p,j,k}$, and on the traffic constraint functions $F_x(I)$ of the individual flows. This kind of dependency on the dynamic system status must be removed in order to perform delay computations at configuration time.

In the following sections, we describe how we first eliminate the dependency on the traffic constraint functions. Then we eliminate the dependency on the number of flows on each input link. The result is a delay formula that can be applied without knowledge about flow population.

B.1 Removing the Dependency on Individual Traffic Constraint Functions

We now show that the aggregated traffic function $F^i_{p,j,k}(I)$ can be bounded by replacing the individual traffic constraint functions $F_x(I)$ by a common upper bound, which is independent of input link j.

The delay on each server can now be formulated without relying on traffic constraint functions within the network of individual flows. The following theorem in fact states that the delay for each flow on each server can be computed by using the constraint traffic functions *at the entrance to the network only*.

Theorem 2: The aggregated traffic of the group of flows $G_{p,j,k}$ is constrained by

$$F_{p,j,k}(I) = \begin{cases} CI, & I \leq \tau_{p,j,k} \\ \mathbf{n}_{p,j,k} \cdot (\boldsymbol{\eta}_{p,k} + \boldsymbol{\rho} I), & I > \tau_{p,j,k} \end{cases}, \qquad (16)$$

where

$$\tau_{p,j,k} = \frac{\mathbf{n}_{p,j,k} \cdot \boldsymbol{\eta}_{p,k}}{C - \mathbf{n}_{p,j,k} \cdot \boldsymbol{\rho}}, \qquad (17)$$

$$\eta^i_{p,k} = \sigma^i + \rho^i Y^i_{p,k}, \qquad (18)$$

and the worst-case delay $d_{p,k}$ of priority-p flows at Server k can be bounded by

$$d_{p,k} \leq \frac{U_{p,k} - V_{p,k} W_{p,k}}{X_{p,k}}, \qquad (19)$$

where

$$U_{p,k} = \sum_{q=1}^{p} \mathbf{n}_{q,k} \cdot \boldsymbol{\eta}_{q,k}, \qquad (20)$$

$$V_{p,k} = C - \sum_{q=1}^{p} \mathbf{n}_{q,k} \cdot \boldsymbol{\rho}, \qquad (21)$$

$$X_{p,k} = C - \sum_{q=1}^{p-1} \mathbf{n}_{q,k} \cdot \boldsymbol{\rho}, \qquad (22)$$

$$W_{p,k} = \max_{j=1}^{L_k} \{ \frac{\mathbf{n}_{p,j,k} \cdot \boldsymbol{\eta}_{p,k}}{C - \mathbf{n}_{p,j,k} \cdot \boldsymbol{\rho}} \}, \qquad (23)$$

and

$$n^i_{q,k} = \sum_{j=1}^{L_k} n^i_{q,j,k}. \qquad (24)$$

The proof of Theorem 2 is given in Appendix A.

The delay computation using Equation (19) still depends on the number of flows on all input links. In the next section, we describe how to remove this dependency.

B.2 Removing the Dependency on the Number of Flows on Each Input Link

As we described earlier, admission control at run-time makes sure that the utilization of Server k allocated to flows of Class i with priority p does not exceed $\alpha^i_{p,k}$. In other words, the following inequality always holds:

$$n^i_{p,k} \rho^i \leq \alpha^i_{p,k} C. \qquad (25)$$

The number of flows on each input link is, therefore, subject to the following constraint:

$$\mathbf{n}_{p,k} \leq \boldsymbol{\gamma}_{p,k} C, \qquad (26)$$

where

$$\gamma^i_{p,k} = \frac{\alpha^i_{p,k}}{\rho^i}. \qquad (27)$$

To maximize the right hand side of (19), we should maximize $U_{p,k}$ and minimize $V_{p,k}$, $X_{p,k}$, and $W_{p,k}$. Under the constraint of (26), these parameters can be bounded for all possible distribution $n^i_{p,j,k}$ of numbers of active flows on all input links, as the following theorem shows:

Theorem 3: If the worst-case queuing delay is experienced by the traffic with priority p at Server k, then,

$$U_{p,k} \leq \sum_{q=1}^{p} (\boldsymbol{\alpha}_{q,k} \cdot \boldsymbol{Z}_{q,k}) C, \qquad (28)$$

$$V_{p,k} \geq (1 - \sum_{q=1}^{p} \|\boldsymbol{\alpha}_{q,k}\|) C, \qquad (29)$$

$$X_{p,k} \geq (1 - \sum_{q=1}^{p-1} \|\boldsymbol{\alpha}_{q,k}\|) C, \qquad (30)$$

and

$$W_{p,k} \geq \frac{\boldsymbol{\alpha}_{p,k} \cdot \boldsymbol{Z}_{p,k}}{L_k - \|\boldsymbol{\alpha}_{p,k}\|}. \qquad (31)$$

where $U_{p,k}$, $V_{p,k}$, $X_{p,k}$, and $W_{p,k}$ are defined in (20), (21), (22), and (23).

The proof of Theorem 3 is given in Appendix B.

If we substitute all the bounds in (28), (29), (30), and (31) into (19), then (3) follows after some algebraic manipulation.

VI. Priority Assignment

The delay computation formulas described in the previous section allow to assign priorities to flows independently of their classes. With appropriate priority assignment algorithms in place, network resources can be significantly better utilized.

Ideally, the priority assignment would be done during the admission control for a new flow, where resource usage can be taken into consideration. This would, however, render the admission control procedure significantly more expensive. We therefore follow the procedure we used earlier for delay verification, and perform the priority assignment off-line, that is, during system configuration.

In order to assign priorities to flows off-line, we must classify and aggregate flows using information (in addition to class membership) that is available before run time. For a network with fixed routers, flows can be classified at each server by their class identification, the source and the destination identification.

In the following we use class and path (in form of source and destination identification) information to assign priorities, where all flows in the same class with the same source and destination have the same priority. This approach has two advantages over more dynamic ones. First, the priority assignment can be done before run time and thus does not burden the admission control procedure at establishment time. Second, the static-priority schedulers need no dynamic information at run time, as the priority mapping for each packet is fully defined by its class identification, and its source and destination identifications. No additional fields in packet headers are needed.

A. Outline of Algorithms

Mapping with increasing complexity can be used to assign priorities to flows:

- One-to-One Mapping: All the flows in a class are assigned the same priority. Flows in different classes are mapped into different priorities. A simple deadline-based mapping can be used to assign priorities to classes with the least deadline getting the highest priority. The advantage of this method is its simplicity. Obviously, this does not take into account more detailed information such as topology and others. We use this mapping as baseline for the comparison with others.
- One-to-Many Mapping: Classes may be partitioned into subclasses for priority assignment purposes, with flows from a class assigned different priorities. Flows in different classes, however, may not share a priority. In Subsection VI-B we present a version of this algorithm. This algorithm can recognize the different requirements of flows in a class and assign them different priorities, hence improving the network performance. The algorithm is still relatively simple, but it may use too many priorities given that it does not allow priorities to be shared by flows from different classes.
- Many-to-Many Mapping: The priority assignment is not constrained by class membership, and flows from different classes can be assigned the same priority. Given its generality, this mapping can achieve better performance than the other two.

B. Details of Algorithms

We will first focus on Algorithm One-to-Many. We will then show that Algorithms One-to-One and Many-to-Many are a special case and generalization of Algorithm One-to-Many, respectively.

Given the limited space, there will be no need to present the other two algorithms in details. The purpose of the static priority assignment algorithm is to generate a *priority assignment table,* which then is used by admission control and is loaded into routers for scheduling purposes. The priority assignment table (see Table I for an example) consists of entries of type $\langle class, source, destination, priority\rangle$. The priority assignment then maps from the first three fields in the entry to the *priority* field.

TABLE I

A EXAMPLE OF PRIORITY ASSIGNMENT TABLE

Class	Source	Destination	Priority
1	node 2	node 3	2
1	node 4	node 7	3
⋮	⋮	⋮	⋮
3	node 6	node 1	1

```
input:  network server graph, flow traffic and deadline
        parameters for all the classes, assigned network
        bandwidth $\alpha_k^i$ for each Class $i$ ($i = 1, \ldots, M$).
output: the priority assignment table and bandwidth
        allocation $\alpha_{p,k}^i$.
1. initialize the priority assignment table, by filling the proper
   class id, source id, and destination id. Initialize the priority
   fields to "undefined".
2. for $i$ from $M$ down to 1 do
       combine all entries of type $\langle i, src, dst, p\rangle$ of Class $i$ into
       subset $S_i$ and push subset $S_i$ onto Stack $SS$;
3. $p = 0$;        /* highest priority */
4. while Stack $SS$ is not empty
       4.1. $p = p + 1$;
       4.2. if $p > N$          /* no more priorities available */
                return "failure";
       4.3. pop a subset $S$ from Stack $SS$;
       4.4. assign $p$ to the priority field of all the entries in $S$;
       4.5. use delay Formula (10) to update the end-to-end
            delay of flows represented by entries in $S$;
       4.6. if no flow in $S$ misses its deadline
                continue;
       4.7. else
                4.7.1. if $S$ consists of a single entry
                           return "failure";
                4.7.2. else
                           4.7.2.1. call Procedure Bi-Partition($S$),
                                    and obtain two subsets: $S_x$, $S_y$
                           4.7.2.2. push $S_y$ and $S_x$ into Stack $SS$;
5. return the current priority assignment table and $\alpha_{p,k}^i$.
```

Fig. 1. Algorithm One-to-Many

Figure 1 shows our One-to-Many priority assignment algorithm. The algorithm uses a stack to store subsets of entries to which the priority fields are to be assigned. Entries in each subset can potentially assigned to the same priority. The subsets are ordered in the stack in accordance to their real-time requirements. The subset with entries that represent flows with the most tight deadline and/or laxity is at the top of the stack.

After its initialization, the algorithm works iteratively. At each iteration, the algorithm first checks whether enough unused priorities are available. If not, the program stops and declares "failure" (Step 4.2). Otherwise, a subset is popped from the stack. The algorithm then assigns the best (highest) available priority to the entries in the subset if the deadlines of the flows represented by those entries can be met. However, if some of the deadline tests cannot be passed, Procedure *Bi-Partition* is called to partition the entries in the subset into two subsets based on their laxity. The idea here is that if we assign a higher priority to entries with little laxity, we may pass the deadline tests for all entries. This is realized by pushing two new subsets into the stack in the proper order and by letting the future iteration deal with the priority assignment. Procedure *Bi-Partition* also assigns bandwidth to the different priorities in the class, that is, properly splits α_k^i to reflect the partitioned subsets.

The program iterates until either it exhausts all the subsets in the stack, in which case a successful priority assignment has been found and the program returns the assignment table or it must declare "failure". The latter happens when either the program runs out of priorities or it cannot meet the timing requirments for a single entry in a subset.

Because the size of a subset is halved at every iteration step, the worst-case time complexity of the algorithm is in the order of $\mathcal{O}(M \log V)$ in the number of delay computations. We will show that this algorithm does perform reasonably well in spite of its low time complexity.

Algorithm One-to-One is a special case of Algorithm One-to-Many presented in Figure 1. For Algorithm One-to-One, no subset partition is allowed (otherwise entries in one class will be assigned to different priorities — a violation of the One-to-One principle). Thus, if we modify the code in Figure 1, so that it returns "failure" whenever a failure on deadline test is found (Step 4.7), it becomes the code for Algorithm One-to-One.

On the other hand, we can generalize Algorithm One-to-Many to become Algorithm Many-to-Many. Recall that Algorithm Many-to-Many allows the priorities to be shared by flows in different classes. Note that sharing a priority is not necessary unless the priorities have been used up. Following this idea, we can modify the code in Figure 1 so that it becomes the code for Algorithm Many-to-Many: At Step 4.2, when it is discovered that all the available priorities have been used up, do not return "failure", but assign the entries with the priority that has just be used. In the case the deadline test fails, assign these entries with a higher priority (until the highest priority has been assigned).

VII. Experimental Evaluation

In this section, we evaluate the performance of the systems that use our new delay analysis techniques and priority assignment algorithms discussed in the previous sections. Recall that we use a utilization-based admission control in our study: As long as the link utilization along the path of a flow does not exceed a given bound, the end-to-end deadline of the flow is guaranteed. The value of this bound therefore gives a good indication about how many flows can be admitted by the network. We define the *maximum usable utilization* (MUU) to be summation of the bandwidth portions that can be allocated to real-time traffic in all classes, and use this metric to measure the performance of the systems. For a given network and a given priority assignment algorithm, the value for the MUU is obtained by performing a binary search in conjunction with the priority assignment algorithm discussed in Section VI.

To illustrate the performance of our algorithms for different settings, we describe two experiments. In the first experiment, we use a fixed network topology and compare the performance of the three algorithms presented in Section VI and measure how the algorithms perform for traffic with varying burstiness. In the second experiment, we measure how the three algorithms behave for networks with different topologies. In the following we describe the setup for the two experiments and discuss the results.

A. Experiment 1

The underlying network topology in this experiment in the classical MCI network topology. All links in the network have a capacity of 100 Mbps. All link servers in the simulated network use a static-priority scheduler with 8 priorities.

We assume that there are three classes of traffic: $\langle$ 640 bits, $32,000$ bps, 50 ms $\rangle$, $\langle$ $1,280$ bits, $64,000$ bps, 100 ms $\rangle$, and $\langle$ $1,920$ bits, $96,000$ bps, 150 ms $\rangle$, where each triple defines σ, ρ, and the end-to-end delay requirement for the class. Any pair of nodes in the simulated networks may request a flow in any class. All the traffic will be routed along shortest paths in terms of number of hops from source to destination. The results of these simulations are depicted in the first row of Table II. In the subsequent rows of the table, the same simulation results are depicted for higher-burtiness traffic. In each row, the burstiness parameter σ is quadrupled.

As expected, Table II shows that the MUU increases significantly with more sophisticated assignment algorithms. The performance improvement of algorithms One-to-Many and Many-to-Many over One-to-One remains constant for traffic with widely different burstiness.

TABLE II
The Comparison of MUU for Different Bursty Delay in the MCI Network

$\frac{\sigma}{\rho}$	Maximum Usable Utilization		
	One-to-One	One-to-Many	Many-to-Many
0.02 s	0.48	0.63	0.73
0.08 s	0.26	0.38	0.43
0.32 s	0.10	0.14	0.17
1.28 s	0.026	0.039	0.050

From Table II, we also see that the traffic burstiness heavily impacts on the MUU. In fact, for very bursty traffic the MUU can get quite low. We would like to point out that, even for very bursty traffic, sufficient amounts of bandwidth can still be designated for real-time traffic.

B. Experiment 2

In the second experiment we keep the setup of Experiment 1, except that we do not vary the burst delay $\frac{\sigma}{\rho}$ of the traffic. Instead, we vary the network topology. We randomly generate network topologies with GT-ITM [21] using the Waxman 2 method described there to generate edges. We generate 50 samples for each kind of networks with different number of nodes ranging from 10 to 20. We classify the generated topologies according to their size in number of nodes, and their diameter.

Figure 2 displays the values for MUU for for small networks (diameter of the networks is less than or equal to 6) and for large networks. We can make the following observations:

- We found that Algorithm Many-to-Many can always achieve the highest MUU among the three algorithms, and Algorithm One-to-Many can achieve higher utilization than Algorithm One-to-One, in the networks with the same number of nodes.

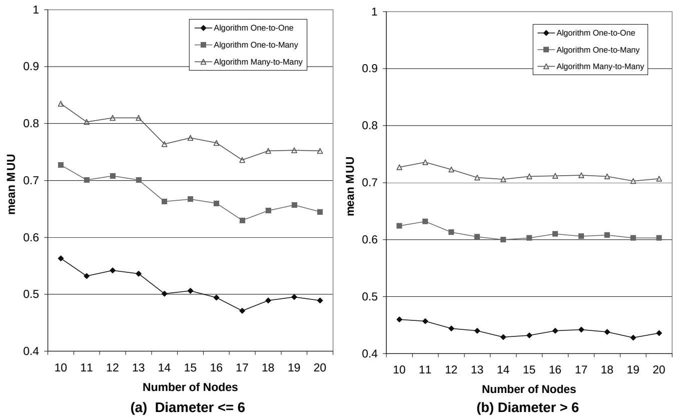

Fig. 2. The MUU Values of Randomly Generated Networks

For example, when the number of nodes is 15, for the case of $Diameter \leq 6$, the mean MUU of Algorithm Many-to-Many is 10.6% higher than that of Algorithm One-to-Many, and is 26.7% higher than that of Algorithm One-to-One. These observations can be explained by the fact that Algorithm Many-to-Many has the highest flexibility in assigning priorities among the three algorithms.

- The diameter of the network has an evident impact on the performance of all the priority assignment algorithms. As the size of the performance decreases. For example, when the the number of nodes is 15, the MUU of Algorithm One-to-Many in the case of $Diameter \leq 6$, is 7.4% higher than that in the case of $Diameter \leq 6$. The reason is that flows in large networks (in the sense of diameter) usually suffer larger end-to-end delay than in small networks.

VIII. Conclusions

In this paper, we have proposed a methodology for providing absolute differentiated services for real-time applications in networks that uses static priority schedulers. Given that static-priority schedulers are widely supported in current routers, we believe that this approach is a practical and effective to support real-time applications in the existing network.

We use a configuration-time verification step to determine safe utilization levels of servers, and so reduce the admission. Admission control at run time then is reduced to a sequence of simple utilization tests on the servers along the path of the new flow. Hence, the approach is scalable.

The configuration-time verification step relies on a flow-population-insensitive delay computation. We have extended Cruz's approach and developed a method that allows us to analyze the delays without depending on dynamic information about flow population. Furthermore, we have designed several priority assignment algorithms, which are shown to be effective in achieving high utilization bounds.

Extensive performance evaluation is made to the systems that used our new delay analysis techniques and priority assignment algorithms. We found that Algorithm Many-to-Many could achieve very high network utilization both in a well-known network and randomly generated networks.

Our methodology presented in this paper can be easily extended to deal with statistical delay guarantees. Much progress has been made in derivation of statistical delay bounds [11], [2], [12], [20]. However, all these previous results require information on flow population to obtain the statistical delay bounds. For example, in [11] statistical delay bounds are obtained by using approximated normal distribution, of which the parameters, in turn, depend on the flow population. Our method on eliminating flow-population dependency in delay computation can be applied in this situation to make delay derivation insensitive to flow population. This should help to provide absolute differentiated services to applications that require statistical delay guarantees.

Acknowledgements

This work was partially sponsored by NSF under contract number EIA-0081761, by DARPA under contract number F30602-99-1-0531, and by Texas Higher Education Coordinating Board under its Advanced Technology Program.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Service*, RFC 2474, December 1998.

[2] C. Chang, *Stability, Queue Length, and Delay of Deterministic and Stochastic Queueing Networks*, IEEE Transactions on Automatic Control, 39(5):913-931, May 1994.

[3] Anna Charny and J. Y. Le Boudec, *Delay Bounds in a Network with Aggregate Scheduling*, Proceedings of QOFIS, October 2000.

[4] B. Choi, D. Xuan, C. Li, R. Bettati and W. Zhao, *Scalable QoS Guaranteed Communication Services for Real-Time*, IEEE Proceedings of ICDCS, 2000.

[5] B. Choi and R. Bettati, *Endpoint Admission Control: Network Based Approach*, to appear in IEEE Proceedings of ICDCS, 2001.

[6] Rene L. Cruz, *A Calculus for Network Delay, Part I and Part II*, IEEE Transactions on Information Theory, Vol. 37. January 1991.

[7] Rene L. Cruz, *SCED+: Efficient Management of Quality of Service Guarantees*, IEEE Proceedings of INFOCOM'98, 1998.

[8] A. Dailianas and A. Bovopoulis, *Real-Time Admission Control Algorithms with Delay and Loss Guarantees in ATM networks*, IEEE Proceedings of INFOCOM'94, 1994.

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

[10] V. Firoiu, J. Kurose, and D. Towsley, *Efficient Admission Control for EDF Schedulers*, IEEE Proceedings of INFOCOM'97, 1997.

[11] E. Knightly, *Enforceable Quality of Service Guarantees for Bursty Traffic Streams*, IEEE Proceedings of INFOCOM'98, 1998.

[12] J. Kurose, *On computing Per-session Performance Bounds in High-Speed Multi-Hop Computer Networks*, Proceedings of ACM Sigmetrics'92, 1992.

[13] C. Li, R. Bettati, and W. Zhao, *Static Priority Scheduling for ATM Networks*, IEEE Proceedings of Real-Time Systems Symposium, 1997.

[14] C. L. Liu and J. W. Layland, *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*, J. ACM, Vol. 20, No. 1, 1973, pp.46-61.

[15] J. Liebeherr, D.E. Wrege, and D. Ferrari, *Exact Admission Control in Networks with Bounded Delay Services*, IEEE/ACM Transactions on Networking, 1996.

[16] K. Nichols, V. Jacobson, and L. Zhang, *A Two-bit Differentiated Services Architecture for the Internet*, Internet Draft, November 1997.

[17] R. Sivakumar, T. Kim, N. Venkitaraman and V. Bharghavan, *Achieving Per-Flow Weighted Rate Fairness in a Core Stateless Network*, IEEE Proceedings of ICDCS, 2000.

[18] S. Shenker, C. Partridge and R. Guerin, *Specification of Guaranteed Quality of Service*, RFC2212, September 1997.

[19] I. Stoica and H. Zhang, *Providing Guaranteed Services Without Per Flow Management*, Proceedings of ACM SIGCOMM, 1999.

[20] O. Yaron and M. Sidi, *Performance and Stability of Communication Networks via Robust Exponential Bounds*, IEEE/ACM Transactions on Networking, 1(3):372-385, June 1993.

[21] Eleen W. Zegura, etc., *http://www.cc.gatech.edu/projects/gtitm/*

[22] Z.-L. Zhang, D. Towsley, and J. Kurose, *Statistical Analysis of the Generalized Processor Sharing Scheduling Discipline*, IEEE Journal of Selected Areas in Communications, 13(6):1071-1080, August 1995.

[23] G. Agrawal, B. Chen, W. Zhao, and S. Davari, *Guaranteeing Synchronous Message Deadlines with the Timed Token Protocol,* IEEE Proceedings of ICDCS, 1992.

APPENDIX A: PROOF OF THEOREM 2

Theorem 2: The aggregated traffic of the group of flows $G_{p,j,k}$ is constrained by

$$F_{p,j,k}(I) = \begin{cases} CI, & I \leq \tau_{p,j,k} \\ \mathbf{n}_{p,j,k}\cdot(\boldsymbol{\eta}_{p,k} + \boldsymbol{\rho} I), & I > \tau_{q,j,k} \end{cases}, \tag{32}$$

where

$$\tau_{p,j,k} = \frac{\mathbf{n}_{p,j,k}\cdot\boldsymbol{\eta}_{p,k}}{C - \mathbf{n}_{p,j,k}\cdot\boldsymbol{\rho}}, \tag{33}$$

$$\eta^i_{p,k} = \sigma^i + \rho^i Y^i_{p,k}, \tag{34}$$

and the worst-case delay $d_{p,k}$ of priority-p flows at Server k can be bounded by

$$d_{p,k} \leq \frac{U_{p,k} - V_{p,k} W_{p,k}}{X_{p,k}}, \tag{35}$$

where

$$U_{p,k} = \sum_{q=1}^{p} \mathbf{n}_{q,k}\cdot\boldsymbol{\eta}_{q,k}, \tag{36}$$

$$V_{p,k} = C - \sum_{q=1}^{p} \mathbf{n}_{q,k}\cdot\boldsymbol{\rho}, \tag{37}$$

$$X_{p,k} = C - \sum_{q=1}^{p-1} \mathbf{n}_{q,k}\cdot\boldsymbol{\rho}, \tag{38}$$

$$W_{p,k} = \max_{j=1}^{L_k}\{\frac{\mathbf{n}_{p,j,k}\cdot\boldsymbol{\eta}_{p,k}}{C - \mathbf{n}_{p,j,k}\cdot\boldsymbol{\rho}}\}, \tag{39}$$

and

$$n^i_{q,k} = \sum_{j=1}^{L_k} n^i_{q,j,k}. \tag{40}$$

In order to prove Theorem 2, we need following lemmas:

Lemma 1: The aggregated traffic of the group of flows $G^i_{p,j,k}$ is constrained by

$$F^i_{p,j,k}(I) = \begin{cases} CI, & I \leq \tau^i_{p,j,k} \\ n^i_{p,j,k}(\eta^i_{p,k} + \rho^i I), & I > \tau^i_{p,j,k} \end{cases}, \tag{41}$$

where

$$\tau^i_{p,j,k} = \frac{n^i_{p,j,k}\eta^i_{p,k}}{C - n^i_{p,j,k}\rho^i}. \tag{42}$$

Proof: For any flow x in the group of flows $G^i_{p,j,k}$, let Y_x be the total worst-case queuing delay experienced by flow x before arriving at Server k. Suppose that $Y^i_{p,k}$ is the maximum of the worst-case queueing delays:

$$Y_x \leq Y^i_{p,k}. \tag{43}$$

Since $H^i(I)$ is the source traffic function of flow x, according to Theorem 2.1 in [6], we have

$$F_x(I) \leq H^i(I + Y_x) \leq H^i(I + Y^i_{p,k}). \tag{44}$$

We can, therefore, bound $F^i_{p,j,k}$ as follows:

$$F^i_{p,j,k}(I) \leq \sum\nolimits_{x \in G^i_{p,j,k}} F_x(I) \tag{45}$$

$$\leq \sum\nolimits_{x \in G^i_{p,j,k}} H^i(I + Y^i_{p,k}). \tag{46}$$

Substituting (2) into (46), we have

$$F^i_{p,j,k}(I) \leq \min\{n^i_{p,j,k}C, n^i_{p,j,k}(\eta^i_{p,k} + \rho^i I)\}. \tag{47}$$

On the other hand, the total amount of traffic that can be transmitted over input link j of Server k during any time interval I is constrained by the link capacity C, i.e.,

$$F^i_{p,j,k}(I) \leq CI. \tag{48}$$

Synthesizing (47) and (48), we verify the values of $F^i_{p,j,k}(I)$ and $\tau^i_{p,j,k}(I)$ as claimed. ■

Similarly, bounds can be defined for the aggregated traffic of group of flows $G_{p,j,k}$ as follows:

$$F_{p,j,k}(I) = \min\{CI, \sum\nolimits_{i=1}^{M} F^i_{p,j,k}\}. \tag{49}$$

Now we are ready to prove Theorem 2.

Proof: Note that each $F^i_{p,j,k}(I)$ is a two-piecewise linear continuous function, and $\sum_{i=1}^{M} F^i_{p,j,k}(I)$ is still a piecewise linear continuous function. The value $\tau^i_{p,j,k}$ identifies the intersection of the two linear segments, and is called the *flex point* of $F^i_{p,j,k}(I)$. All $\tau^i_{p,j,k}$'s are also flex points of $\sum_{i=1}^{M} F^i_{p,j,k}(I)$.

Notice that $\tau_{p,j,k} \geq \tau^i_{p,j,k}$ for all classes i, thus, (32) and (33) hold [2].

Following [13], assuming that a static priority scheduling discipline at the server, we have the following formula, which indicates how long an newly arrival packet of Class i with priority p can be delayed at Server k:

$$d_{p,k} = \frac{1}{C}\max_{I>0}(\sum_{q=1}^{p-1}\sum_{j=1}^{L_k} F_{q,j,k}(I + d_{p,k}) + \sum_{j=1}^{L_k} F_{p,j,k}(I) - CI). \tag{50}$$

The worst-case queuing delay $d_{p,k}$ suffered by the traffic with priority p at Server k will happen at [3]

$$I = \max_{j=1}^{L_k}\{\tau_{p,j,k}\}. \tag{52}$$

[2] If $I \leq \tau_{p,j,k}$, then $\sum_{i=1}^{M} F^i_{p,j,k}(I) \geq CI$; if $I > \tau_{p,j,k}$, then $\sum_{i=1}^{M} F^i_{p,j,k}(I) = \mathbf{n}_{p,j,k}\cdot(\boldsymbol{\eta}_{p,k} + \boldsymbol{\rho} I)$.

[3] Let $\tau_{p-,j,k}$ and $\tau_{p,j,k}$ be the flex points of traffic constraint function for traffic coming from the input link j of Server k with *priority higher than* p and with *priority* p, respectively. Further, let $T_{p-,j,k}$ be the maximum busy interval of the traffic constraint function for traffic coming from input link j of Server k with priority higher than p, and $\tau_{\max}$ is the maximum flex point for the total aggregate traffic in (50). Define

$$\tilde{\tau} = \max_{j=1}^{L_k}\{\tau_{p-,j,k}\},\ \hat{\tau} = \max_{j=1}^{L_k}\{\tau_{p,j,k}\}. \tag{51}$$

We know that $\tau_{\max} = \max(\tilde{\tau} - d_{p,k}, \hat{\tau})$. Here $\tilde{\tau} - d_{p,k} \leq 0$ since $d_{p,k} \geq T_{p-,j,k} \geq \tilde{\tau}$. So $\tau_{\max} = \hat{\tau}$. Let s be the slope of the aggregate traffic function. We find that $s \leq C$ if $I \leq \tau_{max}$; $s \geq C$ if $I \geq \tau_{max}$.

We can, therefore, eliminate the max operator from (50). By substituting (32), (33) and (52) into (50), with some algebraic manipulation we have

$$d_{p,k} = \frac{U_{p,k} - V_{p,k} W_{p,k}}{X_{p,k}}, \tag{53}$$

where $U_{p,k}$, $V_{p,k}$, $W_{p,k}$, and $X_{p,k}$ are defined in (36), (37), (38) and (39), respectively. ■

Appendix B: Proof of Theorem 3

Theorem 3: If the worst-case queuing delay is experienced by the traffic with priority p at Server k, then,

$$U_{p,k} \quad \leq \quad \sum_{q=1}^{p} (\boldsymbol{\alpha}_{q,k} \cdot \boldsymbol{Z}_{q,k}) C, \tag{54}$$

$$V_{p,k} \quad \geq \quad (1 - \sum_{q=1}^{p} \|\boldsymbol{\alpha}_{q,k}\|) C, , \tag{55}$$

$$X_{p,k} \quad \geq \quad (1 - \sum_{q=1}^{p-1} \|\boldsymbol{\alpha}_{q,k}\|) C, \tag{56}$$

and

$$W_{p,k} \quad \geq \quad \frac{\boldsymbol{\alpha}_{p,k} \cdot \boldsymbol{Z}_{p,k}}{L_k - \|\boldsymbol{\alpha}_{p,k}\|}. \tag{57}$$

where $U_{p,k}$, $V_{p,k}$, $X_{p,k}$, and $W_{p,k}$ are defined in (36), (37), (38) and (39).

In order to prove Theorem 3, we need the following lemma:

Lemma 2: The worst-case queuing delay at Server k by traffic of Class i with priority p is experienced when the number of flows $n_{p,k}^{i}$ is maximized, i.e. [4]

$$\mathbf{n}_{p,k} = \boldsymbol{\gamma}_{p,k} C. \tag{58}$$

Proof: By (50), we know that the larger $F_{q,j,k}(I)$, for $q = 1, \ldots, p$, the larger $d_{p,k}$. Furthermore, since $F_{q,j,k}(I)$ is the aggregated traffic of Class i with priority q at Server k, we know that the larger $n_{q,k}^{i}$, the larger $F_{q,j,k}(I)$. Therefore, when the number of flows on each link is maximized, then the traffic of Class i with priority p will experience the worst-case queuing delay at the server. ■

Now we are ready to prove Theorem 3.

Proof: Substituting (58) into (36), (37) and (38), we have

$$U_{p,k} \quad \leq \quad (\sum_{q=1}^{p} \boldsymbol{\gamma}_{q,k} \cdot \boldsymbol{\eta}_{q,k}) C, \tag{59}$$

$$V_{p,k} \quad \geq \quad C - (\sum_{q=1}^{p} \boldsymbol{\gamma}_{q,k} \cdot \boldsymbol{\rho}) C, \tag{60}$$

and

$$X_{p,k} \quad \geq \quad C - (\sum_{q=1}^{p-1} \boldsymbol{\gamma}_{q,k} \cdot \boldsymbol{\rho}) C. \tag{61}$$

since $\boldsymbol{\gamma}_{q,k} \cdot \boldsymbol{\eta}_{q,k} = \boldsymbol{\alpha}_{q,k} \cdot \boldsymbol{Z}_{q,k}$ and $\boldsymbol{\gamma}_{q,k} \cdot \boldsymbol{\rho} = \|\boldsymbol{\alpha}_{q,k}\|$, $U_{p,k}, V_{p,k}, X_{p,k}$ can be verified as claimed.

We can bound $W_{p,k}$ by $\widetilde{W}_{p,k}^{0}$, where $\widetilde{W}_{p,k}^{0}$ is the solution to the following optimization problem if we treat all variables $n_{p,j,k}^{i}$ as real numbers.

$$\text{Minimize} \quad \widetilde{W}_{p,k} = \max_{j=1}^{L_k} \{\tau_{p,j,k}\} \tag{62}$$

$$= \max_{j=1}^{L_k} \{ \frac{\mathbf{n}_{p,j,k} \cdot \boldsymbol{\eta}_{p,k}}{C - \mathbf{n}_{p,j,k} \cdot \boldsymbol{\rho}} \}, \tag{63}$$

$$\text{Subject to} \quad \mathbf{n}_{p,j,k} \geq \mathbf{0}, \;\; j = 1, \ldots, L_k, \tag{64}$$

$$\mathbf{n}_{p,k} = \boldsymbol{\gamma}_{p,k} C. \tag{65}$$

Without loss of generality, we assume that the input links are ordered according to the size of the flex points:

$$\widetilde{W}_{p,k}^{0} = \tau_{p,1,k} \geq \tau_{p,2,k} \geq \ldots \geq \tau_{p,L_k,k}. \tag{66}$$

- First, We can show that, when $\widetilde{W}_{p,k}$ reaches its optimal value $\widetilde{W}_{p,k}^{0}$, all inequalities in (66) will become *equalities*, i.e.,

$$\widetilde{W}_{p,k}^{0} = \tau_{p,1,k} = \tau_{p,2,k} = \ldots = \tau_{p,L_k,k}. \tag{67}$$

Otherwise, there exists some inequality $\tau_{p,j_0,k} > \tau_{p,j_0+1,k}$. It's easy to show that $\tau_{p,j,k}$ is an increasing continuous function with respect to any $n_{p,j,k}^{i}$. There must exist a nonzero $n_{p,j_0,k}^{i_0}$, then by choosing sufficiently small ϵ, decreasing $n_{p,j_0,k}^{i_0}$ by ϵ, increasing $n_{p,j_0+1,k}^{i_0}$ by ϵ, and also keeping them nonnegative, we have $\tau_{p,j_0,k} \geq \tau_{p,j_0+1,k}$. We notice that (66) is still true, but $\tau_{p,j_0,k}$ is decreased, thus $\tau_{p,j_0-1,k} > \tau_{p,j_0,k}$. Following this way, eventually we can decrease $\tau_{p,1,k}$. This contradicts to the assumption that $\tau_{p,1,k}$ is an optimal value.

- Second, applying the formula

$$\frac{x_1}{y_1} = \frac{x_2}{y_2} = \ldots = \frac{x_{L_k}}{y_{L_k}} = \frac{x_1 + x_2 + \ldots + x_{L_k}}{y_1 + y_2 + \ldots + y_{L_k}} \tag{68}$$

in (67), we have

$$\widetilde{W}_{p,k}^{0} = \frac{\sum_{j=1}^{L_k} \mathbf{n}_{p,j,k} \cdot \boldsymbol{\eta}_{p,k}}{\sum_{j=1}^{L_k} (C - \mathbf{n}_{p,j,k} \cdot \boldsymbol{\rho})} = \frac{\boldsymbol{\alpha}_{p,k} \cdot \boldsymbol{Z}_{p,k}}{L_k - \|\boldsymbol{\alpha}_{p,k}\|}. \tag{69}$$

As the values for the $n_{p,j,k}^{i}$'s in (39) are restricted to integers, $W_{p,k}$ is bounded as follows:

$$W_{p,k} \geq \widetilde{W}_{p,k}^{0} = \frac{\boldsymbol{\alpha}_{p,k} \cdot \boldsymbol{Z}_{p,k}}{L_k - \|\boldsymbol{\alpha}_{p,k}\|}. \tag{70}$$

■

[4] In general, $\gamma_{p,k}^{i} C$ is not necessarily an integer. However, in a modern practical system it is very large, we can assume that $\lfloor \gamma_{p,k}^{i} C \rfloor \approx \gamma_{p,k}^{i} C$. For example, if we consider a Gigabit router, $C = 1 \times 10^9$ bps, for voice traffic $\rho^i = 32,000$ bps, if $\alpha_{p,k}^{i} = 15\%$, then $\gamma_{p,k}^{i} C = 4,687.5$.