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. We 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 [18], [20], [22] 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., guarantee 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 [18]. At the other end of spectrum, *relative* differentiated services seek to provide per-hop, per-class relative services [10]. 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, military command and control systems, 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 the context of differentiated-services framework, real-time applications 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 [22]. In this paper, we consider networks that use *static-priority* schedulers. This type of scheduler is supported in many routers currently available, 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

Glossary

Term	Description
$G^i_{p,j,k}$; $G^i_{p,k}$	The set of all flows with priority p of Class i going through Server k from input link j; the set of all flows with priority p of Class i going thought Server k from all input links.
$f^i_{p,k,j}(t)$	The amount of the traffic for group $G^i_{p,j,k}$ during time interval $[0,t)$.
$F^i_{p,k,j}(I)$	The traffic constraint function of $f^i_{p,k,j}(t)$.
σ^i; ρ^i	The burst size and the average rate of a flow of Class i respectively.
D^i	The end-to-end deadline requirement of traffic of Class i.
$d_{p,k}$	The worst-case queuing delay suffered by the traffic with priority p at Server k.
$Y^i_{p,k}$	The maximum of the worst-case delays experienced by group $G^i_{p,k}$.
$n^i_{p,j,k}$; $n^i_{p,k}$	The number of flows in group $G^i_{p,j,k}$; the number of flows in group $G^i_{p,k}$.
$\alpha^i_{p,k}$	The ratio of the link server bandwidth allocated to group $G^i_{p,k}$.
$d_{p,k}$	The worst-case queuing delay suffered by the traffic with priority p at Server k.
$T_{j,k}$	The maximum busy interval of the traffic with priority no lower than p at Server k.
L_k	The number of input links for Link Server k.
M	The number of classes.
P	The maximum number of available priorities.
V	The set of all link servers in the network.

An earlier version of this work was published in the *Proceedings of IEEE Infocom*, Anchorage, Alaska, April 22-26, 2001.

Shengquan Wang, Riccardo Bettati, and Wei Zhao are with the Department of Computer Science, Texas A&M University, College Station, TX 77843. E-mail: {swang, bettati, zhao}@cs.tamu.edu. Dong Xuan is with the Department of Computer Information and Science, the Ohio-state University, Columbus, OH 43210. E-mail: xuan@cis.ohio-state.edu.

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.

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 of the end-to-end delay 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 rest of this paper) 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 [21]. 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 in the `diffserv` domain is how to verify the correctness of a utilization bound at configuration time. Obviously, the verification will have to rely on a delay analysis method. We will follow the approach proposed by Cruz [7] for analyzing delays. Cruz's approach must be adapted to be applicable in a flow-unaware environment however. The delay analysis proposed in [7] 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 yet. We will develop a method that allows us to analyze delays without depending on the dynamic information about flows.

As priority assignment has direct impact on the delay performance of individual packets in static-priority networks, 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 [5]. We study more generalized priority assignment algorithms, where the flows in a class may be assigned different priorities and flows from different classes may have the 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 related previous 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 formula that is insensitive to the flow population. In Section VI, we discuss our heuristic algorithms for priority assignments. 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 for future work are given in Section VIII.

II. Previous Works

A good survey on recent work in both absolute and relative differentiated services has been done in [20]. Here, we compare our work with others from the view point of providing absolute differentiated services. Nichols et al. [18] 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 [8] 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) [22]. Our work is based on static priority scheduling algorithm, which is relatively simple and widely supported.

Admission control has been investigated widely [9], [11], [17]. 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, the admission control needs to explicitly compute and verify delays for the new and existing flows [15]. This procedure is very expensive with increasing numbers of flows. In such cases, a utilization-based admission control dramatically reduces this complexity.

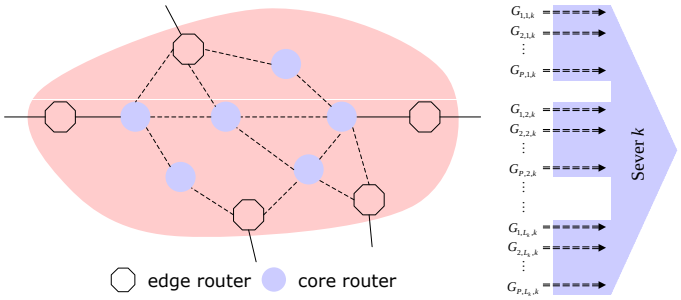
In its basic form, UBAC was first proposed in [16] for preemptive scheduling of periodic tasks on a simple processor. A number of utilization-based tests are known for centralized systems (e.g., 69% and 100% for periodic tasks on a single server using rate-monotonic and earliest-deadline-first scheduling, respectively[16]), or distributed systems (such as 33% for synchronous traffic over FDDI networks [26]). 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 [4] 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 shaping parameters at the entrance to the network. The work in [4] only considers FIFO scheduling. Also, delay bounds are not tight, but 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 [7], Cruz proposed a two-priority assignment scheme for a ring network. The work in [15] 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 (b) traffic model

Fig. 1. Network and Traffic Model

A. Network Model

The `diffserv` architecture distinguishes two types of routers (as shown in Fig. 1(a)): *Edge routers* are located at the boundary of the network and provide support for traffic policing (e.g., leaky bucket). *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. Consequently, the network can be modeled as a graph where the link servers are nodes and are connected through either links in the network or paths within routers, which both make up the set of edges in the graph. We assume output link server k is of capacity C_k and has L_k input link servers with capacities $C_{j,k}, j = 1, \ldots, L_k$. [1]

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 *flow route*, 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 class traffic separately. Let α_k^i denote the portion of bandwidth reserved for class-i traffic at Server k. We assume static-priority schedulers with support for P distinct priorities in the routers. The bandwidth assigned to class-i traffic at Server k is further partitioned into portions $\alpha_{p,k}^i$, one for each class-i traffic with priority p at that server. We note that $\alpha_k^i = \sum_{q=1}^{P} \alpha_{q,k}^i$ (the question of how much bandwidth to assign to each priority traffic will be discussed in Section VI). We aggregate flows into *groups*. All class-i flows with priority p going through Server k from Input Link j form the group $G_{p,j,k}^i$, and all flows with priority p going through Server k from input link j form the group $G_{p,j,k}$ as shown in Fig. 1(b).

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 [7].

Definition 1: The *traffic function* $f(t)$ is defined as the amount of the traffic in a group 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 $F_{p,j,k}^i(I)$ and $F_{p,j,k}(I)$ to express the traffic constraint function for group $G_{p,j,k}^i$ and group $G_{p,j,k}$ 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 constraint function* for any class-i traffic flow, which is constrained at the entrance to the network by

$$H^i(I) \leq \sigma^i + \rho^i I. \tag{2}$$

A leaky bucket and the traffic constraint function for a shaped class i flow are illustrated in Fig. 2. Since the QoS requirement

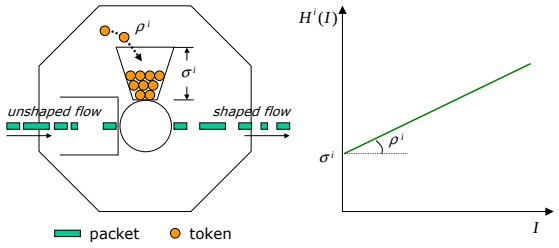


Fig. 2. A Leaky Bucket in Edge Router and the Traffic Constraint Function for a Shaped Class i Flow

of traffic (in our case, end-to-end delay) is specified on a class-by-class basis, we can use characteristic parameters $\langle \sigma^i, \rho^i, D^i \rangle$ to represent class-i traffic, where D^i is defined as the end-to-end deadline requirement of any class-i packet. A queue is stable in the sense of bounded queue length and bounded delay for customers if the long-term average rate of the input traffic is smaller than the capacity [3]. A network is said to be stable if all the data packets experience bounded delays within the network [19]. As long as the utilization of all individual links is less than 100% (this can be achieved by admission control), the stability can be guaranteed [14]. In this work, we perform the delay analysis in a stable network. We use $d_{p,k}$ to denote the local worst-case delay suffered by any packet 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:

[1] In the following, if the context is clear, we will use server to refer to the output link server.

1. *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.
2. *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. Once a new flow arrives, each link along the flow route will be checked to see if there is sufficient bandwidth available. If this is satisfied on all links along the flow route, then the new flow will be admitted and the available bandwidth of all links along the flow route will be decreased by the requested bandwidth; otherwise, the new flow will be blocked. Once the flow terminates, the bandwidth requested by the flow will be released at each link along the flow route.
3. *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.

Among the above three modules, the first module, utilization bound verification, is of most importance. Utilization-based admission control works around a utilization bound. The correctness of this utilization bound need to be verified at configuration time. We propose to realize this critical function in three steps:
1. We first obtain a general delay formula which depends on dynamic flow information.
2. We then remove the dependence in the delay formula to get a flow-population-insensitive delay formula.
3. Finally, we verify the correctness of the utilization bound by applying the flow-population-insensitive delay formula.

Among the above three steps, Step 3 is relatively straightforward once we have the flow-population-insensitive delay formula. We can apply the delay formula to check, under the given utilization bound, whether the end-to-end delay bound in each feasible path satisfies the deadline requirement.

Accordingly, in the rest of this paper, we will focus on Step 1 and Step 2. Before we proceed, we would like to address a related issue. While utilization-based admission control significantly reduces the admission control overhead, excessive connection establishment activity can still add substantial strain to the admission control components. In [6], the authors describe ways to reduce resource allocation overhead at run time by appropriately pre-allocating resources. They proposed an endpoint-based admission control system based on a *sink-tree* resource management strategy, which is particularly well suited for `diffserv`-based architecture. Generally speaking, sink trees are used to aggregate connections according to their egress nodes. The root of a sink tree is then the egress router, and the leaves are the ingress routers. By allocating resources so that each ingress router knows how much resource is ahead for each path towards each egress router, the admission control can be immediately performed at the entrance of the network. Since each egress node has its own sink tree, every possible pair of source and destination node has its own unique path in a sink tree. Consequently, wherever a flow arrives at an ingress node, the node determines the sink tree for the new flow, based on the destination node. Then, the path to the destination and the resources available on the path is determined automatically. An admission decision can then be made at the very node where a flow arrives. By performing the admission decision at the endpoints, the flow setup latency and the signaling overhead are kept to a minimum. How to construct sink-trees is described in detail in [6]. This paper will not address this issue further.

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 system have been derived for a variety of scheduling algorithms [15]. 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.

In the absence of such information, 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. [2]

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

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

where

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

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

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

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

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

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

$$c_k = \frac{1}{C_k} \sum_{j=1}^{L_k} C_{j,k}, \quad (8)$$

and $S^i_{q,k}$ is the set of all sub-routes used by class-i traffic with priority q *upstream from Server* k. [3]

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

- 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 (3) is independent from this kind of information. The values of $\boldsymbol{\sigma}$, $\boldsymbol{\rho}$, $\boldsymbol{\alpha}_{q,k}$, and c_k 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 *relative burstiness* of class-i traffic. The relative burstiness of class-i traffic is the time required for class-i traffic to get to burst size σ^i at the average rate ρ^i. The delay formula depends on the *relative burstiness* and the bandwidth allocation. As $\frac{\sigma^i}{\rho^i}$ or $\alpha^i_{q,k}$ increases, $d_{p,k}$ increases.
- We note that $d_{p,k}$ in (3) depends on $Y^i_{q,k}$. By (6), $Y^i_{q,k}$ is the maximum of the worst-case aggregated delays experienced by all class i packets with priority q upstream from Server k (see Fig.3). The value of $Y^i_{q,k}$, in turn, depends on the delays experienced at some 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. Define V as the set of all link servers in network G, recall that P is the maximum number of available priorities, then we use the $(P \times |V|)$-dimensional vector $\boldsymbol{d}$ to denote the upper bounds of the delays suffered by the traffic with all priorities at all servers:

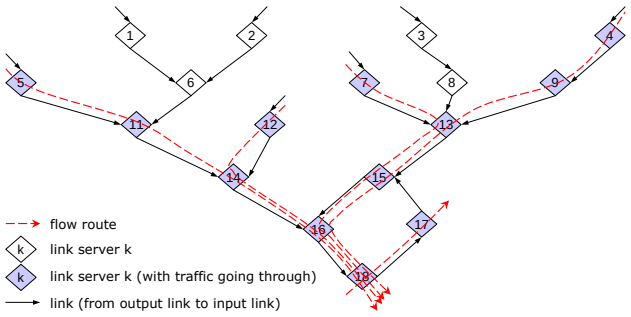


Fig. 3. The Computation of $Y^i_{q,k}$: Consider any class-1 traffic with priority 1 in the network. There are 4 flow routes going through Server 16 ($5 \to 18$, $12 \to 18$, $7 \to 18$, and $4 \to 18$), therefore $Y^1_{1,16} = \max\{d_{1,5} + d_{1,11} + d_{1,14},\ d_{1,12} + d_{1,14},\ d_{1,7} + d_{1,13} + d_{1,15},\ d_{1,4} + d_{1,9} + d_{1,13} + d_{1,15}\}$.

$$\boldsymbol{d} = (d_{1,1}, d_{1,2}, \cdots, d_{1,V}, d_{2,1}, d_{2,2}, \cdots, d_{2,|V|}, \cdots, \cdots, \cdots, \cdots, d_{P,1}, d_{P,2}, \cdots, d_{P,|V|}). \quad (9)$$

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

$$\boldsymbol{\Phi}(\boldsymbol{d}) = (\Phi_{1,1}(\boldsymbol{d}), \Phi_{1,2}(\boldsymbol{d}), \cdots, \Phi_{1,|V|}(\boldsymbol{d}), \Phi_{2,1}(\boldsymbol{d}), \Phi_{2,2}(\boldsymbol{d}), \cdots, \Phi_{2,|V|}(\boldsymbol{d}), \cdots, \cdots, \cdots, \cdots, \Phi_{P,1}(\boldsymbol{d}), \Phi_{P,2}(\boldsymbol{d}), \cdots, \Phi_{P,|V|}(\boldsymbol{d})). \quad (10)$$

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

$$\boldsymbol{d} = \boldsymbol{\Phi}(\boldsymbol{d}). \quad (11)$$

- Once the worst-case delay bound for any packet with any priority at each server is obtained, the end-to-end worst-case delay for packets with priority p on flow route $\mathcal{R}$ can be computed as $d^{e2e}_{p,\mathcal{R}} = \sum_{k\in\mathcal{R}} d_{p,k}$.
- The worst-case delay bound suffered by a packet with specific priority is only affected by the traffic with the equal or higher priorities. [4] The best-effort traffic has the lowest priority, and it will not affect the delay suffered by any real-time traffic, which has higher priorities.

In some special cases, delay formulas independent of network topology can be derived. This is the case, for example, in a network with a single real-time class traffic assigned a single priority in a network of identical link servers and identical allocations of bandwidth to the class on all servers. In this case, we simplify the notation to let $\sigma = \sigma^1, \rho = \rho^1, Y_k = Y^1_{1,k}, C = C_k, \alpha = \alpha^1_{1,k}$ and $L = \max_k\{L_k\}$. If we loosen the bound on Y_k, we will have an explicit delay formula as follows:

Corollary 1: Let d be the maximum of worst-case delays suffered by all real-time class packets across all link servers in the network. If

$$\alpha < \frac{1}{1 + (h-2)(1-\frac{1}{L})}, \quad (12)$$

then

$$d \le \frac{1}{\frac{1}{r} - (h-1)} \frac{\sigma}{\rho}. \quad (13)$$

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

$$d^{e2e} \le \frac{h}{\frac{1}{r} - (h-1)} \frac{\sigma}{\rho}, \quad (14)$$

where

$$r = \alpha \frac{L-1}{L-\alpha}, \quad (15)$$

and h is the length of the longest flow route in the network.

These delay formulas do not depend on the network topology except for the length h of the longest flow route. We note that a very similar result was derived using a different approach in [4].

The proof of *Corollary 1* is given in APPENDIX C.

[3] The exception is that $d_{p,k} = 0$ as $c_k = \|\boldsymbol{\alpha}_{p,k}\| = 1$ or $\|\hat{\boldsymbol{\alpha}}_{p,k}\| = 0$.

[4] Here we ignore blocking due to non-preemption of packets.

In a given network, if we represent flows between servers as directed links between servers and there are no loops in the resulting graph, this network is called *feedforward* network [13]. If a link server is $k-1$ hops away from the farest source in the resulting graph, we define it as layer-k link server. In Fig.3, there are 5 flow routes, $5 \to 18$, $12 \to 18$, $7 \to 18$, $4 \to 18$, and $18 \to 17$. The longest flow route is $4 \to 18$, thus $h = 6$. This network is feedforward since there is no loop in the resulting directed graph. We find that Servers $4, 5, 7, 12$ are all at Layer 1, Servers $8, 11$ are both at Layer 2, Servers $13, 14$ are both at Layer 3, and Servers $15, 16, 18, 17$ are at Layer $4, 5, 6, 7$, respectively.

In a *feedforward* network, traffic is progressively pushed forward along the directed links though the network without loops; therefore, we can get a tighter delay bound that is independent of the network topology as shown in the following corollary:

Corollary 2: Let $\hat{d}_k$ be the maximum of worst-case delays suffered by any real-time class packet at layer-k link servers, then we have the following delay bound:

$$\hat{d}_k \leq r(r+1)^{k-1}\frac{\sigma}{\rho}, \tag{16}$$

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

$$d^{e2e} \leq ((r+1)^{\hat{h}}-1)\frac{\sigma}{\rho}, \tag{17}$$

where r is defined in (15) and $\hat{h}$ is the number of layers in the feedforward network ($\hat{h} \geq h$).

Generally, if $\hat{h}$ is not much greater than h, the delay bound in (17) is much tighter than the bound in (14). In Fig.3, $h = 6, \hat{h} = 7, L = 3$, if $\sigma = 640$bits, $\rho = 32,000$bps, $\alpha = 20\%$, then $\frac{1}{1+(h-2)(1-\frac{1}{L})} = 0.27$, $((r+1)^{\hat{h}}-1)\frac{\sigma}{\rho} = 0.031$s and $\frac{h}{\frac{1}{r}-(h-1)}\frac{\sigma}{\rho} = 0.060$s.

The proof of *Corollary 2* is given in APPENDIX D.

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 about flow population.

Recall that $G^i_{p,j,k}$ is the set of class-i traffic flows with priority p entering Server k through the jth input link. Suppose that the group $G^i_{p,j,k}$ at some 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,

$$F^i_{p,j,k}(I) = \sum\nolimits_{x \in G^i_{p,j,k}} F_x(I), \tag{18}$$

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

$$F_{p,j,k}(I) = \min\{C_{j,k}I, \sum_{i=1}^{M} F^i_{p,j,k}(I)\}, \tag{19}$$

where $C_{j,k}$ is the capacity of the jth input link of Server k.

In a stable network, the worst-case delay $d_{p,k}$ suffered by any priority-p packet at Server k can then easily be formulated in terms of the aggregated traffic constraint functions and the service rate C_k of the server according to [15]:

$$d_{p,k} = \frac{1}{C_k}\max_{I<T_{p,k}}(\mathcal{F}_{p,k}(I+d_{p,k}) - C_k I), \tag{20}$$

where

$$\mathcal{F}_{p,k}(I+d_{p,k}) = \sum_{q=1}^{p-1}\sum_{j=1}^{L_k} F_{q,j,k}(I+d_{p,k}) + \sum_{j=1}^{L_k} F_{p,j,k}(I), \tag{21}$$

$$T_{p,k} = \min\{I>0 : \sum_{q=1}^{p}\sum_{j=1}^{L_k} F_{q,j,k}(I) \leq C_k I\}. \tag{22}$$

Equation (20) indicates how long a newly-arriving packet with priority p can be delayed at Server k in a stable network. It describes the maximum worst-case delay suffered by a packet due to delay by higher-priority packets that arrived before or during the time the packet is queued, and same-priority packets queued before the arrival of the packet.

In (20), $\mathcal{F}_{p,k}(I+d_{p,k})$ and $C_k I$ form an arrival curve and a service curve, respectively; and $T_{p,k}$ denotes the maximum busy interval of the traffic with priority no lower than p at Server k. In other words, Server k never processes packets with priority equal to or higher than p for more than $T_{p,k}$ consecutive time units. The arrival curve $\mathcal{F}_{p,k}(I+d_{p,k})$, the service curve $C_k I$, and the worst-case delay $d_{p,k}$ are illustrated in Fig. 4. [5]

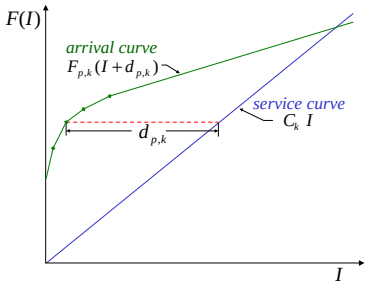


Fig. 4. Arrival Curve $\mathcal{F}_{p,k}(I+d_{p,k})$, Service Curve $C_k I$, and Worst-case Delay $d_{p,k}$.

To get the value of $d_{p,k}$, we first need to find the point where the worst-case delay in (20) is achieved. This allows us to ignore the maximum function in (20). Assume the arrival curve $\mathcal{F}_{p,k}(I+d_{p,k})$ is differentiable almost everywhere for $I > 0$. Define $s(I)$ as the piece-wise slope (average rate) of the arrival curve $\mathcal{F}_{p,k}(I+d_{p,k})$ at interval I, *i.e.*, $s(I) = \frac{\mathrm{d}\mathcal{F}_{p,k}(I+d_{p,k})}{\mathrm{d}I}$. As we know, the arrival curve $\mathcal{F}_{p,k}(I+d_{p,k})$ is an increasing and concave function. From Fig. 4, we know that the worst-case delay is maximized at the point from which the slope of the aggregate traffic function becomes smaller than C_k.

Substituting (18) and (19) into (20), we observe that the above delay formula depends on flow population. In fact, (20) 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

[5] Here, we slightly abuse the terms "curve" and "function".

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 $F_x(I)$ with 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 $G_{p,j,k}$ is constrained by

$$F_{p,j,k}(I) = \begin{cases} C_{j,k} I, & I \le \tau_{p,j,k} \\ \boldsymbol{n}_{p,j,k} \cdot (\boldsymbol{\eta}_{p,k} + \boldsymbol{\rho} I), & I > \tau_{p,j,k} \end{cases} \quad (23)$$

where

$$\tau_{p,j,k} = \frac{\boldsymbol{n}_{p,j,k} \cdot \boldsymbol{\eta}_{p,k}}{C_{j,k} - \boldsymbol{n}_{p,j,k} \cdot \boldsymbol{\rho}}, \quad (24)$$

$$\eta^i_{p,k} = \sigma^i + \rho^i Y^i_{p,k}, \quad (25)$$

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

$$d_{p,k} \le \frac{U_{p,k} - V_{p,k} W_{p,k}}{X_{p,k}}, \quad (26)$$

where

$$U_{p,k} = \sum_{q=1}^{p} \boldsymbol{n}_{q,k} \cdot \boldsymbol{\eta}_{q,k}, \quad (27)$$

$$V_{p,k} = C_k - \sum_{q=1}^{p} \boldsymbol{n}_{q,k} \cdot \boldsymbol{\rho}, \quad (28)$$

$$X_{p,k} = C_k - \sum_{q=1}^{p-1} \boldsymbol{n}_{q,k} \cdot \boldsymbol{\rho}, \quad (29)$$

and

$$W_{p,k} = \frac{\boldsymbol{n}_{p,j',k} \cdot \boldsymbol{\eta}_{p,k}}{C_{j',k} - \boldsymbol{n}_{p,j',k} \cdot \boldsymbol{\rho}} \quad (30)$$

is defined as the point which satisfies that $s(I) > C_k$ as $I < W_{p,k}$ and $s(I) \le C_k$ as $I \ge W_{p,k}$. In (27) – (30),

$$n^i_{q,k} = \sum_{j=1}^{L_k} n^i_{q,j,k}. \quad (31)$$

The proof of *Theorem 2* is given in APPENDIX A. The delay computation using Equation (26) 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 q does not exceed $\alpha^i_{q,k}$. In other words, the following inequality always holds:

$$n^i_{q,k} \rho^i \le \alpha^i_{q,k} C_k. \quad (32)$$

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

$$n^i_{q,k} \le \frac{\alpha^i_{q,k}}{\rho^i} C_k. \quad (33)$$

To maximize the right hand side of (26), we should maximize $U_{p,k}$ and minimize $V_{p,k}$, $X_{p,k}$, and $W_{p,k}$. Under the constraint of (33), these parameters can be bounded for all possible distribution $n^i_{q,j,k}$ 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 any priority-p packet at Server k, then,

$$U_{p,k} \le \sum_{q=1}^{p} (\boldsymbol{\alpha}_{q,k} \cdot \boldsymbol{Z}_{q,k}) C_k, \quad (34)$$

$$V_{p,k} \ge (1 - \sum_{q=1}^{p} \|\boldsymbol{\alpha}_{q,k}\|) C_k, \quad (35)$$

$$X_{p,k} \ge (1 - \sum_{q=1}^{p-1} \|\boldsymbol{\alpha}_{q,k}\|) C_k, \quad (36)$$

and

$$W_{p,k} \ge \frac{\boldsymbol{\alpha}_{p,k} \cdot \boldsymbol{Z}_{p,k}}{c_k - \|\boldsymbol{\alpha}_{p,k}\|}. \quad (37)$$

where $U_{p,k}$, $V_{p,k}$, $X_{p,k}$, $W_{p,k}$ are defined in (27) – (30), $Z_{p,k}$ is defined in (5), *i.e.*, $Z^i_{q,k} = \frac{\sigma^i}{\rho^i} + Y^i_{q,k}$, and c_k is defined in (8), *i.e.*, $c_k = \frac{1}{C_k} \sum_{j=1}^{L_k} C_{j,k}$.

The proof of *Theorem 3* is given in APPENDIX B.

If we substitute all the bounds in (34) – (37) into (26), then, after some algebraic manipulation, we have

$$d_{p,k} \le \frac{1}{(1 - \sum_{q=1}^{p-1} \|\boldsymbol{\alpha}_{q,k}\|)} \Big(\sum_{q=1}^{p} \boldsymbol{\alpha}_{q,k} \cdot \boldsymbol{Z}_{q,k} - \frac{(1 - \sum_{q=1}^{p} \|\boldsymbol{\alpha}_{q,k}\|)}{c_k - \|\boldsymbol{\alpha}_{p,k}\|} \boldsymbol{\alpha}_{p,k} \cdot \boldsymbol{Z}_{p,k} \Big). \quad (38)$$

(3) follows after some definitions of parameters in (38). Hence, *Theorem 1* is proved.

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.

In the following, we use class and path (in form of source and destination address) information to assign priorities, with all flows in the same class and with the same source and destination having 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 the time of establishment. 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:

- *Algorithm One-to-One:* 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. We use this mapping as the baseline for comparison with other approaches.
- *Algorithm One-to-Many:* 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 since it does not allow priorities to be shared by flows from different classes.
- *Algorithm Many-to-Many:* 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; hence, there will be no need to present the other two algorithms in detail.

TABLE I
AN 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

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.

Fig. 5 shows our *One-to-Many* priority assignment algorithm. The input of this algorithm is the network server graph, flow routes, characteristic parameters $\langle \sigma^i, \rho^i, D^i\rangle$ for each flow class, and the network bandwidth α_k^i for each class i at each server k. We need no knowledge about the exact amount of traffic or the number of flows. The value for the α_k^i can be pre-determined for different class traffic by some policies. The algorithm will return the priority assignment table and bandwidth allocation $\alpha_{p,k}^i$ for each class i traffic with priority p at each server k or return "FAILURE" as the output.

The algorithm uses a stack to store subsets of entries of which the priority fields are to be assigned. Entries in each subset can potentially be assigned to the same priority. The subsets are ordered in the stack in accordance to their real-time requirements. Given an entry $\langle i, src, dst, p\rangle$ in a subset, assume that the flow route from src to dst is $\mathcal{R}$ with length $h(\mathcal{R})$, then we define its per-hop laxity as

$$phl_{i,src,dst,p} = \frac{1}{h(\mathcal{R})}(D_i - d_{p,\mathcal{R}}^{e2e}), \tag{39}$$

where D_i is the end-to-end deadline requirement and $d_{p,\mathcal{R}}^{e2e}$ is the computed worst-case end-to-end delay along route $\mathcal{R}$. If the per-hop laxity is less than 0, packets may miss their deadline. The subset with entries that represent flows with the smallest 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 *PartitionSet* (as shown in Fig. 6) is called to partition the entries in the subset into two subsets based on their per-hop 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

Input: Network server graph, flow routes, characteristic parameters $\langle \sigma^i, \rho^i, D^i \rangle$ for each flow class, assigned network bandwidth α_k^i, $i = 1, \ldots, M$.
Output: Priority assignment table and bandwidth allocation $\alpha_{p,k}^i$ that is the portion of bandwidth reserved for class-i traffic with priority p at Server k, or "FAILURE" if no such bandwidth allocation can be found.

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. $p = 0$; /* intial value of priority */
4. while Stack SS is not empty
   4.1. $p = p + 1$;
   4.2. if $p > P$ /* 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 (11) to update the end-to-end delay of flows represented by entries in S;
   4.6. if all per-hop laxities of entries in $S \geq 0$ 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 *PartitionSet(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. 5. Algorithm One-to-Many

assignment. Procedure *PartitionSet* also assigns bandwidth to the traffic with different priorities in the same class. The procedure splits bandwidth according to the ratio of the cardinality of the partitioned subset over the one of the original subset. For example, in this procedure, if S is partitioned into S_x and S_y, then $\alpha_{p,k}^i$ will be split into $\frac{|S_x|}{|S|}\alpha_{p,k}^i$ and $\frac{|S_y|}{|S|}\alpha_{p,k}^i$. In our experiments, we use "partition by the half number of entries" (Step 3.1), *i.e.*, $|S_x| = |S_y| = \frac{1}{2}|S|$.

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 real-time requirments for a single entry in a subset.

Because the size of a subset is split 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 Fig. 5. For *Algorithm One-to-One*, no subset

Input: subset S
Output: subsets S_x and S_y, and bandwidth re-allocation $\alpha_{p,k}^i$

1. compute the per-hop laxity for each entry in subset S;
2. sort subset S in the increasing order of per-hop laxities;
3. partition S into S_x and S_y (for any entry $s_x \in S_x$ and $s_y \in S_y$, define their corresponding per-hop laxity as $phl(s_x)$ and $phl(s_y)$), such that
   3.1. $|S_x| = |S_y| = \frac{1}{2}|S|$, or
   3.1. $phl(s_x) < \widetilde{phl}(s) \leq phl(s_y)$, where $\widetilde{phl}(s)$ is the mean value of per-hop laxities in S, or
   3.3. $phl(s_x) < 0 \leq phl(s_y)$;
4. split $\alpha_{p,k}^i$ into $\frac{|S_x|}{|S|}\alpha_{p,k}^i$ and $\frac{|S_x|}{|S|}\alpha_{p,k}^i$.

Fig. 6. Procedure **P**artitionSet(S)

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 Fig. 5 so that it returns "FAILURE" whenever a failure on a 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 Fig. 5 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 been used. In the case the deadline test fails, assign these entries with a higher priority (until the highest priority has been assigned).

VII. Performance 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 of 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.

Fig. 7 illustrates the detailed flow chart for MUU Computation. Given a network server graph, flow routes, and characteristic parameters for each class flow (burst, average rate, deadline), we assume that all links have the same utilization, and bandwidth is allocated for different classes traffic by a given ratio pre-determined by policies. For any input of link utilization $u = u_k = \sum_{q=1}^{P} \|\boldsymbol{\alpha}_{q,k}\|$, we calculate the worst-case delay

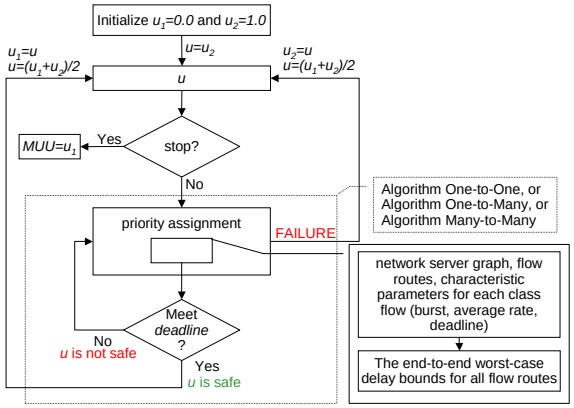


Fig. 7. Maximum Usable Utilization (MUU) Computation.

with our delay analysis methods. Then, we can verify whether the utilization is safe or not to make end-to-end delay meet the deadline requirement. Using the binary searching method, we can obtain the maximum usable utilization (MUU) (The condition of "*stop?*" in Fig. 7 could be "$u_2 - u_1$ is less than the given infinitesimal number or the iteration number is beyond the given limit?").

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

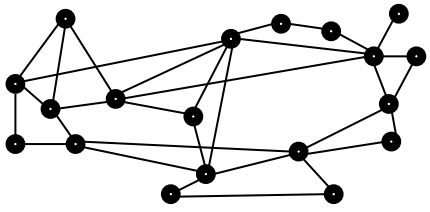


Fig. 8. MCI Network Topology

The underlying network topology in this experiment is the classical MCI network topology as shown in Fig. 8. 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. We assume that $\alpha_k^1 : \alpha_k^2 : \alpha_k^3 = 0.05 : 0.10 : 0.20$, which is pre-determined by some policy before hand governing the operation of the network. 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 the 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 experimental results are depicted for higher-burstiness traffic. In each row, the relative burstiness $\frac{\sigma}{\rho}$ 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 relative burstiness.

TABLE II
THE COMPARISON OF MUU FOR DIFFERENT RELATIVE BURSTINESS 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	0.03	0.04	0.05

In TABLE II, we 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 relative burstiness of the traffic. Instead, we vary the network topology. We randomly generate network topologies with GT-ITM [24] using the Waxman 2 method described there to generate edges. We classify the generated topologies according to their size in number of nodes, and in their diameter. Performance data are collected based on a sample of 50 networks per number of nodes (overall 550 networks).

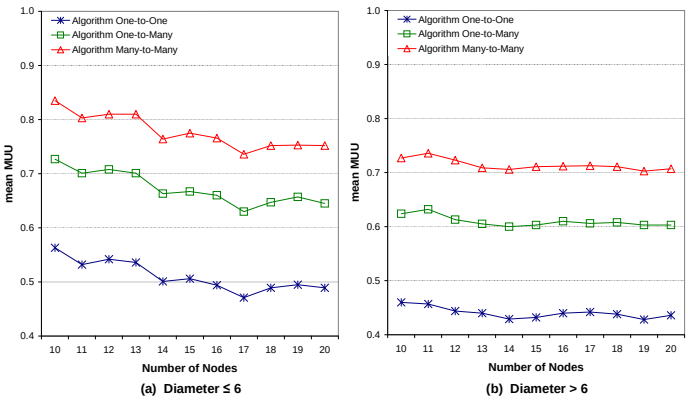


Fig. 9. The MUU Values of Randomly Generated Networks

Fig. 9 displays the mean values for MUU for small networks (diameter of the networks is less than or equal to 6) and for larger

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 better max utilization than *Algorithm One-to-One*, in the networks with the same number of nodes. 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. For example, when 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 $Diameter \geq 6$. The reason is that flows in big networks (in the sense of diameter) usually suffer larger end-to-end delay than in small networks.

C. Discussion

In this paper, we use a *worst-case* delay analysis, which follows Cruz's work [7] on worst-case delay analysis. One of our main contributions is the scalable delay analysis techique, which makes offline delay computation possible and so allows the adaptation of utilization-based admission control mechanism can be adopted. To achieve this, we have to remove flow-population information from the delay analysis, which therefore makes the worse-case delay bound more loose. Generally speaking, the looser the worse-case delay bound, the lower the achieved maximum usable utilization. By the experimental data, however, the utilization assigned to real-time traffic is still very feasible, *i.e.*, the delay bound is acceptable. For example, as relative burstiness is 0.02s, up to 70% utilization can be assigned to real-time traffic by *Algorithm Many-to-Many*. On the other hand, the residue bandwidth will not be wasted, and best-effort traffic (which is assigned the lowest priority) can make full use of it.

Considering both run-time delay computation and offline delay computation, we find that there is a trade-off between computational complexity and maximum usable utilization. Run-time delay computation has more run-time admission overhead and, however, larger usable utilization than offline delay computation. In our model, the distribution of flow arrivals is unknown (we only know the characteristic parameters $\langle \sigma^i, \rho^i, D^i \rangle$ for each class flow). An adaptation method can be adopted to achieve a balance in the trade-off. One possible way is that the utilization can be configured dynamically. During the reconfiguration, all end-to-end delays need to be recomputed and all deadline requirements must be met. It may need more expensive computation but should provide a higher maximum usable utilization.

VIII. Conclusions

In this paper, we have proposed a methodology for providing absolute differentiated services for real-time applications in networks that use static priority schedulers. Given that static-priority schedulers are widely supported in current routers, we believe that this approach is 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 runtime, 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 use our new delay analysis techniques and priority assignment algorithms. We find that *Algorithm Many-to-Many* can achieve very high network utilization both in a well-known network and in 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 [12], [3], [13], [23]. However, all these previous results require information on flow population to obtain the statistical delay bounds. For example, in [12], statistical delay bounds are obtained by using approximated normal distribution, of which the parameters, in turn, depend on the flow population. Our method of 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 guarantee.

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] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry, *The COPS (Common Open Policy Service) Protocol*, Internet-Draft, Feberary 1999.

[3] C. Chang, *Stability, queue length, and delay of deterministic and stochastic queueing networks*, IEEE Transactions on Automatic Control, 39(5):913-931, May 1994.

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

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

[6] B. Choi and R. Bettati, *Endpoint Admission Control: Network Based Approach*, Proceedings of IEEE ICDCS, April 2001.

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

[8] Rene L. Cruz, *SCED+: efficient management of quality of service guarantees*, Proceedings of IEEE Infocom, March – April 1998.

[9] A. Dailianas and A. Bovopoulis, *Real-time admission control algorithms with delay and loss guarantees in ATM networks*, Proceedings of IEEE Infocom, June 1994.

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

[11] V. Firoiu, J. Kurose, and D. Towsley, *Efficient admission control for EDF schedulers*, IEEE Proceedings of Infocom, April 1997.

[12] E. Knightly, *Enforceable quality of service guarantees for bursty traffic streams*, Proceedings of IEEE Infocom, March – April 1998.

[13] J. Kurose, *On computing per-session performance bounds in high-speed multi-hop computer networks*, Proceedings of ACM Sigmetrics, June 1992.

[14] C. Li, A. Raba, and W. Zhao, *Stability in ATM networks*, Proceedings of IEEE Infocom, April 1997.
[15] C. Li, R. Bettati, and W. Zhao, *Static priority scheduling for ATM networks*, Proceedings of IEEE Real-Time Systems Symposium, 1997.
[16] C. L. Liu and J. W. Layland, *Scheduling algorithms for multiprogramming in a hard real-time environment*, Journal of the ACM, Vol. 20, No. 1, 1973.
[17] J. Liebeherr, D.E. Wrege, and D. Ferrari, *Exact admission control in networks with bounded delay services*, IEEE/ACM Transactions on Networking, 4(6):885 – 901, December 1996.
[18] K. Nichols, V. Jacobson, and L. Zhang, *A Two-bit Differentiated Services Architecture for the Internet*, Internet Draft, November 1997.
[19] A. Parekh, *A generalized processor sharing approach to flow control in integrated services networks*, PhD thesis, Department of Electrical Engineering and Computer Science, M.I.T., 1992.
[20] R. Sivakumar, T. Kim, N. Venkitaraman and V. Bharghavan, *Achieving Per-Flow Weighted Rate Fairness in a Core Stateless Network*, Proceedings of IEEE ICDCS, April 2000.
[21] S. Shenker, C. Partridge and R. Guerin, *Specification of Guaranteed Quality of Service*, RFC2212, September 1997.
[22] I. Stoica and H. Zhang, *Providing Guaranteed Services Without Per Flow Management*, Proceedings of ACM SIGCOMM, August – September 1999.
[23] 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.
[24] Eleen W. Zegura, et al., *http://www.cc.gatech.edu/projects/gtitm/*
[25] 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.
[26] G. Agrawal, B. Chen, W. Zhao, and S. Davari, *Guaranteeing Synchronous Message Deadlines with the Timed Token Protocol*, Proceedings of IEEE ICDCS, June 1992.

APPENDIX A: PROOF OF *Theorem 2*

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

$$F_{p,j,k}(I) = \begin{cases} C_{j,k}I, & I \le \tau_{p,j,k} \\ \boldsymbol{n}_{p,j,k}\cdot(\boldsymbol{\eta}_{p,k} + \boldsymbol{\rho}I), & I > \tau_{p,j,k} \end{cases} \quad (40)$$

where

$$\tau_{p,j,k} = \frac{\boldsymbol{n}_{p,j,k}\cdot\boldsymbol{\eta}_{p,k}}{C_{j,k} - \boldsymbol{n}_{p,j,k}\cdot\boldsymbol{\rho}}, \quad (41)$$

$$\eta^i_{p,k} = \sigma^i + \rho^i Y^i_{p,k}, \quad (42)$$

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

$$d_{p,k} \le \frac{U_{p,k} - V_{p,k}W_{p,k}}{X_{p,k}}, \quad (43)$$

where

$$U_{p,k} = \sum_{q=1}^{p} \boldsymbol{n}_{q,k}\cdot\boldsymbol{\eta}_{q,k}, \quad (44)$$

$$V_{p,k} = C_k - \sum_{q=1}^{p} \boldsymbol{n}_{q,k}\cdot\boldsymbol{\rho}, \quad (45)$$

$$X_{p,k} = C_k - \sum_{q=1}^{p-1} \boldsymbol{n}_{q,k}\cdot\boldsymbol{\rho}, \quad (46)$$

and

$$W_{p,k} = \frac{\boldsymbol{n}_{p,j',k}\cdot\boldsymbol{\eta}_{p,k}}{C_{j',k} - \boldsymbol{n}_{p,j',k}\cdot\boldsymbol{\rho}} \quad (47)$$

is defined as the point which satisfies that $s(I) > C_k$ as $I < W_{p,k}$ and $s(I) \le C_k$ as $I \ge W_{p,k}$. In (44) – (47),

$$n^i_{q,k} = \sum_{j=1}^{L_k} n^i_{q,j,k}. \quad (48)$$

Theorem 2 includes two main results: one is the aggregation function, and the other is the worst-case delay bound. To prove these results, in the following, we introduce two lemmas: *Lemma 1* tries to get the expression of the aggregation function and *Lemma 2* tries to derive the worst-case delay.

Lemma 1: The aggregated traffic of group $G_{p,j,k}$ is constrained by (40).

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

$$Y_x \le Y^i_{p,k}. \quad (49)$$

Since $H^i(I)$ is the source traffic function of flow x, according to Theorem 2.1 in [7], we have

$$F_x(I) \le H^i(I + Y_x) \le H^i(I + Y^i_{p,k}). \quad (50)$$

Recall that the source traffic function is $H^i(I) \le \sigma^i + \rho^i I$. Therefore, by (50), we can bound $F^i_{p,j,k}$ as follows:

$$F^i_{p,j,k}(I) \le \sum\nolimits_{x\in G^i_{p,j,k}} F_x(I) \quad (51)$$

$$\le \sum\nolimits_{x\in G^i_{p,j,k}} H^i(I + Y^i_{p,k}) \quad (52)$$

$$\le n^i_{p,j,k}(\eta^i_{p,k} + \rho^i I). \quad (53)$$

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.,

$$F^i_{p,j,k}(I) \le C_{j,k}I. \quad (54)$$

Synthesizing (53) and (54), we have

$$F^i_{p,j,k}(I) = \begin{cases} C_{j,k}I, & I \le \tau^i_{p,j,k} \\ n^i_{p,j,k}(\eta^i_{p,k} + \rho^i I), & I > \tau^i_{p,j,k} \end{cases} \quad (55)$$

where

$$\tau^i_{p,j,k} = \frac{n^i_{p,j,k}\eta^i_{p,k}}{C_{j,k} - n^i_{p,j,k}\rho^i}. \quad (56)$$

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

$$F_{p,j,k}(I) = \min\{C_{j,k}I, \sum\nolimits_{i=1}^{M} F^i_{p,j,k}(I)\}. \quad (57)$$

Notice that $\tau_{p,j,k} \ge \tau^i_{p,j,k}$ for all classes i. $\sum_{i=1}^{M} F^i_{p,j,k}(I) \ge C_{j,k}I$ as $I \le \tau_{p,j,k}$ and $\sum_{i=1}^{M} F^i_{p,j,k}(I) = \boldsymbol{n}_{p,j,k}\cdot(\boldsymbol{\eta}_{p,k} + \boldsymbol{\rho}I)$ as $I > \tau_{p,j,k}$. Thus, (40) holds as claimed. ■

Recall that given a static priority scheduling discipline at the server, we have the following formula that indicates how long a newly-arrived priority-p packet can be delayed at Server k [15]:

$$d_{p,k} = \frac{1}{C_k} \max_{I<T_{p,k}} (\mathcal{F}_{p,k}(I + d_{p,k}) - C_k I), \quad (58)$$

where

$$\mathcal{F}_{p,k}(I + d_{p,k}) = \sum_{q=1}^{p-1} \sum_{j=1}^{L_k} F_{q,j,k}(I + d_{p,k}) + \sum_{j=1}^{L_k} F_{p,j,k}(I), \quad (59)$$

$$T_{p,k} = \min\{I > 0 : \sum_{q=1}^{p} \sum_{j=1}^{L_k} F_{q,j,k}(I) \le C_k I\}. \quad (60)$$

Applying (40) into (59), we can obtain the explicit expression of $\mathcal{F}_{p,k}(I + d_{p,k})$ in (59). To get the explicit expression of $d_{p,k}$, we need to find the point where the worst-case delay in (58) is suffered, and then we can remove the maximum symbol. The following lemma tries to address this issue:

Lemma 2: Define $\tilde{\tau}$ as the point after which the slope of the aggregate traffic function becomes smaller than C_k, and then the worst-case queuing delay $d_{p,k}$ suffered by the traffic with priority p at Server k will happen at

$$I = \tilde{\tau} = \frac{\boldsymbol{n}_{p,j',k} \cdot \boldsymbol{\eta}_{p,k}}{C_{j',k} - \boldsymbol{n}_{p,j',k} \cdot \boldsymbol{\rho}}, \quad (61)$$

for some specific j', where $1 \le j' \le L_k$.

Proof: Note that each $F^i_{p,j,k}(I)$ is a two-piece-wise linear continuous function, and $\sum_{i=1}^{M} F^i_{p,j,k}(I)$ is still a piece-wise 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 $\sum_{i=1}^{M} F^i_{p,j,k}(I)$. We know that all traffic constraint functions $F_{q,j,k}(I)$'s are piece-wise, and so is the aggregated traffic constraint functions $\mathcal{F}_{p,k}(I + d_{p,k})$. We try to find the flex point $\tilde{\tau}$ so that this flex point maximizes the delay in (58) (as shown in Fig.10).

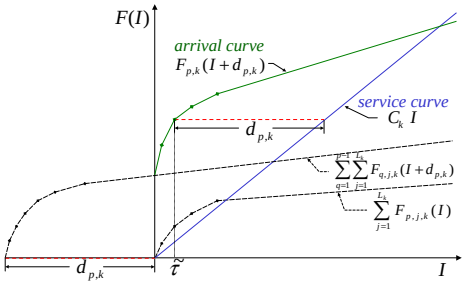


Fig. 10. The Detailed Illustration of Arrival Curve $\mathcal{F}_{p,k}(I + d_{p,k})$

The aggregated traffic function consists of two parts: the aggregated traffic function with priorities higher than p and the one with priority p.

• *The aggregated traffic function with priorities higher than p:* Define $\tau_{p-,k}$ as the maximum of the flex points of the traffic constraint function $F_{q,j,k}(I)$ with priority $q < p$. Let $T_{p-,k}$ be the maximum busy interval of the aggregated traffic constraint function $\sum_{q=1}^{p-1} \sum_{j=1}^{L_k} F_{q,j,k}(I)$. As we know $d_{p,k} \ge T_{p-,k} \ge \tau_{p-,k}$. Therefore, the aggregated traffic function $\sum_{q=1}^{p-1} \sum_{j=1}^{L_k} F_{q,j,k}(I + d_{p,k})$ has no flex points as $I \ge 0$.

• *The aggregated traffic function with priority p:* Let $\tau_{p,j,k}$ be the flex point of the traffic constraint function $F_{p,j,k}(I)$. It is also the flex point of $\mathcal{F}_{p,k}(I + d_{p,k})$. As we know $T_{p,k} \ge \tau_{p,j,k}$ for all j's.

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

$$I = \tilde{\tau} = \frac{\boldsymbol{n}_{p,j',k} \cdot \boldsymbol{\eta}_{p,k}}{C_{j',k} - \boldsymbol{n}_{p,j',k} \cdot \boldsymbol{\rho}}, \quad (62)$$

for some specific j', where $1 \le j' \le L_k$. ■

Now, we are ready to prove *Theorem 2*.

Proof: (40) can be obtained directly from *Lemma 1*. Let us derive (43). By *Lemma 2*, we know that the worst-case queuing delay $d_{p,k}$ suffered by the traffic with priority p at Server k will happen at $I = \tilde{\tau} = \frac{\boldsymbol{n}_{p,j',k} \cdot \boldsymbol{\eta}_{p,k}}{C_{j',k} - \boldsymbol{n}_{p,j',k} \cdot \boldsymbol{\rho}}$ for some specific j', where $1 \le j' \le L_k$. Substituting (55) and (61) into (58), we can, therefore, eliminate the max operator from (58) as follows:

$$\begin{aligned} d_{p,k} \quad \le \quad & \frac{1}{C_k} \Big(\sum_{q=1}^{p-1} \sum_{j=1}^{L_k} \boldsymbol{n}_{q,j,k} \cdot (\boldsymbol{\eta}_{q,k} + \boldsymbol{\rho}(\tilde{\tau} + d_{p,k})) \\ & + \sum_{j=1}^{L_k} \boldsymbol{n}_{p,j,k} \cdot (\boldsymbol{\eta}_{p,k} + \boldsymbol{\rho}\tilde{\tau}) \Big). \end{aligned} \quad (63)$$

By appropriately redefining some parameters and some algebraic manipulation in (63), we have

$$d_{p,k} \le \frac{U_{p,k} - V_{p,k} W_{p,k}}{X_{p,k}}, \quad (64)$$

where $U_{p,k}$, $V_{p,k}$, $W_{p,k}$, and $X_{p,k}$ are defined in (44) – (47), respectively. ■

APPENDIX B: PROOF OF *Theorem 3*

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

$$U_{p,k} \quad \le \quad \sum_{q=1}^{p} (\boldsymbol{\alpha}_{q,k} \cdot \boldsymbol{Z}_{q,k}) C_k, \quad (65)$$

$$V_{p,k} \quad \ge \quad (1 - \sum_{q=1}^{p} \|\boldsymbol{\alpha}_{q,k}\|) C_k, \quad (66)$$

$$X_{p,k} \quad \ge \quad (1 - \sum_{q=1}^{p-1} \|\boldsymbol{\alpha}_{q,k}\|) C_k, \quad (67)$$

and

$$W_{p,k} \quad \ge \quad \frac{\boldsymbol{\alpha}_{p,k} \cdot \boldsymbol{Z}_{p,k}}{c_k - \|\boldsymbol{\alpha}_{p,k}\|}. \quad (68)$$

where $U_{p,k}$, $V_{p,k}$, $X_{p,k}$, $W_{p,k}$ are defined in (44) – (47), $Z_{p,k}$ is defined in (5), *i.e.*, $Z^i_{q,k} = \frac{\sigma^i}{\rho^i} + Y^i_{q,k}$, and c_k is defined in (8), *i.e.*, $c_k = \frac{1}{C_k} \sum_{j=1}^{L_k} C_{j,k}$.

In order to prove *Theorem 3*, we need two lemmas.

Before presenting these lemmas, we define the worst-case of arrival curve $\tilde{\mathcal{F}}_{p,k}(I+d_{p,k})$ as the case that, in the arrival curve $\mathcal{F}_{p,k}(I+d_{p,k})$, $\sum_{j=1}^{L_k} F_{p,j,k}(I)$ becomes

$$\tilde{F}_{p,k}(I) = \min\{\sum_{j=1}^{L_k} C_{j,k}I, \sum_{j=1}^{L_k} \boldsymbol{n}_{p,j,k}\cdot(\boldsymbol{\eta}_{p,k}+\boldsymbol{\rho}I)\}, \quad (69)$$

then let us introduce the first lemma:

Lemma 3: The worst-case queuing delay at Server k suffered by any packet with priority p can be experienced if the arrival curve becomes the worst-case $\tilde{\mathcal{F}}_{p,k}(I+d_{p,k})$, and furthermore we have

$$W_{p,k} = \tilde{\tau} = \frac{\sum_{j=1}^{L_k} \boldsymbol{n}_{p,j,k}\cdot\boldsymbol{\eta}_{p,k}}{\sum_{j=1}^{L_k}(C_{j,k} - \boldsymbol{n}_{p,j,k}\cdot\boldsymbol{\rho})}, \quad (70)$$

where $W_{p,k}$ is defined in (47).

Proof: Note that given all specific $\boldsymbol{n}_{q,j,k}$'s, $\sum_{j=1}^{L_k} F_{p,j,k}(I) \le \tilde{F}_{p,k}(I)$ for general $F_{p,j,k}(I)$, therefore, $\mathcal{F}_{p,k}(I+d_{p,k}) \le \tilde{\mathcal{F}}_{p,k}(I+d_{p,k})$ for general $\mathcal{F}_{p,k}(I+d_{p,k})$ as illustrated in Fig. 11. By (58), we know that the larger $\mathcal{F}_{p,k}(I+d_{p,k})$, the larger

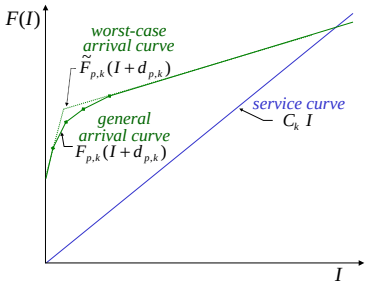


Fig. 11. Worst-case Arrival Curve $\tilde{\mathcal{F}}_{p,k}(I+d_{p,k})$

$d_{p,k}$. Therefore, the worst-case arrival curve will experience the worst-case delay. Note that $\tilde{\tau}$ defined in (70) is the only flex point of the function (69). Therefore, the worst-case delay will happen at $I=\tilde{\tau}$, and then $W_{p,k}=\tilde{\tau}$.

On the other hand, we show that the worst-case arrival curve can be achieved for some specific $\boldsymbol{n}_{p,j,k}$'s. Note that if all flex points $\tau_{p,j,k}(I)$'s of $F_{p,j,k}(I)$'s are equal, *i.e.*,

$$\tau_{p,1,k} = \tau_{p,2,k} = \ldots = \tau_{p,L_k,k}. \quad (71)$$

Applying the formula

$$\frac{x_1}{y_1} = \frac{x_2}{y_2} = \cdots = \frac{x_{L_k}}{y_{L_k}} = \frac{x_1+x_2+\cdots+x_{L_k}}{y_1+y_2+\cdots+y_{L_k}} \quad (72)$$

to (71), we have

$$\tau_{p,1,k} = \tau_{p,2,k} = \ldots = \tau_{p,L_k,k} = \tilde{\tau}. \quad (73)$$

That means $\sum_{j=1}^{L_k} F_{p,j,k}(I)$ becomes $\tilde{F}_{p,k}(I)$, and, hence, the arrival curve $\mathcal{F}_{p,k}(I+d_{p,k})$ becomes the worst-case arrival curve $\tilde{\mathcal{F}}_{p,k}(I+d_{p,k})$. For this worse-case,

$$W_{p,k} = \tilde{\tau} = \frac{\sum_{j=1}^{L_k} \boldsymbol{n}_{p,j,k}\cdot\boldsymbol{\eta}_{p,k}}{\sum_{j=1}^{L_k}(C_{j,k} - \boldsymbol{n}_{p,j,k}\cdot\boldsymbol{\rho})}. \quad (74)$$

■

Lemma 4: The worst-case queuing delay at Server k suffered by any class-i packet with priority q is experienced only if the number of flows $n^i_{q,k}$ is maximized, *i.e.*,

$$n^i_{q,k} = \gamma^i_{q,k} C_k, \quad (75)$$

where

$$\gamma^i_{q,k} = \frac{\alpha^i_{q,k}}{\rho^i}. \quad (76)$$

Proof: By (58), we know that the larger $F_{q,j,k}(I)$, the larger $d_{p,k}$. Furthermore, since $F_{q,j,k}(I)$ is the aggregated class-i traffic with priority q at Server k, we know that the larger $n^i_{q,k}$, the larger $F_{q,j,k}(I)$. Therefore, when the number of flows on each link is maximized, then any class-i packet with priority p will experience the worst-case queuing delay at the server, *i.e.*,

$$n^i_{q,k} = \gamma^i_{q,k} C_k. \quad (77)$$

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

Now, we are ready to prove *Theorem 3*.

Proof: By *Lemma 3* and *Lemma 4*, we obtain (75) and (70). Substituting (75) into (44) – (46) and (70), we have

$$U_{p,k} \le (\sum_{q=1}^{p} \boldsymbol{\gamma}_{q,k}\cdot\boldsymbol{\eta}_{q,k})C_k \quad (78)$$

$$V_{p,k} \ge C_k - (\sum_{q=1}^{p} \boldsymbol{\gamma}_{q,k}\cdot\boldsymbol{\rho})C_k \quad (79)$$

$$X_{p,k} \ge C_k - (\sum_{q=1}^{p-1} \boldsymbol{\gamma}_{q,k}\cdot\boldsymbol{\rho})C_k \quad (80)$$

and

$$W_{p,k} \ge \frac{(\boldsymbol{\gamma}_{p,k}\cdot\boldsymbol{\eta}_{p,k})C_k}{\sum_{j=1}^{L_k} C_{j,k} - (\boldsymbol{\gamma}_{p,k}\cdot\boldsymbol{\rho})C_k}. \quad (81)$$

since $\boldsymbol{\gamma}_{q,k}\cdot\boldsymbol{\eta}_{q,k} = \boldsymbol{\alpha}_{q,k}\cdot\boldsymbol{Z}_{q,k}$, $\boldsymbol{\gamma}_{q,k}\cdot\boldsymbol{\rho} = \|\boldsymbol{\alpha}_{q,k}\|$, and $c_k = \frac{1}{C_k}\sum_{j=1}^{L_k} C_{j,k}$, $U_{p,k}, V_{p,k}, X_{p,k}$ and $W_{p,k}$ can be verified as claimed. ■

APPENDIX C: PROOF OF COROLLARY 1

Corollary 1: Let d be the maximum of worst-case delays suffered by all real-time class packets across all link servers in the network. If

$$\alpha < \frac{1}{1+(h-2)(1-\frac{1}{L})}, \quad (82)$$

then

$$d \le \frac{1}{\frac{1}{r}-(h-1)}\frac{\sigma}{\rho}, \quad (83)$$

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

$$d^{e2e} \le \frac{h}{\frac{1}{r}-(h-1)}\frac{\sigma}{\rho}, \quad (84)$$

where

$$r = \alpha \frac{L-1}{L-\alpha}, \tag{85}$$

and h is the length of the longest flow route in the network.

Proof: Recall that d is the maximum of worst-case delays suffered by all real-time class packets across all link servers in the network, and Y_k is the maximum of worst-case delays suffered by any real-time class packet upstream from Server k, therefore,

$$Y_k \leq (h-1)d. \tag{86}$$

On the other hand, by (3), we have

$$d \leq r(\frac{\sigma}{\rho} + Y_k). \tag{87}$$

Combining (86) and (87), as $\frac{1}{r} > (h-1)$, i.e., $\alpha < \frac{1}{1+(h-2)(1-\frac{1}{L})}$, we have

$$d \leq \frac{1}{\frac{1}{r} - (h-1)} \frac{\sigma}{\rho}. \tag{88}$$

Furthermore, the maximum end-to-end delay can be bounded as follows:

$$d^{e2e} \leq h\, d \leq \frac{h}{\frac{1}{r} - (h-1)} \frac{\sigma}{\rho}. \tag{89}$$

■

APPENDIX D: PROOF OF COROLLARY 2

Corollary 2: Let $\hat{d}_k$ be the maximum of worst-case delays suffered by any real-time class packet at layer-k link servers, then we have the following delay bound:

$$\hat{d}_k \leq r(r+1)^{k-1} \frac{\sigma}{\rho}, \tag{90}$$

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

$$d^{e2e} \leq ((r+1)^{\hat{h}} - 1) \frac{\sigma}{\rho}, \tag{91}$$

where r is defined in (15) and $\hat{h}$ is the number of layers in the feedforward network ($\hat{h} \geq h$).

Proof: (Little abusing parameters $\hat{d}_k$ and Y_k) We also define $\hat{d}_k$ as the worst-case delays bound suffered by any real-time class packet at layer-k link servers, and Y_k as the worst case queuing delay bound suffered by any real-time class packet upstream from layer-k link server, i.e.,

$$Y_k = \sum_{l=1}^{k-1} \hat{d}_l. \tag{92}$$

By (3), we have

$$\hat{d}_k = r(\frac{\sigma}{\rho} + Y_k). \tag{93}$$

Therefore, by (92) and (93), we have

$$\hat{d}_k - \hat{d}_{k-1} = r\hat{d}_{k-1}, \tag{94}$$

and then,

$$\hat{d}_k = (r+1)\hat{d}_{k-1} = \cdots = (r+1)^{k-1}\hat{d}_1. \tag{95}$$

We know $\hat{d}_1 = r\frac{\sigma}{\rho}$, therefore, $\hat{d}_k$, the maximum of worst-case delays suffered by any real-time class packet at layer-k link servers, can be bounded as follow:

$$\hat{d}_k \leq r(r+1)^{k-1} \frac{\sigma}{\rho}, \tag{96}$$

and Y_k, the maximum of worst-case delays suffered by any real-time class packet upstream from layer-k link server, can be bounded as follow:

$$Y_k \leq \sum_{l=1}^{k-1} \hat{d}_l \leq \sum_{l=1}^{k-1} r(r+1)^{l-1} \frac{\sigma}{\rho} = ((r+1)^{k-1} - 1) \frac{\sigma}{\rho}. \tag{97}$$

Therefore, the maximum end-to-end delay can be bounded as follows:

$$d^{e2e} \leq Y_{\hat{h}+1} \leq ((r+1)^{\hat{h}} - 1) \frac{\sigma}{\rho}. \tag{98}$$

■