

Efficient Connection Admission Control Algorithms for Adaptive QoS Real-time Connections over ATM Networks^{*}

WEIJIA JIA

Department of Computer Science, City University of Hong Kong, 83 Tat Chee Ave., Kowloon, Hong Kong
wjia@cs.cityu.edu.hk

WEI ZHAO

Department of Computer Science, Texas A&M University, College Station, TX 77843-3112 USA
zhao@cs.tamu.edu

Abstract. In this paper, we extend the traditional QoS model into one that is adaptive. We address how to admit real-time connections under this new model. We investigate various search techniques in connection admission and assess their tradeoffs under different performance criteria. We show that our connection admission control algorithm with heuristic search techniques can increase the probability that a connection is admitted, and provide the best possible QoS to admitted connections while at the same time maintaining the execution time of the algorithm at a reasonable level. Many existing CAC algorithms for real-time connections use direct testing techniques to determine if deadlines constraints can be satisfied. This is usually time consuming due to the extensive computation involved in obtaining delay bounds. We introduce an indirect testing method that can substantially reduce the execution time of CAC algorithms.

1 INTRODUCTION

In this paper, we address issues related to providing connection-oriented services to real-time applications. Our work differs from the previous studies in that we use an adaptive connection model so that a connection can receive the best possible Quality-of-Service (QoS), depending on the available resources. We aim at design and analysis of Connection Admission Control (CAC) algorithms for adaptive real-time connections. We choose ATM networks to carry out this study. ATM networks support connection-oriented communication, have high bandwidth, and have been selected by many emerging real-time applications.

Adaptive Quality-of-Service (QoS) has becoming important area for the communication networks, especially for the applications in future internet. With increasing power and resources, future (next generation) Internet services, may still have to face the various increasing demanding of QoS such as real-time, multicasting and anycasting [20, 21]. A traditional *real-time connec-*

tion is defined as one that has to meet a stringent bound on the end-to-end delay of its cells. In other words, such a connection specifies its QoS in terms of its traffic characteristics (e.g., bandwidth and burstiness) and a cell-transfer deadline. Real-time connections are useful for supporting distributed real-time applications such as supervisory command and control systems used in manufacturing, chemical processing, nuclear plants, telemedicine, warships, etc.

In connection-oriented communication, a connection must be admitted before data associated with the connection can be transmitted. Admission of a new connection is conditioned on the ability of the network to satisfy the specified QoS of the new connection without violating the QoS guarantees made to those connections that are already in the system. Traditionally, the QoS requirements are specified by fixed values (e.g., bandwidth = 4 Mb/s, deadline = 30 ms, etc.). This is a simplistic approach because it ignores the fact that for many applications, QoS requirements do not have to be constant. For example, consider an application that needs to transmit a stream of live video data through a network and has QoS requirements for bandwidth, burstiness, and deadline. If the network cannot satisfy the

^{*} An earlier version of this paper has been presented at the International Conference on Parallel and Distributed Systems, 1998. This work has been supported by City University of Hong Kong.

QoS requirements, the application may be willing to accept a “poorer” QoS, rather than being simply rejected. For the “poorer” QoS, we mean less bandwidth, less burstiness, larger deadline, or a combination of these. On the other hand, if there are more resources available such that better (than specified) QoS can be provided, then the application may also want to receive the improved QoS in order to increase its own performance as well.

In this paper, we introduce *adaptive real-time connection* (ARTC). An ARTC is a generalized traditional real-time connection. QoS specifications of an adaptive real-time connection are given by a region in the QoS parameter space, rather than fixed values. Thus, during connection admission, the network can adapt to resource availability. In this way, the best possible QoS from the specified region can be offered, if the connection is admissible. Thus, the CAC algorithm for this kind of connections has three objectives:

- to increase the probability that a connection can be admitted;
- to provide the best possible QoS to admitted connections, subject to availability of network resource; and
- to maintain the execution time of connection admission control algorithm at a reasonable level.

The first two objectives are directly related to the overall network performance provided to the users. The flexibility in QoS specification of adaptive real-time connections provides opportunity to achieve them. The third objective is particularly challenging because adaptation may take time, potentially worsening the execution time of the CAC algorithm. In this study, we investigate several CAC algorithms for adaptive real-time connections. Our goal is to develop CAC algorithms that can achieve the first two objectives without compromising the third one. We will demonstrate that one of the CAC algorithms we introduce in this paper can achieve high connection admission probability and offer good QoS to the admitted connections while at the same time not requiring as much of execution time.

To further improve the execution time of the CAC algorithms, we propose a new testing method. To admit a new real-time connection, one has to check if the deadline of the new connection can be satisfied without violating the QoS guarantees made to the existing connections. Many existing CAC algorithms for real-time connections use direct testing techniques to determine if deadline constraints can be satisfied. With the direct testing method, the delays of connections are computed first, then they are checked against the corresponding deadlines. The direct testing method is usually time consuming due to the extensive computation in obtaining delay bounds. In this paper, we will introduce an *indirect testing* method that can substantially reduce the execution time of CAC algorithms.

Considerable progress has been made recently towards solving the connection admission problem in ATM networks for real-time applications. In computation of end-to-end delay bounds, the basic approach has been viewing a connection as being served by a sequence of servers and computing the worst case delays suffered by a connection at each of the servers [1,2,3,4]. A key factor that affects the end-to-end delay of connections is the service discipline used in multiplexing different connections on one link at an ATM switch. Many of the previous studies on meeting end-to-end deadlines in ATM networks concentrated on designing and analyzing scheduling policies for ATM switches. A modified FCFS scheduling scheme called FIFO⁺ was proposed and studied in [8]. A switch scheduling policy called “Stop and Go” was presented in [9]. A virtual clock-scheduling scheme in which cells are prioritized by a virtual time stamp assigned to them was discussed in [10]. Ferrari and Verma [11] and Zheng and Shin [12] studied the use of Earliest Deadline First scheduling in wide area networks. Scheduling policies based on fair queuing and the derivation of the corresponding delay bound were discussed in [13]. Zhang and Ferrari [14] discussed how local deterministic delay bounds can be guaranteed over an ATM link for bursty traffic, even when the sum of peak rates of all the connections exceeds one. Deterministic delay bounds in networks were also studied by Yates, Kurose and Towsley in [15] and by Cruz in [1,2,16]. A negotiation of user-level QoS requirements was discussed in [19,20] presented a negotiation technique for multimedia applications. Most of these previous studies considered the fixed QoS requirements and all used direct testing methods. In this paper, we consider admission control of connections with adaptive QoS requirement in conjunction with a new indirect testing method.

The rest of this paper is organized as follows. In Section 2, we introduce network and connection models. In Section 3, we present a general framework for the problem at hand. Several CAC algorithms are discussed in Section 4. The indirect testing method, which improves CAC algorithms, is addressed in Section 5. Section 6 summarizes the paper and discusses future work.

2 NETWORK AND CONNECTION MODELS

In this section, the preliminary concepts and ARTC model will be discussed. Notations and terminology are also introduced for the use of the model and algorithm.

2.1 ATM NETWORKS

In an ATM network, hosts are connected to ATM switches and ATM switches are connected to each other by physical links. An ATM switch consists of input ports, a switching fabric, and output ports. When an ATM cell arrives at an input port of a switch, it is transported by the switching fabric to the output port where it will then be transmitted into the physical link connected to an output port.

We are interested in the worst case delay of cells. For this purpose, we model the ATM network as a collection of servers. A server is an abstraction of a network component that is traversed by the cells of a connection. Therefore, the input ports, the switching fabric, the output ports, and physical links can be modeled as servers.

The servers are classified into two categories [1,2]: constant servers and variable servers. A constant server is the one that offers a constant delay to each cell that traverses through it and does not by itself change the traffic flow characteristics of a connection. For example, physical links and the switch fabric are constant servers. The function of an input port is to de-multiplex the arriving cells based on the information in the cell header. This is achieved in constant time by the hardware associated with the input port. Thus, we can also model the input port of an ATM switch as a constant de-multiplexor server. To simplify the delay analysis, we can subtract the appropriate constant delays encountered by a connection from its deadline. Consequently, we eliminate all the constant delays from consideration and focus only on the output port controllers. In the rest of this paper, we assume that all the deadlines of connections have been modified under this assumption.

The functionality of an output port of a switch is more complex. An output port may simultaneously receive cells belonging to different connections. Cells may be buffered at an output port and transmitted in an order that is determined by the scheduling discipline. Thus, an output port should be considered as a variable server since the delay suffered by a cell varies depending upon the queue length in the buffer.

The scheduling policy at an output port controller determines the order of the cells (from different connections) being transmitted. Typical scheduling policy utilized in the ATM switches is either FIFO or priority driven. For example, the LatisCell switch by Bay Networks uses the FIFO policy while the ForeRunner switch from the Fore Systems adopts a priority driven scheduling algorithm. In this paper, we only consider the FIFO scheduling policy.

Output port controllers can also be classified as regulated or non-regulated. For a regulated output port controller, cells from a particular connection will be regulated before being transmitted over the link. Regulating traffic limits the burstiness of the traffic, increases the stability of the network, and simplifies the delay analysis [1,2]. Therefore, we focus ourselves on the regulated output port controllers.

A leaky bucket regulator is a typical mechanism used for traffic regulation [1,2]. It consists of a token bucket and a cell buffer. The cells from a connection associated with the leaky bucket are queued at the cell buffer. A waiting cell is transmitted if at least one token is available in the token bucket and a token is consumed whenever a cell is transmitted. Tokens are stored at the token buffer with a maximum size of b and tokens of a leaky

bucket are generated automatically at a rate of r . In this paper, we assume that the output port controllers are equipped with leaky bucket regulators.

2.2 ADAPTIVE REAL-TIME CONNECTIONS

There are two kinds of QoS for a connection: *the QoS requested* by the connection which is denoted as Q^r , and *the QoS offered* by the system which is denoted as Q . For a traditional real-time connection, Q^r is given by a *triple*: (β, ρ, D) where β and ρ characterize connection's traffic: during any time interval of length I , the total number of cells that enters the network from the source will be upper-bounded by $\beta + \rho I$. D is the cell-transfer deadline: the end-to-end delay of a cell should never be more than D . If a traditional real-time connection is admitted, its offered QoS will be exactly the same as the requested one. That is, $Q^r = Q$.

For our adaptive real-time connections (ARTC), conceptually, *QoS* is the quality of networks which are required as min-max limits (the worst and the best services). In our algorithm of ARTC, in order to discuss the *QoS* in terms of quantitative research, *QoS* is differentiated as the *requested QoS* and the *offered QoS*. The requested QoS of an ARTC is specified by a *pair of triples*: $(Q^{r\text{-best}}, Q^{r\text{-worst}})$ where

$$Q^{r\text{-best}} = (\beta^{\max}, \rho^{\max}, D^{\min}) \quad (2-1)$$

and

$$Q^{r\text{-worst}} = (\beta^{\min}, \rho^{\min}, D^{\max}) \quad (2-2)$$

These six parameters define a region in the QoS parameter space. We call this region the *requested-QoS region*. See Figure 1 for an illustration of such a region. This region presents the QoS values that are acceptable by the new connection. That is, if $Q = (\beta, \rho, D)$ is the QoS offered to the new connection, then following inequalities must be satisfied:

$$\beta^{\min} \leq \beta \leq \beta^{\max}, \quad (2-3)$$

$$\rho^{\min} \leq \rho \leq \rho^{\max}, \quad (2-4)$$

and

$$D^{\min} \leq D \leq D^{\max}. \quad (2-5)$$

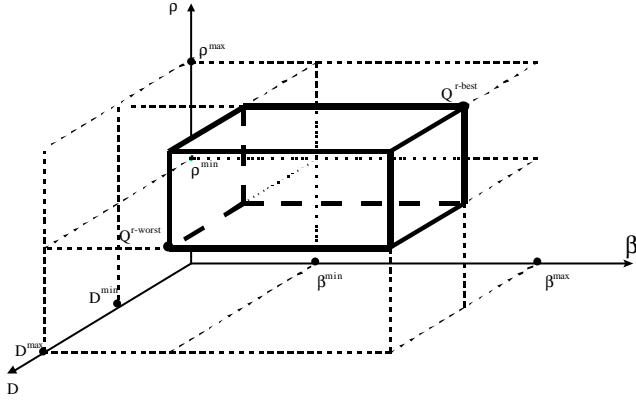


Figure 1. Requested-QoS region of a new connection

For the convenience of discussion, we define a relation among QoS triples: for $Q = (\beta, \rho, D)$, $Q' = (\beta', \rho', D')$, $Q \preceq Q'$ if and only if $\beta \leq \beta'$, $\rho \leq \rho'$, and $D \geq D'$. With this notation, an offered QoS (say, Q) must satisfy

$$Q^{r\text{-worst}} \preceq Q \preceq Q^{r\text{-best}}. \quad (2-6)$$

The goal of a CAC algorithm for adaptive real-time connections is to determine a suitable offered QoS in the requested-QoS region of a new connection such that the deadline constraints of the new and existing connections are all satisfied. As we will see in the next section, this may involve a search in the requested-QoS region. To reduce the complexity of the search, we propose to only search discrete points in the requested-QoS region. In particular, we partition each parameter dimension in a requested-QoS region into P equal-length intervals. The lengths of intervals at each dimension are as follows:

$$\Delta \mathbf{b} = \frac{\mathbf{b}^{\max} - \mathbf{b}^{\min}}{P}, \quad (2-7)$$

$$\Delta \mathbf{r} = \frac{\mathbf{r}^{\max} - \mathbf{r}^{\min}}{P}, \quad (2-8)$$

and

$$\Delta D = \frac{D^{\max} - D^{\min}}{P}. \quad (2-9)$$

We will only search the points on boundaries of intervals. That is, the QoS considered by a CAC algorithm will have its value in the form of $(\beta^{\max} - i \cdot \Delta \beta, \rho^{\max} - j \cdot \Delta \rho, D^{\min} + k \cdot \Delta D)$ where $i, j, k = 0, 1, \dots, P$. Note that there are total $(P+1)^3$ discrete points of this kind in the requested-QoS region. Once P is properly selected, these many points should provide sufficient number of choices for a CAC algorithm. Clearly, the value of P will have an impact on the system performance. We will address this issue in Section 4.

3 A GENERAL FRAMEWORK

In this section, we will first define the function of admission control algorithms. We will then introduce the performance metrics used for evaluating CAC algorithms. A general flowchart for CAC algorithms will also be presented.

3.1 PROBLEM DEFINITION

Before we go further discussion of CAC algorithm, for simplicity of discussion, in our algorithm, we only consider three parameters in this paper, i.e., the bursts, arrival rate and deadline. The other parameters such as *lose rate* and *connection cost* are not discussed due to different modeling. First, we introduce some notations. Let M_i be the i^{th} connection. Let

$$Q_i = (\beta_i, \rho_i, D_i). \quad (3-1)$$

denote its offered QoS, if M_i is admitted. Let $s(i, j)$ be the identity of the j -th server in the path of connection M_i and K_i be the total number of variable servers serving connection M_i . Thus, H_i , the sequence of servers serving connection M_i is given by

$$H_i = \langle s(i,1), s(i,2), \dots, s(i,j), \dots, s(i, K_i) \rangle \quad (3-2)$$

Let $d_{i,j}$ be the worst case delay experienced by a cell from connection M_i at server j . Then, d_i , the worst case end-to-end cell delay experienced by a cell of connection M_i is given by

$$d_i = d_{i,s(i,1)} + d_{i,s(i,2)} + \dots + d_{i,s(i,K_i)}. \quad (3-3)$$

Finally, let $\vec{d}^{[N]}$ be the vector whose components are the worst case end-to-end delays of N connections, i.e.,

$$\vec{d}^{[N]} = (d_1, d_2, \dots, d_N). \quad (3-4)$$

and let $\vec{D}^{[N]}$ be the deadline vector whose components are the (offered) deadlines of N connections.

$$\vec{D}^{[N]} = (D_1, D_2, \dots, D_N). \quad (3-5)$$

Given two vectors $\vec{X}^{[N]} = (X_1, X_2, \dots, X_N)$ and

$\vec{Y}^{[N]} = (Y_1, Y_2, \dots, Y_N)$ of size N , we denote

$$\vec{X} \leq \vec{Y} \quad (3-6)$$

if for $i = 1, 2, \dots, N$, $X_i \leq Y_i$.

Using these notations, the *function* of a CAC algorithm can be defined as follows: Assume that \mathbf{M} is the set of N adaptive real-time connections that have already been admitted by the ATM network. That is,

$$\mathbf{M} = \{M_1, M_2, \dots, M_i, \dots, M_N\}. \quad (3-7)$$

Now, connection M_{N+1} requests for admission. M_{N+1} has a requested-QoS region defined by $(Q_{N+1}^{r-best}, Q_{N+1}^{r-worst})$. The connection admission control algorithm should determine an offered QoS, $Q_{N+1} = (\beta_{N+1}, \rho_{N+1}, D_{N+1})$, for connection M_{N+1} such that

$$Q_{N+1}^{r-worst} \leq Q \leq Q_{N+1}^{r-best}, \quad (3-8)$$

and

$$\vec{d}^{[N+1]} \leq \vec{D}^{[N+1]}. \quad (3-9)$$

If such an offered QoS exists, M_{N+1} is admitted; otherwise it is rejected. (3-9) may need some explanation. Recall that for an existing connection M_i ($1 \leq i \leq N$), its offered QoS has been guaranteed by the system. When the new connection (M_{N+1}) is introduced, we need to make sure that a guarantee made to any existing connection will not be violated. M_{N+1} has no impact on the traffic parameters (β_i and ρ_i) of M_i ($1 \leq i \leq N$), because they are enforced by regulators in the individual switches. However, M_{N+1} may increase the delays of cells from M_i if both connections share some servers. Thus, individual end-to-end delays of connections M_1, \dots, M_N must be re-tested in order to ensure that they are still in the ranges defined by their offered QoS. The details on derivation of $\vec{d}^{[N+1]}$ is given in Appendix A.

3.2 PERFORMANCE METRICS

We propose the following performance metrics for evaluating CAC algorithms.

1. *Admission probability* (AP). This is defined as the probability that a connection can be admitted upon the arrival of its request. For a session, the admission probability can be estimated by

$$AP \approx \frac{\text{Number of connections admitted}}{\text{Number of connections arrived}} \quad (3-10)$$

2. *Relative QoS Error* (RE). Let M be an admitted connection which has its requested QoS $Q^r = (Q^{r-best}, Q^{r-worst}) = ((\beta^{max}, \rho^{max}, D^{min}), (\beta^{min}, \rho^{min}, D^{max}))$ and offered QoS $Q = (\beta, \rho, D)$. The relative QoS error for M is defined as follows:

$$RE = \frac{\sqrt{\left(\frac{\mathbf{b}^{max} - \mathbf{b}}{\Delta \mathbf{b}}\right)^2 + \left(\frac{\mathbf{r}^{max} - \mathbf{r}}{\Delta \mathbf{r}}\right)^2 + \left(\frac{D - D^{min}}{\Delta D}\right)^2}}{\sqrt{\left(\frac{\mathbf{b}^{max} - \mathbf{b}^{min}}{\Delta \mathbf{b}}\right)^2 + \left(\frac{\mathbf{r}^{max} - \mathbf{r}^{min}}{\Delta \mathbf{r}}\right)^2 + \left(\frac{D^{max} - D^{min}}{\Delta D}\right)^2}} \quad (3-11)$$

RE measures the relative distance between the offered QoS (Q) and Q^{r-best} . That is, the less the RE is, the closer the offered QoS is to Q^{r-best} . In particular, RE is zero if $Q = Q^{r-best}$. RE = 1 if $Q = Q^{r-worst}$. RE is between zero and one if Q is any other point in the requested-QoS region. For a system, we are usually interested in the *average* of relative QoS errors of all admitted connections.

3. *Execution Time* (ET). This is defined as the time taken for a CAC algorithm to make an admission decision.

3.3 A FLOWCHART OF CAC ALGORITHMS

It is not an easy task to design a CAC algorithm that can realize good and balanced performance among all the metrics. Before proceeding to report the details of CAC algorithms, we would like to introduce a general flowchart (See Figure 2). The flowchart shows the high-level structure and operations of CAC algorithms. The flowchart helps us understand important design issues and thus comes out with better CAC algorithms.

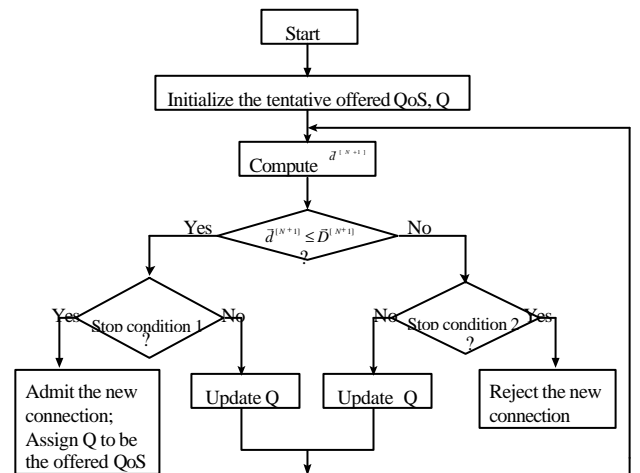


Figure 2. A general flowchart of CAC algorithm

The flowchart implies that a CAC algorithm is basically a search algorithm: it searches in the requested- QoS region in order to find an offered QoS that can satisfy both (3-8) and (3-9). Thus, the performance of a CAC algorithm depends on a critical issue: *completeness of the search*.

3.4 TWO BASIC THEOREMS

Based on the definition of “ \preceq ” relationship, we can establish two theorems as follows:

Theorem 1. Let Q and Q' be two points in a requested-QoS region. If $Q' \preceq Q$, then $RE(Q') \geq RE(Q)$.

Proof: Let $Q = (\beta, \rho, D)$ and $Q' = (\beta', \rho', D')$. $Q' \preceq Q$ implies $\rho' \leq \rho$, $\beta' \leq \beta$, and $D' \geq D$. That is,

$$\rho^{\max} - \rho' \geq \rho^{\max} - \rho, \quad (3-12)$$

$$\beta^{\max} - \beta' \geq \beta^{\max} - \beta, \quad (3-13)$$

and

$$D' - D^{\min} \geq D - D^{\min}. \quad (3-14)$$

Substituting (3-12), (3-13), and (3-14) into (3-11), we have

$$RE(Q') = \frac{\sqrt{(\mathbf{b}_{\max} - \mathbf{b})^2 + (\mathbf{r}_{\max} - \mathbf{r})^2 + (D - D_{\min})^2}}{\sqrt{(\mathbf{b}_{\max} - \mathbf{b}_{\min})^2 + (\mathbf{r}_{\max} - \mathbf{r}_{\min})^2 + (D_{\max} - D_{\min})^2}} \geq \frac{\sqrt{\left(\frac{\mathbf{b}_{\max} - \mathbf{b}}{\Delta \mathbf{b}}\right)^2 + \left(\frac{\mathbf{r}_{\max} - \mathbf{r}}{\Delta \mathbf{r}}\right)^2 + \left(\frac{D - D_{\min}}{\Delta D}\right)^2}}{\sqrt{\left(\frac{\mathbf{b}_{\max} - \mathbf{b}_{\min}}{\Delta \mathbf{b}}\right)^2 + \left(\frac{\mathbf{r}_{\max} - \mathbf{r}_{\min}}{\Delta \mathbf{r}}\right)^2 + \left(\frac{D_{\max} - D_{\min}}{\Delta D}\right)^2}} = RE(Q).$$

Q.E.D.

Theorem 1 implies that the relative QoS error $RE(Q)$ is a non increasing function of Q in term of “ \preceq ” relationship.

Theorem 2. Let Q and Q' be two points in a requested-QoS region. If $Q \preceq Q'$, then $\bar{d}^{[N+1]}(Q) \leq \bar{d}^{[N+1]}(Q')$.

See Appendix B for a proof of this theorem. Theorem 2 implies that if a system with an offered QoS (say Q) cannot make the deadline constraints satisfied, then the system with Q' where $Q \preceq Q'$ cannot either.

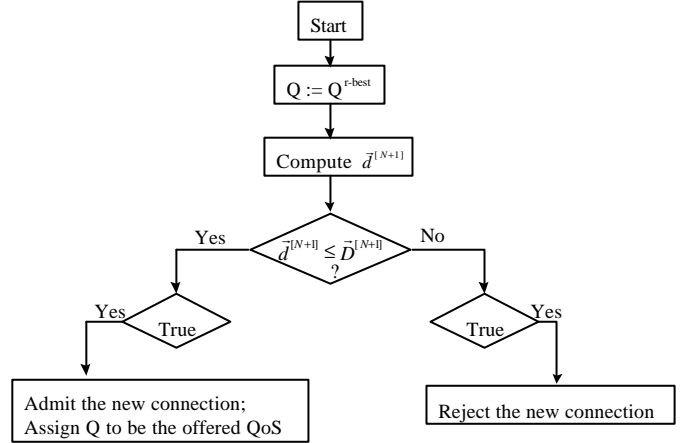
We will use these two theorems in design our CAC algorithms as discussed in the next section.

4 CONNECTION ADMISSION CONTROL ALGORITHMS

In this section, we will discuss four CAC algorithms. These algorithms use different search strategies and result in different performance in terms of AP, RE, and ET.

4.1 ALGORITHMS WITHOUT ANY SEARCH

The simplest CAC algorithms are the ones that do not perform any search at all. With such CAC algorithms, a specific QoS from the requested-QoS region is chosen to be the tentative offered QoS. If with this tentative offered QoS, delay constraints



of all the connections can be satisfied, the new connection is admitted; otherwise, it is rejected. In either case, no other offered QoS will be considered. Two particular algorithms of this kind are as follows:

- *The best-QoS algorithm (BQ).* With this algorithm, the tentative offered QoS is assigned as $Q^{\text{r-best}}$ of the new connection. See Figure 3 for the flowchart of this algorithm.
- *The worst-QoS algorithm (WQ).* With this algorithm, the tentative offered QoS is assigned as $Q^{\text{r-worst}}$ of the new connection. The flowchart is similar to Figure 3, thus it is omitted here.

Figure 3. Flowchart of the Best-QoS (BQ) algorithm

Comparing the flowcharts of algorithms BQ and WQ with the general one shown in Figure 2, we see that the stop conditions in these two algorithms are always true, implying no iteration will be carried out.

Clearly, the above algorithms will have the least execution time because they do not perform any search at all. The best-QoS algorithm can actually achieve the smallest relative QoS error because every connection it admits will have zero relative QoS error. However, it may unnecessarily reject some connections. As a result, it will have the least admission probability. On the other hand, the worst-QoS algorithm can actually achieve the highest admission probability, but it will have the worst relative QoS error because every connection it admitted has a relative QoS error 100%. That is, these two algorithms do not achieve a good balance between AP and RE. We will use them as baseline algorithms to compare the performance of other algorithms.

4.2 AN EXHAUSTIVE SEARCH ALGORITHM

Here, we discuss a CAC algorithm that does search in the requested-QoS region and chooses an offered QoS with the best (i.e., the minimum) relative QoS error (RE). A simple way to do this is to search all the points in the requested-QoS region. Obviously, this will take too much time for any non-trivial situation.

By Theorems 1 and 2, we know that in term of “ \succeq ” relationship, the delay vector is a non decreasing function of Q while the relative QoS error RE(Q) is a non increasing function of Q. This observation suggests that the CAC algorithm should examine the points in the requested-QoS region in an increasing order of RE values up to the point where the deadline constraints can be satisfied or the point which is itself $Q^{\text{-worst}}$. In the former case, the new connection is accepted and in the latter case, the new connection is rejected.

One issue has to be addressed if we want to realize the above idea in an algorithm. Because RE(Q) is a real number, it is difficult, if not impossible, to enumerate points in the decreasing order of RE(Q) values. To deal with this problem, we define a function as follows: for $Q = (\beta, \rho, D)$,

$$f(Q) = \left(\frac{\mathbf{b}^{\max} - \mathbf{b}}{\Delta \mathbf{b}}\right)^2 + \left(\frac{\mathbf{r}^{\max} - \mathbf{r}}{\Delta \mathbf{r}}\right)^2 + \left(\frac{D^{\min} - D}{\Delta D}\right)^2 \quad (4-1)$$

Note that by the definitions of $\Delta \beta$, $\Delta \rho$, and ΔD , $f(Q)$ is an integer. In particular, $f(Q^{\text{r-best}}) = 0$ and $f(Q^{\text{r-worst}}) = 3P^2$ where P is the number of intervals in a QoS dimension.

It is easy to see that if for any two points (say Q and Q') in a requested-QoS region, $RE(Q') < RE(Q)$, then $f(Q') < f(Q)$, and vice versa. That is, instead of searching all the points in the increasing order of RE, the algorithm can equivalently search all the points in an increasing order of the $f()$ values. Using the values of $f()$ instead of RE to guide the search greatly simplifies the algorithm.

Note that there are usually multiple points for a given $f()$ value. The algorithm should exhaustively examine all the points for a given $f()$ value before looking for the points with a larger $f()$ value. For this purpose, let us define $QS(C)$ to be a set of QoS triples where $f(Q) = C$. That is,

$$QS(C) = \{Q | Q \text{ is in the requested-QoS region and } f(Q) = C\}. \quad (4-2)$$

Using these notations, we develop an *exhaustive search algorithm* (ES). The flowchart of the algorithm is shown in Figure 4. The algorithm intends to search the requested-QoS region and tries to find an offered QoS which can have the least $f()$ value (hence, the least RE value) such that all the deadline constraints of connections are satisfied. The algorithm starts with C equal to zero. It generates set $QS(C)$. It then repeatedly removes an element (Q) from $QS(C)$ and tests if with this Q, the deadline constraints can be satisfied. If the test passes, then the algorithm admits the new connection. If no test is passed but $QS(C)$ becomes empty, the C value is

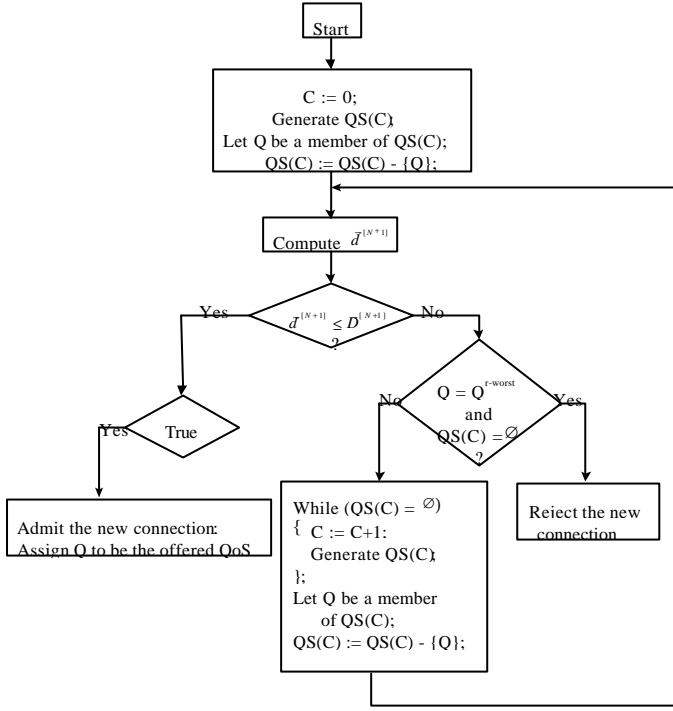


Figure 4. Flowchart of the exhaustive search (ES) algorithm

increased and a new $QS(C)$ is generated. The algorithm iterates until it either admits the new connection or all the points in the requested-QoS region have been examined. In the latter case, the new connection is rejected.

In comparison with the best-QoS (BQ) and the worst-QoS (WQ) algorithms, the ES algorithm should achieve a better balance between the admission probability and relative QoS error. The ES algorithm makes the best effort to admit a connection with the smallest RE possible while the BQ and WQ algorithms only attempt some extreme points in the requested-QoS region. A problem with the ES algorithm is its execution time. The total number of points in a requested-QoS region is $(P + 1)^3$. In the worst case, the ES algorithm will examine all of them. This may take too much time to be acceptable by many practical applications.

4.3 A HEURISTIC SEARCH ALGORITHM

In this subsection, we develop a *heuristic search algorithm* (HS). Different from the ES algorithm discussed in the last subsection, the heuristic search algorithm will not search entire requested-QoS region, it only searches on the line segment connecting $Q^{r-best} = (\beta^{max}, \rho^{max}, D^{min})$ and $Q^{r-worst} = (\beta^{min}, \rho^{min}, D^{max})$ in the requested-QoS region. Note that there are only $P + 1$ points on this line segment. Let these points be R_0, R_1, \dots, R_P . They are in the form of

$$R_i = (\beta^{min} + i*\Delta\beta, \rho^{min} + i*\Delta\rho, D^{max} - i*\Delta D) \quad (4-3)$$

where $i = 0, 1, \dots, P$. Note that $R_0 = Q^{r-worst}$ and $R_P = Q^{r-best}$. Because the HS algorithm will search fewer number of points, its execution time can be much less than that of the ES algorithm.

Before describing the details of the algorithm, we would like to make some observations on the points along the line segment. Based on the definition of “ \preceq ”, and Theorems 1 and 2, we have

$$Q^{r-worst} = R_0 \preceq R_1 \preceq \dots \preceq R_P = Q^{r-best}, \quad (4-4)$$

$$\begin{aligned} RE(Q^{r-worst}) = RE(R_0) &> RE(R_1) > \dots > RE(R_P) \\ &= RE(Q^{r-best}), \end{aligned} \quad (4-5)$$

and

$$\begin{aligned} &\vec{d}^{[N+1]}(Q^{r-worst}) \\ &= \vec{d}^{[N+1]}(R_0) \leq \vec{d}^{[N+1]}(R_1) \leq \dots \leq \vec{d}^{[N+1]}(R_P) \\ &= \vec{d}^{[N+1]}(Q^{r-best}) \end{aligned} \quad (4-6)$$

Note that the line segment connecting $Q^{r-best} = (\beta^{max}, \rho^{max}, D^{min})$ and $Q^{r-worst} = (\beta^{min}, \rho^{min}, D^{max})$ contains relatively a small number of points in comparison with that in the entire required QoS region. However, (4-4), (4-5), and (4-6) suggest that the $P + 1$ points on the line segment represent a wide range of QoS values, implying that an algorithm uses these $P + 1$ points may not have to compromise much on choosing proper offered QoS for a new connection.

Furthermore, (4-4), (4-5), and (4-6) indicate that the $P + 1$ points on the line segment are *ordered* in terms of the relative error and delay values. This suggests that a binary search technique can be used in order to further reduce the execution time of the algorithm.

See Figure 5 for the flowchart of this algorithm. The algorithm maintains a window which defines the range of points to be searched inclusively. Initially, the window is assigned as follows:

$$[R_{low}, R_{up}] = [R_0, R_P], \quad (4-7)$$

where *low* and *up* are the indices of the lower bound and upper bound of the window, respectively. Once a window is determined, R_{test} , a point in the window is chosen as the tentative offered QoS, where $low \leq test \leq up$.

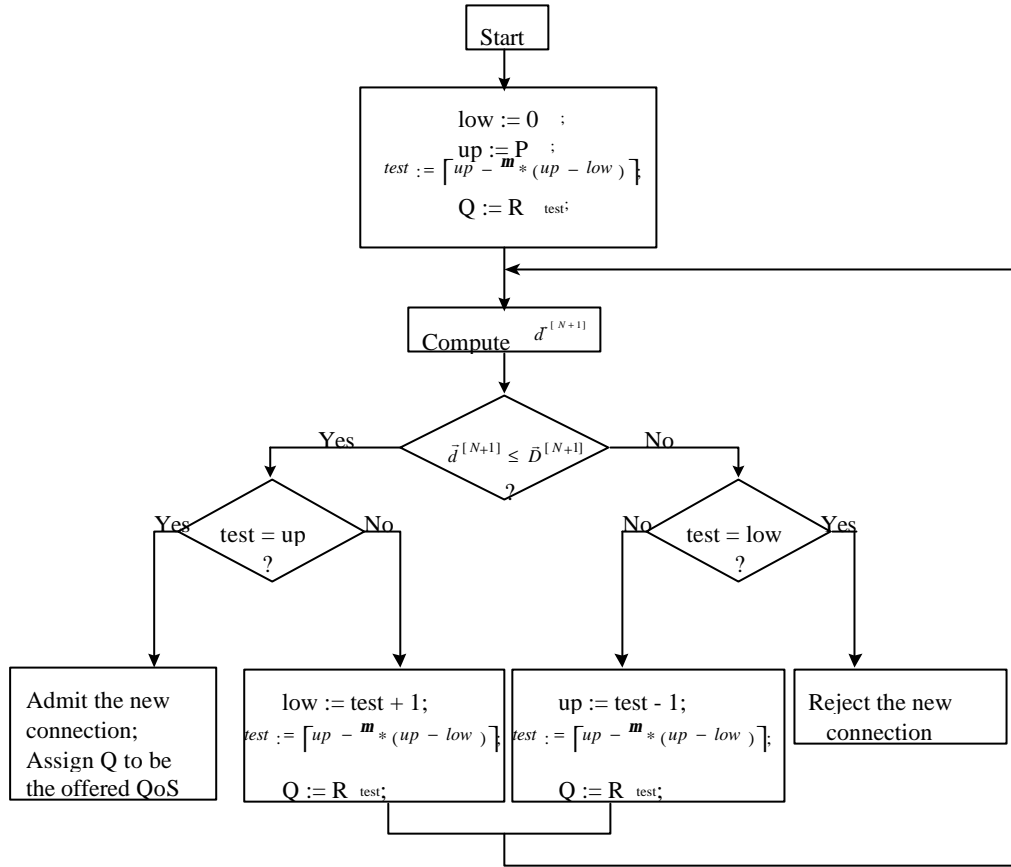


Figure 5. Flowchart of the heuristic search(HS) algorithm

For a given window $[R_{low}, R_{up}]$, selection of R_{test} is based on the system load. We observe that if the system load is very heavy, then the chance for a connection to be admitted with small RE value is small. Hence, R_{test} should be close to R_{low} . On the other hand, if the system load is light, then R_{test} should be close to R_{up} . By choosing R_{test} in such a biased manner, the algorithm can converge faster. In particular, we compute R_{test} by the following formula:

$$test = \lceil up - \mu * (up - low) \rceil \quad (4-8)$$

where μ is the maximum server utilization. The utilization of a server is computed by summation of all the ρ values of the connections that are served by the server.

Once the value of R_{test} is determined, the delay vector is computed. Then, we test if the deadline constraints can be satisfied, we have two cases to be considered:

Case 1. R_{test} can make the deadline constraints satisfied. If $R_{test} = R_{up}$, then the algorithm has already searched the entire window including the upper bound. Hence, the algorithm terminates and the new connection is admitted with the offered QoS R_{test} . Otherwise, we will have to modify the window and con-

tinue the search. Note that in this case, we do not have to consider points $R_{low}, R_{low+1}, \dots, R_{test-1}$ any more. Due to Theorem 1, these points will not result in better (smaller) RE than R_{test} . Thus, we only need to consider the points between R_{test+1} and R_{up} inclusive. That is, the window should be updated as follows:

$$[R_{low}, R_{up}] = [R_{test+1}, R_{up}]. \quad (4-9)$$

Case 2 R_{test} cannot make the deadline constraints satisfied. In this case, if $R_{test} = R_{low}$, then the algorithm has already searched the entire window including the lower bound. No point in the window can make the deadline constraints satisfied. The algorithm terminates and the new connection is rejected. Otherwise, we do not have to consider points $R_{test+1}, R_{test+2}, \dots, R_{up}$. This is because if R_{test} cannot make the deadline constraints satisfied, none of these points can (due to Theorem 2). In this case, the window should be updated as follows:

$$[R_{low}, R_{up}] = [R_{low}, R_{test-1}]. \quad (4-10)$$

In either case, if the algorithm does not terminate, then a new R_{test} is chosen between R_{low} and R_{up} . The algorithm repeats the above steps.

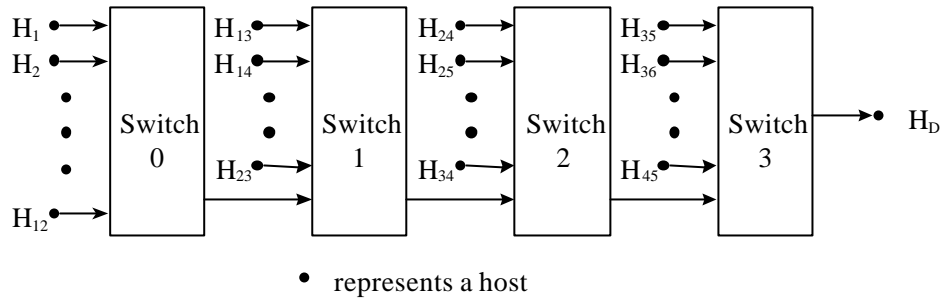


Figure 6. ATM network architecture used in the simulation

In brief, we use three kinds of heuristic techniques in this algorithm:

1. We do not search the entire requested-QoS region but constrain the search to be performed only on the line segment connecting R_{low} and R_{up} .
2. We perform a binary search over the segment, rather than a sequential one.
3. We select the tentative offered QoS in a biased manner, depending on the system current loading condition.

All these heuristics intend to reduce the time taken to search without compromising too much on other performance measures. As we will demonstrate in the next subsection, this goal is indeed achieved.

4.4 PERFORMANCE EVALUATION

In this subsection, we will evaluate the performance of the CAC algorithms introduced in this paper. We first describe the system architecture considered and then present the performance results.

4.4.1 Experimental system setup

The network which we consider consists of four switches. They are connected as shown in Figure 6. To obtain the performance data, we use a discrete event simulation model to simulate the ATM network. The simulation program is written in C++ programming language and runs in a PENTIUM 586(166MHz)/Windows 95 environment. There are several system parameters: $1/I$ is defined as the mean inter-arrival time between the requests of connection establishments; t is the mean lifetime of each connection. Both the inter-arrival time and the lifetime are chosen from exponential distributions. β^{min} , ρ^{min} , and D^{min} of a connection are generated by exponential distributions. Once they are generated, β^{max} , ρ^{max} , and D^{max} are assigned as follows:

$$\beta^{max} = (1 + \alpha_\beta)\beta^{min} \quad (4-11)$$

$$\rho^{max} = (1 + \alpha_\rho)\rho^{min} \quad (4-12)$$

and

$$D^{max} = (1 + \alpha_D)D^{min} \quad (4-13)$$

where $\alpha_\beta > 0$, $\alpha_\rho > 0$, and $\alpha_D > 0$. They are called *QoS range factors*.

The maximum demand link utilization is given $u = I * \bar{r} / t$ where $\bar{r} = (\bar{r}^{max} + \bar{r}^{min}) / 2$. The destination of all the connections is host H_D , and the source host of a connection is randomly selected from hosts H_1, H_2, \dots, H_{45} .

4.4.2 Performance Results

Sensitivity of P

Recall that P is the number of intervals along a QoS parameter dimension in a requested-QoS region. In some sense, P presents the granularity of the search that can be performed by a CAC algorithm. The larger the P is, the finer the granularity is. We evaluate the impact of the value of P on the system performance. The results are shown in Figure 7. We test three different values of P: P = 4, P = 8, and P = 16. In the simulation, the QoS range factors are 40%. The algorithm used in this experiment is the ES algorithm. For the other algorithms, the results are similar. We do not show them here due to the space limitation.

From Figure 7 (a), we see that the different P values result in almost identical admission probability (AP). That is, AP is not much sensitive to the values of P. From Figure 7 (b), we see that the average relative QoS error is sensitive to the values of P. The lesser the P is, the larger the relative error is. That is, a finer granularity of search can improve the QoS offered to the connections. But, we note that the difference of the average relative error is much smaller when P changes from 8 to 16 than when it changes from 4 to 8. Figure 7 (c) shows that the execution time of the CAC algorithm is very sensitive to the values of P. The average execution time increases proportionally as the value of P increases. From these observations, it seems to us that P = 8 is perhaps the most proper selection of P. Some performance

measures may suffer too much if the value of P is too large or too small. We will use $P = 8$ in the rest of this paper.

Comparison of Four Algorithms

We now compare the performance of four algorithms which we have introduced in this paper. The results are shown in Figure 8. From this figure, we make the following observations:

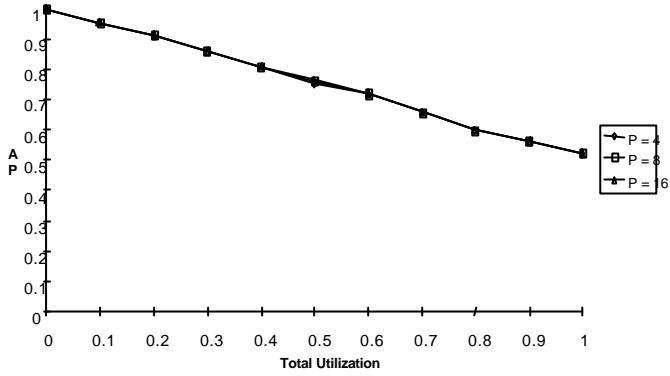
- As expected, the best-QoS algorithm (BQ) has the worst admission probability (AP) but the best average relative QoS error (RE). On the other hand, the worst-QoS algorithm (WQ) has the best AP but the worst RE (RE = 100%). Thus, these two algorithms cannot achieve a good balance between AP and RE. Both the BQ and WQ algorithms have the least average execution time (ET) because they do not perform any search at all.
- The two search algorithms, the exhaustive search (ES) and heuristic search (HS), do better in achieving a good balance between AP and RE. They have more or less comparable values on the relative errors and admission probability. Note that only $P + 1$ points are available to be searched by the HS algorithm, which are substantially less than that of the ES algorithm. However, because those $P + 1$ points offer a wide range of representative QoS values, the HS algorithm does not seem to suffer much of the performance in terms of AP and RE.
- However, the average execution time of the HS and ES algorithms is very different. The average execution time of the exhaustive search algorithm (ES) is much more than that of heuristic search algorithm (HS). For example, when the total utilization is 0.6, the average execution time taken by the ES algorithm is 71,900 ms while the time taken by the HS algorithm is only 780 ms. This is mainly because the number of iterations taken by the HS algorithm is much less.
- More interestingly, the average execution time of the HS algorithm does not always increase with the utilization like other algorithms do. For example, when the total utilization is changed from 0.6 to 0.8, the time taken by the HS algorithm is reduced from 780 ms to 600 ms. This is the effect of our biased binary search. In the biased binary search, the search point (R_{test}) is set in accordance with the system load. Hence, when the load is very heavy, the search window reduces faster than the median load, resulting in less execution time.
- Over all, the heuristic search algorithm achieves reasonably high admission probability, low relative QoS error, and low execution time. Thus, our design objectives stated in Section 1 are realized with this algorithm.

5 IMPROVED CAC ALGORITHMS

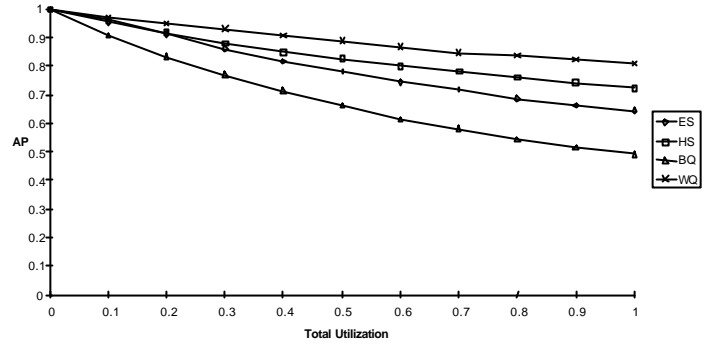
Consider the general flowchart shown in Figure 2 again. From our experiments, we find that a CAC algorithm usually spends much of the execution time in computing the delay vector which is needed in order to test if the deadline constraints can be satisfied. In this section, we consider methods to improve the efficiency of the testing method.

5.1 THE BASIC IDEA

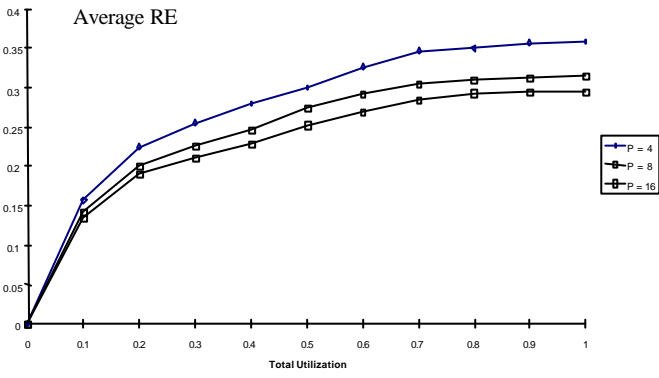
The basic idea is to use *feasible region* in testing. For a given connection that requests admission, a feasible region is a region in a QoS parameter space such that if the offered QoS of the new connection is in the region, then all the deadline constraints of the new and existing connections can be satisfied. That is, any offered QoS in the feasible region will make (3-9) hold. We call this method as *indirect testing* method because it does not directly check the delays against to the deadlines. Instead, it tests if the offered QoS is in the feasible region. As we will show, our new indirect testing method can reduce execution time of CAC algorithms substantially.



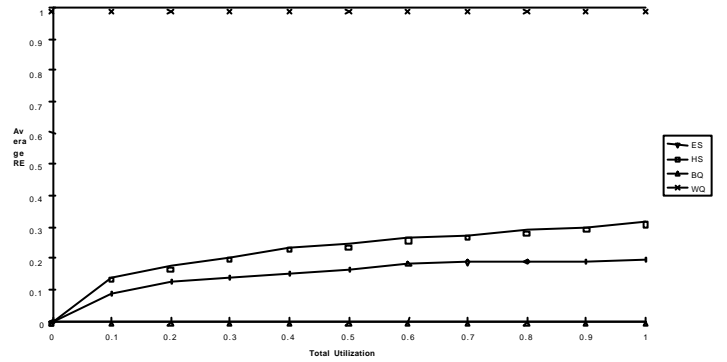
(a) Admission



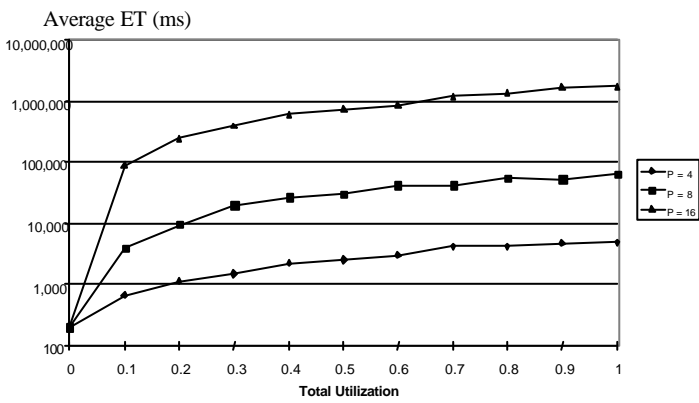
(a) Admission Probability



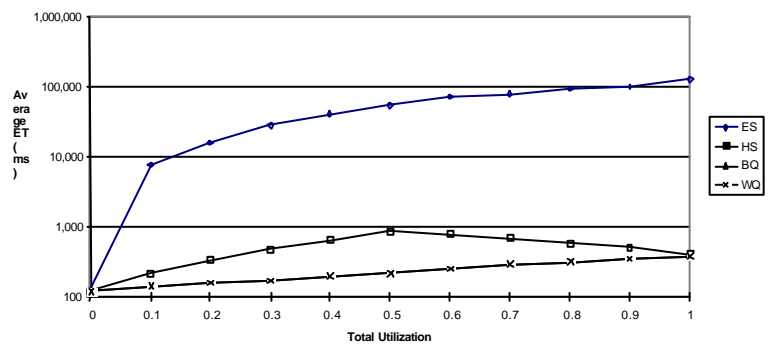
(b) Average QoS Error



(b) Average relative QoS Error



(c) Average Execution Time



(c) Average Execution Time(ms)

Figure 7. Sensitivity of P

Figure 8. Performance comparisons of four algorithms

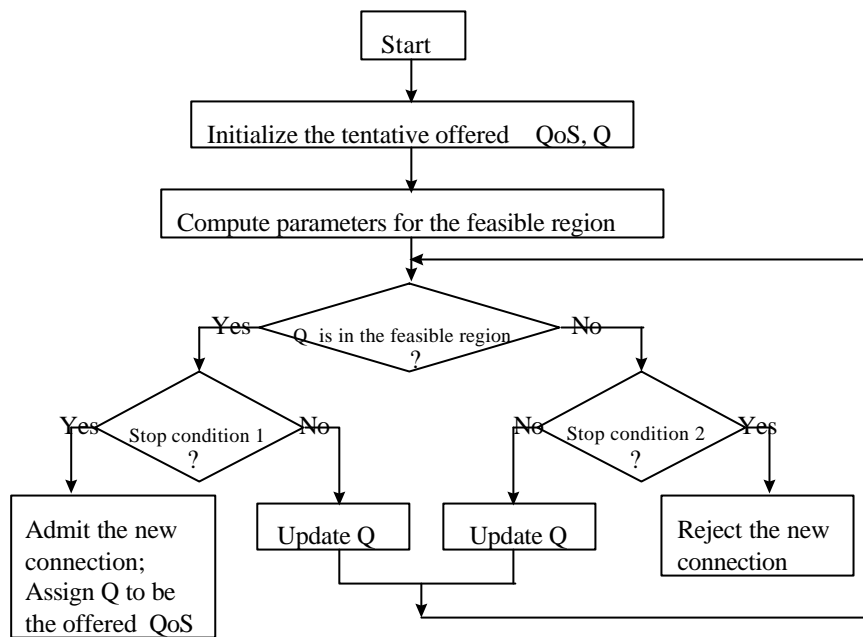


Figure 9. A general flowchart of CAC algorithm with the indirect testing method

By the definition of the feasible region, the admission problem now becomes the one to find an offered QoS which can be in both the feasible region and the requested-QoS region of the new connection. Formally, for a new connection, let Q^r be its requested-QoS region and Q^f be its feasible-QoS region. The new connection can be admitted if

$$Q^r \cap Q^f \neq \emptyset; \quad (5-1)$$

otherwise, it is rejected. Thus, once the feasible region is obtained, (5-1) instead of (3-8) and (3-9) can be used to determine if the new connection should be admitted.

While the method suggested by using (5-1) seems to be simple and straightforward, it has two problems:

1. The intersection operation is complicated because it involves intersecting two polyhedrons in multiple dimension space.
2. Once we find that (5-1) holds, we will have to determine an offered QoS which should be a point in $Q^r \cap Q^f$. This will have to be a search process if we want to find the offered QoS that can minimize the relative QoS error.

Given that a search in QoS parameter space is unavoidable, we propose to integrate the search process suggested in Figure 2 with the idea of indirect testing. The general flowchart for the improved CAC algorithms is shown in Figure 9. In comparison

with Figure 2, we see that instead of computing the delay vector and then testing if the deadlines can be satisfied, the new procedure will compute the feasible region and test if the tentative offered QoS is in the feasible region. We expect the new procedure to take much less time in execution due to the following reasons:

- Usually, it takes much less time to derive parameters of the feasible region than to compute the delay vector.
- In Figure 2, the delay vector has to be computed in every iteration because the delay vector depends on the tentative offered QoS. However, with the new procedure shown in Figure 9, the parameters of the feasible region only need to be computed once, because they are independent from the offered QoS.

The questions which remain to be addressed are how to derive a feasible region and how much the performance (execution time) can be improved.

5.2 DERIVATION OF FEASIBLE REGION

For a single switch system, we can establish the following theorem which helps to derive a feasible region.

Theorem 3. Let a system have N existing connections which have their offered QoS parameters given (3-1). Any point $(\beta_{N+1}, \rho_{N+1}, D_{N+1})$ in the feasible region of connection M_{N+1} satisfies inequalities (5-2 through 5-4), then its delay can be guaranteed. The inequalities are shown below

$$0 \leq \rho_{N+1} \leq 1 - \mu; \quad (5-2)$$

and, if $D_{\min} = \min_{i=1, \dots, N+1}(D_i) \leq \bar{\omega} + \mu^* \xi$,

$$0 \leq \beta_{N+1} \leq D_{\min} - \bar{\omega} + \xi^*(1 - \mu - \rho_{N+1}); \quad (5-3)$$

otherwise, if $D_{\min} > \bar{\omega} + \mu^* \xi$,

$$0 \leq \beta_{N+1} \leq (D_{\min} - \bar{\omega}) * (1 - \rho_{N+1}) / \mu, \quad (5-4)$$

where

$$\mu = \sum_{i=1, \dots, N} \rho_i, \quad \bar{\omega} = \sum_{i=1, \dots, N} \beta_i,$$

and

$$\xi = \max_{i=1, \dots, N} (\beta_i / (1 - \rho_i)).$$

Proof: We ought to prove if the offered QoS of the new connection is in the feasible region, then the deadline constraints defined in (3-9) can always be satisfied. Let the offered QoS be $(\beta_{N+1}, \rho_{N+1}, D_{N+1})$ and assume that it is in the feasible region. That is, (5-2), (5-3), and (5-4) are satisfied. The theorem can then be proved by considering two cases: $D_{\min} = D_{N+1}$ and $D_{\min} \neq D_{N+1}$.

Case 1. $D_{\min} = D_{N+1}$. We have two subcases: $D_{N+1} \leq \mathbf{v} + \mathbf{m}^* \mathbf{x}$ and $D_{N+1} > \mathbf{v} + \mathbf{m}^* \mathbf{x}$.

Subcase 1.1. $D_{N+1} \leq \mathbf{v} + \mathbf{m}^* \mathbf{x}$. By (5-4), we have

$$\begin{aligned} \beta_{N+1} &\leq D_{\min} - \bar{\omega} + \xi^*(1 - \mu - \rho_{N+1}) \\ &\leq \bar{\omega} + \mu^* \xi - \bar{\omega} + \xi^*(1 - \mu - \rho_{N+1}) \\ &\leq \xi^*(1 - \rho_{N+1}) \end{aligned} \quad (5-5)$$

By (A-9),

$$\begin{aligned} d_* &= \mathbf{b}_{N+1} + \mathbf{x}^* \mathbf{r}_{N+1} + \mathbf{v} + (\mathbf{m} - 1) * \mathbf{x} \\ &\leq D_{N+1} - \mathbf{v} - (\mathbf{m} - 1) * \mathbf{x} - \mathbf{x}^* \mathbf{r}_{N+1} \\ &\quad + \mathbf{x}^* \mathbf{r}_{N+1} + \mathbf{v} + (\mathbf{m} - 1) * \mathbf{x} = D_{N+1} \end{aligned} \quad (5-6)$$

where d_* is the worst case delay of cells at the server¹.

Subcase 1.2. $D_{N+1} > \mathbf{v} + \mathbf{m}^* \mathbf{x}$. We consider two subcases: $\mathbf{b}_{N+1} \geq \mathbf{x} - \mathbf{x}^* \mathbf{r}_{N+1}$ and

$$\mathbf{b}_{N+1} < \mathbf{x} - \mathbf{x}^* \mathbf{r}_{N+1}.$$

Sub-subcase 1.2.1. $\mathbf{b}_{N+1} \geq \mathbf{x} - \mathbf{x}^* \mathbf{r}_{N+1}$. By (A-9),

$$\begin{aligned} d_* &= \mathbf{b}_{N+1} + (\mathbf{r}_{N+1} + \mathbf{v} - 1) * \frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} + \mathbf{v} \\ &\leq \frac{D_{N+1} - \mathbf{v}}{\mathbf{m}} (1 - \mathbf{r}_{N+1}) + (\mathbf{r}_{N+1} + \mathbf{m} - 1) \\ &\quad * \frac{D_{N+1} - \mathbf{v}}{\mathbf{m}} (1 - \mathbf{r}_{N+1}) + \mathbf{v} \\ &\quad \frac{D_{N+1} - \mathbf{v}}{\mathbf{m}} (1 - \mathbf{r}_{N+1}) + \mathbf{v} \\ &= \frac{D_{N+1} - \mathbf{v}}{\mathbf{m}} (1 - \mathbf{r}_{N+1}) + (\mathbf{r}_{N+1} + \mathbf{m} - 1) \\ &\quad * \frac{D_{N+1} - \mathbf{v}}{\mathbf{m}} + \mathbf{v} = D_{N+1} \end{aligned} \quad (5-7)$$

Sub-subcase 1.2.2. $\mathbf{b}_{N+1} < \mathbf{x} - \mathbf{x}^* \mathbf{r}_{N+1}$.

Once again, by (A-9),

$$\begin{aligned} d_* &= \mathbf{b}_{N+1} + \mathbf{x}^* \mathbf{r}_{N+1} + \mathbf{v} + (\mathbf{m} - 1) * \mathbf{x} \\ &\leq \mathbf{x} - \mathbf{x}^* \mathbf{r}_{N+1} + \mathbf{x}^* \mathbf{r}_{N+1} + \mathbf{v} + (\mathbf{m} - 1) * \mathbf{x} \\ &= \mathbf{v} + \mathbf{m}^* \mathbf{x} \\ &\leq D_{N+1} \end{aligned} \quad (5-8)$$

By (5-6), (5-7), and (5-8), we have

$$d_* \leq D_{N+1}. \quad (5-9)$$

That is, for any i , $1 \leq i \leq N + 1$,

$$d_i \leq d_* \leq D_{N+1} \leq D_i. \quad (5-10)$$

Thus,

$$\vec{d}^{[N+1]} \leq \vec{D}^{[N+1]}. \quad (5-11)$$

¹ In (A-9), parameters $\bar{\omega}$, μ , ξ , have indexed by the server id. Because we are dealing a single switch network here, the server index is omitted. Also note that for $i = 1, 2, \dots, N + 1$, $m_{ij} = 1$ if j is one, otherwise it is zero.

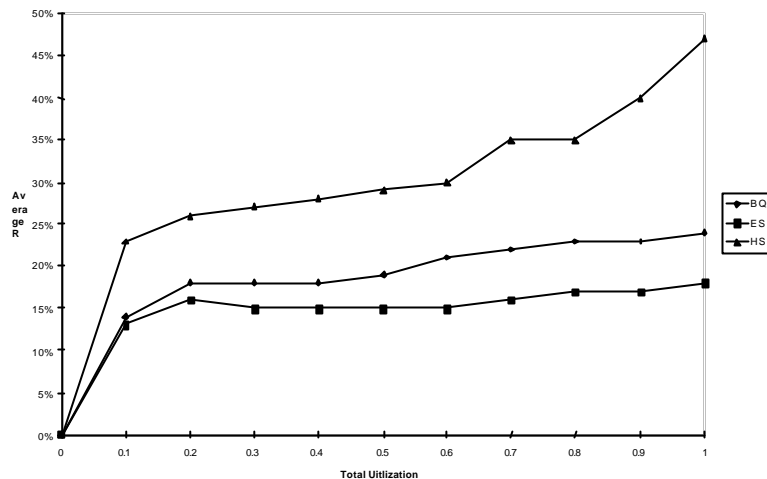


Figure 10. Performance improvement by the indirect testing method.

The proof for the case 2 (i.e. $D_{\min} \neq D_{N+1}$) is similar, we leave it to our reader as an exercise.

Q.E.D.

A similar approach can be taken to derive feasible region for a multiple switch network. However, the process is much more complicated. We will not present it here due to the space limitation.

5.3 PERFORMANCE EVALUATION OF IMPROVED CAC ALGORITHMS

In this subsection, we would like to assess the performance improvement due to using the indirect testing method. We modify the CAC algorithms accordingly to incorporate the new indirect testing method. For each algorithm, we evaluate the performance improvement by the following ratio:

$$R = \frac{E[ET] - E[ET']}{E[ET]} \quad (5-12)$$

where $E[ET]$ and $E[ET']$ are the average execution time with the original (direct testing) and the improved (indirect testing) algorithms, respectively.

The same network used in Section 4.4 was simulated. Simulation results are shown in Figure 10. Because the BQ algorithm has the exact same execution time as the WQ algorithm, only the result for the BQ algorithm is shown. From this figure, we have the following observations:

- The improvement to the ES algorithm is the smallest. This is because the ES algorithm in each iteration has to compute the values of $f()$ which also takes considerable amount of time.

- The improvement to the BQ (as well as WQ) algorithm is between the ES and HS algorithms. This is because the BQ (as well as WQ) algorithm does not do search at all. As a result, delay computation in the original program takes less portion of time than that of the HS algorithm.
- The improvement to the HS algorithm is the largest among these algorithms. This is because the HS algorithm uses three kinds of heuristic techniques and the operations for the HS algorithm are simpler. As a result, percentage improvement of the HS algorithm is greater. This further justifies our recommendation of the HS algorithm.

6 FINAL REMARKS AND FUTURE DIRECTIONS

In this paper, we addressed admission control problem for adaptive real-time connections. Different from the traditional QoS model, the QoS requirements for our adaptive real-time connections are given as a region in QoS parameter space. This allows the network to be more flexible when it intends to admit a new connection. We investigate various search techniques used in the CAC algorithms. We study the tradeoffs among performance measurements. We propose an indirect testing technique that can further reduce the execution time of the CAC algorithms. We demonstrate that the heuristic search algorithm we introduced in this paper can achieve high connection admission probability and offer good QoS to the admitted connections while at the same time not requiring much of execution time.

The work reported in this paper is preliminary. Many extensions can be made. The deadline constraints we considered in this paper are “hard” in the sense that once the connections are admitted, they must always be satisfied. It is possible to consider connections with soft real-time constraints. With our CAC algorithms reported in this paper, the offered QoS to an admitted

connection is fixed once the connection has been selected. The network performance can be further improved if we also adjust the offered QoS for admitted connections whenever necessary. This will be particularly useful when dealing with faulty situations.

In our simulation, we have provided several traffic parameters such as mean inter-arrival time between the requests of connection establishments and the mean lifetime of each connection. Both the inter-arrival time and the lifetime are chosen from exponential distributions. QoS parameters β , ρ and D of a connection are also generated by exponential distributions. These parameters provided to the ATM network are used for the purpose of analyzing the availability of our CAC algorithm. However some insight must be given here on the characteristics of the applications used when more QoS parameters are taken into consideration. In fact, it is difficult, if not impossible, for an application to specify all its QoS parameters and then to negotiate with the CAC algorithm. Future work of the CAC system should be able to derive the QoS parameters of a new connection request so as the parameters can be applied to the testing with the existing parameters of feasible region. Auto-configuration of QoS parameters is desirable and becoming important for the future multimedia applications in which both QoS requirements and Internet network connections (resources) will be highly dynamic, fast-speed and heterogeneous.

Acknowledgment

Authors are grateful to the referees for valuable comments which help the improvement of the paper. Yi-bo Xue has helped for the work. This work was partially sponsored by City University of Hong Kong under grants 7000700 and 7000664. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing City University of Hong Kong.

7 REFERENCES

- [1] R. L. Cruz. A calculus for network delay. *IEEE Transactions on Information Theory*, 37(1):114-131, Jan. 1991.
- [2] R. L. Cruz. A calculus for network delay, part 2: Network analysis. *IEEE Transactions on Information Theory*, 37 (1): 132-141, Jan. 1991.
- [3] A. Raha, S. Kamat and W. Zhao. Guaranteeing end-to-end deadlines in ATM networks. *Proceedings of ICDCS'95*, June 1995.
- [4] A. Raha, S. Kamat, and W. Zhao. Admission control for hard real-time connections in ATN LANs. *Proceedings of IEEE INFOCOM'96*, Mar. 1996.
- [5] ATM Forum. *ATM Forum-ATM User-Network Interface Specification Version 3.1*, 1995.
- [6] S. J. Golestani. A stop-and-go queuing framework for congestion management. *Proceedings of SIGCOMM'90*, pages 8-18, Sept. 1990.
- [7] H. Zhang and D. Ferrari. Rate-controlled static priority queuing. *Proceedings of IEEE INFOCOM'93*, pages 227-236, March 1993.
- [8] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: architecture and mechanism. *Proceedings of ACM SIGCOMM'92*, pages 14-26, Aug. 1992.
- [9] S. J. Golestani. A framing strategy for congestion management. *IEEE Journal on Selected Areas in Communications*, 9(7):1064-1077, Sept. 1991.
- [10] L. Zhang. Virtual clock: a new traffic control algorithm for packet switching networks. *Proceedings of ACM SIGCOMM'90*, pages 19-29, Sept. 1990.
- [11] D. Ferrari and D. C. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, SAC-8(3):368-379, Apr. 1990.
- [12] Q. Zheng and K. G. Shin. On the ability of establishing real-time channels in point-to-point packet-switch networks. *IEEE Trans. Communication*, 1996.
- [13] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. *Proceedings of ACM SIGCOMM'89*, pages 1-12, Sept. 1989.
- [14] H. Zhang and D. Ferrari. Rate-controlled service disciplines. *Journal High Speed Networks*, Vol.3, No.4, pages 389-412, 1994.
- [15] D. Yates, J. Kurose, D. Towsley and M. G. Hluchyj. On per-session delay distributions and the call admission problem for real-time applications with QoS requirements. *Proceedings of ACM SIGCOMM'93*, pages 2-12. Sept. 1993.
- [16] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected areas in Communications*, 13(6):1048-1056, Aug. 1995.
- [17] C. Li, A. Raha and W. Zhao. Stability in ATM networks. *Proceeding of IEEE INFOCOM'97*, March 1997.
- [18] ATM Forum Technical Committee. Private network-network specification interface V1.0 (PNNI 1.0). *af-pnni-0055.000*, Mar. 1996.
- [19] D. Chen, R. Colwell, H. Gelman, P. K. Chrysanthis and D. Mosse. A framework for experimenting with QoS for multimedia services. *Proceeding of Multimedia Computing and Networking*, pages 186-197, Jan. 1996.
- [20] P. J. Reynolds, H. W. Peter Beadle. Quality of service support for networked multimedia workstations, *Proceeding of Multimedia Computing and Networking*, pages 198-210, Jan. 1996.
- [21] S. Deering and R. Hinden. Internet Protocol, Version 6, Specification. *RFC 1883*, Ipsilon, Networks, Dec. 1995.

[22] D. Xuan, W. Jia, and W. Zhao. Routing Algorithms for Anycast Messages, *Proceedings ICCP98*, Aug., 28-31, 1998, Minneapolis, USA, pp. 122-130.

APPENDIX A: COMPUTING THE WORST CASE END-TO-END DELAY

In this appendix, we present the method of computing the worst case end-to-end delay of a connection.

Let $d_{i,j}$ be the worst case delay experienced by a cell from connection M_i at server j . Then, d_i , the worst case end-to-end cell delay experienced by a cell of connection M_i is given by

$$d_i = d_{i,s(i,1)} + d_{i,s(i,2)} + \cdots + d_{i,s(i,K_i)} \quad (\text{A-1})$$

where $s(i, 1)$, $s(i, 2)$, $s(i, K_i)$ are the servers on path of the connection M_i .

Given (A-1), we need to consider how to compute $d_{i,j}$ here.

We let $m_{i,j}$ denote the membership of connection M_i enters into server j . That is,

$$m_{i,j} = \begin{cases} 1, & \text{if connection } M_i \text{ enters into server } j; \\ 0, & \text{otherwise.} \end{cases} \quad (\text{A-2})$$

We let $F_{i,j}(I)$ be the maximum number of M_i 's cells that can arrive at server j in any interval of length I . By [3], for any server, using the FCFS scheduling policy, the maximum delay, $d_{i,j}$, is given by

$$d_{i,j} = \max_{I \geq 0} \left(\sum_{i=1}^{N+1} m_{i,j} * F_{i,j}(I) - I \right). \quad (\text{A-3})$$

By [17], $F_{i,j}(I)$ is a piecewise continuously linear function

of variable I , so is $\sum_{i=1}^{N+1} m_{i,j} * F_{i,j}(I) - I$. The function

$\sum_{i=1}^{N+1} m_{i,j} * F_{i,j}(I) - I$ is maximized at $I = \mathbf{z}_j$, which is given by

$$\mathbf{z}_j = \max_{i \in \{1,2,\dots,N+1\}} \left\{ m_{i,j} * \frac{\mathbf{b}_i}{1 - \mathbf{r}_i} \right\}. \quad (\text{A-4})$$

Thus,

$$\begin{aligned} d_{i,j} &= \sum_{i=1}^{N+1} m_{i,j} * F_{i,j}(\mathbf{z}_j) - \mathbf{z}_j \\ &= \sum_{i=1}^{N+1} m_{i,j} * (\mathbf{b}_i + \mathbf{r}_i * \mathbf{z}_j) - \mathbf{z}_j \\ &= \sum_{i=1}^{N+1} m_{i,j} * \mathbf{b}_i + \left(\sum_{i=1}^{N+1} m_{i,j} * \mathbf{r}_i - 1 \right) * \mathbf{z}_j \end{aligned} \quad (\text{A-5})$$

Let $\mathbf{v}_j = \sum_{i=1}^N m_{i,j} * \mathbf{b}_i$ and $\mathbf{m}_j = \sum_{i=1}^N m_{i,j} * \mathbf{r}_i$. Then,

$$\begin{aligned} d_{i,j} &= \mathbf{v}_j + \mathbf{b}_{N+1} + (\mathbf{m}_j + \mathbf{r}_{N+1} - 1) * \mathbf{z}_j \\ &= \mathbf{b}_{N+1} + \mathbf{z}_j * \mathbf{r}_{N+1} + \mathbf{v}_j + (\mathbf{m}_j - 1) * \mathbf{z}_j \end{aligned} \quad (\text{A-6})$$

Let $\mathbf{x}_j = \max_{i \in \{1,2,\dots,N\}} \left\{ m_{i,j} * \frac{\mathbf{b}_i}{1 - \mathbf{r}_i} \right\}$, then there are two

cases: $\mathbf{z}_j = \mathbf{x}_j$ and $\mathbf{z}_j = \frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}}$. That is,

$$\frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} \leq \mathbf{x}_j \text{ and } \frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} > \mathbf{x}_j.$$

Case 1: $\frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} \leq \mathbf{x}_j$. By (A-6)

$$d_{i,j} = \mathbf{b}_{N+1} + \mathbf{x}_j * \mathbf{r}_{N+1} + \mathbf{v}_j + (\mathbf{m}_j - 1) * \mathbf{x}_j. \quad (\text{A-7})$$

Case 2: $\frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} > \mathbf{x}_j$. By (A-6)

$$\begin{aligned} d_{i,j} &= \mathbf{b}_{N+1} + \mathbf{r}_{N+1} * \frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} + \mathbf{v}_j + (\mathbf{m}_j - 1) * \frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} \\ &= \mathbf{b}_{N+1} + (\mathbf{r}_{N+1} + \mathbf{m}_j - 1) * \frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} + \mathbf{v}_j \end{aligned} \quad (\text{A-8})$$

In summary,

$$d_{i,j} = \begin{cases} \mathbf{b}_{N+1} + \mathbf{x}_j * \mathbf{r}_{N+1} + \mathbf{v}_j + (\mathbf{m}_j - 1) * \mathbf{x}_j, \\ \text{if } \frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} \leq \mathbf{x}_j \\ \mathbf{b}_{N+1} + (\mathbf{r}_{N+1} + \mathbf{m}_j - 1) * \frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} + \mathbf{v}_j, \\ \text{otherwise } \frac{\mathbf{b}_{N+1}}{1 - \mathbf{r}_{N+1}} > \mathbf{x}_j \end{cases} \quad (\text{A-9})$$

$$\mathbf{z}_{s(i,j)} = \max_{i \in \{1,2,\dots,N+1\}} \left\{ m_{i,s(i,j)} * \frac{\mathbf{b}_i}{1 - \mathbf{r}_i} \right\}. \quad (\text{B-6})$$

Substituting (B-1) and (B-3) into (B-2), we have

$$d_i(Q) \leq d_i(Q'). \quad (\text{B-7})$$

Because of arbitrariness of index i , and the definition of (3-6)

$$\bar{d}^{[N+1]}(Q) \leq \bar{d}^{[N+1]}(Q'). \quad (\text{B-8})$$

APPENDIX B PROOF OF THEOREM 2

Theorem 2. Let Q and Q' be two points in a requested-QoS region. If $Q \preceq Q'$, then $\bar{d}^{[N+1]}(Q) \leq \bar{d}^{[N+1]}(Q')$.

Proof: Let $Q = (\beta, \rho, D)$ and $Q' = (\beta', \rho', D')$. $Q \preceq Q'$ implies $\rho \leq \rho'$, $\beta \leq \beta'$, and $D \geq D'$. That is,

$$\frac{\mathbf{b}}{1 - \mathbf{r}} \leq \frac{\mathbf{b}'}{1 - \mathbf{r}'}. \quad (\text{B-1})$$

Consider an arbitrary connection. (say M_i , $1 \leq i \leq N + 1$). By (A-1), d_i , the worst case end-to-end cell delay experienced by a cell of connection M_i is given by

$$d_i = d_{i,s(i,1)} + d_{i,s(i,2)} + \dots + d_{i,s(i,K_i)} \\ = \sum_{j=1}^{K_i} d_{i,s(i,j)} \quad (\text{B-2})$$

where $s(i, 1), s(i, 2), \dots, s(i, K_i)$ are the servers on path of connection M_i .

By (A-6),

$$d_{i,s(i,j)} = \mathbf{v}_{s(i,j)} + \mathbf{b}_i + (\mathbf{r}_i + \mathbf{m}_{s(i,j)} - 1) * \mathbf{z}_{s(i,j)} \quad (\text{B-3})$$

$$\text{where } \mathbf{v}_{s(i,j)} = \sum_{i=1}^N m_{i,s(i,j)} * \mathbf{b}_i, \quad (\text{B-4})$$

$$\mathbf{m}_{s(i,j)} = \sum_{i=1}^N m_{i,s(i,j)} * \mathbf{r}_i, \quad (\text{B-5})$$

Biographies of Authors

Weijia Jia received his BSc and MSc from Center South University of Technology, China in 1982, 1984, and PhD from Faculty Polytechnic of Mons, Belgium in 1993, respectively. Before joining City University of Hong Kong as an assistant professor, he has been a research fellow at German National Research Center for Computer Science (GMD) in St. Augustin. His research interest includes fault-tolerant distributed systems, real-time communication protocols and Internet computer networks and protocols for next generation Internet. He is a member of IEEE Computer Society.

Wei Zhao is a full Professor in the Department of Computer Science, Texas A&M University, USA. Dr. Zhao is active in research and development in the areas of communication networks, fault tolerant real-time systems, distributed operating systems. He received the Outstanding Paper Award in the IEEE International Conference on Distributed Computing Systems in 1992. He is an inventor of two U.S. patents on using high speed networks for time-critical applications. Dr. Zhao has been an editor for the IEEE Transactions on Computers. He has served the Chair and Program Committee Chair for many IEEE International Conferences.