Admission Control for Hard Real-Time Connections in ATM LANs

Amitava Raha Sanjay Kamat Wei Zhao

Department of Computer Science
Texas A & M University
College Station, TX 77843-3112 *

Abstract

A CAC algorithm must efficiently determine if a new connection can be admitted by verifying that its QoS requirements can be met without violating those of previously admitted connections. In hard real-time systems, the QoS requirements are specified in terms of end-to-end cell deadlines and no cell loss due to buffer overflow. A CAC algorithm must account for interdependencies among connections caused by statistical multiplexing of cells in ATM networks. Arbitrarity of network topology may lead to cyclic dependencies among various connections. We present an efficient CAC algorithm that addresses the above issues. The algorithm uses a traffic descriptor called the maximum traffic rate function to effectively compute bounds on end-to-end delays of connections and buffer requirements within the network. Our work differs from most previous work in that it does not require traffic restoration inside the network.

1 Introduction

We address the problem of admitting *hard real-time* (HRT) connections in an ATM local area network. A hard real-time connection specifies its Quality of Service (QoS) in terms of a hard cell-transfer deadline and the requirement of no cell loss due to buffer overflow. Connections supporting distributed hard real-time applications such as supervisory command and control systems used in manufacturing, chemical processing, nuclear plants, telemedicine, warships, etc., can be characterized as HRT connections.

The problem of admitting a hard real-time connection in an ATM network is as follows. Consider a network that has already admitted a set of N hard real-time connections $\{M_1, M_2, \ldots, M_N\}$ with each connection receiving its requested Quality of Service. Let a request for a new hard real-time connection M_{N+1} arrive. Now the network must efficiently determine if M_{N+1} can receive its requested Quality of Service without violating the guarantees already provided to connections $\{M_1, M_2, \ldots, M_N\}$.

Thus, a key issue in admission of hard real-time connections in an ATM LAN is derivation of delay bounds of connections. This is a challenging task. Determining delay bounds has been the pivotal issue in the development of real-time technology [12, 20, 21, 24]. Much work has been concentrated on centralized systems [13]. In general, obtaining delay bounds in a LAN has been difficult due to the distributed nature of the problem. There are generally two approaches for doing so: *synthesis* and *decomposition*.

With the synthesis approach, the entire network is considered to be a single server. Such an approach gives reasonable bounds only if one or few applications access the network at a time. Therefore, this approach has been adopted only for small and simple networks such as 802.5 token ring [11, 22], DQDB [19], and FDDI [1, 11].

With the *decomposition* approach, the network is decomposed into *servers*. Each connection is viewed as being served by a sequence of servers. The worst case end-to-end delays is obtained by summing the upper bounds of the delays suffered by a connection at each of the servers [5, 15, 17]. The advantage of the decomposition approach is that it provides the basis for a general and modular analysis of the network, similar to the analysis of electrical circuits. We adopt the network *decomposition* approach for the computation of the end-to-end cell delays.

To analyze the delay bounds at each server it is necessary to have a description of a connection's traffic at the input of the server. Many traffic descriptors have been proposed in the literature [2, 5, 17]. In order to explicitly model the traffic characteristics of connections, we adopt a maximum rate function introduced in [17] and used in [18]. While much of the previous work [7, 8, 9, 10, 23, 25] assume the existence of additional mechanisms within the network to tailor the connection traffic so that the traffic at the input of each server adheres to a specific traffic characterization. The use of the maximum function frees us from this assumption.

Much of the previous studies on meeting end-to-end deadlines in ATM networks have concentrated on designing and analyzing scheduling policies for ATM switches [4, 7, 8, 9, 10, 15, 23, 25, 26]. In this work, we assume that FCFS scheduling discipline is used at ATM switches. Because of its simplicity the FCFS scheduling policy has been used in existing commercial ATM switches. Consequently, the analysis and results presented in this work are directly applicable

*The work reported in this paper was supported in part by the Office of Naval Research under Grant N00014-95-J-0238 and Texas Advanced Technology program under grant 999903-204.

to most existing networks. However, the methodology used in designing our CAC algorithm is general and the approach can be extended to systems using other scheduling policies.

Point-to-point topology of ATM networks and statistical multiplexing of cells makes the derivation of delay bounds complicated in comparison with other local area networks. Specifically, admitting a new connection perturbs the traffic of some of the existing connections, necessitating a re-evaluation of the end-to-end cell delays of these connections. Furthermore, because of the arbitrariness of network topology it is possible that the connections form a feedback loop, creating a cyclic dependency among the connections. These cyclic dependencies among the connections complicates the delay analysis [5, 15]. Most previous work assumes that either the cyclic dependencies do not exist or are eliminated by some internal network control mechanism (e.g., traffic regulation, reshaping by dedicated hardware and framing) [5, 9, 3, 6]. A major contribution of this study is that we develop a CAC algorithm that explicitly takes into account the possible cyclic dependencies among connections without using any special network control mechanisms.

The rest of this paper is organized as follows. In Section 2, we give an overview of our methodology. In Section 3, we present and discuss our CAC algorithm. In Section 4, we show that our algorithm is effective and efficient by demonstrating that there is a high probability that a new connection is admitted in a normally loaded system. Section 5 concludes the paper with a discussion of our approach and future work.

2 Preliminaries

In this section, we present the preliminary concepts and techniques which we employ for deriving an upper-bound for the end-end cell delays of a connection. We will also introduce some of the notations and terminology we use in the rest of this paper.

An upper bound on the end-to-end delay of a cell is obtained by summing the worst case delays a cell may experience at every network component it traverses. The methodology to compute these delays is based on the following three steps:

Network decomposition. The idea behind this step is to model the network as a set of servers that serve individual connections [5, 17]. Those servers that offer only constant delays to a connection's cells and do not change the cell traffic characteristics of a connection are considered separately from those which offer variable delays to cells and hence affect a connection's traffic.

Connection-Server graph construction. As a consequence of the above step, each connection is represented as a path in a graph whose nodes are servers which potentially affect the connection's traffic.

Individual Server analysis. The objective in this step is to compute the worst case delay suffered by a connection at each of the servers that offer a variable amount of delay to cells.

In the next three subsections we examine these steps in some detail.

2.1 Network decomposition

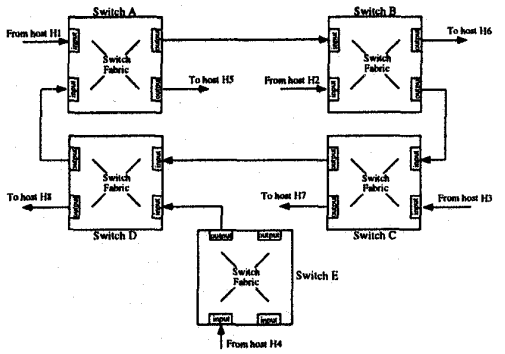

Figure 1: An ATM LAN with 5 switches

This step involves modeling the network as a collection of servers. A server is an abstraction of a network component that is traversed by a connection's cells. In an ATM LAN, hosts are connected to ATM switches and various ATM switches are connected to each other using physical links. Figure 1 shows an ATM LAN consisting of 5 switches. Thus, switches and communication links are two key components in an ATM LAN. As shown in Figure 1, the switch itself consists of input ports, a switching fabric, and output ports. An ATM cell arrives at an input port of a switch, is switched by the switching fabric to an output port, and is transmitted along the physical link associated with the output port. In the network decomposition step, we model the input ports, the switching fabric, the output ports, and the physical links as servers serving ATM connections.

The servers are classified into two categories: *constant servers* and *variable servers*. A constant server is one that offers a constant amount of delay to each cell that uses it and does not by itself change the traffic flow characteristics of a connection. For example, physical links and the switching fabric are constant delay line servers. The function of an input port is to demultiplex 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 demultiplexor server.

The functionality of an output port of a switch is more complex. An output port may simultaneously receive cells belonging to different connections competing for transmission on the link associated with the output port. Thus, cells may be buffered at an output port and transmitted in an order that is determined by the scheduling discipline employed by the switch hardware.[1] First Come First Serve (FCFS) is the most commonly used discipline. Hence, we model output ports of switches as FCFS multiplexor servers.

[1] In most ATM switches cells of HRT connections are assigned high priority and cells from low priority (non-time-constrained) connections are transmitted when the queue of high priority cells is empty.

Note that an FCFS multiplexor server must be considered as a variable server since the delay suffered by a cell in this server varies depending upon the queue length in the buffer. Consequently, the traffic characteristics of a connection at the output of this server may differ from those at the input.

As an example of network decomposition, consider the ATM LAN shown in Figure 1. Figure 2 shows the same network modeled as a collection of servers serving four connections M_1, M_2, M_3, and M_4. Although this example may not be representative of a typical ATM LAN, it is used to illustrate important concepts discussed in this paper. We shall use this example throughout the paper.

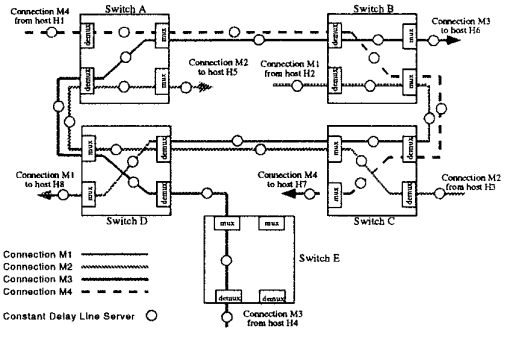

Figure 2: Example of network decomposition

2.2 Connection-Server graph construction

As mentioned earlier, introduction of a new connection in an ATM LAN may affect the delays suffered by some of the previously admitted connections. The purpose of Connection-Server graph construction is to identify such interdependencies.

First, note that network decomposition into servers allows us to view each connection as a stream of cells served by a sequence of constant and variable servers. For example, consider connection M_1 from Host H2 to Host H8 shown in Figure 2. M_1 traverses 7 delay line servers (4 physical links and 3 switching fabrics) and 3 demultiplexor servers (input ports of 3 switches) all of which are constant servers. M_1 also traverses 3 multiplexor servers (output ports of 3 switches) which are variable servers. Recall that the constant servers serving M_1 only add a fixed amount of delay to M_1's cells and do not change M_1's traffic characteristics. Hence, their impact on M_1 can be accounted for by simply subtracting the total delay suffered by M_1 at these servers from M_1's end-to-end deadline. The same holds for other connections. In the rest of the paper, we assume that the deadlines of connections are modified in such a way. Consequently, we eliminate all the constant servers from further consideration and focus only on the variable servers in the remainder of the paper. We will often omit the prefix 'variable' when referring to variable servers to avoid repetitiousness.

Now we can view a connection as being served by a sequence of variable servers only. Let K be the total number of network components modeled as variable servers. In the example shown in Figure 2, K equals 9. Each of these servers is given a unique server-id which is an integer in the range of $1 \cdots K$. A Connection-Server graph is constructed as a labeled, directed graph with the servers as its nodes. A directed edge is introduced from server m to server n if there is a connection that is served by server m followed by server n. The edge is labeled by all the connections that use the servers in immediate sequence. Figure 3 shows Connection-Server graph corresponding to the set of connections shown in Figure 2. The sources and destinations of connections are also shown in the Connection-Server graph to facilitate the discussion of our CAC algorithm later.

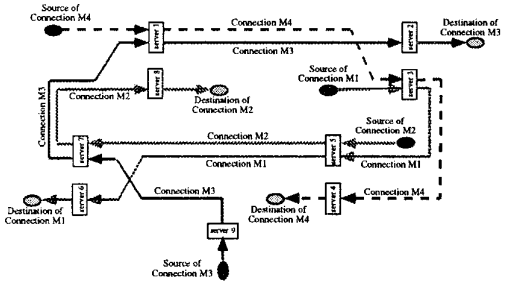

Figure 3: Connection-Server graph construction

We denote the sequence of servers serving connection M_i by

$$H_i = < s(i,1), s(i,2), \ldots, s(i,j), \ldots, s(i,K_i) >, \quad (1)$$

where K_i is the total number of servers serving connection M_i and $s(i,j)$ denotes the server-id of the j^{th} server in the connection's path. For example, from Figure 3, we see that H_1, the sequence of servers for connection M_1, is $< 3,5,6 >$. Clearly, H_i, the sequence of servers serving M_i, must be a valid directed path in the Connection-Server graph. If server s is one of the servers in H_i, such that $s = s(i,k), k \neq K_i$, then function $next_i(s)$ is defined as

$$next_i(s) \quad = \quad s(i,k+1). \quad (2)$$

$next_i(s)$ will be used in our CAC algorithm.

2.3 Individual server analysis

As stated before, the objective of the network decomposition step is to be able to compute the worst case end-to-end delays of a connection as sum of the worst case delays encountered at individual servers. The construction of the Connection-Server graph captured the dependencies among the servers. Next we need to examine how delays at individual servers can be computed. This is the main objective of the server analysis step. In this paper, we assume that time is normalized in terms of the cell transmission time in the ATM network. That is, time is considered a discrete quantity with the cell transmission time being taken as one time unit.

We will need some notations to facilitate the discussion of server analysis. Let $d_{i,s}$ be the upper bound on the delay experienced by a cell of M_i at server s.

We let $d_{i,s} = 0$ if s does not belong to M_i's connection path specified by H_i. Let $\vec{d}(n)$ be a vector; that is,

$$\vec{d}(n) = (d_1, d_2, \ldots, d_i, \ldots, d_n), \quad (3)$$

where d_i is an upper bound on the end-to-end delay experienced by a cell of connection M_i. We will compute d_i as

$$d_i = \sum_{j=1}^{K_i} d_{i,s(i,j)}. \quad (4)$$

2.3.1 Traffic descriptor

From the Connection-Server graph we can determine all the connections that share a network server. Hence, the delay at every server can be obtained if the input traffic pattern of all the connections sharing the server is known. The traffic pattern of a connection at a point in the network is characterized by a *traffic descriptor* [2]. It must be noted that due to multiplexing at ATM switches the traffic pattern of a connection at any point in the network need not be the same as that at its source [2, 5, 17]. In this paper, we consider the following traffic descriptor:

Maximum rate function descriptor: This descriptor uses the notation $\Gamma(I)$ to specify the maximum arrival rate of cells in an interval of length I. Equivalently, a maximum of $I \cdot \Gamma(I)$ cells belonging to the connection may arrive in an interval of length I.

The reader may note that a connection's *actual* traffic pattern may differ from that implied by the traffic descriptor used to describe the connection's traffic. We need the *maximum rate function* descriptor because it specifies the worst case behavior of the traffic and helps us to derive bounds on the delays suffered by a connection's cells and on the queue lengths at servers.

For connection M_i, we denote the maximum rate function at the input of server s ($s = 1, 2, \ldots, K$), by $\Gamma_{i,s}(I)$. However, if server s is not part of H_i, M_i's connection path, then $\forall I$, $\Gamma_{i,s}(I) = 0$. At M_i's source, its maximum rate function is given by $\Gamma_{i,s(i,1)}(I)$ which is assumed to be specified by the requesting application during the connection set-up procedure. Note that $\Gamma_{i,s(i,j+1)}(I)$, the maximum rate function for M_i at the input of server $s(i, j+1)$ is same as the one at the output of the server $s(i, j)$. In Section 2.3.3, we will present the result for computing the maximum rate function of a connection at the output of a server.

2.3.2 Delay and queue length bounds

Now consider an FCFS server s. Consider the case where the maximum rate function traffic descriptors of all the connections at the input to server s are known. Then, the following result [5, 16] can be used to find an upper bound on the delay experienced by a cell and the maximum queue length at server s.

THEOREM 2.1 *Consider the connection M_i that traverses server s. If $d_{i,s}$ is the upper bound on the delay experienced by connection M_i (measured in cell transmission time units) and q_s is the maximum queue length at server s then*

$$d_{i,s} = q_s = \lceil \max_{I \le L_s} (\sum_m I\Gamma_{m,s}(I) - I) \rceil \quad (5)$$

where L_s is the length of the longest busy interval at server s and is given by

$$L_s = \min(I \mid \sum_m \Gamma_{m,s}(I) \le 1). \quad (6)$$

Theorem 2.1 can be proved by applying Theorem 4.1 given in [5].

2.3.3 Derivation of internal traffic descriptor

Recall that connection M_i passes through a sequence of servers that is given by H_i in (1).

Let us assume that $d_{i,s(i,1)}, d_{i,s(i,2)}, \ldots, d_{i,s(i,k)}$, which are upper bounds on the delays suffered by M_i at the first k ($k < K_i$) servers in its connection path, are known. Then upper bounds on $\Gamma_{i,s(i,k+1)}(I)$, the maximum rate function values for connection M_i at the output of server $s(i,k)$ are given by the following theorem.

THEOREM 2.2

$$\Gamma_{i,s(i,k+1)}(I) \le \min(1, (1 + \frac{c_i^k}{I})\Gamma_{i,s(i,1)}(I + c_i^k)) \quad (7)$$

where c_i^k is the sum of the upper bounds on the delays experienced by M_i's cells at all the upstream servers from $s(i,1)$ to $s(i,k)$ and is given by

$$c_i^k = \sum_{1 \le l \le k} d_{i,s(i,l)}. \quad (8)$$

Theorem 2.2 is a generalization of Theorem 2.1 in [5], where the maximum rate function at the output of an FCFS server was obtained in terms of that at the input to the server. Theorem 2.2 gives an upper bound on a connection's traffic rate inside the network in terms of that at the source. This is useful in practice. Most often, sources in hard real-time systems generate regular traffic (eg. periodic) for which $\Gamma(I)$ can be described by a closed form expression. Thus, Theorem 2.2 facilitates efficient computation of maximum traffic rate functions inside the network, making a fast CAC algorithm feasible. Our performance results will also demonstrate that the delay upper bounds computed using (7) with (5) are reasonable in that the CAC algorithm has a high probability of connection admission for normal loads.

3 The CAC algorithm

In this section, we will first elucidate some fundamental requirements of a connection admission control (CAC) algorithm for hard real-time (HRT) systems. Then, we present an efficient CAC algorithm for HRT systems and establish its properties.

3.1 Requirements

Recall from the previous section that admission of a new connection can perturb the traffic of existing connections. This perturbation is not limited to the connections that share a server with the new connection and may spread to other connections in the system. Thus, a CAC algorithm must take into account the extent of such perturbation and re-evaluate the delays and queue lengths affected by the perturbation.

In general, any connection admission control algorithm has to satisfy the following two required properties.

Property 1. First, the algorithm must *terminate*. In other words, the connection admission algorithm must either admit the new connection or reject it within a bounded time. Furthermore, the time taken by the CAC algorithm has a direct impact on the time required for connection establishment. Therefore, it is desirable to have an *efficient* CAC algorithm which takes a short time to admit or reject a connection.

Property 2. A CAC algorithm must be *correct* in the sense that if the new HRT connection is admitted then the end-to-end cells delays of all connections (the existing and the newly admitted) must be less than or equal to their deadlines and there must be no buffer overflow.

To formalize the second property, we introduce the following notations and conventions:

Let $\vec{d}^*(n)$ be a vector of size n.[2] It represents the end-to-end delays associated with n connections, that is,

$$\vec{d}^*(n) = (d_1^*, d_2^*, \ldots, d_i^*, \ldots, d_n^*), \quad (9)$$

where d_i^* is a random variable that denotes the end-to-end delay of a cell belonging to M_i.

Let $\vec{D}(n)$ be a vector of size n. It represents the end-to-end deadlines associated with n connections, that is,

$$\vec{D}(n) = (D_1, D_2, \ldots, D_i, \ldots, D_n), \quad (10)$$

where D_i is the deadline associated with M_i.

Let $\vec{q}^*(K)$ be a vector of size K. It represents the queue lengths, that is

$$\vec{q}^*(K) = (q_1^*, q_2^*, \ldots, q_s^*, \ldots, q_K^*), \quad (11)$$

where q_s^* is a random variable that denotes the queue length at server s.

Let $\vec{B}(K)$ be a vector of size K. It represents the buffer capacities of the servers, that is,

$$\vec{B}(K) = (B_1, B_2, \ldots, B_s, \ldots, B_K), \quad (12)$$

where B_s denotes the buffer capacity at server s.

[2]The reader may notice that we explicitly specify the size of the vector. This is to avoid the confusing of the number of connections being taking into consideration. However, when the context is clear we shall omit the specification of the size.

Given two vectors $\vec{X}(n) = (X_1, X_2, \ldots, X_n)$ and $\vec{Y}(n) = (Y_1, Y_2, \ldots, Y_n)$ of size n, we say that

$$\vec{X} \leq \vec{Y} \quad \text{if } (\forall i,\ 1 \leq i \leq n,\ X_i \leq Y_i) \quad (13)$$

and

$$\vec{X} < \vec{Y} \quad \text{if } ((\vec{X} \leq \vec{Y}) \text{ and } (\exists i,\ 1 \leq i \leq n,\ X_i < Y_i)). \quad (14)$$

In terms of the above notations, the correctness property can be stated as follows: if the new connection M_{N+1} is admitted then

$$\vec{d}^*(N+1) \leq \vec{D}(N+1) \quad (15)$$

and

$$\vec{q}^*(K) \leq \vec{B}(K), \quad (16)$$

where N is the number of connections previously admitted into the system.

Our objective is to develop an efficient CAC algorithm that is efficient (Property 1) and that satisfies (15) and (16) (Property 2).

3.2 Motivation

Given the network decomposition methodology, a straight forward approach to solving the CAC problem is to identify all the servers impacted by the new connection. Then all the servers impacted by admitting the new connection can be re-analyzed.

However, there is an inherent problem in such an approach. Consider the system presented in Figure 2. Let $\{M_1, M_2, M_3\}$ be the connections that already exist in the system and M_4 be the new request. The Connection-Server graph in Figure 3 shows the situation if M_4 were to be admitted.

The reader may note that there is a cyclic dependency in the connection-server graph. For example, M_4 shares server 3 with M_1 affecting M_1's traffic. Since M_1 later shares server 5 with M_2, the behavior of server 5 and connection M_2 may be impacted by introduction of M_4. Furthermore, M_2 shares server 7 with M_3 and M_3 shares server 1 with M_4 itself, forming a dependency loop!

With such a dependency loop one cannot analyze the impacted servers in a straight forward sequential manner. Our CAC algorithm is an iterative procedure. During the iterations it explicitly and cyclically traces and analyzes those servers being impacted.

In general, a system with this kind of cyclic dependency may not be *stable* in the sense that the delays and queue lengths may not be bounded. This has created a great deal of difficulty in analysis [5, 15]. Particularly, when an iterative method is used to analyze the system, one may run into the risk that the procedure may not converge. We are dealing with hard real-time systems which have stringent deadline and buffer requirements. Our CAC algorithm exploits the restrictions imposed by hard real-time systems to solve the convergence problem in a potentially unstable system.

3.3 Important data structures

Before we present the CAC algorithm, we discuss some important data structures that are used in the algorithm. We assume that the network management system, which invokes the CAC algorithm maintains the following data structures.

dM, a matrix used by the network to store the current value of upper bounds of the cell delays experienced by connections at servers in the ATM LAN. That is, for $1 \leq s \leq K$,

$$dM_{i,s} = d_{i,s}, \tag{17}$$

where $d_{i,s}$ is defined in (5).

$\vec{q}(K) = (q_1, q_2, \ldots, q_K)$, a vector used by the network to store the upper bound on the queue size at every server in the network. The default initial value of an element of $\vec{q}(K)$ is 0.

$\vec{D}(N) = (D_1, D_2, \ldots, D_N)$, which is defined in (10).

$\vec{B}(K) = (B_1, B_2, \ldots, B_K)$, which is defined in (12).

$\vec{I}(N) = (\Gamma_{1,s(1,1)}(I), \Gamma_{2,s(2,1)}(I), \ldots, \Gamma_{N,s(N,1)}(I))$, the input traffic vector, where $\Gamma_{i,s(i,1)}(I)$ presents the maximum rate function of connection M_i at the source.

Once a new connection admission request arrives, the new connection (say M_{N+1}) presents the following information to the system:

$H_{N+1} = < s(N+1,1), \ldots, s(N+1, K_{N+1}) >$, i.e., the connection path of M_{N+1}.

D_{N+1}, the cell transfer deadline of connection M_{N+1}.

$\Gamma_{N+1,s(N+1,1)}(I)$, the maximum rate function of connection M_{N+1} at the source.

The network management system passes dM, $\vec{q}(K)$, $\vec{D}(N)$, $\vec{I}(N)$, H_{N+1}, D_{N+1}, $\Gamma_{N+1,s(N+1,1)}$ and $\vec{B}(K)$ to the CAC algorithm. In addition to the above input data structures, the CAC algorithm uses the following internal data structures.

$dM^{internal}$, a matrix internally used by the CAC algorithm.

$\vec{d}(N+1)$, a vector of size $N+1$. It is internally used by the CAC algorithm to store the computed upper bounds on the end-to-end cell delays.

$\vec{q}^{internal}(K)$, a vector internally used by the CAC algorithm to store the computed upper bounds on the queue length at the servers.

Impact_server_list is an ordered list of the server-ids. When the CAC algorithm detects that a server would be affected by admission of the new connection, it appends the corresponding server-id at the end of *Impact_server_list*. The CAC algorithm uses *Impact_server_list* to determine the order in which the servers affected by the new connection are to be analyzed.

3.4 The Algorithm

The pseudocode for the CAC algorithm is given in Figure 3. It is an iterative procedure which efficiently determines the servers that would be affected if the new connection M_{N+1} were to be admitted and re-analyzes them. The algorithm has three major phases.

Initialization phase (Lines 1 - 12)
In this phase, the algorithm copies the system data structures dM and $\vec{q}(K)$ into its internal working space. The matrix $dM^{internal}$ of the algorithm is initialized as follows. For $(1 \leq s \leq K)$,

$$dM^{internal}_{i,s} = \begin{cases} dM_{i,s} & \text{if } 1 \leq i \leq N, \\ 0 & \text{if } i = N+1. \end{cases}$$

Also, $\vec{D}(N+1)$ and $\vec{d}(N+1)$ are constructed for the set of connections under consideration which includes the new connection and the existing ones.

Since all the servers in the connection path of M_{N+1} are directly impacted if M_{N+1} were to be admitted, the algorithm initializes *Impact_server_list* with the server-ids of all the servers in M_{N+1}'s path, i.e., with elements of H_{N+1}.

Iteration phase (Lines 13 - 30)
This is the main body of the algorithm and consists of a while loop. The iterative procedure begins with the current status of the network being the existing set of connections. Then, it systematically traces the impact of the new connection request, M_{N+1}. In each iteration, there are three major operations :

1. First, the algorithm removes the first server in *Impact_server_list* for analysis.

2. Next, the algorithm computes an upper bound on the queue length and delay at the server being analyzed. The results presented in Section 2.3 are used to achieve this.

3. Because an increase in delay at this server may change the input traffic characteristics of the succeeding servers, the succeeding servers are also appended to *Impact_server_list*. The $next_i(s)$ function defined in Section 2.2 facilitates this process. Thus, the algorithm systematically traces the perturbation caused in the network due to connection M_{N+1}.

The iteration process terminates if the current value of $\vec{d}(N+1)$ or $\vec{q}(K)$ violates the QoS requirements of the $N+1$ connections. The iteration also stops if *Impact_server_list* is empty. An empty *Impact_server_list* implies that the queue lengths and the delays at every server have reached a stable value.

Verification phase (Lines 31 - 33)
In this phase, the algorithm examines the cause of the termination of the while loop. The algorithm accepts the new connection if the deadline and buffer requirements of all the connections can be met, i.e., if $\vec{d}(N+1) \leq \vec{D}(N+1)$ and $\vec{q}(K) \leq \vec{B}(K)$. If the new connection is accepted, the algorithm also returns the updated values of dM, $\vec{q}(K)$, $\vec{D}(N+1)$, and $\vec{I}(N+1)$ to the network management system.

Connection_Admission_Control (dM, $\vec{q}(K)$, $\vec{D}(N)$, $\vec{\Gamma}(N)$, H_{N+1}, D_{N+1}, $\Gamma_{N+1,s(N+1,1)}$, $\vec{B}(K)$)

1. $\vec{q}^{internal}(K) = (q_1, q_2, \ldots, q_K)$;
2. $\vec{D}$(N+1) $= (D_1, D_2, \ldots, D_N, D_{N+1})$;
3. $dM^{internal} = dM$;
4. **for all** s **do**
5. $dM^{internal}_{N+1,s} = 0$;
7. **end for all**
8. **for** $i = 1$ **to** $N+1$ **do**
9. $d_i = \sum_{s \in H_i} dM^{internal}_{i,s}$;
10. **end for**
11. $\vec{d}$(N+1) $= (d_1, d_2, \ldots, d_N, d_{N+1})$;
12. $Impact\_server\_list = < s(N+1,1), s(N+1,2), \ldots, s(N+1, K_{N+1}) >$;
13. **while(** $(Impact\_server\_list \neq \emptyset)$ **and** $(\vec{d}(N+1) \leq \vec{D}(N+1))$ **and** $(\vec{q}^{internal}(K) \leq \vec{B}(K))$ **do**
14. s = first_element($Impact\_server\_list$);
15. $Impact\_server\_list = Impact\_server\_list - (s)$;
16. $old\_q = q_s^{internal}$;
17. $q_s^{internal}$ = compute_queue_length(s);
18. **for** ($i = 1$ **to** $N+1$) **do**
19. **if** $s \in H_i$ **then**
20. $dM^{internal}_{i,s} = q_s$;

/* If delay or queue length at server s changes, add the succeeding server to $Impact\_server\_list$ */

21. **if** $((old\_q \neq q_s^{internal})$ **and** $(next_i(s) \notin Impact\_server\_list))$ **then**
22. append_to_list($Impact\_server\_list$, $next_i(s)$);
23. **end if**
24. **endif**
25. **end for**
26. **for** $i = 1$ **to** $N+1$ **do**
27. $d_i = \sum_{s \in H_i} dM^{internal}_{i,s}$;
28. **end for**
29. $\vec{d}$(N+1) $= (d_1, d_2, \ldots, d_N, d_{N+1})$;
30. **end while**
31. **if** $((\vec{d}(N+1) \leq \vec{D}(N+1))$ **and** $(\vec{q}(K) \leq \vec{B}(K)))$ **then**
32. **return**(Accept, $\vec{D}(N+1)$, $dM^{internal}$, $\vec{q}^{internal}(K)$, $\vec{\Gamma}(N+1)$);
33. **else return**(Reject).

Figure 3: Pseudocode for the CAC algorithm

3.5 Properties of the algorithm

In this section, we establish properties of the CAC algorithm. For the proofs please refer to [16].

Let $\vec{d}^k$ be vector $\vec{d}(N+1)$ at the end of the k^{th} iteration. Let $\vec{d}^0$ be the vector $\vec{d}(N+1)$ which is computed at the initialization phase in the algorithm.

LEMMA 3.1 *During an execution of the algorithm, before its termination there is a sub-sequence of iterations (iterations $I_0, I_1, I_2, \ldots, I_j, \ldots$) where $I_0 < I_1 < I_2 < \ldots I_{j-1} < I_j \ldots$ such that for $j > 1$,*

$$I_j - I_{j-1} \leq K \tag{18}$$

and

$$\vec{d}^{I_{j-1}} < \vec{d}^{I_j}. \tag{19}$$

LEMMA 3.2 *If the CAC algorithm accepts a connection after G iterations then $1 \leq i \leq N+1$, $1 \leq s \leq K$,*

$$q_s^* \leq q_s, \tag{20}$$

and

$$d_i^* \leq d_i, \tag{21}$$

where d_i^* is the end-to-end delay experienced by a cell of connection M_i and q_s^* is the queue length at server s.

THEOREM 3.1 *The CAC algorithm terminates and is correct.*

Furthermore, the CAC algorithm is quite efficient. For example, for a link utilization of 50% and 120 connections, the average time to admit a new connection for an implementation of the CAC algorithm executing in a Sun/Solaris environment was 200 milliseconds.

4 Performance Evaluation

In this section, we evaluate the performance of the CAC algorithm discussed in the previous sections. We will first define performance metrics, then describe the system architecture considered and present the performance results.

We use *Admission Probability (AP(U))* for evaluating a CAC algorithm for hard real time connections.

$AP(U)$ is defined as the probability that HRT connections are admissible conditioned on the average utilization of the inter-switch physical links in the network being U.

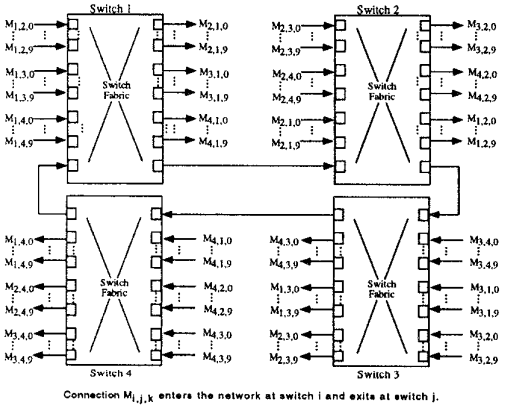

Figure 4: The network system evaluated

The network we consider in this paper consists of four 32 x 32 ATM switches. That is, each switch has 32 input lines and 32 output lines. As shown in Figure 4, there are 120 connections in the system. Connection $M_{i,j,k}$ is the kth connection that enters the

network at switch i and leaves the network at switch j. The connections in the network form a symmetric pattern. The system is arranged in such a way that 60 connections share one inter-switch link at each stage. Several systems with different architectures have also been evaluated. The results are similar and are not presented here due to the space limitation.

Since our target applications are hard real-time systems, we consider the Source Traffic Descriptor (STD [2]) for the HRT connections to be the traditional HRT source traffic model. That is, the source traffic is assumed to be *periodic* and the STD is described by the parameters (C, P), where P is the period of the message and C is the number of cells in a message. Although the source traffic of a connection is periodic, due to multiplexing in the network the periodicity of the connection traffic may no longer be maintained within the network. As mentioned earlier, we use the maximum rate function to characterize the traffic of connections inside the network.

To obtain the performance data, we developed a program to simulate the network system. The program was written in the C programming language and run in a Sun/Solaris environment. In each run of the program 1000 connection sets were randomly generated. For each connection, the total number of cells per period were chosen from a geometric distribution with mean 10. Similar results have been obtained with different settings of parameters. We do not present them here due to space limitations.

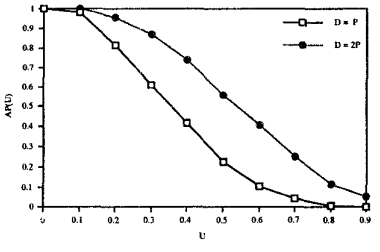

Figure 5: Admission Probability vs. Link Utilization: unregulated source.

Figure 5 shows the admission probability results for our sample network. The performance figures are corresponding to two values of D_i, (P_i and $2P_i$). It is common practice in a hard real-time system that deadlines are associated with periods [13, 14]. From Figure 5, we can make the following observations:

1. In general, we found that the admission probability is sensitive to the average link utilization. As the utilization increases the admission probability decreases. This is expected because higher the network utilization, the more difficult it is for the system to admit a set of connections.

2. When the end-to-end cell deadlines of the connections are increased, the admission probability shows an obvious improvement. For example, when $U = 0.4$, the admission probability increases from about 40% to 80% when the average deadline increases from P to 2P.

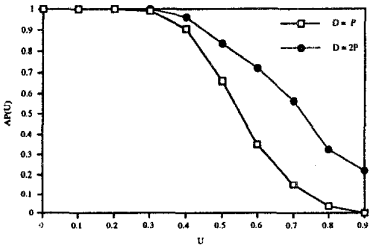

Figure 6: Admission probability vs. Link utilization: regulated source.

So far, we have considered a network management system which uses the CAC algorithm to control connection admission without modifying the input traffic of the connections. There has been increased interest in controlling the delays of connections by appropriately regulating the connection traffic at the entrance of the network. By regulating the input traffic, its burstiness can be controlled. This tends to reduce the adverse impact of burstiness on the end-to-end delays of other connections. We now consider the system with traffic regulation at the source. We adopt the method proposed in [18] to select parameters of the traffic regulation mechanism (e.g., leaky bucket) so that an appropriate level of regulation is maintained.

Figure 6 shows the admission probability results for the regulated system. In comparison with the data in Figure 5, we see that by using input traffic regulation method of [18] in conjunction with our CAC algorithm, the admission probability can be improved. The $AP(U)$ is almost 100% for values of U as high as 30%.

5 Final Remarks

In this paper we addressed the connection admission control problem in an ATM LAN supporting hard real-time applications. The key issue in solving this problem was obtaining reasonable upper bounds on the end-to-end delays of connections. We took a network decomposition approach, in which the network is modeled as a collection of servers. This approach has been used by several researchers before. However, our work significantly differs from the previous work by making the following contributions:

We use the maximum rate function $\Gamma(I)$ to describe a connection's traffic at any point in the network. The use of this traffic descriptor frees us from several restrictive assumptions often made in previous analysis. For example, the use of a single value to define the overall worst case rate of a connection leads to over allocation of resources to connections lowering the network resource utilization. Some of the previous work assumes reconstruction of a connection's source traffic within the network using traffic shaping. Such traffic shaping mechanisms within the network also introduce additional delays and are not easy to implement on a per-connection basis. Our approach based on the maximum rate function does not require reconstruction of a connection's original traffic inside the network.

A potential concern one may have on the use of the maximum rate function is the need for computational

resources (time or space) to obtain values of $\Gamma(I)$ for different values of I. We provide a simple and efficient method of computing $\Gamma(I)$ values for a connection at any point in the network in terms of the corresponding values at the connection's source.

When a new connection is admitted, not only is the status of servers traversed by the new connection affected but also the perturbation may spread to other servers in the system. More seriously, it has been recognized that the arbitrary topology of the network may cause a cyclic relationship among the connections that impact each other. Our CAC algorithm explicitly accounts for the extent of the perturbation caused by admitting a new connection and also deals with the possible cyclic dependencies.

We formally established the desired properties of our CAC algorithm. Specifically, we showed that the algorithm always terminates, regardless of the existence of cyclic dependencies among the connections and servers. Our algorithm is correct in the sense that it accepts a connection only if the QoS requirements of the new connection together with the existing ones can be met.

We evaluated the performance of our algorithm in terms of the connection admission probability. We found that with our CAC algorithm, a connection has a high probability of being admitted under light or medium load conditions, which are typical of most hard real-time systems.

This work can be extended in several ways. It would be interesting to consider scheduling disciplines other than FCFS. Another important issue in hard real-time systems is to consider good approximations [17] of the explicit traffic descriptor so as to improve the execution overheads of the algorithm, although this will result in a corresponding decrease in the admission probability.

References

[1] G. Agrawal, B. Chen, W. Zhao, and S. Davari. Guaranteeing synchronous message deadlines in high speed token ring networks with timed token protocol. *Proc. of IEEE ICDCS*, pg 468–475, June 1992.

[2] ATM Forum. *ATM Forum- ATM User-Network Interface Specification Version 3.1*, 1995.

[3] P. Boyer, F. Guillemin, M. Servel, and J. Coudreuse. Spacing cells protects and enhances utilization of ATM network links. *IEEE Network*, 6(5):38–49, Sept. 1992.

[4] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *Proc. of ACM SIGCOMM'92*, pg 14–26, Aug. 1992.

[5] R. L. Cruz. A calculus for network delay. *IEEE Tran. on Inf. Th.*, 37(1):114–141, Jan. 1991.

[6] A. Dailianas and A. Bovopoulis. Real-time admission control algorithms with delay and loss guarantees in ATM networks. *Proc. of INFOCOM'94*, pg 1065–1072, 1994.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Proc. of ACM SIGCOMM'89*, pg 1–12, Sept. 1989.

[8] D. Ferrari and D. C. Verma. A scheme for real-time channel establishment in wide-area networks. *JSAC*, SAC-8(3):368–379, Apr. 1990.

[9] S. J. Golestani. A framing strategy for congestion management. *JSAC*, 9(7):1064–1077, Sept. 1991.

[10] C. R. Kalmanek, H. Kanakia, and S. Keshav. Rate controlled servers for very high-speed networks. *Proc. of IEEE Global Telecommunications Conf.*, pg 300.3.1–300.3.9, Dec. 1990.

[11] S. Kamat and W. Zhao. Real-time schedulability of two token ring protocols. *Proc. of IEEE ICDCS*, pg 347–354, May 1993.

[12] S. Kamat and W. Zhao. Performance comparison of two token ring protocols for real-time communication. S. Son, editor, *Principles of Real-Time Sys.*. Prentice Hall, 1994.

[13] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *JACM*, 20(1):46–61, Jan. 1973.

[14] N. Malcolm and W. Zhao. Hard real-time communication in multiple-access networks. *Journal of Real-Time Sys.*, Jan. 1995.

[15] A. K. J. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*. PhD thesis, EECS, MIT, 1992.

[16] A. Raha. *Real Time Communication in ATM Networks*. PhD thesis, Dept. of CS, Texas A&M Univ.,1996. In preparation.

[17] A. Raha, S. Kamat, and W. Zhao. Guaranteeing end-to-end deadlines in ATM networks. *Proc. of IEEE ICDCS*, June 1995.

[18] A. Raha, S. Kamat, and W. Zhao. Using traffic regulation to meet end-to-end deadlines in ATM networks. *Proc. of IEEE ICNP-95*, 1995.

[19] L. Sha, S. S. Sathaye, and J. K. Strosnider. Scheduling real-time communication on dual-link networks. *Proc. of IEEE RTSS*, pg 188–197, Dec. 1992.

[20] J. A. Stankovic. Misconceptions about real-time computing: A serious problem for next generation systems. *IEEE Computer*, 21(10):10–19, Oct. 1988.

[21] J. A. Stankovic and K. Ramamritham, editors. *Hard Real-Time Sys.*. IEEE Computer Society Press, 1988.

[22] J. K. Strosnider, T. Marchok, and J. Lehoczky. Advanced real-time scheduling using the IEEE 802.5 token ring. *Proc. of IEEE RTSS*, pg 42–52, Dec. 1988.

[23] L. Trajkovic and S. J. Golestani. Congestion control for multimedia services. *IEEE Network*, 6(5):20–26, Sept. 1992.

[24] A. M. van Tilborg and G. M. Koob. *Foundations of Real-Time Comp.: Scheduling and Resource Management*. Kluwer Academic Publishers, 1991.

[25] H. Zhang and S. Keshav. Comparison of rate-based service disciplines. *Proc. of ACM SIGCOMM'91*, pg 113–121, Sept. 1991.

[26] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. *Proc. of ACM SIGCOMM'90*, pg 19–29, Sept. 1990.