Admission control for hard real-time connections in ATM LANs

A.Raha, W.Zhao, S.Kamat and W.Jia

Abstract: A connection admission control (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. To achieve these objectives a CAC algorithm must account for the traffic characteristics of connections. Furthermore, arbitrary topology of the network may lead to cyclic dependencies among various connections. An efficient CAC algorithm is presented that addresses these 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. The 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 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 characterised 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 $\{M_1, M_2, ..., M_N\}$ 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 $\{M_1, M_2, ..., M_N\}$.

A key issue in admission of hard real-time connections in an ATM LAN is the derivation of delay bounds of connections. This is a challenging task. Arbitrary topology of ATM networks along with statistical multiplexing of cells belonging to different connections makes the derivation complicated in comparision with other local area network. 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, arbitrary topology loops make the delay analysis complicated [1, 2]. Therefore determining delay bounds has been the pivotal issue in the development of real-time technology [3–6]. In general, obtaining delay bounds in a LAN has been difficult due to the distributed nature of the problem. There are roughly two approaches: synthesis and decomposition.

With the synthesis approach, the entire network is considered as 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 [7, 8], DQDB [9], and FDDI [7, 10].

With the decomposition approach, the network is decomposed into servers. Each ATM connection is viewed as being served by a sequence of servers. The worst case end-to-end cell delays are obtained by summing the upper bounds of the delays suffered by a connection at each of the servers [1, 2, 11]. 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 method of network decomposition in the computation of the end-to-end cell delays.

To analyse 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 [1, 11, 12]. To explicitlv model the traffic characteristics of connections we adopt a maximum rate function introduced in [11] and used in [13]. Much of the previous work [14–19] assumes 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 characterisation. The use of the maximum function to model traffic within the network frees us from this assumption.

In this paper we design and analyse a connection admission control (CAC) algorithm that addresses these problems. Our algorithm has two important properties: it

IEE Proceedings online no. 20010300

DOI: 10.1049/ip-com:20010300

Paper first received 28th February 2000 and in revised form 23rd January 2001

A. Raha is with the Fujitsu Software Corporation, 3055 Orchard Drive, San Jose, CA 95134, USA

W. Zhao is with the Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA

S. Kamat is with the IBM T J. Watson Research Center, Yorktown Heights, NY 10598-0704, USA

W. Jia is with the Department of Computer Science, City University of Hong Kong, 83 Tat Chee Ave., Kowloon, Hong Kong

explicitly models the traffic characterstics of connections in the network, and it takes into account interaction of the new connection's traffic with that of the old ones and addresses the possibility of cyclic dependencies among those connections. We show that our algorithm is effective and efficient. We demonstrate that there is a high probability that a new connection be admitted in a normally loaded system. Our experimental data also show that it takes a relatively short time for the CAC algorithm to make the admission decision, an essential feature for on-line use.

2 Preliminaries

We present the preliminary concepts and techniques employed for deriving the wors-case end-to-end cell delay of a connection. We also introduce some of the notations and terminologies we use in the rest of this paper.

The worst-case end-to-end delay experienced by 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 [1, 11]. Those servers that offer only constant delays to a connection's cells without changing 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 previous step, each connection is represented as a path in a graph whose nodes are servers which potentially affect the connection's traffic.

Indiuidual server analysis. The objective of this step is to compute the worst-case delay suffered by a connection and characterise its traffic at the output of each of the servers that offer a variable amount of delay to cells.

The following three subsections examine these steps in detail.

2.1 Network decomposition

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. Fig. 1 shows an ATM LAN consisting of five switches. Thus, switches and communication links are two key components in an ATM LAN. As shown in Fig. 1, a switch itself consists of input ports, a switching fabric, and output ports. An ATM cell arrives at an input port of a switch, is transported by the switching fabric to an output port, and 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 the one that offers a constant delay to each cell passing it but does not by itself change the traffic flow characteristics of a connection. For example physical links and the switchmg 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 demultiplexer server. We further assume that in the high priority queue the cells are served on the first come first serve (FCFS) basis.

As an example of network decomposition, consider the ATM LAN shown in Fig. 1. Fig. 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.

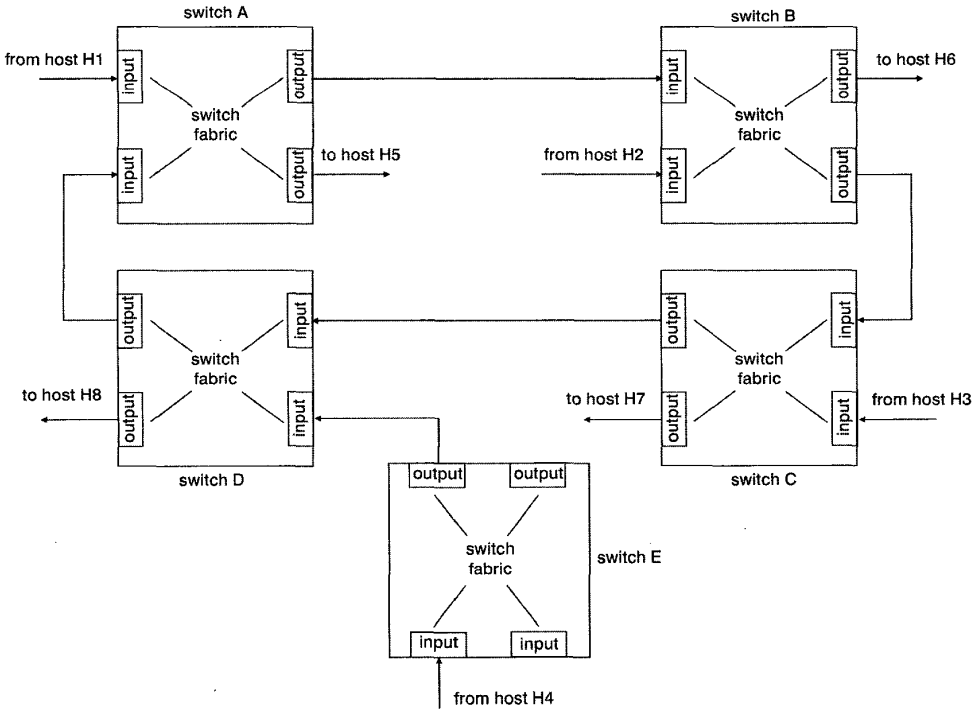

Fig. 1 *ATM LAN with five switches*

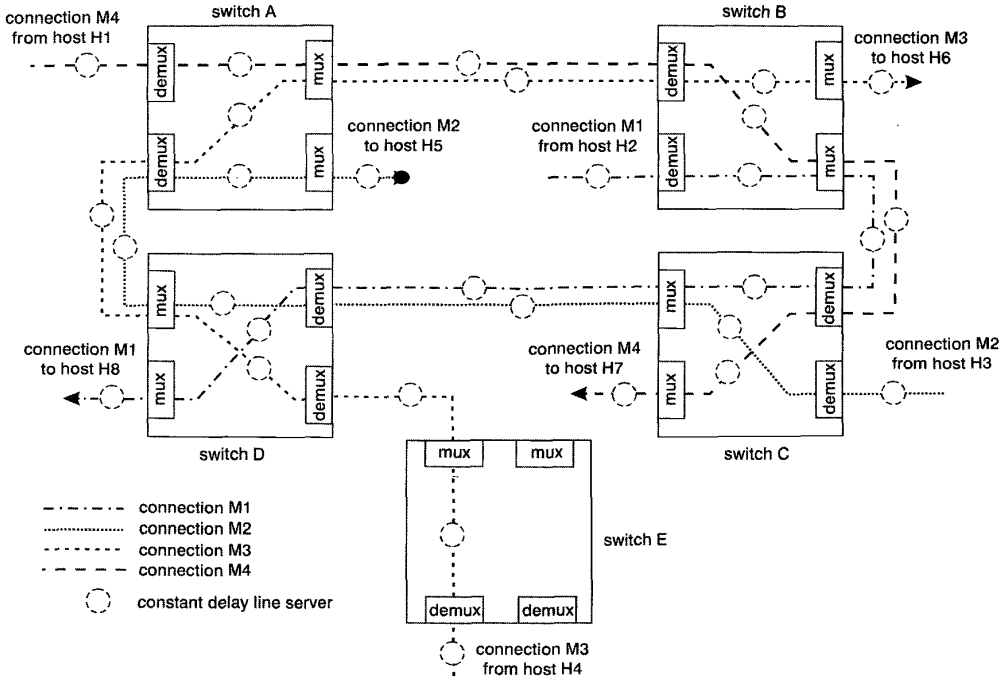

Fig. 2 *Example of network decomposition*

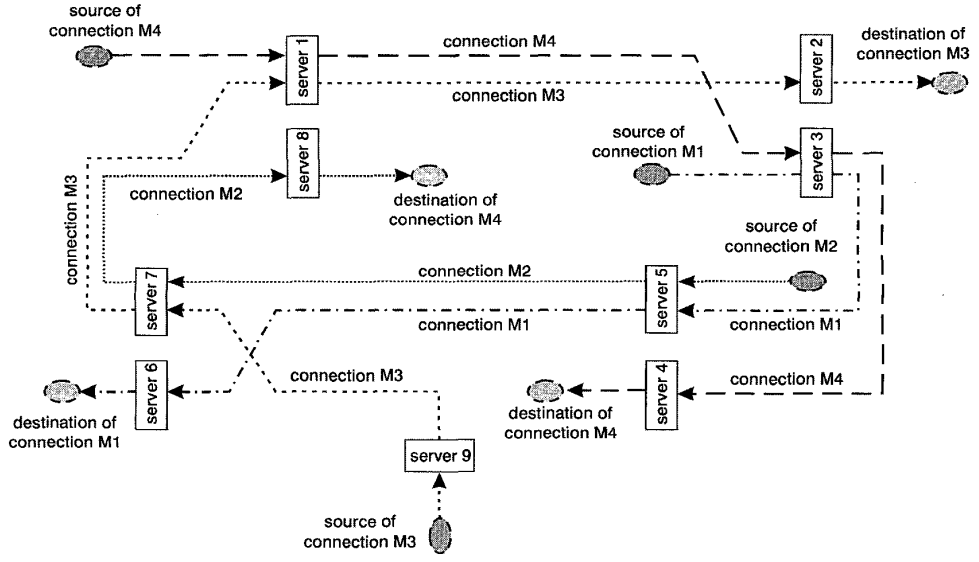

Fig. 3 *Connection–server graph construction*

2.2 Connection–server graph construction

Introduction of a new connection into 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 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 2 to host 8 (Fig. 2). M_1 traverses seven delay line servers (four physical links and three switching fabrics) and three demultiplexer servers (input ports of three switches) all of which are constant servers. M_1 also traverses three multiplexer servers (output ports of three 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 without changing 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. In so doing, 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 modifier 'variable' when referring to variable servers to avoid repetition.

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 modelled as variable servers. In the example shown in Fig. 2, K equals 9. Each of these servers is given a unique identity, an integer ranging from 1 to K. A connection–server graph is constructed

as a labelled, directed graph with the servers as its nodes. A directed edge is introduced from server m to server n if there is a connection served by server m followed by server n. The edge is labelled by the connection that uses the servers in immediate sequence. Fig. 3 shows the connection–server graph corresponding to the system shown in Fig. 2. The sources and destinations of connections are also shown in the connection–server graph to facilitate the later discussion of our CAC algorithm.

We denote the sequence of servers serving connection M_i by

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

where K_i is the total number of servers serving connection M_i, and $s(i, j)$ the identity of the jth server in the path of connection M_i. For example, from Fig. 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 x is one of the servers in H_i, such that $x = s(i, k)$, $k \neq K_i$, then function $next_i(x)$ is defined as

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

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

2.3 Individual server analysis

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. We assume that time is normalised in terms of the cell transmission time in the ATM network. That is, time is considered a discrete quantity with the cell transmission time taken as unit time.

We need some notations to facilitate the discussion. Let $d_{i,s}$ be the worst-case 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

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

where d_i is the worst-case 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 input traffic patterns of all the connections sharing the server are known. The traffic pattern of a connection at a point in the network is characterised by a traffic descriptor [2]. 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 [1, 11, 12].

We use the maximum rate function. $\Gamma(I)$ as our traffic descriptor. $\Gamma(I)$ specifies the maximum arrival rate of cells in any interval of length I. Equivalently, at most $I\Gamma(I)$ cells belonging to the connection may arrive in an interval of length I.

A connection's actual traffic pattern may differ from that implied by the traffic descriptor used to describe the connection's traffic. The *maximum rate function* specifies the worst case behaviour of the traffic. This information is necessary for 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, ..., K), by $\Gamma_{i,s}(I)$. However, if server s does not belong to 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 the same as the one at the output of server $s(i, j)$. In Section 2.3.3 we will present a method 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. Assume that the maximum rate function traffic descriptors of all the connections at the input to server s are known. Then the following result from [1, 20] can be used to find the worst-case delay experienced by a cell and the maximum queue length at server s.

Theorem 1. Consider connection M_i that traverses server s. $d_{i,s}$ the worst-case delay experienced by connection M_i, and q_s, the maximum queue length at server s, are given by

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

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

$$L_s = \min \left(I \middle| \sum_m \Gamma_{m,s}(I) \leq I \right) \quad (6)$$

Theorem 1 can be proved by applying theorem 4.1 given in [3].

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 = <$s(i, 1)$, $s(i, 2)$, ..., $s(i, j)$, ..., $s(i, K_i)$>. Assume that $d_{i,s(i,1)}$, $d_{i,s(i,2)}$, ... $d_{i,s(i,k)}$, the worst-case delays suffered by M_i at the first k ($k < K_i$) servers in its connection path are known. Then the 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

$$\Gamma_{i,s(i,k+1)}(I) \leq \min \left(1, (1 + \frac{c_{i,k}}{I}) \Gamma_{i,s(i,1)}(I + c_{i,k}) \right) \quad (7)$$

where $c_{i,k}$ is given by

$$c_{i,k} = \sum_{1 \leq l \leq k} d_{i,s(i,l)} \quad (8)$$

The proof of theorem 2 is given in the Appendix (Section 8.1).

Theorem 2 gives an upper bound on a connection's maximum rate function at the input of server $s(i, k + 1)$ in terms of the parameters of the source traffic. Theorem 2 is a generalisation of theorem 2.1 in [3], where the maximum rate function at the output of an FCFS server was obtained in terms of the parameters of traffic at the input to the server. This is useful in practice. Most often, sources in hard real-time systems generate regular traffic (e.g. periodic) for which $\Gamma(I)$ can be described by a closed-form expression. Thus, theorem 2 facilitates efficient computation of maximum traffic-rate functions inside the network, making an efficient CAC algorithm feasible. Our performance results will also demonstrate that the delay bounds computed by using eqns. 5 and 7 are reasonable in that the CAC algorithm has a high probability of connection admission for normal loads.

3 Connection admission control algorithm

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

Any connection admission control algorithm has to satisfy the following two properties:

Property 1: 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 ones) must be no more than their deadlines and there must be no buffer overflow.

Property 2: 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 that takes a short time to make an admission decision.

To formalise the property we introduce the following notations and conventions:

• Let $\vec{d}(n)$ be a vector of size n. (We explicitly specify the size of the vector; this is to avoid the confusion of the number of connections considered. However, when the context is clear we omit this specification). $\vec{d}^*(n)$ represents the end-to-end delays associated with n connections, that is

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

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. $\vec{D}(n)$ represents the end-to-end deadlines associated with n connections. That is

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

where D_i is the deadline associated with M_i.

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

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

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

• Let $\vec{B}(K)$ be a vector size K. $\vec{B}(K)$ represents the buffer capacities of the server s. That is

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

where B_s denotes the buffer capacity at server s.

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

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

and

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

In terms of these notations the correctness property can be stated as follows: if the new connection $M_N + 1$ is admitted then

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

and

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

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

3.2 Motivation

We know that once a new connection is admitted its traffic can perturb the traffic of existing connections. This perturbation is not limited to the connections that share a server with the new connection but may spread to other connections in the system. Thus, to make it correct a CAC algorithm must take into account the extent of such perturbation and re-evaluate the delays and queue lengths affected by the perturbation.

Given the network decomposition methodology, a straightforward approach to solving the 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-analysed.

However, there is an inherent problem in such an approach. Consider the system presented in Fig. 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 Fig. 3 shows the situation in which M_4 has been admitted.

There is a cyclic dependency in the connection–server graph. For example, M_4 shares server 3 with M_1, hence affecting M_1's traffic. Since M_1 later shares server 5 with M_2, the behaviour of server 5 and connection M_2 may be impacted by the 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 analyse the impacted servers in a simple sequential manner. Our CAC algorithm is an iterative procedure. During the iterations it explicitly and cyclically traces and analyses those servers impacted.

In general, a system with such 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 [1, 2]. Particularly, when an iterative method is used to analyse the system one may run into such risk as the procedure may not converge, violating the termination requirement. We are dealing with hard real-time systems which have stringent dead line 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 used in the algorithm. We assume that the network management system, which invokes the CAC algorithm. maintains the following data structures:

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

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

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

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

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

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

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

Once a new connection admission request arrives, the new connection ($M_N + 1$) presents the following information to the system:

- $H_{N+1} = <s(N + 1, 1), ..., s(N + 1, K_{N+1})>$, 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, $\vec{q}(K)$, $\vec{D}(N)$, $\vec{\Gamma}(N)$, H_{N+1}, D_{N+1}, $\Gamma_{N+1,s(N+1,1)}$ and $\vec{B}(K)$ to the CAC algorithm. In addition to these 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, an ordered list of the server identities. When the CAC algorithm detects that a server would be affected by admitting the new connection, it will append the corresponding identity 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 analysed.

3.4 Algorithm

The pseudocode for the CAC algorithm is given in Fig. 4. It is an iterative procedure which efficiently determines and re-analyses the servers that will be affected if the new connection M_{N+1} is admitted. The algorithm has three phases.

3.4.1 Initialisation phase, lines 1–11: In this phase the algorithm copies the system data structures dM and $\vec{q}(K)$ into its internal working space. Matrix $dM^{internal}$ of the algorithm is initialised as follows. For $1 \le s \le K$,

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

Also, $\vec{D}(N + 1)$ and $\vec{d}(N + 1)$ are constructed for the set of connections under consideration,including the new connec-

```
Connection_Admission_Control (dM, q(K), D(N), Γ(N), H_{N+1}, D_{N+1}, Γ_{N+1,s(N+1,1)}, B(K))
/* Initialisation phase: Constructing d(N + 1) and D(N + 1) for connection set */
1.  q^internal(K) = q(K);
2.  D(N + 1) = (D_1, D_2, ..., D_N, D_{N+1});
3.  dM^internal = dM;
4.  for all s do
5.      dM^internal_{N+1,s} = 0;
6.  end for all
7.  for i = 1 to N + 1 do
8.      d_i = Σ_{s∈H_i} dM^internal_{i,s};
9.  end for
10. d(N + 1) = (d_1, d_2, ..., d_N, d_{N+1});
11. Impact_server_list = < s(N + 1, 1), s(N + 1, 2), ..., s(N + 1, K_{N+1}) >;
/* Iteration phase */
12. while ((Impact_server_list ≠ ∅) and (d(N + 1) ≤ D(N + 1)) and (q^internal(K) ≤ B(K))) do
13.     s = first_element(Impact_server_list);
14.     Impact_server_list = Impact_server_list −;
15.     old_q = q_s^internal;
16.     q_s^internal = compute_queue_length(s);
17.     for (i = 1 to N + 1) do
18.         if s ∈ H_i then
19.             dM^internal_{i,s} = q_s;
/* If delay or queue length at server s changes, add the succeeding server to Impact_server_list */
20.             if ((old_q ≠ q_s^internal) and
                    (next_i(s) ∉ Impact_server_list)) then
21.                 append_to_list(Impact_server_list, next_i(s));
22.             end if
23.         end if
24.     end for
25.     for i = 1 to N + 1 do
26.         d_i = Σ_{s∈H_i} dM^internal_{i,s};
27.     end for
28.     d(N + 1) = (d_1, d_2, ..., d_N, d_{N+1});
29. end while
/* QoS verification phase */
30. if ((d(N + 1) ≤ D(N + 1)) and (q^internal(K) ≤ B(K))) then
31.     return(Accept, D(N + 1), dM^internal, q^internal(K), Γ(N + 1));
32. else return(Reject).
```

Fig. 4 *Pseudocode for CAC algorithm*

tion and the existing ones. Since all the servers in the connection path of M_{N+1} will be directly impacted if M_{N+1} is admitted, the algorithm initialises *Impact_server_list* with the identities of all the servers in M_{N+1}'s path.

3.4.2 Iteration phase, lines 13–29: 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 with 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

(i) First, the algorithm removes the first server in *Impact_server_list* for analysis.

(ii) Next, the algorithm computes an upper bound on the queue length and then the delay at the server analysed. The results presented in Section 3.3 are used to achieve this.

(iii) 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 $next_l(s)$ function defined in Section 3.2 facilitates this process. Thus, the algorithm systematically traces the perturbation caused by connection M_{N+1} in the network.

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.

3.4.3 Verification phase, lines 30–32: 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 $\vec{\Gamma}(N+1)$ to the network management system.

Table 1: Values of *Impact_server_list* in the example

Iteration k	*Impact_server_list* at beginning of iteration	Server analysed during iteration	*Impact_server_list* at end of iteration
1	<1, 3, 4>	1	<3, 4,2>
2	<3, 4, 2>	3	<4, 2, 5>
3	<4, 2, 5>	4	<2, 5>
4	<2, 5>	2	<5>
5	<5>	5	<6, 7>
6	<6, 7>	6	<7>
7	<7>	7	<8, 1>
8	<8, 1>	8	<1>
9	<1>	1	<2, 3>
10	<2, 3>	2	<3>
11	<3>	3	<4, 5>
:	:	:	:
:	:	:	:
:	:	:	:

3.5 Some remarks

Consider the system presented in Fig. 2. Let $\{M_1, M_2, M_3\}$ be the set of connections that already exist in the system and M_4 be one newly requested. The connection–server graph in Fig. 3 shows the situation in which M_4 is admitted. During the execution of the CAC algorithm, servers impacted by admitting the new connection are identified and re-analysed. For illustration, assume that all the successor servers are impacted during the first 11 iterations. Table 1 shows the members of *Impact_server_list* and the servers analysed during the iterations. The reader is encouraged to verify the contents of Table 1 by tracing the algorithm. This will help understand the algorithm. 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 behaviour 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! From Table 1, it is clear that our CAC algorithm is able to capture this dependency. The algorithm explicitly and cyclically traces and analyses those servers impacted during the iterations.

Furthermore, note that during an execution of our CAC algorithm the iteration stops whenever the delay or queue length goes beyond the allowable bounds. This automatically solves the convergence problem in a potentially unstable system. Section 8.2 formally establishes the termination property.

4 Performance evaluation

We first define performance metrics, then describe the system architecture considered and present the performance results.

4.1 Metrics

Consider the following two metrics for evaluating a CAC algorithm for hard real-time connections.

(i) *Admission probability (AP(U))*: defined as the probability that HRT connections are admissible on condition that the average utilisation of the inter-switch physical links in the network is U.

(ii) *Execution time (ET(U))*: defined as the average time taken for the CAC algorithm to make an admission decision given that the average utilisation of the inter-switch links in the network is U.

It is desirable to have a CAC algorithm with high admission probability and low execution time. However, there is an obvious trade-off between the two. A CAC algorithm that blindly rejectes all the admission requests should have almost zero execution time but certainly the lowest admission probability. On the other hand, one may design a CAC algorithm with higher admission probility by computing tighter delay bounds which may result in a longer execution time. In the design of our CAC algorithm. we have carefully balanced both factors. The data shown in the following subsections will demonstrate that our CAC algorithm can achieve reasonable admission probability aud yet maintains a low execution time.

4.2 Experimental system setup

Here, we report the results of performance evaluation for a svstem architecture shown in Fig. 5. The results from the other systems are similar and therefore are not presented here.

The network considered consists of four 32 × 32 ATM switches. That is, each switch has 32 input lines and 32 output lines. As shown in Fig. 5. 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.

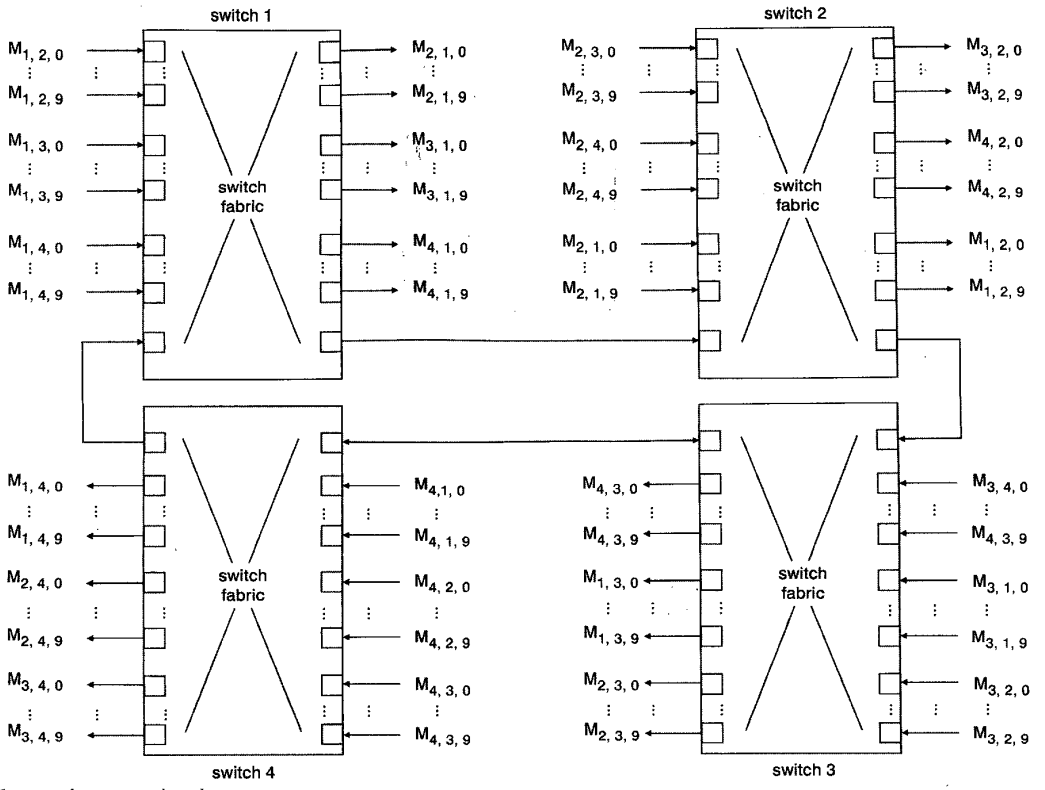

Fig. 5 *The network system evaluated*
Connection $M_{i,j,k}$ enters the network at switch i and exits at switch j

Several systems with different architectures have also been evaluated, the results are similar.

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 characterise 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 1,000 connection sets were randomly generated. For each connection, the total number of cells per period was chosen from a geometric distribution with mean ten. Similar results have been obtained with different settings of parameters.

4.3 Admission probability

Fig. 6 shows the admission probabilitv results for our sample network. The performance figures are corresponding to two values of deadline D_i, (P_i and $2P_i$). It is common practice in a hard real-time system that deadlines are associated with periods [21, 22]. From Fig. 6, we can make the following observations:

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

• 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 $2P$.

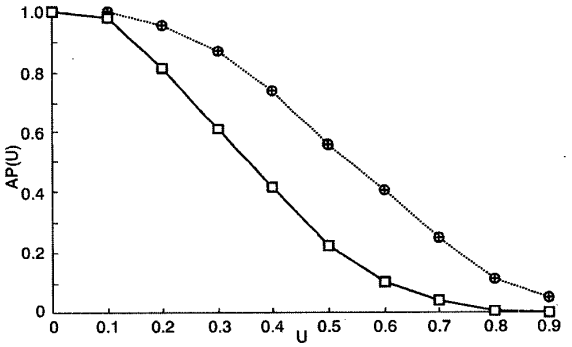

Fig. 6 *Admission probability against link utilisation: unregulated source (system S)*
—□— $D = P$
····⊕···· $D = 2P$

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 an increasing 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 [15] to select parameters of the traffic regulation mechanism (e.g. leaky bucket) so that an appropriate level of regulation is maintained.

Fig. 7 shows the admission probabilitv results for the regulated system (called system R). In comparison with the data in Fig. 6 we see that by using input traffic regulation 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%.

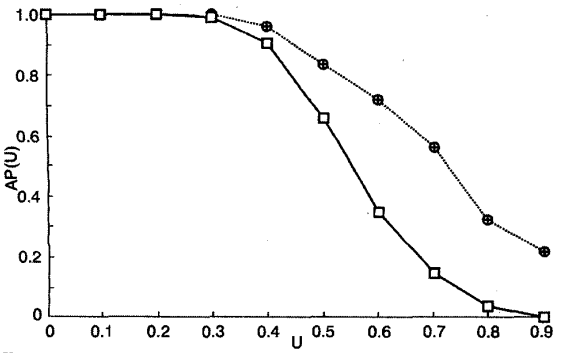

Fig. 7 *Admission probability against link utilisation: regulated source (system R)*
—□— $D = P$
····⊕···· $D = 2P$

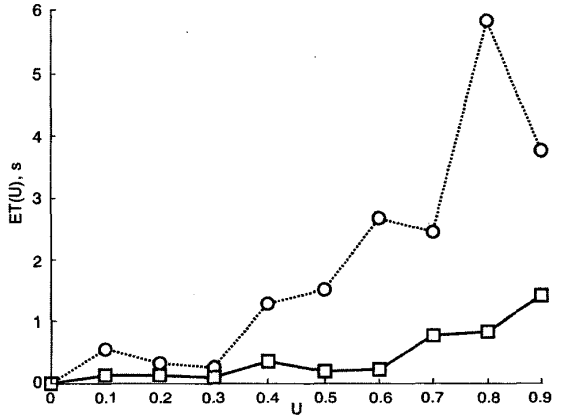

Fig. 8 *Execution time results with and without regulation*
—□— system S
····○···· system R

4.4 Execution time

Here we evaluate the efficiency of our CAC algorithm in terms of the time taken by our CAC algorithm to make an admission decision. It is desirable that any online CAC algorithm has a small execution time. Fig. 8 presents the results for both system S and system R. In the case of the regulated system, the execution time reported included that taken to select proper regulation parameters.

From Fig. 8 we can make the following observations:

• We found that in terms of $ET(U)$ our CAC algorithm performed very well when it was used alone. The maxmium average time required by the algorithm was less than 1.4 s. For a large range of utilisation, (e.g. U = 0 to 60%), the $ET(U)$ was less than 0.36s.

• The $ET(U)$ is increased when the CAC algorithm is used in conjunction with the regulation method. This is because of the additional overheads due to the regulation parameter setting method. However, even in this case the maximum $ET(U)$ is about 6s. For a large range of utilisation (e.g. U = 0 to 60%), the $ET(U)$ is less than 2.7s.

• From Fig. 8, we can also observe that the $ET(U)$ may not monotonically increase as the average link utilisation increase. With a little reflection we notice that the average link utilisation impacts the $ET(U)$ of the CAC algorithm in two different ways.

• When the link utilisation increases, so does the load of the server. Consequently, the length of the busy interval at the server also increases. This results in an increase in the time taken to compute the delay upper bound at the server. As a result, the $ET(U)$ of the CAC algorithm increases.

• On the other hand, an increase of link utilisation may reduce the number of iterations during an execution of the CAC algorithm. This is because a high link utilisation increases the likehood that the deadline and buffer requirements will be violated. Therefore the CAC algorithm terminates in 'reject' with less number of iterations. As a result, the increased link utilisation may reduce the $ET(U)$.

These two opposing effects explain the nonmonotonical phenomenon of $ET(U)$ as a function of the link utilisation.

5 Conclusions

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 modelled as a collection of servers. This approach has been used by several researchers before [1, 11]. 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. A potential concern one may have on the use of the maximum rate function is the need for computational resources (time, space or both) to obtain values of $\Gamma(I)$ for different values of I. By generalising the result in [1], we provide a simple and efficient method for computing $\Gamma(I)$ values for a connection at any point in the network.

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. Thus, our algorithm is correct in the sense that it accepts a connection only if the QoS requirements of the new connection and existing ones can be met. The performance of our algorithm is throughly evaluated. We found that the algorithm is effective and efficient in the sense that it can achieve a reasonable admission probability and yet maintain a low execution time.

This work can be extended in several ways. An important issue in hard real-time systems is to consider good approximations [11] of the explicit traffic descriptor so as to improve the execution overheads of the algorithm, although this will result in a corresponding decrease in admission probability.

6 Acknowledgments

This work was partially sponsored by the City University of Hong Kong under grant 7100130 and 7001060, and UGC (University Grant Council) Hong Kong under grant 9040511-CityU 1076-00E. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official polices or endorsements, either express or implied, of the UGC Hong Kong or the City University of Hong Kong.

7 References

1 CRUZ, R.L.: 'A calculus for network delay', *IEEE Trans. Inf. Theory*, 1991, **37**, (1), pp. 114–141
2 PAREKH, A.K.J.: 'A generalised processor sharing approach to flow control in integrated services networks'. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1992
3 KAMAT, S., and ZHAO, W.: 'Performance comparison of two token ring protocols for real-time communication' *in* SON, S. (Ed.): 'Principles of real-time systems' (Prentice Hall, 1994)
4 STANKOVIC, J.A.: 'Misconceptions about real-time computing: A serious problem for next generation systems', *Computer*, 1988, **21**, (10), pp. 10–19
5 STANKOVIC, J.A., and RAMAMRITHAM, K. (Eds.): 'Hard real-time systems' (IEEE Computer Society Press, 1988)
6 VAN TILBORG, A.M., and KOOB, G.M.: 'Foundations of real-time computing: Scheduling and resource management' (Kluwer Academic Publishers, 1991)
7 KAMAT, S., and ZHAO, W.: 'Real-time schedulability of two token ring protocols'. Proceedings of the IEEE international conference on *Distributed computing systems*, May 1993, pp. 347–354
8 STROSNIDER, J. K., MARCHOK, T., and LEHOCZKY, J.: 'Advanced real-time scheduling using the IEEE 802.5 token ring'. Proceedings of the IEEE symposium on *Real-time systems*, December 1988, pp. 42–52
9 SHA, L., SATHAYE, S.S., and STROSNIDER, J.K.: 'Scheduling real-time communication on dual-link networks'. Proceedings of the IEEE Symposium on *Real-time systems*, December 1992, pp. 188–197
10 AGRAWAL, G., CHEN, B., ZHAO, W., and DAVARI, S.: 'Guaranteeing synchronous message deadlines in high speed token ring networks with timed token protocol'. Proceedings of the 12th IEEE international conference on *Distributed computing systems*, June 1992, pp. 468–475
11 RAHA, A., KAMAT, S., and ZHAO, W.: 'Guaranteeing end-to-end deadlines in ATM networks'. Proceedings of the 15th IEEE international conference on *Distributed computing systems*, June 1995,
12 'ATM user-network interface specification version 3.1'. ATM Forum, 1995
13 RAHA, A., KAMAT, S., and ZHAO, W.: 'Using traffic regulation to meet end-to-end deadlines in ATM networks'. Presented at the 1995 international conference on *Network protocols* (ICNP-95), 1995,
14 DEMERS, A., KESHAV, S., and SHENKER, S.: 'Analysis and simulation of a fair queueing algorithm'. Proceedings of ACM SIGCOMM'89, September 1989, pp. 1–12
15 FERRARI, D. , and VERMA, D.C.: 'A scheme for real-time channel establishment in wide-area networks', *IEEE J. Sel. Area. Commun.*, 1990, **8**, (3), pp. 368–379
16 GOLESTANI, S.J.: 'A framing strategy for congestion management', *IEEE J. Sel. Area. Commun.*, 1991, **9**, (7), pp. 1064–1077
17 KALMANEK, C.R., KANAKIAG, H., and KESHAV, S.: 'Rate controlled servers for very high-speed networks'. Proceedings of IEEE conference on *Global telecommunications*, December 1990, pp. 300.3.1–300.3.99
18 TRAJKOVIC, L., and GOLESTANI, S.J.: 'Congestion control for multimedia services', *IEEE Netw.*, 1992, **6**, (5), pp. 20–26
19 ZHANG, H., and KESHAV, S.: 'Comparison of rate-based service disciplines'. Proceedings of ACM SIGCOMM'91, September 1991, pp. 113–121
20 RAHA, A.: 'Real time communication in ATM networks'. PhD thesis, Department of Computer Science, Texas A&M University 1995, in preparation
21 LIU, C.L., and LAYLAND, J.W.: 'Scheduling algorithms for multiprogramming in a hard-real-time environment', *J. Assoc. Comput. Mach.*, 1973, **20**, (1), pp. 46-61
22 MALCOLM, N., and ZHAO, W.: 'Hard real-time communication in multiple-access networks', *J. Real-Time Syst.*, 1995

8 Appendix

8.1 Proof of theorem 2

Theorem 2

$$\Gamma_{i,s(i,k+1)}(I) \leq \min\left(1, (1 + \frac{c_{i,k}}{I})\Gamma_{i,s(i,1)}(I + c_{i,k})\right) \tag{18}$$

where $c_{i,k}$ is given by

$$c_{i,k} = \sum_{1 \leq l \leq k} d_{i,s(i,l)} \tag{19}$$

Proof: We prove this theorem by induction. By definition, $\Gamma(I) \leq 1$. Thus it is sufficient to show that

$$\Gamma_{i,s(i,k+1)}(I) \leq (1 + \frac{c_{i,k}}{I})\Gamma_{i,s(i,1)}(I + c_{i,k}) \tag{20}$$

where $c_{i,k}$ is given by

$$c_{i,k} = \sum_{1 \leq l \leq k} d_{i,s(i,l)} \tag{21}$$

Base step: k = 1. When $k = 1$. $\Gamma_{i,s\,(i,k+1)}(I) = \Gamma_{i,s(i,2)}(I)$ is the maximum rate function of M_i at the output of the first server traversed by M_i. Consider server $s(i, 1)$ and an arbitrary time instant (say t). Let $a_{i,s(i,1)}\,(t - d_{i,s(i,1)}, t + I)$ $t \geq 0$, be the number of M_i's cells that arrive at the input of $s(i, 1)$ in interval $[t - d_{i,s(i,1)}, t + I)$. Let $x_{i,s(i,1)}(t, t + I)$, $t \geq 0$ be the number of M_i's cells that are transmitted by $s(i, 1)$ in interval $[t, t + I)$. Thus, by definition,

$$I\Gamma_{i,s(i,2)}(I) = \max_t\left(x_{i,s(i,1)}(t, t + I)\right) \tag{22}$$

$d_{i,s(i,1)}$ is the worst-case delay experienced by M_i's cells at server $s(i, 1)$. Therefore if a cell of connection M_i is transmitted by server $s(i, 1)$ at time t, $t \geq 0$, then the earliest time that the cell can arrive at the input of server $s(i, 1)$ is $t - d_{i,s(i,1)}$. Hence

$$x_{i,s(i,1)}(t, t + I) \leq a_{i,s(i,1)}(t - d_{i,s(i,1)}, t + I) \tag{23}$$

But by definition

$$a_{i,s(i,1)}(t - d_{i,s(i,1)}, t + I) \leq (I + d_{i,s(i,1)})\Gamma_{i,s(i,1)}(I + d_{i,s(i,1)}) \tag{24}$$

By substituting eqn. 24 in eqn. 23 we obtain

$$x_{i,s(i,1)}(t, t + I) \leq (I + d_{i,s(i,1)})\Gamma_{i,s(i,1)}(I + d_{i,s(i,1)}) \tag{25}$$

Due to the arbitrarity of t and eqn. 22 we have

$$I\Gamma_{i,s(i,2)}(I) \leq (I+d_{i,s(i,1)})\Gamma_{i,s(i,1)}(I+d_{i,s(i,1)}) \tag{26}$$

Dividing by I on both sides, eqn. 26 becomes

$$\Gamma_{i,s(i,2)}(I) \leq (1+\frac{d_{i,s(i,1)}}{I})\Gamma_{i,s(i,1)}(I+d_{i,s(i,1)}) \tag{27}$$

Hence the base case holds.

Induction hypothesis: Let the maximum rate function at the input of the kth server in M_i's path be given by

$$\Gamma_{i,s(i,k)}(I) \leq (1 + \frac{c_{i,k-1}}{I})\Gamma_{i,s(i,1)}(I + c_{i,k-1}) \tag{28}$$

where $c_{i,k-1}$ is given by

$$c_{i,k-1} = \sum_{1 \leq l \leq k-1} d_{i,s(i,l)} \tag{29}$$

Induction step: $\Gamma_{i,s\,(i,k+1)}(I)$ is the maximum rate function of M_i at the output of the kth server traversed by M_i. By using similar arguments discussed in the base case we have

$$I\Gamma_{i,s(i,k+1)}(I) \leq (I + d_{i,s(i,k)})\Gamma_{i,s(i,k)}(I + d_{i,s(i,k)}) \tag{30}$$

Substituting eqn. 28 in eqn. 30 we obtain

$$I\Gamma_{i,s(i,k+1)}(I) \leq (I + d_{i,s(i,k)})(1 + \frac{c_{i,k-1}}{I + d_{i,s(i,k)}}) \times \Gamma_{i,s(i,1)}(I + d_{i,s(i,k)} + c_{i,k-1})) \tag{31}$$

Thus

$$I\Gamma_{i,s(i,k+1)}(I) \leq (I + d_{i,s(i,k)} + c_{i,k-1}) \times \Gamma_{i,s(i,1)}(I + d_{i,s(i,k)} + c_{i,k-1})) \tag{32}$$

But

$$c_{i,k-1} + d_{i,s(i,k)} = \sum_{1 \leq l \leq k-1} d_{i,s(i,l)} + d_{i,s(i,k)} \tag{33}$$

$$= \sum_{1 \le l \le k} d_{i,s(i,l)} \quad (34)$$

$$= c_{i,k} \quad (35)$$

Substituting eqn. 35 in eqn. 32 we have

$$I\Gamma_{i,s(i,k+1)}(I) \le (I + c_{i,k})\Gamma_{i,s(i,1)}(I + c_{i,k}) \quad (36)$$

That is

$$\Gamma_{i,s(i,k+1)}(I) \le (1 + \frac{c_{i,k}}{I})\Gamma_{i,s(i,1)}(I + c_{i,k}) \quad (37)$$

Therefore the theorem follows.

8.2 Termination property of the algorithm

To prove the termination property of the algorithm we need some lemmas. For notational convenience, let $\vec{d}^{[k]}$ be vector $\vec{d}$ at the end of the kth iteration. Let $\vec{d}^{[0]}$ be the vector $\vec{d}$, which is computed at the initialisation phase in Fig. 4.

Lemma 1: During any execution of the CAC algorithm, the following inequality holds: for $k \ge 0$,

$$\vec{d}^{[k]} \le \vec{d}^{[k+1]} \quad (38)$$

Furthermore, the strict inequality ($\vec{d}^{[k]} < \vec{d}^{[k+1]}$) holds if a server is appended to *Impact_server_list* during iteration $k + 1$.

Proof: Consider the first iteration. Before the iteration we have $\vec{d} = \vec{d}^{[0]}$ and at the end of the iteration we have $\vec{d} = \vec{d}^{[1]}$. During this iteration, server $s = s(N + 1, 1)$ is analysed. Consider a previously admitted connection M_i. If M_i passes through this server, then $d_{i,s}$, M_i's delay bound at this server is updated. Let $d_{i,s}^{old}$ and $d_{i,s}^{new}$ be the old and new values of this delay bound, respectively. Thus by definition

$$d_i^{[1]} = \begin{cases} d_i^{[0]} & \text{if } M_i \text{ does not pass through server } s \\ d_i^{[0]} + d_{i,s}^{new} - d_{i,s}^{old} & \text{otherwise} \end{cases} \quad (39)$$

Considering the fact that $d_{i,s}^{new}$ takes into account the impact of the new connection (M_{N+1}) while $d_{i,s}^{old}$ does not, from eqn. 5 we have

$$d_{i,s}^{new} - d_{i,s}^{old} = \lceil \max_{1 \le I \le L_s^{new}} (\sum_{i=1}^{N+1} I\Gamma_{i,s}(I) - I) \rceil - \lceil \max_{1 \le I \le L_s^{old}} (\sum_{i=1}^{N} I\Gamma_{i,s}(I) - I) \rceil \quad (40)$$

where L_s^{new} and L_s^{old} are the new and old maximum busy interval lengths computed by using eqn. 6. Substituting eqn. 6 in eqn. 40 we can show that $d_{i,s}^{new} - d_{i,s}^{old} \ge 0$. Thus $d_i^{[0]} \le d_i^{[1]}$. It then follows that $\vec{d}^{[0]} \le \vec{d}^{[1]}$. Furthermore, if a server is added to *Impact_server_list* during iteration 1, then we have $old\_q < q_s^{internal}$. This implies that $d_i^{[0]} < d_i^{[1]}$ if connection M_i passes through the server. Because there will be at least one connection doing so, $\vec{d}^{[0]} < \vec{d}^{[1]}$. We have now established the lemma for the case of $k = 0$. By mathematical induction, the general case ($k > 0$) can be proved.

To prove the following lemma, define *size_of_list*(k) as the size of *Impact_server_list* at the end of the kth iteration. Let *size_of_list*(k)(0) be the size of *Impact_server_list* before the iteration phase, i.e. size of list H_{N+1}.

Lemma 2: During an execution of the CAC algorithm, if $\vec{d}^{[k]} = \vec{d}^{[k+1]}$ then

$$size\_of\_list(k + 1) = size\_of\_list(k) - 1 \quad (41)$$

Proof: In the algorithm given in Fig. 4, the first element in *Impact_server_list* is removed (see line 14) during every iteration. Thus, the size of the *Impact_server_list* is reduced by one. If $\vec{d}^{[k]} = \vec{d}^{[k+1]}$, then the queue length of the server is not changed. Hence, from line 20 of Fig. 4, no new element is appended to *Impact_server_list*. The lemma then follows.

Lemma 3: During an execution of the algorithm. before its termination there is a subsequence of iterations (iterations I_0, I_1, I_2, ..., I_j ...) where $I_0 < I_1 < I_2 < ..., I_{j-1} < I_j$ ... such that for $j > 1$

$$I_j - I_{j-1} \le K \quad (42)$$

and

$$\vec{d}^{[I_{j-1}]} < \vec{d}^{[I_j]} \quad (43)$$

Proof: This subsequence of iterations can be established as follows: Let $I_0 = 0$. Recall that in the algorithm before the first iteration, *Impact_server_list* was initialised by the identities of all the servers traversed by M_{N+1}, that is by H_{N+1}. We have

$$size\_of\_list(0) \le K \quad (44)$$

Thus, by the end of the Kth iteration, every server in the initial *Impact_server_list* has been analysed at least once. Using lemmas 1 and 2, it is obvious that either the algorithm has terminated, or between the first iteration and the Kth iteration, there must be at least one iteration (let it be iteration x) during which the list is appended with an element. This means that compared with $\vec{d}^{[x-1]}$, at least one of the elements in $\vec{d}^{[x]}$ has increased. By eqn. 14,

$$\vec{d}^{[x-1]} < \vec{d}^{[x]} \quad (45)$$

Because $\vec{d}^{[I_0]} \le \vec{d}^{[x-1]}$, we have

$$\vec{d}^{[I_0]} < \vec{d}^{[x]} \quad (46)$$

Note that $x - I_0 \le K$. Let I_1 be x. Thus, we have iteration I_1 identified. By mathematical induction, the subsequent iterations can be identified.

The following lemma gives the delay and buffer size invariance which holds for each iteration of the CAC algorithm.

Lemma 4: If the CAC algorithm accepts a connection then $1 \le i \le N + 1$, $1 \le s < K$,

$$d_i^* \le d_i \quad (47)$$

and

$$q_s^* \le q_s \quad (48)$$

where d_i^* is the end-to-end delay experienced by a cell of connection M_i and qs^* is the queue length at server 5. The formal proof of this property is given in [13]. Intuitively lemma 4 can be explained as follows. Consider connections M_1, ..., M_N, that are already admitted by the network before the arrival of the connection request M_{N+1}. Since these connections are admitted by the network, their QoS is not violated before the arrival of the new request. Therefore, eqns. 47 and 48 must be true before the new connection request M_{N+1} is considered. The cells from M_{N+1} will change the cell arrival traffic at the servers in the network. Therefore if M_{N+1} is accepted by the network, then the worst-case delays suffered by the cells of all the connections may change due to the perturbation caused by the cells of the new connection. This perturbation originates from server $s(N + 1, 1)$, the server at which the cells of connection M_{N+1} enter the network and may propagate along different paths. Hence, if we assume that eqns. 47 and 48 are not true then there must exist a server (say s) along a

perturbation path originating from $s(N+1, 1)$, for which the computed values of $d_{i,s}$ and q_s are incorrect. However, the CAC algorithm traces all the possible perturbation path 5 by using the list *Impact_server_list*. Further, the CAC algorithm uses the results of theorems 1 and 2 to determine the worst-case cell delay, maximum queue length, and bounds on output traffic flows at every server in *Impact_server_list*. Therefore, no such server s can exist. Hence, the lemma holds.

Theorem 3: The CAC algorithm terminates.

Proof: We show this by contradiction. Assuming that there is a system (e.g. K servers and $N+1$ connections) such that the CAC algorithm does not terminate.

Given the contradiction assumption, the algorithm iterates infinitely. However, from lemma 3 we know that there is a subsequence of iterations (iterations I_0, I_1, I_2, ..., I_j ...) where $I_0 < I_1 < I_2 < ..., I_{j-1} < I_j ...$ and

$$\vec{\boldsymbol{d}}^{[I_0]} < \vec{\boldsymbol{d}}^{[I_1]} < \vec{\boldsymbol{d}}^{[I_2]} < \ldots \vec{\boldsymbol{d}}^{[I_{j-1}]} < \vec{\boldsymbol{d}}^{[I_j]} < \ldots \tag{49}$$

Thus, $\{\vec{\boldsymbol{d}}^{[I_0]}, \vec{\boldsymbol{d}}^{[I_1]}, \vec{\boldsymbol{d}}^{[I_2]}, \ldots \vec{\boldsymbol{d}}^{[I_{j-1}]}, \vec{\boldsymbol{d}}^{[I_j]}, \ldots\}$ is a monotonically increasing sequence. From eqn. 5, we know that the elements in $\vec{\boldsymbol{d}}$ take positive integer values. Thus, eqn. 49 implies that there must be an iteration (say I_x) such that

$$\vec{\boldsymbol{d}}^{[I_x]} > \vec{\boldsymbol{D}} \tag{50}$$

This is one of the termination conditions in the iteration. Thus, the execution of the algorithm must terminate. This is a contradiction. Thus the theorem is proved.

Theorem 4: The CAC algorithm is correct.

Proof: Assume that the CAC algorithm shown in Fig. 4 accepts a connection after E iterations. Since the connection is accepted by the CAC algorithm, we must have $\vec{\boldsymbol{d}}^{[E]}(N+1) \leq \vec{\boldsymbol{D}}(N+1)$ and $\vec{\boldsymbol{q}}^{[E]}(K) \leq \vec{\boldsymbol{B}}(K)$ (see line 31). That is for $1 \leq i \leq N+1$, we have

$$d_i^{[E]} \leq D_i \tag{51}$$

But by lemma 4 we have $d_i^* \leq d_i^{[E]}$. Therefore, for $1 \leq i \leq N+1$,

$$d_i^* \leq D_i \tag{52}$$

By using eqns. 9, 13 and 52 we get

$$\vec{\boldsymbol{d}}^*(N+1) \leq \vec{\boldsymbol{D}}(N+1) \tag{53}$$

Similarly, using lemma 4 we can show

$$\vec{\boldsymbol{q}}^*(K) \leq \vec{\boldsymbol{B}}(K) \tag{54}$$

Hence the CAC algorithm is correct.