As appeared in: Proceedings of the Real-Time Systems Symposium (RTSS'97), San Francisco, CA, Dec. 1997.

Static Priority Scheduling for ATM Networks

Chengzhi Li Riccardo Bettati Wei Zhao

Department of Computer Science
Texas A & M University
College Station, TX 77843-3112
FAX 409 - 847 - 8578
Phone 409 - 845 - 5098
Email: {chengzhi,bettati,zhao}@cs.tamu.edu

Abstract

Static-priority scheduling is popular for traffic scheduling in ATM switches because it is less costly than dynamic priority scheduling while being sensitive to the delay constraints of connections. We study delay computation and priority assignment problems in an ATM networks with static priority scheduling. Given an ATM network with arbitrary topology, it is possible that the traffic on it may become unstable (i.e., packet delays become unbounded) due to the potential cyclic dependency of the traffic. An unstable network is definitely unacceptable for many delay sensitive applications. We start by formally deriving a simple condition under which the network is guaranteed to be stable. We then develop a numerical method to compute worst-case end-to-end delays in an ATM network with arbitrary topology. Convergence of the method is formally proved and a closed form for the computing error is obtained. Despite of its advantages, static-priority scheduling remains sensitive to proper priority assignment. We describe two simple priority assignment methods, which we show to outperform other commonly used methods.

1. Introduction

In this paper, we study ATM networks that use *static-priority scheduling* as their cell transmission policy. Traditionally, FCFS (first-come-first-served) has been the discipline used for scheduling cell transmission. While inexpensive, FCFS is not sensitive to delay requirements of applications. Hence it makes it difficult to support multiple levels of quality of service as proposed in the ATM standard. On the other hand, *dynamic-priority* scheduling is capable of providing delay-sensitive cell transmission. However, dynamic-priority schemes are known to be difficult to implement in high-speed networks. At present there is no ATM product that provides dynamic link scheduling. *Static-priority* scheduling can be considered a compromise between FCFS and dynamic-priority scheduling. It is relatively inexpensive in comparison with dynamic-priority scheduling while providing delay-sensitive communication to applications.

Three key problems must be addressed when using static-priority scheduling in ATM networks.

- *End-to-End Delay Computation.* Usually, the end-to-end delay can be obtained using a *decomposition* approach. In this approach, the network is decomposed into *servers*, and each ATM connection is viewed as traversing a sequence of servers. The worst-case end-to-end cell delays are obtained by summing up the upper bounds of the delays suffered by a connection at each of the servers [4, 17, 18].
- *Stability.* Because the topology of an ATM network can be arbitrary, it is possible that traffic of one connection interacts with that of another, resulting in a cyclic dependency among the connections. In this case, the system may become unstable in the sense that some of the delays are unbounded [4, 17]. Even if the system is stable, an iterative procedure must be used for the delay computation to accommodate these feedback effects. In [19], the stability problem was addressed for systems with FCFS scheduling. Determining worst-case delays with FCFS scheduling in a system with traffic dependencies was dealt with in [14].
- *Priority Assignment.* The delay bounds for individual cells depend on the priorities assigned to their respective connections. This assignment should be sensitive to the deadline requirements of the connections in order to maximize the chance that the entire connection set can be admitted, i.e., all the deadlines of connections can always be satisfied. In [15] various priority assignment methods were examined and compared for systems with a single server or with traffic regulation mechanisms.

In this paper, we are addressing the three issues described above in the context of ATM networks. Differently from the previous work, we concentrate on ATM switches with static-priority schedulers. In addition, we allow for networks with arbitrary topology and therefore need to address the stability problem. We do not assume that traffic regulation mechanisms are in place. For a stable system with arbitrary topology, we describe an efficient iterative procedure to compute the worst-case end-to-end delay of cells. We formally prove the convergence of the procedure and derive a closed form for the computation error. No analysis on computation error for this type of iterative delay computation was reported before.

Based on the delay-computation method developed, we study five priority assignment algorithms. As the base method for comparison, we assign the same priority to all connections, effectively scheduling cells in FCFS manner. The second algorithm assigns the priority in a deadline-monotonic manner [1, 13]. That is, the smaller the relative deadline of a connection, the higher is its priority. The third algorithm combines the first two by recursively partitioning the connection set into subsets of connections. Priorities are assigned at subset level, based on deadline information, and the connections in each subset are scheduled FCFS order. Our performance comparison shows that this partition method outperforms the first two methods in all the situations tested.

It is to be expected that an algorithm that is aware of the topology or the load of the system performs better than strictly deadline driven. We therefore go on to study a basic priority assignment approach described by Cruz [4] for a limited class of system configurations. We integrate the scheme proposed by Cruz with our partition scheme, in fact proposing an algorithm that is both delay aware and topology aware. Our evaluations show that the integrated approach by a large margin outperforms the remaining four algorithms.

The rest of this paper is organized as follows: In Section 2, we review the previous work and compare it with the approach taken in this paper. Section 3 describes a formal model of connections and ATM networks. The worst-case delay computation method is developed in Section 4, while Section 5 deals with priority assignment methods and their performance evaluations. In Section 6, we conclude the paper with a discussion of possible extensions.

2. Previous Work

Determining delay bounds has been the pivotal issue in the development of real-time technology [12, 21, 22, 23, 24]. Much work has focused on centralized systems [16]. In general, obtaining delay bounds in a network environment is difficult due to the distributed nature of the problem. Nevertheless, considerable progress has been made recently towards obtaining delay bounds for network traffic in a variety of settings. Much of the previous studies on meeting end-to-end deadlines in ATM networks have concentrated primarily on the design and the analysis of *scheduling policies* for ATM switches [3, 6, 8, 11, 17, 19, 25]. The efficient implementation of most of these schemes turns out to be difficult in high-performance switches. Therefore, most ATM switches have very simple support for traffic scheduling, typically in form of FCFS or static priority. In this work, we assume that a static-priority scheduling discipline is used at the ATM switches. Most previous work also assumes that either cyclic dependencies among connections do not exist or that they are eliminated by some internal network control mechanism (e.g., traffic regulation, reshaping by dedicated hardware, and framing) [2, 4, 5, 11]. We explicitly take into account possible cyclic dependencies among connections without using such mechanisms. We investigate and obtain conditions under which the system is stable and are able to determine the delay bounds under such conditions.

3. Connection and Network Modes

Real-time systems typically operate in a *modal* fashion; at any given instant of time the system operates in one of a set of known modes [20]. System operation in a specific mode is characterized by the execution and the requirements of a previously defined set of applications. Thus, each operational mode of the system has an associated set of connections that must be established among applications that need to communicate. It is critical for these systems to ensure that the deadlines of all the connections in such a set are met.

Let $\mathcal{M} = \{M_1, M_2, \ldots, M_N\}$ denote the set of N connections that must be admitted for a successful operation of the system in a particular mode. Let D_i be the relative end-to-end *deadline* of cells belonging to connection M_i. The deadline is a fixed performance requirement. On the other hand, let d_i denote the worst case end-to-end *delay* of cells belonging to connection M_i. This delay varies with the load experienced by the network. Hence, the deadline requirements of a set of connections are met if the delays do not exceed the specified end-to-end deadlines.

3.1. ATM Networks

In an ATM network, hosts are connected to ATM *switches* and ATM switches are connected to each other using physical *links*. As an example, consider the network with four switches shown in Figure 1. Although this example may not be representative of a typical ATM network, it is used to illustrate a number of important concepts discussed in this paper. An ATM switch consists of *input ports*, a *switching fabric*, and *output ports*. When an ATM cell arrives at an input port of a switch, it is delivered by the switching fabric to an output port, and is transmitted to the next host or switch on the physical link associated with the output port. We model the ATM network as a collection of *servers*. A server is an abstraction of a network component

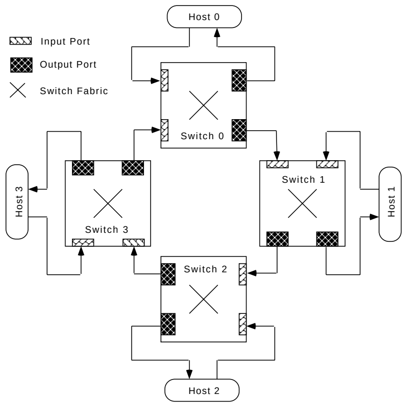


Figure 1. ATM Network with Four Switches

– be it an input port, a switching fabric, an output port, or a physical link – which is used by a connection.

Traditionally, servers are classified into two categories: *constant servers* and *variable servers* [4, 18, 19]. A constant server delays each cell by a constant amount of time. It does not, by itself, change the traffic flow characteristics of a connection. For example, physical links and nonblocking switching fabrics are constant servers. In switches without input buffering, input ports can be modeled as constant demultiplexer servers: arriving cells are demultiplexed based on the information in the cell header, a process that can be done in constant time.

The behavior of output ports of a switch, on the other hand, 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 used in the switch hardware. Note that a multiplexor server must be modeled 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 that at the input.

To simplify the delay analysis, we can eliminate all constant servers as follows: First, we subtract the constant delay of each server from the deadline of each connection traversing that server. Then we set the constant delay of each constant server to zero. Consequently, we can focus only on the output port controllers. In the rest of this paper, we assume that all the deadlines of connections have been modified as described above.

The scheduling policy at the output port of the ATM switch determines the order in which cells from connections that use the outgoing link are transmitted. Hence, the scheduling policy has a direct impact on the delays experienced by the cells of a connection as they are buffered at the output port. As a result, the traffic of a connection typically becomes more *bursty*. This increase in burstinesss may in turn affect the delay experienced by cells of other connections, and thus, perturb other traffic flows in the network.

In this paper, we consider ATM switches with static-priority schedulers at their output ports. With static-priority scheduling, every connection is assigned a priority at each server it traverses. We call the priority assignment *static* if it is time-independent. A priority assignment is said to be *fixed* if the priority assigned to the connection is the same on all the servers it traverses.

3.2. Connection-Server Graphs

We use the above abstraction for connections and servers to construct a *connection-server graph*. A connection-server graph is constructed as a labeled, directed graph with the servers as its nodes. A directed edge is introduced from Server k to Server j if there is a connection that is served by Server k and Server j in sequence. The sources and destinations of connections are also part of the connection-server graph. The connection-server graph is used to facilitate the following discussion on network stability.

Servers and their output links. Servers in a connection-server graph are multiplexors with a single output link. The topology of a connection-server graph is therefore fully determined by two sets of variables: $P(j)$ specifies the set of all servers whose output traffic enters Server j, and $C(j)$ is the set of all connections that traverse Server j. We call the servers in $P(j)$ the *predecessor* servers of Server j.

Routes. The route of a connection is defined by the sequence of servers traversed by that connection. Let $s(i,j)$ denotes the identity of the j^{th} server in the route of Connection M_i. If S_i is the total number of servers serving Connection M_i, the *route* of M_i is the sequence G_i of servers serving that connection.

$$G_i = < s(i,1), s(i,2), \ldots, s(i,j), \ldots, s(i,S_i) > . \quad (1)$$

The *partial route* $G_{i,j}$ is the set of servers traversed by a cell of Connection M_i from the source *up to and including* Server j. In other words, $G_{i,j}$ contains all the servers used by M_i upstream from Server j, including Server j.

3.3. Meeting Deadlines

Main Issues. As we mentioned earlier, to effectively analyze the end-to-end cell delays in ATM networks, three important issues must be addressed.

1. *System Stability:* An ATM network is *stable* if and only if the end-to-end delays of each connection can be bounded. Stability cannot be taken for granted in networks with arbitrary topology. If the connection-server graph has cycles, there may be feedback effects in the queue lengths for servers on the cycle. Due to the presence of these feedback effects, the system may be potentially unstable, with connections having unbounded cell delays. Obviously, it is not a fruitful exercise to determine delay bounds in a potentially unstable system. Therefore, determination of system stability is a critical step in the computation of communication delay bounds.

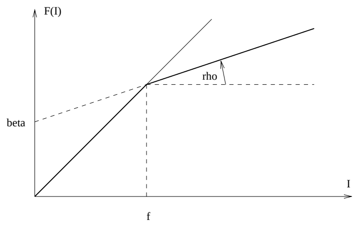


Figure 2. Linear Traffic Bounding Function F with Parameters β and ρ.

2. *End-to-end cell delay bounds:* In a network that is shown to be stable the end-to-end delay of a connection is the sum of the worst-case local delays at each server along its route. Therefore, in order to derive the end-to-end delay of a connection we derive the worst-case local delay, that is, the upper bound on the delay a cell of the connection experiences at a particular server, for each of the servers on its route.
3. *Priority assignment:* The priorities assigned to connections on the servers control their local delay bounds. The assignment should be sensitive to the deadline requirements of the connections in order to maximize the chance that all the deadline requirements of the connections can be met, and the connections can be admitted.

Traffic Characterization. In order to allow for an analytical delay calculation, the traffic is characterized in form of a *traffic bounding function*, which defines the maximum number of bits that can arrive as function of the time interval. This bounding function $F_{i,j}(I)$ specifies the maximum number of cells that can arrive at Server j for Connection M_i during any interval of length I. In the following discussion, we will assume that the traffic entering the network is bounded by a linear traffic function [4] of the form

$$F(I) = \min(I, \beta + \rho * I), \tag{2}$$

which models a traffic that is shaped by a token bucket of size β and rate ρ, followed by a leaky bucket with the rate equal to the link speed, which we normalize to one. In the following we will call a traffic that is characterized by Equation (2) to be *bound by a* (β, ρ) *function*. In particular, we assume the traffic of Connection M_i at its entrance $s(i, 1)$ to the network to be bound by a (β, ρ) function with parameters β_i and ρ_i.

As illustrated in Figure 2, the linear bounding function consists of two linear segments, one with a rate of one, and one with rate ρ. Following the terminology in [9], we call the intersection point of the two segments the *flex* point, which is denoted as f in Figure 2.

In order to quantify the effect connections have on each other, we want to cluster the connections sharing a particular link that are assigned the same priority, and represent them by a single traffic bounding function. For this, we define $\mathcal{F}_{p,j}(I)$ to be the aggregated traffic of connections with priority p on the output link of Server j. That is, $\mathcal{F}_{p,j}(I)$ is the maximum number of cells of connections with priority p that can leave Server j during any interval of length I. Similarly, we let $\mathcal{J}_{p,j}$ be the aggregated cell traffic of connections with priority *higher than or equal to* p on the output link of Server j. That is, $\mathcal{J}_{p,j}(I)$ is the maximum number of cells of connections with priority higher than or equal to p that can leave Server j during any interval of length I. These traffic bounding functions will be useful in the following delay analysis.

Worst-Case Delays. Let d_i be the *worst-case end-to-end delay* experienced by Connection M_i. We define $d_{p,j}$ to be the worst-case delay experienced at Server j by a connection with priority p. If Connection M_i is assigned priority $\pi(i, j)$ at Server j, then $d_{\pi(i,j),j}$ is the *worst case local delay* of Connection M_i at Server j, and the end-to-end delay for M_i can be formulated as the sum of the worst-case local delays on its route:

$$d_i = \sum_{j=1}^{S_i} d_{\pi(i,s(i,j)),s(i,j)}. \tag{3}$$

Assuming a system with K servers and P priorities per server, in the following discussion we will use the KP-dimensional vector $\vec{d}$ of delays at all priorities at all the servers in the system, i.e.,

$$\vec{d} = (d_{1,1}, d_{2,1}, \cdots, d_{P,1}, \cdots, d_{1,K}, \cdots, d_{P,K})^\top. \tag{4}$$

4. Stability and Delay Analysis

In this section, we first derive the expression for the local delays, based on which we formulate a simple criteria for stability. We then proceed to develop an iterative method for computing the local delays. Once the local delays have been calculated, the end-to-end worst-case delay of each connection can be obtained by adding up the local delays, as illustrated in Equation (3).

4.1. Traffic Bounding Functions

We argued earlier that knowing the traffic bounding functions of connections allows us to analytically compute delay bounds. Unfortunately, the shape of the traffic changes as the latter is routed through the servers. In order to determine delay bounds at servers inside the network, the effect servers have on the traffic shape must be known. The following theorem allows to determine the traffic bounding function $F_{i,j}(I)$ for Connection M_i at the input of Server j, when the source traffic of M_i is bound by a (β, ρ) function as defined in Equation (2).

Theorem 1 *For any connection M_i whose traffic at its entrance to the network is bound by a (β, ρ) traffic constraint function with parameters β_i and ρ_i, the traffic of M_i at the output of any server j on its route is bound by a (β, ρ) function with parameters $\bar{\beta}$ and $\bar{\rho}$, where*

$$\bar{\beta} = \beta_i + \rho_i \sum_{g \in G_{i,j}} d_{\pi(i,g),g} \tag{5}$$

$$\bar{\rho} = \rho_i . \tag{6}$$

According to Theorem 1, as the traffic moves along its route, it remains bounded by a (β, ρ) function. The rate ρ remains constant along the route, but the burstiness β increases as a function of the accumulated worst-case delay on the route. This theorem generalizes earlier results by Cruz [4] and Raha *et al* [19], where the traffic characteristic function at the output of an FCFS server was obtained in terms of that at the immediate input to the server [4] or at the source [19]. In Theorem 1 we extend these results to servers with static-priority scheduling. As we will see, Theorem 1 facilitates the efficient computation of worst-case delays in networks with static-priority scheduling.

Cells from different connections in an ATM network may be multiplexed at the multiplexor of a switch and transmitted over its output link. In order to determine cell delays, we must characterize the aggregate cell traffic over a single link. The following theorem describes the aggregate traffic for an arbitrary set of connections over a particular link.

Theorem 2 *Let $\mathcal{M}^*$ be any subset of the connections that traverse Server j. Its aggregate traffic on the output link of Server j is bounded by a (β, ρ) function with parameters $\bar{\beta}$ and $\bar{\rho}$, where*

$$\bar{\beta} = \sum_{i \in \mathcal{M}^*} \left(\beta_i + \rho_i \sum_{g \in G_{i,j}} d_{\pi(i,g),g} \right) \quad (7)$$

$$\bar{\rho} = \sum_{i \in \mathcal{M}^*} \rho_i . \quad (8)$$

It follows that every aggregate traffic inside the network is bound by a (β, ρ) function, and we can determine the parameters by applying Equation (7). In particular, we can use Equation (7) to determine the traffic bounding functions $\mathcal{F}_{p,j}$ and $\mathcal{J}_{p,j}$ for the aggregate traffic with priority equal or higher than the given priority p, respectively, at the output link of Server j. This is of use in analyzing the delays at the servers in the network, as we describe in the following section.

4.2. Expressions for Local Delays

Once the traffic bounding functions of both the traffic entering the network and the traffic inside the network are known, the local delays for every connection at each switch can be determined. Formula (9) indicates how long a newly arriving cell with priority p can be delayed at a given switch j in a stable network. As defined earlier, $P(j)$ denotes the set of predecessor servers to Server j.

$$d_{p,j} = \max_{0 < t \leq T_{p,j}} \Big(\sum_{k \in P(j)} \mathcal{J}_{p-1,k}(t + d_{p,j}) + \sum_{k \in P(j)} \mathcal{F}_{p,k}(t) - t \Big) + 1 . \quad (9)$$

In this formula, $T_{p,j}$ denotes the *maximum busy interval* for all connections with priority equal to or higher than p on Server j. In other words, Server S_j never processes cells with priority equal to or higher than p for more than $T_{p,j}$ consecutive time units. Formula (9) describes the maximum delay of a cell as the time the cell is delayed by *higher-priority* cells, which arrived before or while the cell is queued, and *same-priority* cells, which were there before the arrival of the new cell. If no higher-priority connections "join" a set of connections at a server, then the delays at that server are zero; the traffic simply flow through the server. This is illustrated by the following lemma.

Lemma 1 *If all the traffic with priority higher than or equal to p at Server j comes from only one previous server then $d_{p,j} = 0$.*

Lemma 1 holds because the cells have been ordered at the output link of the previous server. Server j now simply forwards the cells. Given the shape of the traffic bounding functions, we can formulate the maximum busy interval $T_{p,j}$ using the following closed form.

Theorem 3 *The maximum busy interval $T_{p,j}$ at Server j for all connections assigned priority higher than or equal to p is given by*

$$T_{p,j} = \min_{t>0} \{t | \sum_{k \in P(j)} \mathcal{J}_{p,k}(t) - t < 0\}$$

$$= \frac{\sum_{i \in C(j), \pi(i,j) \leq p} [\beta_i + \rho_i * \sum_{g \in G_{i,j}, g \neq j} d_{\pi(i,g),g}]}{1 - \sum_{i \in C(j), \pi(i,j) \leq p} \rho_i} . \quad (10)$$

The summations in Equation (10) go over all connections that traverse Server j and are assigned a priority higher or equal to p. We note that the term $d_{p,j}$ occurs on both sides of Equation (9). This means that the local delays cannot be determined directly, but are solutions to the system of equations defined by Equation (9). Two key problems must be addressed: First, we need to know the condition under which the system is stable. If the system is not stable, no solutions to Equation (9) exist. Second, assuming the system is stable, we need to solve Equation (9) efficiently. Solving the equation directly is very expensive because of the "max" operation on its right-hand side. The following theorem provides a means to effectively determine whether a system is stable. We use the term χ to indicate whether the delay for a connection on a given server has an effect on the delay for that connection on servers downstream. For this purpose, we define $\chi_{i,q,s,p,j}$ to be 1 if Server s is upstream from Server j on the route of Connection M_i, and the connection is assigned priority q and p on Server s and j, respectively. The value for $\chi_{i,q,s,p,j}$ is zero otherwise. For each server j we define a server j^* among the predecessor servers of Server j, whose output traffic flowing through Server j has the largest flex point. We say that Server j^* is *critical* for Server j.

Theorem 4 *The worst case delay $d_{p,j}$ at Server j for cells of connections with priority p is*

$$d_{p,j} = \Pi_{p,j} + \sum_{s=1}^{K} \sum_{q=1}^{P} C_{q,s,p,j} d_{q,s} , \quad (11)$$

where K is the number of servers in the system, and $\Pi_{p,j}$ and $C_{q,s,p,j}$ are defined as follows:

$$\Pi_{p,j} = \frac{\sum_{i \in C(j), \pi(i,j) \le p} \beta_i + 1}{1 - \sum_{i \in C(j), \pi(i,j) < p} \rho_i}$$
$$+ \frac{\sum_{i \in C(j), \pi(i,j) \le p} \rho_i - 1}{1 - \sum_{i \in C(j), \pi(i,j) < p} \rho_i} * \frac{\sum_{i \in C(j^*), \pi(i,j^*) = p} \beta_i}{1 - \sum_{i \in C(j^*), \pi(i,j^*) = p} \rho_i}$$

where $j^* \in P(j)$ is a critical server for Server j, and

$$C_{q,s,p,j} = \frac{\sum_{i \in C(j), \pi(i,j) \le p} \rho_i * \chi_{i,q,s,p,j}}{1 - \sum_{i \in C(j), \pi(i,j) < p} \rho_i} +$$
$$\frac{\sum_{i \in C(j), \pi(i,j) \le p} \rho_i - 1}{1 - \sum_{i \in C(j), \pi(i,j) < p} \rho_i} * \frac{\sum_{i \in C(j^*), \pi(i,j^*) = p} \rho_i * \chi_{i,q,s,p,j}}{1 - \sum_{i \in C(j^*), \pi(i,j^*) = p} \rho_i}.$$

The equations in Theorem 4 can be written as a system of equations as follows:

$$\vec{d} \quad = \quad \vec{\Pi} + \mathcal{C} \cdot \vec{d}\,, \tag{12}$$

where $\vec{\Pi} = (\Pi_{1,1}, \Pi_{2,1}, \cdots, \Pi_{P,1}, \cdots, \Pi_{P,K})^\top$, and $\mathcal{C}$ is given as following

$$\begin{bmatrix}
0 & C_{2,1,1,1} & \cdots & C_{P,1,1,1} & \cdots & C_{P,K,1,1} \\
C_{1,1,2,1} & 0 & \cdots & C_{P,1,2,1} & \cdots & C_{P,K,2,1} \\
\vdots & \ddots & \ddots & & \vdots & \\
 & & \ddots & 0 & \ddots & \vdots \\
\vdots & & & \ddots & \ddots & C_{P,K,P-1,K} \\
C_{1,1,P,K} & C_{2,1,P,K} & \cdots & C_{P,1,P,K} & \cdots & 0
\end{bmatrix}$$

We simplify the notation for the system of equations (12) by denoting

$$z_{p,j}(\vec{d}) \quad = \quad \Pi_{p,j} + \sum_{s=1}^{K} \sum_{q=1}^{P} C_{q,s,p,j} * d_{q,s} \tag{13}$$

and $\vec{Z}(\vec{d}) = [z_{1,1}(\vec{d}), \cdots, z_{P,K}(\vec{d})]^\top$, and

$$\vec{d} \quad = \quad \vec{Z}(\vec{d}) \ . \tag{14}$$

Although Equation (14) may at first sight look linear, it is not. The values of $\Pi_{p,j}$ and $C_{q,s,p,j}$ depend on the choice of Server j^* as critical server, and so indirectly on the delay d_{i,j^*} on that server.

If we define $C^k_{q,s,p,j}$ to be the value for $C_{q,s,p,j}$, if Server k in $P(j)$ were to be the critical server for Server j, then the upper bound $\tilde{C}_{q,s,p,j}$ on $C_{q,s,p,j}$ can be defined as

$$\tilde{C}_{q,s,p,j} = max_{k \in P(j)} \{C^k_{q,s,p,j}\} \tag{15}$$

and, similarly,

$$\tilde{\Pi}_{p,j} = max_{k \in P(j)} \{\Pi^k_{p,j}\}. \tag{16}$$

4.3. Stability Criteria and Delay Computation

The following theorem then formulates a criteria for the stability of the system.

Theorem 5 *For a given priority assignment, if*

$$\nu = \max_{p,j} (\sum_{s=1}^{K} \sum_{q=1}^{P} \tilde{C}_{q,s,p,j}) < 1 \tag{17}$$

then the system is stable, where $\tilde{C}_{q,s,p,j}$ is defined in (15).

Results similar to Theorem 4 and Theorem 5 were obtained earlier by Cruz [4] and Li *et al* [14] for systems with FCFS schedulers. As FCFS scheduling is a special case of priority scheduling, our result here generalizes the earlier work. In addition, Theorem 5 gives a concise criterium for stability. Furthermore, Theorem 4 provides the basis for efficiently finding the worst case delays.

Equation (14) can be solved by using a simple iterative procedure as follows: Let $\vec{d}^{[0]}$ represent the KP-dimensional delay vector at the beginning of the first iteration, and let $\vec{d}^{[n]}$ the same vector at the end of the n^{th} iteration. Before the first iteration, vector $\vec{d}^{[0]}$ is initialized to be

$$\vec{d}^{[0]} := (1, \ 1, \ \ldots, \ 1)^\top \ . \tag{18}$$

During the n^{th} iteration, the new value for $\vec{d}$ is computed as follows:

$$\vec{d}^{[n]} := \vec{Z}(\vec{d}^{[n-1]}) \ . \tag{19}$$

In order to demonstrate the convergence of this iterative procedure, we need to determine the error between $\vec{d}$ and $\vec{d}^{[n]}$, the vector at the end of the n^{th} iteration. That is, we need to establish the difference between the value of $d_{p,j}$ computed at the end of the n^{th} iteration and the real value of $d_{p,j}$. For the iteration procedure to converge, this difference must become zero for large values of n. The following theorem gives an estimation of $\vec{d}$ at the end of the n^{th} iteration. We use the notation $\|\cdot\|$ to denote the maximum norm.

Theorem 6 *For a stable system, if the iterative procedure defined by (18) and (19) is used to solve (14), then at the end of the n^{th} iteration the following holds:*

$$\|\vec{d} - \vec{d}^{[n]}\| \le \frac{(\nu)^n}{1-\nu} * \|\vec{d}^{[1]} - \vec{d}^{[0]}\| \ , \tag{20}$$

where ν was defined in Equation (17).

The convergence of the iterative procedure follows as a corollary, given that we showed earlier (Theorem 5) that in a stable system $\nu < 1$ holds, and therefore

$$\lim_{n \to \infty} \frac{\nu^n}{1-\nu} = 0. \tag{21}$$

As the value for n increases, the right-hand side of Equation (20) tends to go to zero. Hence, the iterative procedure converges.

As a result we have an effective scheme that - given a set of connections and their routes in a network with arbitrary topology and static-priority scheduling on the links - determines (1) the stability of the system, and (2) the local delays of connections at the switches. This scheme assumes (β, ρ) traffic bounding functions at the entrance to the network and does not rely on traffic regulation inside the network.

5. Priority Assignment

Given static-priority schedulers in the servers, the probability that a set of connections can be established depends on the way the prioritites on the servers are assigned to connections. Unfortunately, the following theorem indicates that it is very unlikely that an efficient optimal priority assignment algorithm can be found.

Theorem 7 *Given a general-topology connection-server graph and a set of connections $\mathcal{M}$, the problem of finding a priority assignment $\pi(i, j)$ of connections to servers so that every connection M_i meets its end-to-end delay D_i is $\mathcal{NP}$-complete.*

Given that an efficient optimal assignment algorithm is unlikely to exist, we compare a number of heuristics, which assign priorities with consideration for either the deadlines of connections, or the topology of the underlying network, or both.

5.1. The Algorithms

The trivial approach simply assigns the same priority to all connections on all servers. The scheduling policy on all servers then degenerates to FCFS scheduling. FCFS does not take into account deadline information, and therefore the performance can be expected to be poor. In the following evaluation we will be using FCFS as a baseline.

5.1.1 Deadline Based Heuristics

We can expect that the performance improves if the priority assignment reflects the message urgency. Intuitively, the smaller the relative deadline of a connection is, the higher its priority should be. The *relative deadline monotonic* (RDM) algorithm [1, 13] assigns priorities in this order. For single-server systems and periodic workload, RDM is known to be an optimal static-priority assignment algorithm. Interestingly, our evaluations show that RDM does not perform well when connections traverse multiple servers. In some cases it even underperforms FCFS! This effect is particularly strong when deadline variations are small, and deadlines do not provide a sufficient decision basis for fixed-priority assignments.

The two algorithms FCFS and RDM can be combined into a simple scheme that starts with an FCFS assignment and successively modifies priorities of connections to take message urgency into consideration. This leads to Algorithm *Partition*, which is described in Figure 3. This algorithm repeatedly partitions the connection set into an increasing number of subsets in accordance with message laxities. It then assigns the different priorities to the connections in the different subsets.

Algorithm *Partition:*

- **Step 1:** Assign the same priority to all connections. All connections are initially in the same subset.
- **Step 2:** Compute the delays for each connection using the method described in Section 4.2. If all connections pass the deadline test, stop; return the current priority assignment.
- **Step 3:** For each subset in which a connection fails to pass the deadline test, perform the following steps:
  Step 3.1: If the subset consists of a single element, stop; the algorithm was not able to find a feasible priority assignment.
  Step 3.2: Partition the subset of connections into two subsets and assign connections to the subsets in increasing order of their laxity (defined to be the difference between the deadline and the computed delay).
- **Step 4:** For Server j, assign priority p to a connection if the server is on the path of the connection, and the connection is in the pth subset.
- **Step 5:** Return to Step 2.

Figure 3. Algorithm *Partition* for Assigning Priorities to Connections

The iteration stops when all connections pass the deadline test, and the whole set of connections is admissible, or when a subset with only one connection needs to be further partitioned because the connection does not meet the deadline. In that case, no more partitions can be done, and the algorithm declares failure. Because the size of the smallest subset of connections is halved at every iteration step, the worst-case cost of the algorithm is order $O(\lg n)$ in the number of delay computations.

In its basic form, Algorithm *Partition* compares relative deadlines for deciding how to partition the connection set into urgent and non-urgent connections. Connections that traverse a larger number of servers tend to experience more delay, which is not considered when the algorithm simply compares relative deadlines. In the evaluations described below we therefore make the decisions how to partition the connection set by using the *modified relative deadline* D'_i, which is defined as the relative deadline D_i of connection M_i divided by the number of servers traversed by M_i: $D'_i := D_i/S_i$. In this way the length of connections is accounted for when making priority assignments.

5.1.2 Topology Based Heuristics

The priority assignment algorithms described above share the following two characteristics: (1) Each connection is assigned the same priority on all the servers on its route, and (2) the priority assignment is independent of the topology of the network or the load of individual servers. Better results should be expected when priorities are allowed to vary between servers, and are assigned with regard to the underlying topology.

A very simple priority assignment that takes into account the network topology was described by Cruz [4] for the case of a ring. Cruz proposed a two-priority scheme, in which connections are assigned a low priority on the first server when they join the network. On all the other servers, connections are assigned a high priority. In other words, cells already in the ring have higher priority than cells that join the ring. Cruz argued that assigning priorities in this way leads to less disturbance of traffic and hence improves delay bounds. We call this method the "Cruz Algorithm".

We now consider an integrated algorithm, which not only uses the location and topology information, but also the timing information (i.e., laxities) to assign priorities. It can be considered as an integration of Cruz' algorithm and Algorithm *Partition*. Figure 4 describes this approach, which we call Algorithm *Integrated*.

```
Algorithm Integrated:
  • Step 1: Assign the same priority to all connections. All connections are initially in the same subset.
  • Step 2: Compute the delays for each connection using the method described in Section 4.2. If all connections pass the deadline test, stop; return the current priority assignment.
  • Step 3: For each subset in which a connection fails to pass the deadline test, perform the following steps:
        Step 3.1: If the subset consists of a single element, stop; the algorithm was not able to find a feasible priority assignment.
        Step 3.2: Partition the subset of connections into two subsets and assign connections to the subsets in increasing order of their laxity (defined to be the difference between the deadline and the computed delay).
  • Step 4: For Server $j$, increase the priority by one for the connections that are not in the first (highest-priority) subset and join the network at Server $j$.
  • Step 5: Compute the delays for each connection using the method described in Section 4.2. If all connections pass the deadline test, stop; return the current priority assignment.
  • Step 6: For Server $j$, assign priority $p$ to a connection if the server is on the path of the connection, and the connection is in the $p$th subset.
  • Step 7: Return to Step 2.
```

Figure 4. Algorithm *Integrated* for Assigning Priorities to Connections

Except for Step 4 and Step 5, Algorithm *Integrated* is identical to Algorithm *Partition*. After repartitioning the connections in Step 3, but before re-assigning the priorities in Step 6, Algorithm *Integrated* slightly lowers the priorities of connections at their entrance to the network, essentially following the idea of Cruz' algorithm. This is done by assigning a lower priority ($p + 1$) to a connection in the pth subset at Server j if the latter is the server at the entrance to the network. If this assignment is not successful, Algorithm *Integrated* resorts to the priority assignment of Algorithm *Partition* by assigning priority p to every connection in the pth subset.

5.2. Performance Evaluation

In this subsection, we evaluate the performance of the five priority assignment algorithms discussed in the previous subsection. We will first define a performance metric, then describe the system configuration considered and present the performance results.

Performance Metric. We quantify the performance of an algorithm by measuring the *Admission Probability* ($AP(U)$) for a given link utilization U, that is, the probability that a set of connections are admissible conditioned on the average utilization of the links in the network being U.

Topology and Traffic Load. We consider ATM networks with a specialized ring topology. This topology has been used as an representative benchmark by Cruz, Gallager, and Parekh to study the problem of delay stability in ATM networks for FCFS servers [4, 17]. Henceforth, we shall refer to this topology as the *Cruz-Gallager-Parekh* (C-G-P) ring.

The architecture of the C-G-P ring is described as follows. The system consists of K 2×2 switches and K connections, $M_1, \ldots, M_i, \ldots, M_K$. Each server has a distinct identity id, where $id = 1, 2, \ldots, 2{*}K$ and every connection has an acyclic path that traverses K servers. For connection M_i, the first server, $s(i, 1)$, is Server i and the following servers are

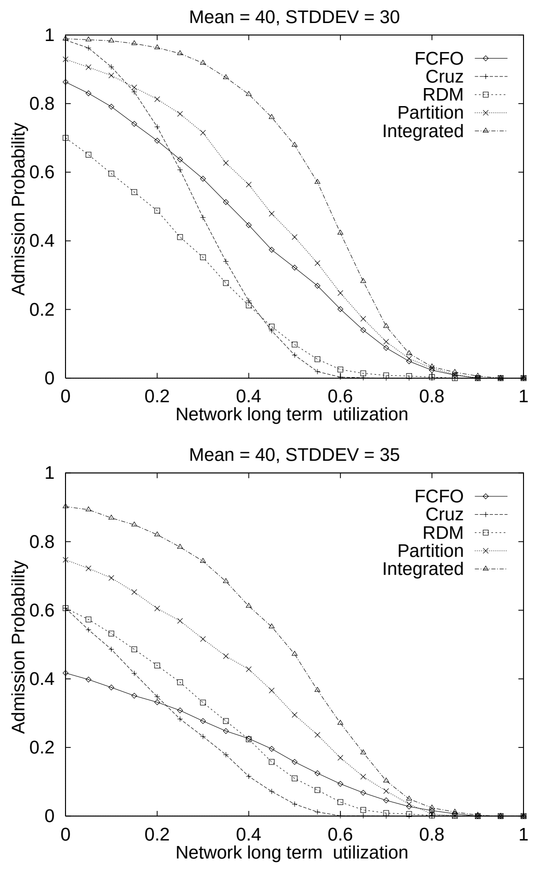
$$s(i,j) = \begin{cases} 1 + (i + j - 2) \bmod K & 1 \le j \le K - 1 \\ K + i & j = K. \end{cases}$$

Figure 5 shows an example of a C-G-P ring, which is the connection server graph of the ring with four switches depicted in Figure 1. The source traffic for each connection in the C-G-P ring is constrained by a (β, ρ) traffic bounding function as defined in Equation (2).

Simulation Methodology. We measured the performance by simulating the behavior of the five algorithms with randomly generated connection sets. For each data point $1,000$ connection sets were randomly generated, and each was was tested for admission. For each connection, β_i and ρ_i were chosen from uniform distributions, and the relative deadlines of connections where chosen from a general exponential distribution. Statistics were collected from the sample set to estimate the admission probability defined earlier. For all measurements, the 99-percentile confidence intervals are below 1% of the admission probability range. Similar results have been obtained with different network topologies and settings of parameters.

5.2.1 Numerical Results and Observations

Figure 6 shows the admission probability results for our example network. The performance figures are corresponding



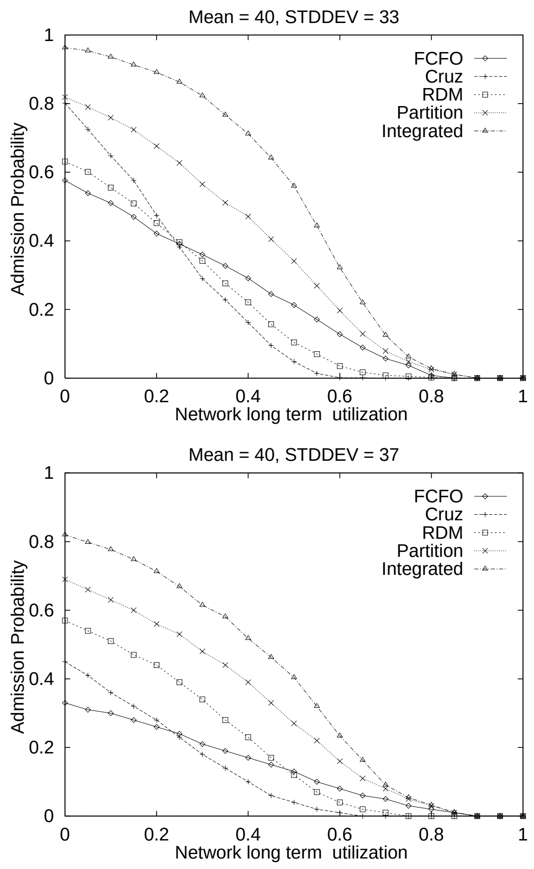
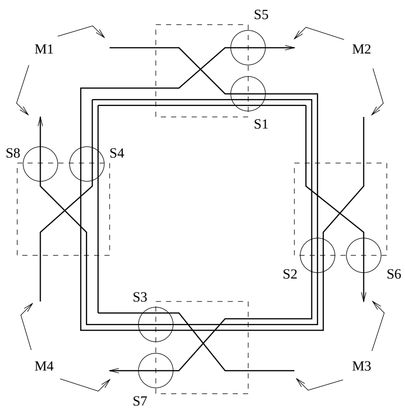


Figure 6. Admission Probability vs. Long-Term Utilization



	$s(i,1)$	$s(i,2)$	$s(i,3)$	$s(i,4)$
M_1	1	2	3	5
M_2	2	3	4	6
M_3	3	4	1	7
M_4	4	1	2	8

Figure 5. Cruz-Gallager-Parekh Ring with 4 Switches

to the case when the average relative deadline is 40, and the standard deviation of deadlines (STD(D)) ranges from 30 to 37. From these figures, we can make the following observations:

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

Second, in all the cases tested, it is seen that Algorithm *Partition* performs far better than both the FCFS and the RDM methods. This comes from the fact that, whenever a feasible assignment can be found by either FCFS or RDM, Algorithm *Partition* finds it as well. Sometimes, the improvement is significant. For example, when link utilization is 0.4 and STD(D) = 33 (Figure 6), either FCFS or RDM admitted no more than 30% of connection sets while our new algorithm can admit around 50% of connection sets.

Furthermore, we observe that the integrated method outperforms all the other four methods, namely FCFS, RDM, Cruz, and Algorithm *Partition*. For example, when the utilization is 50%, the integrated method admits up to 75% more message sets than the partition method, and up to seven times more than RDM. While the results are encouraging, they are not surprising. Recall from the design of this algorithm that it inherently considers all the possible priority assignments that would be examined by any of the other four methods while maintaining the same order of complexity as the partition method. This demonstrates that by properly integrating the timing and topology information, performance can indeed be improved without introducing much overhead.

6. Final Remarks

We have studied ATM networks with static priority scheduling. We developed a condition under which the network is guaranteed to be stable. In an unstable network, the worst case delays could be unbounded. This is not acceptable for many time-constrained applications. We also developed an iterative method for computing worst case end-to-end delays. The convergence of the numerical procedure is formally proved and a closed form for the computation error is derived.

We also addressed the problem of how to assign priorities to connections in such a system. In particular, we analyzed five algorithms: FCFS, relative deadline monotonic (RDM), and Cruz' Algorithm, in addition two combinations thereof, namely Algorithm *Partition* and Algorithm *Integrated*. The first three have been proposed in previous studies, while the latter two are new. Performance evaluations show that the two new algorithms outperform the other three. Sometimes, the performance difference is significant.

This work can be extended in a number of ways. For example, we are currently studying stability issues, delay computation, and priority assignment in connection-based *heterogeneous* networks. To establish the criteria for stability in such networks it will be necessary to investigate characterizations of the traffic within the network. Utilizing a consistent traffic characterization function over a series of network segments is a key step in this process.

Acknowledgment

This work was partially sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-96-1-1076 and by Texas Higher Education Coordinating Board under its Advanced Technology Program with grant number 999903-204. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the U.S. Government, Texas State Government , Texas Higher Education Coordinating Board, or Texas A&M University.

References

[1] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: the deadline-monotonic approach. In *Proc. of the Eighth RTOSS* , May 1991.

[2] P. Boyer, F. Guillemin, M. Servel, and J. Coudreuse. Spacing cells protects and enhances utilization of ATM network links. *IEEE Network*, Sept. 1992.

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

[4] R. L. Cruz. A calculus for network delay, partI, part II. *IEEE Trans. on Information Theory*, Jan. 1991.

[5] A. Dailianas and A. Bovopoulis. Real-time admission control algorithms with delay and loss guarantees in ATM networks. In *Proc. of INFOCOM'94*.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proc. of ACM SIGCOMM'89*, Sept. 1989.

[7] N. Dunford and J. Schwartz. Linear operators, Part I : General theory. In *Interscience publishers*. New York (1958).

[8] D. Ferrari and D. C. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE J. on Selected Areas in Comm.*, Apr. 1990.

[9] V. Firoiu, J. Kurose, and D. Towsley. Efficient Admission Control for EDF Schedulers. In *Proc. of the IEEE INFORCOM'97*.

[10] L. Georgiadis, R. Guérin, V. Peris, and K. Sivarajan. Efficient network QoS provisioning based on per node traffic shaping. *IEEE ACM Trans. on Networking*, Aug. 1996.

[11] S. J. Golestani. A framing strategy for congestion management. *IEEE J. on Selected Areas in Comm.*, Sept. 1991.

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

[13] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, December 1982.

[14] C. Li, A. Raha, and W. Zhao. Stability in ATM networks. In *Proc. of the IEEE INFORCOM'97*.

[15] J. Liebeherr, D.E. Wrege, and D. Ferrari. *"Exact admission control in networks with bounded delay services."*, to appear in *IEEE/ACM Trans. on Networking*.

[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. of the Association for Computing Machinery*, Jan. 1973.

[17] A. K. J. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*. PhD thesis, M.I.T., 1992.

[18] A. Raha, S. Kamat, and W. Zhao. Guaranteeing end-to-end deadlines in ATM networks. In *Proc of the 15th IEEE ICDCS'95*.

[19] A. Raha, S. Kamat, and W. Zhao. Admission control for hard real-time connections in ATM LAN's. In *Proc. of the IEEE INFORCOM'96*.

[20] A. Raha and W. Zhao. Evaluation of admission policies in ATM based embedded hard real-time systems. Technical report, Department of Computer Science, Texas A&M University, June 1994.

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

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

[23] A. M. van Tilborg and G. M. Koob. *Foundations of Real-Time Computing: Formal Specifications and Methods*. Kluwer Adademic Publishers, 1991.

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

[25] H. Zhang and D. Ferrari. Rate-controlled static priority queueing. In *Proc. of IEEE INFORCOM'93*.