Rate Controlled Servers for Very High-Speed Networks

C. R. Kalmanek and H. Kanakia
AT&T Bell Laboratories
Murray Hill, NJ 07974

S. Keshav
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

Future high-speed networks are expected to carry traffic with a wide range of performance requirements. We describe two queue service disciplines, rate-based scheduling and hierarchical round robin scheduling, that allow some connections to receive guaranteed rate and jitter performance, while others receive best effort service. Rate-based scheduling is designed for fast packet networks, while hierarchical round robin is an extension of round robin scheduling suitable for use in networks based on the Asynchronous Transfer Mode (ATM) being defined in CCITT. Both schemes are feasible at rates of one Gigabit/sec. The schemes allow strict bounds on the buffer space required for rate controlled connections and can provide efficient utilization of transmission bandwidth.

Introduction

Future high-speed networks are expected to carry traffic with a wide range of performance requirements. A classic tradeoff in network design is between providing quality of service guarantees on one hand, and achieving efficient utilization of the transmission bandwidth on the other [TYM]. Synchronous circuit switching can provide guaranteed bandwidth and bounds on jitter at the expense of underutilizing bandwidth, whereas datagram networks can optimally utilize bandwidth but cannot provide strong performance guarantees.

In integrated networks, the design problem is more difficult because different classes of traffic have dramatically different performance requirements, expressed in terms of the desired bandwidth, latency, and *jitter.* [1] Real-time video requires high bandwidth, low latency and low jitter. Voice traffic requires low bandwidth, low latency and low jitter. Computer traffic spans a wide range of requirements, from applications requiring low latency for small transfers to those which are quite insensitive to the bandwidth, latency, and jitter achieved. In addition, the operating range of future networks will be very large: we expect trunk speeds from 64 Kb/sec to over 1 Gb/sec. The burden on the network designer is that where there is contention for limited transmission bandwidth or buffer space, control mechanisms must ensure that each traffic class meets its performance objectives.

Rate Control

We propose to separate the virtual circuits or connections provided by the network into two categories: rate controlled connections and best effort connections. In the literature, the use of the term "rate control" is not standardized. We use the term "rate control" to mean controlling both the rate and jitter of a connection. At call setup time, a user requesting a rate controlled connection specifies a desired average service rate and jitter bound. If the call is accepted, the network provides guaranteed bandwidth and jitter bounds, and there may be mechanisms for renegotiation if the network cannot provide a suitable connection. Best effort connections, on the other hand, do not imply (strong) performance guarantees.

As in [ZHA], the rate of a connection is defined over an averaging interval: the rate is determined by dividing the total number of bytes transmitted during an averaging interval by the averaging interval. The value of the averaging interval will determine the allowed burstiness at the source. If the traffic source uses a large averaging interval, the network must have sufficient buffer space at the first node to accomodate a burst.

Our notion of rate control also specifies a bound on the jitter introduced by a server. We first present an intuitive definition of jitter in order to contrast it with the definition that we will use. A server is assumed to introduce delay as packets pass through it: the delay is modeled as having a constant component and a variable component. The jitter introduced by a server is computed by measuring the packet departure time from the server and subtracting the constant component of the delay from the departure time for each packet to give the normalized departure time. The jitter for a connection is the maximum difference over all packets between the normalized departure time and the arrival time at the server. This definition is useful in analytic modeling of the server.

---

[1] Defined below.

We propose an alternative definition of jitter that we believe is more relevant than the above definition in systems design. Jitter is defined to be a short term average rate, where the averaging interval used in computing the jitter is different from the averaging interval used in computing the rate of a connection. We refer to the two averaging intervals as the *rate averaging interval*, and the *jitter averaging interval*. Specifically, the jitter is the maximum number of packets transmitted by the server in the jitter averaging interval. This definition captures a useful property of our rate controlled servers, namely that they *smooth* the output stream. In principle, either averaging interval can range from the time to send one packet to the total connection time. In practice, the rate averaging interval might be on the order of several round trip times. The jitter averaging interval would be smaller and would depend on the size of the receive buffer at the destination.

Two alternative measures of quality for real-time streams have been suggested in recent work [FER], [LL]. These two alternative measures are the source-to-destination delay bound for a packet and the variance in inter-packet gaps observed at a receiver. In transporting real-time video or voice, bounds on absolute delay for a packet are not sufficient unless the jitter is also controlled. The absolute delay time for a packet in transit seems important mainly for applications such as real-time feedback and control systems. The variance in inter-packet gaps would affect quality only when the receiver equipment does not have the intelligence and the memory required to smooth out the variation in inter-packet gaps. In an era of inexpensive microprocessors and low cost memory devices, it seems likely that receiver equipment would be designed with the capability to smooth interpacket packet gaps for a burst of packets. The real factor affecting the cost would be the length of the burst that has to be stored before decoding the data, and it is this burst length which is captured in our definition of jitter. The distinction between our definition of rate control and these alternative definitions is important since we have found that neither the bounds on the delay in a network nor the variance in interpacket gaps observed at a receiver comes cheaply, especially for networks without a synchronous time base. However, we contend that controlling the jitter and rate can be achieved at an acceptable cost.

Traffic Categories

In the paper, we use the following classification of sources of packet traffic. A source that injects packets into the network at fixed time intervals is referred to as a Continuous Bit Rate (CBR) source. Uncompressed video or voice sources are CBR sources of packets. Sources which use compression or adaptive techniques generate packets at variable bit rates. We refer to these types of sources as Variable Bit Rate (VBR) sources. A simple model for a VBR source has an "on-off" traffic pattern: the source sends some number of packets at its peak rate for a period of time, and is quiet for a period of time. We expect that CBR and VBR sources would use rate-controlled connections. Sources with large amounts of data to transfer, such as file transfer protocols, can simulate either a CBR source or VBR source by using rate-based admission control. It should be noted that use of rate control within a network is distinct from the admission control scheme used. A rate-controlled network could support either rate-based or window-based admission control policies.

A source that sends short messages at widely spaced intervals, is referred to as an intermittent source. Distributed computations use interprocess control messages which are typically small and hence are examples of intermittent sources. For an intermittent source, the measure of importance is low end-to-end delay and small probability of data loss. We expect that intermittent sources would use the best effort service.

Goals of Work

The goal of the rate control proposed in this paper is to guarantee the rate and provide an upper bound on the jitter for the rate-controlled connections, while also providing the lowest possible delay and loss probability for best-effort connections. An equally important goal for the network provider is the efficient use of network resources: memory and bandwidth. In this paper we propose two implementations of the proposed rate control scheme. The first implementation is focused on fast packet networks, and the scheduling of packets for service can be achieved by a microprocessor at very high speeds using an algorithm known as rate-based scheduling. The second implementation focuses on networks based on *Asynchronous Transfer Mode (ATM),* where messages are broken up and carried in small, fixed length *cells* that are switched and multiplexed in a uniform manner [CCITT]. This implementation, known as Hierarchical Round Robin Scheduling (HRR), yields the desired performance guarantees by an enhancement of ordinary round robin scheduling.

Previous Work

In a packet switched network, the main point of control is the queuing discipline used at a switch. Two simple schemes used in current wide-area networks have been widely studied: these are First-Come-First-Served (FCFS) policy and Round-Robin (RR) service among active connections. Neither of these schemes differentiates between the quality of service desired by different connections. Variations of FCFS and RR policies have been proposed to guarantee quality of service, either by use in conjunction with rate-based admission control [LL],[SLCG] or by including priorities, either static or dynamic, in each packet or implicit with each connection. One notable variation is the Earliest-Due-Date (EDD) scheme [FV], which marks each packet with a due date and schedules the packet whose due date is next to expire. Although EDD achieves lowest aggregate delay for a network, the per-packet processing may be prohibitive for high-speed networks. None of the schemes that we know of provides the bounds on jitter.

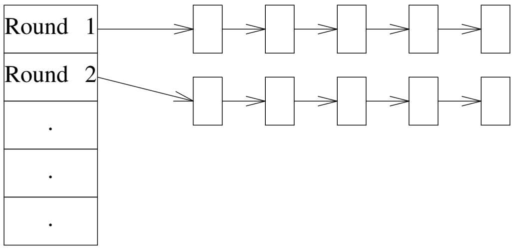
Various recent proposals have addressed the performance objectives for best effort connections. One approach (Jacobson [JAC] and Ramakrishnan-Jain [RJ]) assumes that sources voluntarily reduce their window size when they sense or are informed of network congestion. These schemes do not provide guarantees against data loss, and depend on the sources willingness to cooperate. In another approach [HKM], the source transmission window is adapted dynamically and the network provides guaranteed buffer space equal to the window size for each connection at each queueing point. Any of these schemes can be supported within the context of the service disciplines discussed in this paper, which multiplex rate controlled and best effort traffic at the server.

The service disciplines proposed in this paper meet our goals for rate control. Moreover, our experience in implementation suggests that they are not subsignificantly more costly in memory or CPU requirements than FCFS or RR scheduling. One of the implementations is based on a novel queue structure used to hold waiting packets. The other implementation shows how one can easily modify the basic RR server to perform rate control. These two proposed implementations are equivalent in the sense that they both meet the requirements for a rate-controlled server, and provide a packet transmission order which is similar.

Rate-Based Scheduling

This algorithm, first proposed by one of the authors [KAN], uses a tree-like structure to store packets awaiting their turn. The structure is shown in Figure 1.

We can think of the transmission on the output link as occurring in frames or *rounds* of fixed size. Packets are queued in each round as a singly-linked list, and there is an array which holds pointers to the packets for each round. One of the rounds is the current round from which packets are removed and transmitted. When the server reaches the end of the list, it moves on to the next round. Thus, the service procedure is quite simple. What remains to be discussed is how packets are queued in rounds and how this achieves rate control. The following algorithm is used to queue arriving packets.



Round-based Data Structure For Rate-Based Scheduling
Figure 1

For each connection, a switch keeps four parameters defining a rate and jitter for the connection. These parameters are the service quantum, the last round number, the current count and the interval. The quantum defines the number of packets belonging to this connection that can be sent per round. The last round number indicates the round to which the previous packet from this connection was added. The current count is the number of packets of this connection added so far to the last round. The interval number is an integer which is added to the last round number when the current count equals the quantum. This gives the new last round number, and is used to skip rounds when the rate of the connection is less than 1 packet/round. With these parameters, the algorithm to identify the round for a received packet takes only a few lines of code. We note that the order of packet transmission for a connection is kept the same as the order in which packets arrived.

In order to control jitter, the rounds are all kept of fixed size (bytes). The rate-based server will not insert a packet into an earlier round (a round with round number less than the last round number for the connection) even if there is an earlier round that is not full. This limitation ensures that the jitter is no more than the service quantum over an interval defined as interval number times the round transmission time. In order to handle best effort traffic with minimum delay, packets belonging to best effort sources are queued separately. Whenever a round under transmission runs short of packets, then best effort packets are served in round robin order until the fixed round size is filled. One can reserve some fraction of the bandwidth in each round for best effort packets by restricting the sum of the quanta of currently active connections to be below a predefined fraction of each round.

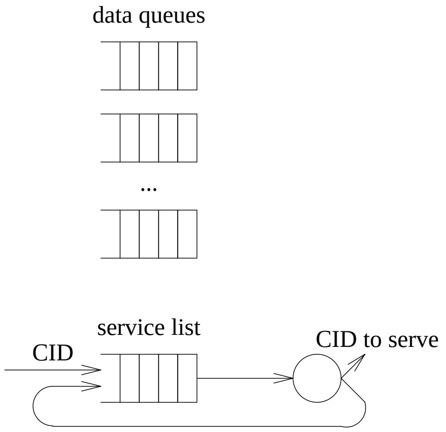
The algorithm described above controls rate and jitter over a wide range of desired rates. An implementation of this in software was measured to run at about 200,000 packets/sec with variable length packets, when run on a 25-MHz MC68020 Microprocessor.

Hierarchical Round Robin Scheduling

We describe below another implementation achieving equivalent order of transmission of packets over a link. This implementation was designed for use in ATM networks. We start by describing an implementation of an ordinary round robin server for fixed-size cells.

Round Robin Server

It is well known that fairness requires some sort of round robin queue service discipline as opposed to traditional first-come-first-served queueing [MOR,



A Round Robin Server Consists of a List of CIDs.
Figure 2

DKS]. Data from each connection is stored in per-connection data queues, so that each connection can be served separately. When a cell arrives, its data is stored in the data queue and its connection identifier (CID) is added to the tail of a *service list.* In order to ensure that each CID is entered on the list only once, there is a flag bit per connection which is set to indicate that the CID is on the service list (the *round robin flag bit*).

The server periodically takes a CID from the head of the service list and serves it for some number of *service quanta.* That is, there is a maximum number of cells that will be taken from the data queue for that CID and put onto the transmission line. The service quantum can be different for different CIDs. If the data queue goes empty, the flag bit is cleared and the server takes another CID from the head of the service list. If the data queue still has data in it, however, the server first returns the CID to the tail of the service list before going on. The situation is shown in Figure 2.

HRR Implementation

We will describe the implementation as a series of successive refinements on a basic idea. We then present the exact details of the algorithm in a form suitable for hardware implementation.

Again, we think of the transmission on the output link as occurring in frames of fixed size. A frame can be measured in time slots where each slot corresponds to one cell served on the output link. We modify the round robin server such that it starts service through its list of CIDs once per frame. For example, if the service quantum for each connection were one cell, the server would serve each CID on its service list at a rate of one cell each frame. Once a particular connection is served, its CID is returned to the end of the list, but we must insure that this connection is not served again until the beginning of the next frame. To do this, we implement each service list as two lists: a *current* list from which CIDs are being served, and a *next* list to which CIDs are added. At the beginning of the frame, we swap the current and next lists.

The HRR server has a hierarchy of service lists, each of which has a different frame length. The topmost list has the shortest frame length, and is used to serve connections that are allocated the highest service rate. We think of each list as having some number of time slots associated with it. We partition the bandwidth on the link between service lists by allocating some fraction of the slots associated with each list to lists that are lower in the hierarchy. When all the timeslots allocated to a particular list are used, we start another round through its service list. At the bottom of the hierarchy, there is a round robin server that handles the best effort traffic.

Consider two lists of connections that require service, labeled 1 and 2. The level 1 list is used for high rate connections, while the level 2 list is used for lower rate connections. (For the moment, we will ignore best effort traffic). We can achieve different service rates by having different frame lengths for the two lists. Successive rounds through the level 1 list are to be started at fixed time intervals FT_1 (in units of cell service time), and successive service rounds through the level 2 list are to be started at fixed time intervals FT_2. Moreover, we *interleave* the service of level 1 and 2 as follows. After serving $b_1 < FT_1$ cells in level 1, we serve $FT_1 - b_1$ cells in level 2. After serving b_1 cells in level 1 again, we serve the next cells in round robin order in level 2, and so on.

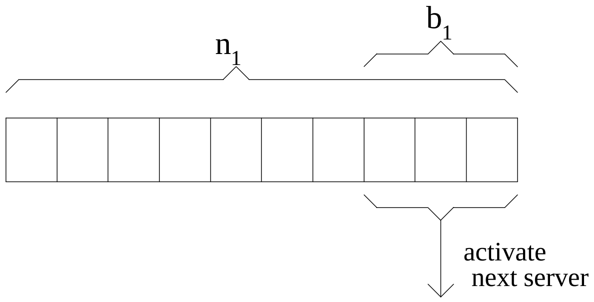
The next refinement is to allow for multiple service quanta each time a connection is picked for service. Given a connection j, we define the service quantum a_j, which is the number of cells served from that connection whenever it is picked for service. Thus, if VCI 29 is given a service quantum of 3, every time that VCI appears in a round robin list, three cells are serviced. There is an obvious problem if this VCI is on a level other than the first. Since the service of that level may be widely separated in time, we will need to remember how many cells from a VCI have been serviced at each level, and when service at that level resumes, this has to be accounted for. Thus, when connection j is picked for service, we load a counter, G_i, with the value of the service quantum a_j. Every time level i services a cell, G_i is decremented till the service quantum is exhausted.

The final refinement of the algorithm is to add a best effort server. If in serving CIDs from any service list we run out of CIDs before exhausting the frame, we fill out the rest of the frame with best effort traffic.

We are now ready to present the complete algorithm, in a form suitable for hardware realization.

Notation

We number the levels $1 \cdots N$, with the highest rate connections at level 1. At level i, we have n_i slots. As before, one slot represents one cell serviced on the output link. We define a boundary $b_i \le n_i$, such that b_i slots are allocated to lists at lower (slower) levels. The remaining $n_i - b_i$ slots at level i are either allocated to



Server at level 1 allocates 3 slots to lower levels.
Figure 3

some connection or are unallocated. The service quantum for connection j is denoted by a_j. Let *queued* $_j$ denote the number of cells queued at a connection at any given time. We shall refer to the HRR implementation as having multiple servers, one per level, where a server is either *active* or *inactive.* A server at level i is active if servers $1..i-1$ are active and have served $n_i - b_i$ cells in each of their frames. Server 1 is always active. (Figure 3 gives an example of a server at level 1 with n_1 equal to 10 slots and where $b_1 = 3$ slots have been allocated to lower levels.)

The frame time, FT_i, for level i is defined to be the time (in units of cell service time) between two services of the first slot in the frame. During each frame time, a server makes one round through its service list.

Detailed Description

Each server i has three counters called NB_i, B_i and G_i. NB_i determines how many cells are served from level i, B_i determines how many cells are served from all levels lower than i, and G_i keeps track of service quanta larger than 1. The algorithm proceeds as follows:

```
1.  Initialize:
     At the beginning of a frame at level $i$,
     $NB_i \leftarrow n_i - b_i$;  $B_i \leftarrow b_i$;  and swap($next_i$, $current_i$).
2.  Server and connection selection:
     Let $i$ be the index of the lowest rate active server.
     If (current service list $i$ is empty and $NB_i$ is non-zero)
         activate BE server for one slot
     else
         server $i$ picks connection $j$ from the head of its current service list
         If ($G_i$ is 0)
             $G_i \leftarrow min(a_j, queued_j)$.
         serve connection $j$ for one slot; decrement $G_i$.
     decrement $NB_i$ and $B_{i-1}, ..., B_1$.
3.  Adjust service list:
     If ($j$'s data queue is empty)
         set $G_i \leftarrow 0$ and clear the round robin flag bit for connection $j$
     else if ($G_i$ is zero)
         server $i$ places connection $j$ at the tail of its next service list
     else server $i$ places connection $j$ at the head of its current service list
4.  Check for change of active server:
     If (any of $B_{i-1}, ..., B_1$ is 0)
         server $i$ becomes inactive.
     else if ($NB_i$ is zero and $B_i$ is non-zero)
         server $i$ makes server $i+1$ active
     Go to step 2.
```

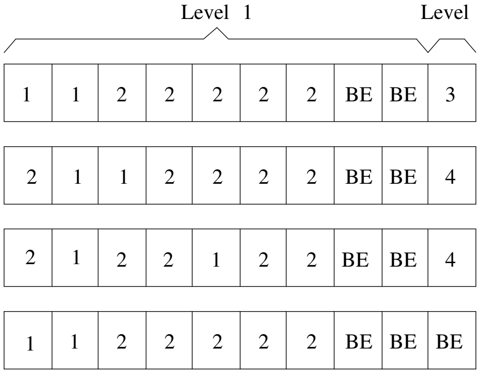
Example

An example will illustrate. Consider two levels with n_1 = 10 slots and n_2 = 10 slots. Also, b_1 = 1 slot, which means that the level 1 server allocates 1 time slot every 10 to the level 2 server. b_2 = 0. (At 45 Mbps with ATM 53 byte cells, these frame times support rates that are multiples of 4.5 Mbps and 450 Kbps, respectively.) Suppose that CID#1 has reserved 9 Mbps, CID#2 has reserved 22.5 Mbps, CID#3 has reserved 450 Kbps, and CID#4 has reserved 900 Kbps. The allocations will look like the following:

	CID#	allocation
Level 1:	1	2
	2	5
Level 2:	3	1
	4	2

The HRR algorithm does not determine the service order within the frame. That is determined by the position of the CID in the service list for a particular service. For example, the above case might result in the service order shown in Figure 4.

The HRR algorithm is able to offer guaranteed bandwidth and an upper bound on jitter to rate controlled connections. The bandwidth received by a connection j at level i is a_j / FT_i cells/sec. The jitter bound is computed as follows. In the worst case, an HRR server could serve connection j for a_j cells at the end of one frame, following by another a_j cells at the beginning of the next frame. If we consider at the interval during which the cells are sent, we see that the jitter bound is $2 * a_j$ cells during one frame time at that level. Thus, the HRR algorithm bounds the size of the elasticity buffer needed at intermediate nodes and at the receiver as desired. We also offer a straightforward computation of the delay bound for an HRR server at an intermediate node. Consider a cell from connection j that arrives at the server just after j was served. This cell may not get sent out until the end of the next frame, so the delay is bounded by $2 * FT_i$, if connection j is at level i. Thus, there is a fixed delay introduced by the server as a side-effect of metering the flow.



1	1	2	2	2	2	2	BE	BE	3
2	1	1	2	2	2	2	BE	BE	4
2	1	2	2	1	2	2	BE	BE	4
1	1	2	2	2	2	2	BE	BE	BE

this frame repeated seven times ...

Example of service order
Figure 4

The HRR scheme can be readily implemented as an Application Specific Integrated Circuit (ASIC). One of the authors and R.C. Restrick have built a queueing engine that implements per-connection queueing for 32,000 CIDs with a memory bandwidth of 1.3 Gbps. A chip has been built which supports sixteen round robin lists of CIDs. Implementation of the HRR service discipline requires support for multiple frame counters to control which round robin list is served next. The chip to implement the counters would be designed with a microprocessor interface to allow flexibility in setting the parameters of the algorithm.

Rate Partitioning

Consider the hierarchical round robin server in the example above with $n_1 = 10$, $n_2 = 10$, $b_1 = 1$, $b_2 = 0$. These parameter values allow up to 9 connections sending at 10% of the total bandwidth, and up to 10 connections sending at 1% of the total bandwidth. Suppose the 11th connection sends a call setup message requesting rate controlled service at 1%. As things stand, we can either allocate more bandwidth to this connection than requested, or we can deny the call. As both are unacceptable, we present a scheme which allows the menu of rates to be changed. Suppose we simply set $b_1 = 2$. This would cause the server at level 2 to be served twice as often, which would halve the frame time and double the rate. The correct solution is to set