Queueing Delays in Rate Controlled ATM Networks

Anindo Banerjea

CS Division, University of California, Berkeley, CA 94704
and International Computer Science Institute, Berkeley

Srinivasan Keshav

AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974

Abstract

This paper addresses the problem of finding the worst case end-to-end delay and buffer occupancy bounds in ATM networks with rate-controlled, non-work conserving servers. A theoretical framework is constructed to analyze such servers in isolation and in tandem. The analysis is based on a simple fluid model, but care is taken so that the computed delay and buffer occupancy values are upper bounds on actual values. A simple algorithm is presented to perform these calculations in linear time. Simulation results compare the computed worst case delays with the actual delays obtained on some simple network topologies. The algorithm is found to predict node delays well for bursty input traffic, but poorly for smooth input traffic. Buffer requirements are predicted well in both cases.

1. Introduction

Recent work has shown that *framed, non-work-conserving* servers can provide end-to-end delay bounds to users who need strict guarantees on network performance [15]. By *framed* we mean that the servers work on the basis of a fixed size interval called the *frametime,* during which they allocate a number of transmission slots to each channel being served. We assume that during each slot a fixed size packet or ATM *cell* is transmitted on the output trunk. By *non-work-conserving* we mean that given enough cells in the queue, the server will send out exactly the allocated number of ATM cells (the *chunksize*) in each frame, even if this will leave the output trunk idle. The server is free to serve these cells during any portion of the frame. If there are fewer than *chunksize* cells in the queue for the channel, the server will send out the available cells, then use the free slots for non-real time traffic. Examples of such servers are those that obey the Hierarchical Round Robin (HRR) and the Stop-and-Go service discipline [8, 7]. These servers simultaneously serve a number of frames of different lengths so that a choice of frametimes is available during channel establishment.

While earlier work considered the behavior of a single server [KKK 90], and of 'smooth' traffic traversing a sequence of identical servers [7, 15], the general case, where a circuit carrying bursty traffic may be served at a different framing interval at each server, was not analyzed. This paper determines the upper bound on the end-to-end queueing delays and buffer requirements at each queueing point for a simplex channel established over any sequence of framed non-work conserving servers. We present a formal modeling of a tandem of such servers and use it to compute the worst case end-to-end delay and buffer requirements. These are then compared with simulation results.

2. Previous Work

Traditionally, end-to-end delays have been computed for Poisson arrivals, and for the average case, since this makes queueing theoretic analysis easier. In contrast, we will assume parametrically constrained inputs and compute worst-case delays. This type of analysis was first proposed by Cruz who used it to analyze some work-conserving scheduling disciplines [2, 3]. In recent work, Cruz has also computed delay bounds for non-work conserving disciplines, but this analysis does not explicitly consider the effects of different frame sizes at different servers, resulting

---

A. Banerjea was supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Hitachi, Ltd., Hitachi America, Ltd., the University of California under a MICRO grant, and the International Computer Science Institute. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations.

in weaker delay bounds [4]. Kurose has computed tight delay bounds not only for the maximum delay, but also for the delay distribution [10]. However, this work assumes a work-conserving service discipline, and a stronger characterization of input traffic than ours.

Parekh and Gallager [12, 13] have computed worst case delays assuming sources were leaky bucket compliant and the servers used the Packetized Generalized Processor Sharing (PGPS) scheduling discipline. This is a work-conserving discipline and so different traffic streams can interact with each other. Thus, their analysis is necessarily more intricate than ours, since with non-work conserving disciplines, competing traffic streams interact minimally. On the other hand, we model the phenomenon of 'slippage' (to be described below) and this introduces complexities that are not found in their analysis. We have used their notion of 'greedy' sources extensively in our work.

Our formal modelling of a message stream and of a server as a transformation on a message stream is based on the recent work of Low and Varaiya [11]. While they use their model mainly to analyze burstiness, we use it for exact transient analysis of a tandem of rate-controlled servers.

Our server model is based on the HRR servers described in [8]. The Stop-and-Go discipline proposed by Golestani [7] is closely related to this discipline, and the techniques we propose can be used to analyze Stop-and-Go as well. As currently described in the literature, the delay and delay jitter bounds for Stop-and-Go are valid only when a connection is allocated the same frametime at all servers along the path, and the source is assumed to be smooth at the time scale of a frametime. Our analysis can be trivially extended to provide delay and delay jitter bounds for Stop-and-Go when a connection can be assigned to different frametimes at different servers, or when it is not smooth at the time scale of a frametime.

3. Network Model

In our model a channel is associated with a static path through the network and has state and reserved resources at each switch along this path. Since a rate-controlled server isolates the traffic flowing along each channel from the traffic on other channels, we focus our attention on the performance of a single channel in the network. We first characterize the input traffic, then present a series of definitions, lemmas and theorems that allow us to present an algorithm to compute the worst case delay and buffer bounds.

3.1. Input Traffic Model

We characterize the traffic entering the channel with three parameters as suggested in [5, 6]:

$x_{\min}$ The minimum cell inter-arrival time.

x_{ave} The minimum average cell inter-arrival time, over **any** time interval of length *I*.

I The averaging interval for calculating x_{ave}.

Recent work has shown that this model accurately describes many types of rate-controlled traffic expected in high-speed networks [14]. We assume that the user of the channel is required to obey these restrictions on the input traffic, and the delay and buffer values calculated need only hold if these restrictions are not violated.

3.2. Definitions

Our notation draws from, and extends, that used in Reference [11]. We define a *message* as a temporal flow of data represented by a bounded non-negative function $m(t)$, $0 \le t \le T$ where $m(t)$ is the instantaneous rate at time *t* in cells/sec and $T < \infty$. The interval $[0, T]$ is called the *time space* and is denoted by *TS*. The *cumulative message curve* $M(t) = \int_0^t m(t)$ and is the amount of the message seen till time *t*. The inverse of $M = M^{-1}$ is defined as the *earliest* time at which $M(t)$ amount of message has been seen. The interval $[0, M(T)]$ is the *message space* and will be denoted by *MS*.

A *server* is a device that takes an arriving message $m_{in}(t)$ and transforms it into a departing message. The arriving message at a server is defined by its arrival curve $A(t) = \int_0^t m_{in}(t)$ and the departing message is defined by the service curve $S_A(t) = \; = \int_0^t m_{out}(t)$. Every server is associated with a queue. A message is stored in the queue from the time that it arrives till the time it is served. The instantaneous amount of data in the server (*queue size*) is denoted by $Q_A(t) = A(t) - S_A(t)$. The *maximum queue size* Q^* is defined as $\underset{t \in TS}{Max}\; Q_{A(t)}$. The *queueing delay* $D_A(m_0) = S_A^{-1}(m_0) - A^{-1}(m_0)$ is the time between the arrival and departure of the m_0 prefix of the message. The *maximum queueing delay* D^* is $\underset{m \in MS}{Max}\; D_A(m)$.

A cumulative message curve $M(t)$ will be called *piecewise linear constrained (PLC)* if it is made of piecewise linear segments. A *greedy* source is one that sends data at the maximum rate possible and at the earliest moment possible. A *parametrically constrained source (PCS)* is a source that obeys the following constraints:

(a) the rate of message transmission $m(t) \le 1/x_{\min}$

(b) over all intervals of time of length *I* and all time *t*, $\int_t^{t+I} m(t)\, dt \le I/x_{ave}$,

where $x_{\min}$, x_{ave} and *I* are the three parameters constraining the source. A *parametrically constrained greedy source (PCGS)* will send at rate $1/x_{\min}$ until I/x_{ave} packets are sent, then will remain idle until time *I*. After this, the behavior will repeat, until the entire message has been sent. Clearly, a PCGS is PLC. The cumulative message curve $M(t)$ for a PCGS source is shown by the dashed line in

Figure 1.

For a given set of parametric constraints $x_{\min}$, x_{ave}, and I a specific interval $[t_0, t_0 + I] \subseteq TS$ is denoted by *IS* and is completely specified by choosing t_0, the origin for the interval. A PCS obeys the average rate rule over any interval *IS*.

A server is called a *fixed rate server* with service rate ρ if its output message

$$m_{out}(t) = \Phi(m_{in})(t)$$
$$= m_{in}(t) \text{ if } m_{in}(t) \le \rho \text{ and } Q_A(t) = 0$$
$$= \rho \text{ if } m_{in}(t) > \rho \text{ or } Q_A(t) > 0.$$

3.3. Exact Analysis of a Tandem of Servers

We now state some theorems about the behavior of fixed rate servers with PCS inputs. We first look at a single server, then consider servers operating in tandem, and finally introduce, define and analyze servers with slippage. For reasons of space, we defer proofs to Reference [1].

3.3.1. Single Server

Lemma 1.1: If traffic generated by a PCS $M_{in}(t)$ is input to a fixed rate server with rate $\rho \ge 1/x_{ave}$, then any interval of time during which $Q_{M_{in}}(t) > 0$ continuously has to be of length $< I$.

Lemma 1.2: If the input to a fixed rate server is PCS then so is the output.

Theorem 1.1: For a fixed rate server of rate $\ge 1/x_{ave}$, the largest possible queueing delay $D*$ and the largest possible queueing size $Q*$ over all possible PCS input messages are achieved by a parametrically constrained greedy source (PCGS).

Corollary: The maximum queue size achieved by any PCS is less than that achieved by a PCGS at time $\frac{I\, x_{\min}}{x_{ave}}$ from the start of the interval.

This tells us that for a single fixed rate server, the worst case queueing behaviour is observed on sending in a PCGS input to the server, and occurs within the first I time units. Thus, we can compute $D*$ and $Q*$ simply by studying the behavior of a PCGS traffic input over a single time interval of duration I.

3.3.2. Tandem of Servers

A tandem of servers is defined to be a set of servers numbered $1 \cdots \mathrm{N}$ such that the output of server i is the input to server $i+1$. There may be a fixed propagation delay in between servers. In this analysis, the delay is irrelevant and is assumed to be 0. For a tandem of fixed rate servers, the service curve of server i is the arrival curve for server $i+1$.

Lemma 2.1: If the input to a fixed rate server is from a PCGS, then the output is PCS. If $\rho > 1/x_{\min}$ then output is identical to the input, otherwise it looks like a PCGS with $x'_{\min} = 1/\rho$ and $x'_{ave} = x_{ave}$.

Theorem 2.1: All of the fixed rate servers in a tandem achieve the largest possible delay and the largest possible queue size when the input to server 1 is PCGS as compared to any other PCS source with the same parameters. Moreover, this occurs within the first I time units of the PCGS stream.

Theorem 2.1 extends the result from the case of a single server to that of a tandem of servers. Thus, even for a tandem, we can calculate bounds by studying a single traffic input for a limited period of time.

3.3.3. Slippage

Consider a source that has been allocated a service rate of, say, 1 cell per frame at some server. A source may try to fool the server in the following way: it sends one cell at the start of the frame time, and when it knows that the that cell has been served, send another cell within the same frame time. If the server schedules the VCI for service immediately upon the arrival of the second cell, then the source will be able to have two cells served per frame time whereas it would have paid for only one. To prevent this, some servers maintain two sets of VCI service lists [8]. If a cell arrives on a VCI that has an idle queue, the VCI is placed in the 'idle' service list. At the end of the frame time, the 'idle' and 'active' service lists are swapped. This prevents sources from overusing the server, but at the cost of added delay for the first cell that arrives to idle VCI. We call this additional delay *slippage*. Slippage may also occur if a switch stores state information for inactive VCIs in secondary storage, so that when a cell arrives to an idle VCI, there may be some time lost in fetching VCI state.

After the slippage time, the flow is continuous until the queue becomes empty. The channel is then again made inactive. In a HRR server, the channel remains inactive up to one frametime, which is the slippage for the server. A *bounded slippage fixed rate server* with slippage bound s and rate ρ has the following properties.

(a) If $Q_{M_{in}}(t) = 0$ then $m_{out}(t) = 0$

(b) If $Q_{M_{in}}(t_1) = 0$ and $Q_{M_{in}}(t_1 +^{\dagger}) > 0$ then $\exists_{t_2 \in TS} \{t_2 \mid t_2 \le t_1 + s\}$ and $\exists_{t_3 \in TS} \{t_3 \mid t_3 > t_2\}$ such that $m_{out}(t) = 0$ in $[t_1, t_2)$, $m_{out}(t) = \rho$ in $[t_2, t_3)$ and $Q_{M_{in}}(t_3) = 0$.

In other words the slippage (the length of the interval during which the server remains inactive after $Q_{M_{in}}(t)$ becomes non-zero) varies from 0 to s.

A *fixed slippage fixed rate server* with fixed slippage s

† $f(t_1+)$ is the limit from the right of f(t) at t_1. That is, the queue becomes non zero after t_1.

and rate ρ is a server of the above type such that in (b) above $t_2 = t_1 + s$. This server is deterministic and the slippage is always s.

The following theorems and lemmas are useful when $s << I$. We expect this to be the common case.

Lemma 3.1: If the message curve $M_{in}(t)$ generated by a PCS is input to a bounded slippage fixed rate server (ρ, s) having $\rho > \frac{I}{x_{ave}(I-s)}$ then any interval of time during which $Q_{M_{in}}(t) > 0$ continuously has to be of length $< I$

Lemma 3.2: If the message curve $M_{in}(t)$ generated by a PCS is input to a bounded slippage fixed rate server (ρ, s) having $\rho > \frac{I}{x_{ave}(I-s)}$ then the output is PCS.

Theorem 3.1: The queueing delay and queuesize of a *bounded* slippage fixed rate server (ρ, s) having rate $\rho > \frac{I}{x_{ave}(I-s)}$ over all possible PCS inputs are bounded by the $D*$ and $Q*$ achieved by a PCGS input to a fixed rate server with *fixed* slippage s.

By Lemma 3.1 and the periodic nature of a PCGS $D*$ and $Q*$ occur within one interval of length I. Thus, we only need analyze the system for the first I time units, with a PCGS source, to find the delay and queue size bounds.

Note: From this point on we assume $\rho > \frac{I}{x_{ave}(I-s)}$.

3.3.4. Tandem of Fixed Rate Servers with Slippage

We now introduce a relative definition of greediness. A message curve $M(t)$ is said to be greedy with respect to some constraint set *CS* where *CS* is a set of message curves if $\underset{N(t) \in CS}{\forall} \; \underset{m \in MS}{\forall} \; M^{-1}(m) \le N^{-1}(m)$.

A server maps a message curve $M(t)$ into a output message curve $S_M(t)$. Thus it can also be thought to map sets of messages corresponding to all legal inputs ($\in$ *CS*) to sets of messages corresponding to all the outputs to the messages in the input set *CS*. Loosely, we can also use the symbol Φ to denote this transformation.

Lemma 4.1: If the input to a fixed slippage fixed rate server is greedy with respect to the set *PCS* of message curves which are PCS, the output is greedy with respect to the set of message curves $\Phi(PCS)$ which are the outputs for the message curves in the input set.

Theorem 4.1: The queueing delay and queuesize of a tandem of *bounded* slippage fixed rate servers (ρ_i, s_i) having rate $\rho_i > \frac{I}{x_{ave}(I-s)}$ over all possible PCS input messages are bounded by the $D*$s and $Q*$s achieved by a PCGS input to a tandem of *fixed* slippage fixed rate servers (ρ_i, s_i).

From the theorems above, we can compute upper bounds on the queueing delay and queue size for a tandem of bounded slippage fixed rate servers for any parametrically constrained input by assuming PCGS input to server 1 and assuming fixed slippage at each of the servers. This is a simplified and deterministic system with fixed and computable curves $M_i(t)$ at the output of each server i. Moreover, by Lemma 3.1, we only need to look at any one interval *IS*.

The system can be analyzed graphically in a manner similar to [13] as follows: We start with $M_0(t)$ as the greedy input curve. Apply the series of transformations Φ_i as defined by the definition for fixed slippage fixed rate servers and the parameters (ρ, s), to this curve for $i = 1 \ldots N$, the number of servers in the tandem, to get the series of intermediate message curves $M_i(t)$.

One way to do this would be to construct $M_0(t)$ as a set of points in the $MS \times IS$ plane, using enough points to ensure accuracy, then apply Φ_1 to each of them to get the output from the first server, and so on for each Φ_i. This is computationally expensive, and also requires too much communication in case the computation needs to be done in a distributed fashion (all the points in $M_i(t)$ need to be carried across from server i to i+1). We need some short hand to describe $M_i(t)$.

3.3.5. Envelope of the Curve

We define the envelope of $M_i(t)$, $\varepsilon_i(t)$, by construction as follows. ε_0 is the PCGS message curve input to server 1, as shown by the dashed line in Figure 1. ε_i the envelope of the output from server i (with rate ρ_i and slippage s_i) is constructed from ε_{i-1} the envelope of the output from server $i-1$ as follows :

Let:

$t_0 = Min\{t \in TS \mid \varepsilon_{i-1}(t) = 0 \text{ and } \varepsilon_{i-1}(t+) > 0\}$

$t_1 = Min\{t \in TS \mid \varepsilon_{i-1}(t) = \rho_i(t - t_0)$

Then: $\varepsilon_i(t)$

$= 0$ for $t < t_0 + s_i$

$= \rho_i(t - t_0)$ for $t_0 \le t < t_1$

$= \varepsilon_{i-1}(t)$ for $t \ge t_1$

$\varepsilon_i(t)$ can be thought of as an upper bound on $M_i(t)$. t_0 is the point where server i becomes active and t_1 is where Q goes to zero. In Figure 2 for server 1, t_0 is the x-coordinate of point **E** and t_1 is the x-coordinate of point **A**.)

Figure 1 to shows how ε_1 (drawn in bold lines) is defined from ε_0 (in the dashed lines). Figure 2 shows ε_2 in bold and ε_1 in the dashed lines.

For the rest of this section $M_i(t)$ refers to the message curve output from server i if the input to server 1 is PCGS.

Lemma 5.1: The maximum queueing delay $D*$ and the maximum queuesize $Q*$ for server i with input $M_{i-1}(t)$ occur in the interval (t_0, t_1). (t_0 and t_1 are from the definition of $\varepsilon_i(t)$.

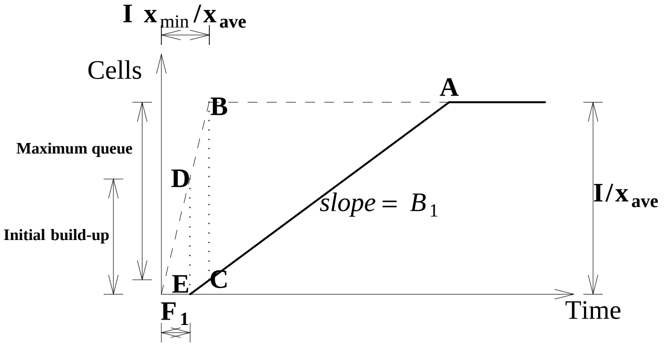

Figure 1. Calculating $\mathcal{E}_1$ from $\mathcal{E}_0$.

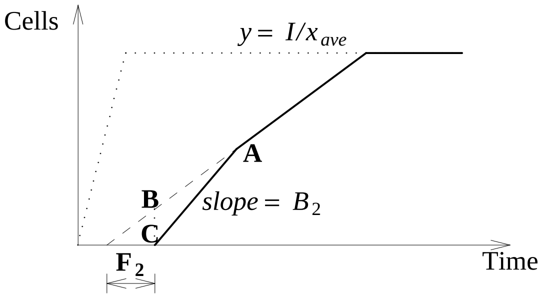

Figure 2. Calculating $\mathcal{E}_2$ from $\mathcal{E}_1$.

Theorem 5.1: If $\mathcal{E}_{i-1}$ is the input to server i, the resulting $Q^*_{\mathcal{E}_{i-1}}$ and $D^*_{\mathcal{E}_{i-1}}$ are upper bounds on $Q_{M_{i-1}}(t)$ and $D_{M_{i-1}}(t)$ observed if $M_{i-1}(t)$ is the input.

The maximum vertical distance between $\mathcal{E}_i$ and $\mathcal{E}_{i-1}$ corresponds to $Q*$ and the maximum horizontal distance to $D*$. In Figure 1 $Q*$ for server 1 is given by the segment BC and $D*$ is given by the segment BA. Calculation of the envelopes is computationally feasible; we will present an algorithm in Section 5. Theorem 5.1 tells us that the values for $Q*$ and $D*$ we get from the envelope calculations are upper bounds on the actual queue size and queuing delay observed for any input.

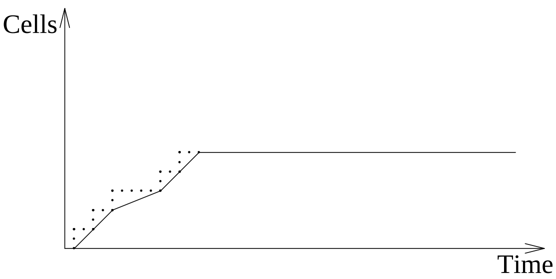

Figure 3. Actual Departure of Traffic vs. Fluid Model.

4. Correcting for the fluid approximation

The above analysis treats the message as a continuous function in time, so that a server sends out the message smoothly at a rate ρ. In reality, a framed, non-work-conserving server sends out cells as a step function where one *chunksize* set of cells are transmitted once every frame-time. A fluid model is a straight line approximation to it. Since the cells in each frame can be sent out at any time within the frame, we must assume the worst case and correct for it. We do this in much the same way as in Reference [13, Chapter 4], where the fluid analysis is extended to the packetized case. To get the maximum number of cells which could have been sent out from $node_j$ by time t, we take the value given by the straight line approximation and apply the following function to it.

$$\text{buffer\_correction}(y) \equiv \left\lceil \left\lfloor \frac{y}{a_j} \right\rfloor + 1 \right\rceil a_j,$$

where a_j is the *chunksize* at server *j*.

This gives us the value of the step immediately over the point on the straight line that we are looking at. In terms of Figure 3, it raises the points on the straight line to the dotted line above it. By taking the floor and adding 1 we ensure that for the points of discontinuity in the step function, we take the higher value.

Similarly to calculate the queueing delay for a cell which is q deep in the queue of a server, we use

$$\text{delay\_correction}(q) \equiv \left\lceil \frac{q}{a_j} \right\rceil F_j,$$

where F_j is the frametime at the server.

5. Computational Algorithm

The analysis above motivates a simple algorithm to calculate the envelope of the message curve. By Theorems 4.1 and 5.1 the envelope provides upper bounds on the delay and queue sizes experienced at each server by any PCS input to server 1. By Lemma 3.1 we only need to look at one interval *IS*. By correcting for the fluid approximation we can get upper bounds for delays suffered by fixed size ATM cells being transmitted *chunksize* cells in each frame.

The envelope $\mathcal{E}_i$ is represented as a doubly linked list of the endpoints of the linear segments of the envelope ($<x,y>$ coordinate pairs), sorted by x coordinates. The initial envelope $\mathcal{E}_i$ describing the worst case input traffic contains three points: $<0,0>$, $<\frac{I\,x_{\min}}{x_{ave}}, \frac{I}{x_{ave}}>$, and $<\infty, \frac{I}{x_{ave}}>$. The C psuedocode in Figure 4 describes the computation for one server in the tandem. It shows how to compute $\mathcal{E}_i$, $Q*$ and $D*$ for server i from $\mathcal{E}_{i-1}$.

```
struct point {
    float x, y;
    struct point *next, *previous;
};
#define buffer_correction (y)  ( ⌈ y / a_{i-1} ⌉ + 1) a_{i-1}
#define delay_correction(q)  ⌈ q / a_i ⌉ F_i
/* Inputs to the algorithm */
float a_{i-1}, a_i;              /* chunksizes */
float F_i, s_i;                  /* frametime, chunksize */
float ρ_i;                       /* service rate*/
/* Input from server i-1*/
struct point *env;               /* ε_{i-1} */
/* Local variables */
struct point *p, *q;
float Q, D, x;
float Q* = D* = 0.0;             /* The results */

for (p = env ; p→x < ∞ ; p = p→next )
        /* Normalize the point, s_i is the slippage */
        p→x = p→x - s_i;
for (p = env→next ; p→x < ∞ ; p = p→next ){
        q = p→previous;
        if (p→x < 0){
        /* Q and D for points before start of service */
                Q = buffer_correction(p→y);
                D = delay_correction(Q) - p→x;
                /* -p→x adds the remaining slippage */
                D* = max(D*, D); Q* = max(Q*, Q);
        }
        if (p→x > 0 and q→x ≤ 0) {
        /* find Q and D for x = 0 */
                Q = buffer_correction(
                        (p→x * q→y - q→x * p→y) / (p→x - q→x));
                D = delay_correction(Q);
                D* = max(D*, D); Q* = max(Q*, Q);
        }
        if (p→x > 0){
        /* for all other points */
                Q = buffer_correction(p→y) - ρ_i * p→x;
                if (Q < 0) break;
                /* we need not look at remaining points */
                D = delay_correction(Q);
                D* = max(D*, D); Q* = max(Q*, Q);
        }
}
/* Construct ε_i for the next server */
env = p;        /* Discard points below the line y = ρ_i x */
q = p→previous;
x = (p→x * q→y - q→x * p→y) / (q→y - p→y + ρ_i * (p→x - q→x));
add_to_envelope(env , new_point(x,ρ_i * x));
add_to_envelope(env , new_point(0.0,0.0));
```

Figure 4. Calculations for $node_i$.

To simplify calculations we normalize the envelope (by subtracting s_i from the x-coordinate of each point) on starting the computation for each node so that the x-intercept of the output envelope is always at the origin. This sets t_0 as defined in Lemma 5.1 to be 0. Then for each point in the interval (t_0, t_1) as defined in Lemma 5.1 we find Q and D and take the maxima. By Lemma 5.1 we only need to look at these points. By the definition of $\mathcal{E}$ the remaining points together with $<t_1, \mathcal{E}_{i-1}(t_1)>$ and $<0,0>$ constitute $\mathcal{E}_i$. Thus, using the analysis of section 3, $Q*$ and $D*$ as calculated by the algorithm are provably the upper bounds on the queuing delay and queue size at server i in the tandem.

The computation can be carried out during channel establishment in a distributed fashion by each intermediate switch along the path. The inputs to the computation are a_i, F_i, s_i, ρ_i, (which are properties of the server i and should be locally known) a_{i-1} and $\mathcal{E}_{i-1}$ (which need to be communicated as a part of the channel establishment control message). The outputs are $Q*, D*$ and $\mathcal{E}_i$.

6. Discussion of Pessimistic Assumptions

The assumptions we have made for our analysis are worst case assumptions and they account for a case that will almost never happen in practice.

To begin with, we get an upper bound on the end-to-end delay by adding the worst case delays for each node. While this analysis is tight in the sense that pathological cases can be constructed such that the bound is achieved, it is very unlikely that the same cell will suffer the worst case delay at each node. Thus, the sum of the per node worst case delays will likely be a loose upper bound on actual end to end delays.

Second, we assume that the source is greedy. If a source is not always greedy, then it will get an end-to-end delay that could be much less than if it were greedy.

Finally, we have assumed in our analysis that the cells that arrive during the slippage period suffer the maximum slippage (of F_i) and *also* get transmitted at the end of their transmission frames, leading to a net delay of $2F_i$. To suffer a slippage of F_i the first cell must come right at the start of the frame of the node, and to suffer an additional F_i delay it must get transmitted at the very end of the next frame. This is a very unlikely combination of events.

To summarize, our analysis is tight, but pessimistic; therefore our worst case delay bounds are likely to be larger than those observed in practice. To get a feel for the kind of maximum delays that might be observed in practice we turned to simulation.

7. Simulation

The first part of this paper presented a worst case analysis of the end-to-end queueing delays suffered by the cells transmitted across a virtual circuit established through a series of non-work conserving rate-controlled switches. Our formal modelling assures us that the analysis is correct, but an interesting question is to determine how accurately the analysis reflects the observed worst case delay and buffering. In Section 6 we pointed out some reasons why we expect our analysis to be pessimistic. In this section, we compare analytical and simulation results when the switches implement HRR scheduling. Specifically, we use two metrics to compare the worst case delay computed analytically to the average and worst case delay in a set of simulation scenarios:

a) the ratio of the average delay to the analytical delay bound.

b) the ratio of the maximum observed delay to the analytical delay bound.

The first metric indicates how pessimistic the analysis is in practice, and the second metric measures its tightness.

At this point, it is useful to consider the question of computing average and maximum delays using simulations. Maximum delay is a point metric, and as such can be observed only in an infinitely long simulation run. Thus, the observed maximum delay is only indicative of the true maximum. The average case, similarly, can only be computed by considering all possible input workloads and all possible server phase relationships that correspond to a single simulation topology. Since this is impossible to simulate, again, our 'average' results are only indicative of the true average. Nevertheless, we would like to introduce enough variability in the simulation so as to approximate this average. To do so, we need to consider the sources of delay variability.

Several factors contribute to the end-to-end delay of a cell. These are the propagation and switching delay, the queueing delay, and, since reservations are made at the average rate, the smoothing delay at the first switch in the path. In these experiments, we ignore the propagation and switching delays, since they are typically of fixed size and independent of the scheduling discipline. The smoothing delay depends purely on the burstiness of the source, and for a greedy source, this is deterministic. Thus, the analysis computes this exactly. The queueing delay arises from variations in service time within a single frametime due to internal burstiness, slippage and the fact that cells queued for service during a frame time are served in random order. Thus, it depends on the ratio of the frame times at adjacent switches, drift between frames at adjacent switches and the intensity of cross traffic. It is the source of variability in the delay.

Since the smoothing, propagation and switching delays are deterministic, in order to compute the average delay using simulations, we need to ensure that the probability that a cell arrives from each source be uniformly distributed during each frametime of each switch. We do so by introducing cross traffic sources that generated parametrically constrained random traffic. Since this traffic takes up a random number of slots in the frame, cells from the source under study achieve the desired random queueing delays. By running simulations for long enough periods, we achieve convergence of the average delay, as described in Section 7.1.

7.1. Simulation Details

All simulations were performed using the REAL simulator [9]. The simulation topology is simple - a series of switches from a source (labeled 1) to a destination (Figure 5). This source sends parametrically constrained greedy traffic, that is, traffic that can be characterized by the parameters $x_{\min}$, x_{ave} and I and such that every part of the message is sent at the earliest time possible.

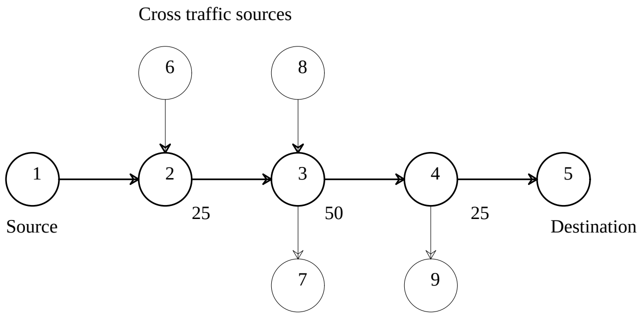

Figure 5. Simulation scenario.

Nodes 2, 3 and 4 are HRR switchesi and node 5 is the destination. Cross traffic sources 6 and 8 generate parametrically constrained traffic, but within these constraints, they send data at random intervals of time (i.e., non-greedily) to corresponding destinations 7 and 9.

For the purposes of this study, we have chosen all cells to be 100 bytes long and all lines with a capacity of 400,000 bits/sec (500 cells/sec). With suitable scaling, these results are equally applicable to higher speed networks For example, if we scale by a factor of 1000, then 1 second of simulated time (and end-to-end delay) would correspond to 1ms, and 1 kbps of bandwidth to 1 Mbps. Since our results do not have any time related constants, they are scale independent,

Switches are assumed to have infinite buffering and the actual number of buffers used is traced to be able to compare against the predicted buffer requirements. The delay values at an individual switch are accurate to 1μs. However, since end-to-end delays are computed using a histogram with a bucket size of 15ms, the values presented are accurate only to plus or minus 15ms.

7.2. Simulation Convergence

One tricky problem with running a simulation is to know when to stop. We determine the length of the simulation by looking at the graph of the average end-to-end delay versus simulation time. For all the scenarios we found that the average delay reached its asymptote well before 50 seconds of simulated time, so all the results are shown for simulations that were run for 50 seconds.

We noted above that the end-to-end delay of a cell depends on several factors. It is impractical to study the effect of all the parameters at the same time, so we study a base case, then vary one parameter at a time and note its effect.

7.3. The base case

In the base case, the source sends data at a constant rate of one 100 byte cell every 40ms (25 cells per second) so that $x_{\min} = x_{ave}$ = 40ms. The frame time at switches 2 and 4 is 50ms, corresponding to 25 slots, and the frame time at switch 3 is 100ms, corresponding to 50 slots. The results for the base case, with cross traffic intensity of 0.5 of the line bandwidth are shown in Table 1.

Scenario	*Observed Mean (s)*	*Observed Max (s)*	*Bound (s)*	*Mean/ Bound (ratio)*	*Max/ Bound (ratio)*
Base	*0.12*	*0.19*	*0.46*	*0.24*	*0.40*

Table 1. Base case end-to-end delay results.

We note that the observed maximum delay is much smaller than the computed maximum delay. This is because of two reasons.

- The input is perfectly smooth, at a rate less than or equal to the allocation at each switch. Thus the slippage delay is the dominant factor in the calculated value, and the worst case slippage is very unlikely (see the discussion in § 6).
- In the worst case the same cell gets the worst possible delay at each switch. In practice, this situation is very unlikely. This is clear when we compare the worst case observed delay at each switch with the worst case computed delay at each switch.

Switch #	*Max. observed*	*Max. computed*	*Ratio*
2	*0.055*	*0.100*	*0.55*
3	*0.098*	*0.210*	*0.46*
4	*0.094*	*0.150*	*0.63*

Table 2. Maximum Delay at Each Switch.

From the table we see that the computed values better approximate the worst case delay than is apparent from comparing the sum. Our pessimistic assumption that the same cell could suffer the worst delay at each switch leads to the problem. This effect is seen in all the simulations. Since, in practice, we are more interested in the accuracy of the end-to-end delay computation, we will not show per-node accuracy for the other cases.

7.4. Effect of cross traffic

We now study the effect of cross traffic on queueing delays. Table 3 shows the observed and computed values of delays as the cross traffic intensity increases.

We see that as the cross traffic increases, the mean delay increases. The reason is that at switch 3, as the intensity of cross traffic increases, the probability that a cell from source 1 is served after cells from competing sources increases. Clearly, this leads to increased queueing delay at that switch.

Cross Traffic	*Observed Mean*	*Observed Max*	*Bound*	*Mean/ Bound*	*Max/ Bound*
0.1	*0.11*	*0.15*	*0.46*	*0.25*	*0.32*
0.3	*0.11*	*0.18*	*0.46*	*0.23*	*0.39*
0.5	*0.12*	*0.18*	*0.46*	*0.26*	*0.39*
0.7	*0.13*	*0.18*	*0.46*	*0.28*	*0.39*
0.9	*0.14*	*0.18*	*0.46*	*0.29*	*0.39*

Table 3. Effect of cross traffic.

Since the effect of cross traffic is small and predictable, we did the remaining experiments only for a cross traffic intensity of 0.5 of the link bandwidth. The results for higher intensities are similar.

7.5. Effect of bursty sources

In the base case, the source sends a continuous stream of data, a cell every 40ms. Since HRR switches make bandwidth reservations for the average rate of a conversation, if a source is bursty, the queueing delay (due to smoothing at the first few switches) increases. This section examines the increase in delay due to bursty sources.

Sources can be bursty due to one of two reasons: the peak to average ratio can be large, or the interval over which the averaging is done can be large. In terms of $x_{\min}$, x_{ave} and I, this corresponds to a larger $\frac{x_{ave}}{x_{\min}}$ ratio or a larger I. We examine both cases below. First consider the situation where $\frac{x_{ave}}{x_{\min}}$ is fixed at 20, and I increases from 1 through 4 seconds.

Ave. Interval	*Observed Mean*	*Observed Max*	*Bound*	*Mean/ Bound*	*Max/ Bound*
1	*0.59*	*0.95*	*1.2*	*0.49*	*0.83*
2	*0.99*	*1.70*	*1.95*	*0.51*	*0.87*
4	*1.79*	*3.40*	*3.5*	*0.51*	*0.97*

Table 4. Effect of increasing averaging interval.

We see that as the interval increases, the observed and the computed maximum delays increase. This is due to the smoothing that happens at the first few switches - other switches have roughly the same maximum delay. To see this, compare the maximum observed delays at each switch as I increases in the table below.

Note that by the time the third switch in the path is reached, there are no more smoothing delays. In general, smoothing delays will be seen only at the set of switches from the source onwards that have strictly non-increasing bandwidth allocations.

I	*Switch 2*	*Switch 3*	*Switch 4*
1	*0.59*	*.30*	*0.09*
2	*1.14*	*.50*	*0.09*
4	*2.29*	*.95*	*0.09*

Table 5. Maximum observed delays at each switch as *I* increases.

In Table 4 we note that the prediction accuracy is high for bursty sources and the accuracy increases as the input becomes more bursty. The reason is that the smoothing delay is deterministic and exactly computed. As the interval size increases, the smoothing delay dominates all the other delays, and so the algorithm becomes more and more accurate.

The effect of increasing the peak to average rate ratio is observed by keeping *I* as 1 sec, and increasing the ratio from 2 to 20 (Table 6).

Ratio	*Observed*		*Bound*	*Mean/*	*Max/*
	Mean	*Max*		*Bound*	*Bound*
2	*0.35*	*0.50*	*0.75*	*0.47*	*0.67*
5	*0.53*	*0.80*	*1.05*	*0.50*	*0.76*
10	*0.57*	*0.90*	*1.15*	*0.50*	*0.78*
20	*0.59*	*1.00*	*1.20*	*0.49*	*0.83*

Table 6. Effect of increasing peak to average ratio.

We see that as the ratio increases, the observed end-to-end delay increases. However, the increase is not as pronounced as with increasing the averaging interval. This suggests the burstiness caused by changing the peak to average ratio is not too important for HRR networks. The reason is simple: because of per-channel queueing, each conversation sees the server as a dedicated server with vacations. As long as arrivals happen during the vacation, the effective end-to-end delay is constant. Thus, even as the ratio increases, there is no appreciable change in the delay. Further, as in the earlier case, as the source becomes more bursty, the prediction accuracy improves.

7.6. Effect of increasing frame ratios

In this set of experiments the ratio of frame sizes at switches 2 and 4 to the frame size at switch 3 is varied from 1:1 to 1:10. As the frame size of switch 3 increases, we expect the smoothing delays at switch 4 to increase, leading to an increased end-to-end delay. One way to view this is to see that a when a large frame is followed by a smaller frame, it introduces burstiness within the network. The greater the ratio, the more the burstiness, and the greater the smoothing delay. The results of the experiments are summarized in Table 7.

From Table 7, we see that expected increase does happen. The increase is small for a ratio of 2, but the delay doubles at a ratio of 5. Thus, we recommend that conversations be placed at similar frame times at consecutive switches. The prediction accuracy does not improve much for increasing

Ratio	*Observed*		*Bound*	*Mean/*	*Max/*
	Mean	*Max*		*Bound*	*Bound*
1	*0.13*	*0.17*	*0.30*	*0.44*	*0.55*
2	*0.12*	*0.18*	*0.46*	*0.39*	*0.53*
5	*0.26*	*0.36*	*0.85*	*0.42*	*0.56*
10	*0.53*	*0.75*	*1.50*	*0.35*	*0.50*

Table 7. Effect of increasing frame size ratio.

frame sizes. This is because the source is sending perfectly smooth traffic and slippage delays dominate the end-to-end delay.

7.7. Effect of increasing number of hops

In this experiment, we increase the number of hops in the network. This is done by adding two switches to the path of source 1, with frame times of 100ms and 50ms respectively. At each new switch, there is a cross traffic source with intensity 0.5. The results are presented in Table 8.

# hops	*Observed*		*Bound*	*Mean/*	*Max/*
	Mean	*Max*		*Bound*	*Bound*
4	*0.12*	*0.18*	*0.46*	*0.39*	*0.53*
6	*0.32*	*0.36*	*0.90*	*0.36*	*0.40*

Table 8. Effect of increasing number of hops.

From Table 8, we see that as the number of hops increases, the observed delay increases. The increase in mean delay is because of the additional smoothing delay at the second of the added switches. The conclusion is clear: as the length of a path in a HRR network increases, the delay is bound to increase. Thus, routing algorithms have an added incentive to find shortest paths, or better, paths along which frame times are non-decreasing, so that smoothing delays are avoided. Again, in this experiment the source is sending perfectly smooth data, leading to low prediction accuracy. Further, note that the accuracy of the algorithm decreases with the length of the path, reinforcing our belief that it best models bursty sources.

7.8. Buffer size prediction

We compare the buffer sizes predicted by the algorithm to the simulated buffer occupancies for two of the above experimental set ups (Table 9).

Switch #	*Computed buffers*	*Observed buffers*
2	*2*	*2*
3	*4*	*3*
4	*3*	*3*

Table 9. Base Case: Buffer Occupancy.

For the base case the observed buffer occupancies are rather small so the scope for the analysis to be wrong is also limited. Therefore also evaluated peak buffer occupancy predictions for bursty input traffic where queueing delays should cause high buffer occupancies. Table 1o present the buffer occupancy when averaging interval $I = 4$, and $\frac{x_{ave}}{x_{\min}}$ is fixed at 20 (as in section 7.3). The buffer requirement predictions are quite accurate, reflecting our exact analysis of queueing transients.

Switch #	*Computed buffers*	*Observed buffers*
2	*94*	*92*
3	*28*	*28*
4	*3*	*3*

Table 10. Buffer Occupancy for Averaging Interval $I = 4$.

7.9. Summary of Simulation Results

Our simulation scenarios explored a number of causes of end-to-end delay in order to see how close the worst case analysis is to observed maximum and average delays. Due to practical limitations, the observed delays are only indicative of the true values.

We found that when the input is smooth, the prediction accuracy is low since the slippage delays dominate the analysis, and the worst possible slippage is very improbable. However, as the sources become more bursty, the accuracy of the analysis substantially improves. The effect of cross traffic is small - it increases the mean queueing delay, but the increase is bounded by one frametime. The largest change in end-to-end delay is from input burstiness caused by a larger averaging interval. For a fixed averaging interval, changing the peak to average sending rate affects the delay distribution negligibly.

The buffer size predictions are accurate for both smooth and bursty inputs. When the sources are smooth, the buffer sizes achieved are small, and so the scope for inaccuracy is limited. When sources are bursty, the buffer occupancy increases, but so does prediction accuracy.

8. Conclusions

We have presented a formal framework for transient queueing analysis in networks of rate-controlled servers. We use this framework to prove that the worst possible queueing delay occurs when the input is greedy and each server has the maximum slippage. Further, the computation of this worst delay needs to consider only one averaging interval of time. From this result, we develop an algorithm to compute worst case queueing delays and buffer requirements. This requires at most $O(n)$ computation at each node.

The analysis is tight, in the sense that pathological cases can be constructed such that at least one cell suffers the predicted delay at each node. However, since the same cell will not suffer the worst case delay, in general the ratio of end to end observed worst case delay to computed delay is not too high, especially for sources transmitting smooth traffic.

We compare the algorithm developed from analysis with observed mean and maximum delays in simulation experiments. The prediction accuracy improves as the source becomes more bursty. If sources are sending data smoothly within their traffic constraints, the worst case delay they receive can be as little as 40% of the predicted worst case. It rises to 70 to 90% for bursty sources. However, prediction of buffer requirements is accurate, for both smooth and bursty sources.

9. Acknowledgments

We would like to thank C.R. Kalmanek, H. Kanakia and D. Ferrari for their advice and helpful comments.

References

[1]A. Banerjea and S. Keshav, ‘‘A Formal Analysis of Queueing Delays in Rate-Controlled Networks, ’’ *Manuscript in preparation*, 1992.

[2]R.L. Cruz, ‘‘A Calculus for Network Delay, Part I: Network Elements in Isolation, ’’ IEEE Trans. on Information Theory, 37 (1991).

[3]R.L. Cruz, ‘‘A Calculus for Network Delay, Part II: Network Analysis, ’’ IEEE Trans. on Information Theory, 37 (1991).

[4]R.L. Cruz, ‘‘Service Burstiness and Dynamic Burstiness Measures: A Framework, ’’ *Preprint*, Submitted to J. on High Speed Networks, April 1992.

[5]D. Ferrari, ‘‘Client Requirements for Real Time Communication Services,’’ *IEEE Communications Magazine,* 28(11), November 1990, also RFC 1193.

[6]D. Ferrari and D. Verma, ‘‘A Scheme for Real-Time Channel Establishment in Wide-Area Networks,’’ *IEEE Journal on Selected Areas in Communications,* 8(3):368-379, April 1990.

[7]S.J. Golestani, ‘‘A Stop-and-Go Queueing Framework for Congestion Management, ’’ *Proc. ACM SigComm 1990,*pp 8-18, Philadelphia, Pennsylvania, September 1990.

[8]C. R. Kalmanek, H. Kanakia and S. Keshav, ‘‘Rate Controlled Servers for Very High Speed Networks,’’ *Conference Record, GlobeCom9 1990,* December 1990, pp. 300.3.1-300.3.9.

[9]S. Keshav, ‘‘REAL: A Network Simulator,’’ CS Tech. Report 88/472, University of California, Berkeley, December 1988.

[10]J. Kurose, ‘‘On Computing Per-session Performance Bounds in High-Speed Multi-Hop Computer Networks’’ *Proc. ACM SigMetrics/Performance 1992*.

[11]S. Low and P.P Varaiya, ‘‘A Simple Theory of Traffic and Resource Allocation in ATM, ’’ *Conference Record, GlobeCom 1991*, December 1991.

[12]A.K. Parekh and R.G. Gallager, ‘‘A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks,’’ Laboratory for Information and Decision Systems, *MIT Technical Report 2076*, 1991.

[13]A.K. Parekh, ‘‘A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks,’’ *PhD Thesis*, MIT, February 1992.

[14]D. Verma, ‘‘Guaranteed Performance Communication in High-Speed Networks,’’ *PhD Thesis*, University of California at Berkeley, November 1991.

[15]H. Zhang and S. Keshav, ‘‘Comparison of Rate-Based Service Disciplines,’’ *Proc. ACM SigComm 1991*, September 1991.