Queueing Delays in Rate Controlled Networks

Anindo Banerjea

The Tenet Group
Computer Science Division
University of California, Berkeley
and
International Computer Science Institute
Berkeley, California

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 networks of rate-controlled, non-work conserving servers.

The calculations are 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.

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.

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 [ZhaKes91]. By *framed* we mean that the servers work on the basis of an interval, called the *frametime*, during which they allocate a number of *transmission slots* to each channel being served. Each slot corresponds to the transmission of a fixed size packet or *cell* 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 cells (the *chunksize*) in each frame, even if this will leave the output trunk idle. 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.

One example of such servers are those that obey the Hierarchical Round Robin (HRR) service discipline [KKK90]. HRR servers maintain of a hierarchy of round robin frames, so that a choice of frametimes is available during channel establishment. While earlier work considered the behavior of a single HRR server [KKK 90], and of a channel traversing a sequence of identical servers [ZhaKes91], the general case, where the frametime could be different at each server, and the input traffic could be bursty, was not analyzed. This paper determines the upper bound on the end-to-end queueing delays and buffer requirements of a simplex channel established over any sequence of HRR servers. We present a novel graphical analysis technique and compare the predicted delay and buffer requirements with simulation results. The analysis extends to a series of Stop-and-Go servers [Gol90] as well.

2. Model of the Network

In our model a channel is associated with a static path through the network and has state and reserved resources at each switch (node) along this path. We focus our attention on the performance of one channel in the network and will analyze the behavior of this channel by induction. That is, given the traffic characteristics at the end of a section of the channel consisting of the first *i-1* nodes (denoted by $channel_{i-1}$) and the node characteristics of $node_i$, we will calculate the characteristics of $channel_i$ (Figure 1). We develop our network model by modeling the input traffic, the nodes, and the traffic as it flows through the network.

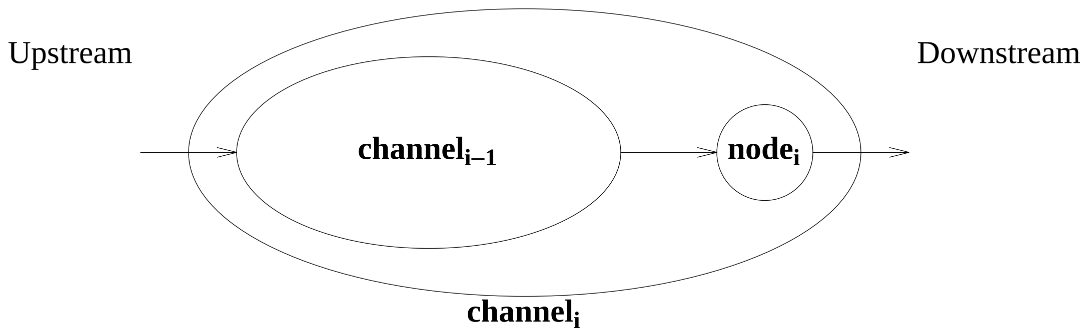

Figure 1. Network Model

2.1. Node Model

For each $node_j$ along the path of the channel, we know the following node characteristics:

a_j *chunksize*, the number of slots allocated to the channel per frame.

F_j *frametime*, the duration of the frame.

B_j reserved rate, the maximum rate at which the node can transmit cells on this channel. ($B_j \equiv a_j / F_j$)

Different channels through a given node will have different values for the above parameters. Since we are concentrating on one channel, we will subscript the parameters just by the node indices.

We assume the physical links between adjacent nodes are acting as constant delay pipes with link delay between $node_{i-1}$ and $node_i$ equal to Δ_i.

2.2. Input Traffic Model

We characterize the traffic entering the channel as suggested in [Fer90, FerVer90]:

$x_{\min}$ 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 most types of rate-controlled traffic expected in high-speed networks [Ver91]. We assume that the user of the channel is bound to obey these restrictions on the input traffic, and the delay and buffer values calculated need only hold if these restrictions are not violated.

2.3. Fluid Approximation

Note that each server is rate-controlled and so will send cells from a channel at its allocated rate until the channel's queue is empty. A simple fluid model, where the output from a server is modeled as a continuous flow at the rate B_j, captures this behavior. This is illustrated in Figure 2, where the X axis shows time elapsed, and the Y axis shows the sequence number of cells sent from $node_j$.

Observe that while the actual traffic is a step function, where one chunksize set of cells are transmitted once every frametime, 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. To get the maximum number of cells which could have been sent out from $node_j$ by time t, we take value given by the straight line approximation and apply the following function to it.

$$\text{fluid\_correction}(y) \equiv \left(\left\lfloor \frac{y}{a_j} \right\rfloor + 1 \right) a_j$$

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 2, it raises the points on the straight line to the dotted line above it. By taking floor and adding one we ensure that for the points of discontinuity in the step function, we take the higher value.

3. Graphical Analysis of Worst Case Behavior

We now introduce a novel technique for analyzing the worst case behavior of a series of rate-controlled servers. Consider the graph of the sequence number of a cell on the output trunk of a node *vs.* time (as in Figure 2). We will use the fluid approximation described earlier. We need only consider the output graph for the worst case arrival pattern at the input, which will repeat every interval I, so the graphs are periodic with period I, and we consider only one period.

The worst case traffic arrival pattern at $node_i$ is the output graph corresponding to $node_{i-1}$. The service at $node_i$ can be represented by a straight line with slope B_i. The output from $node_i$ is then the composition of the input graph and the service graph. The

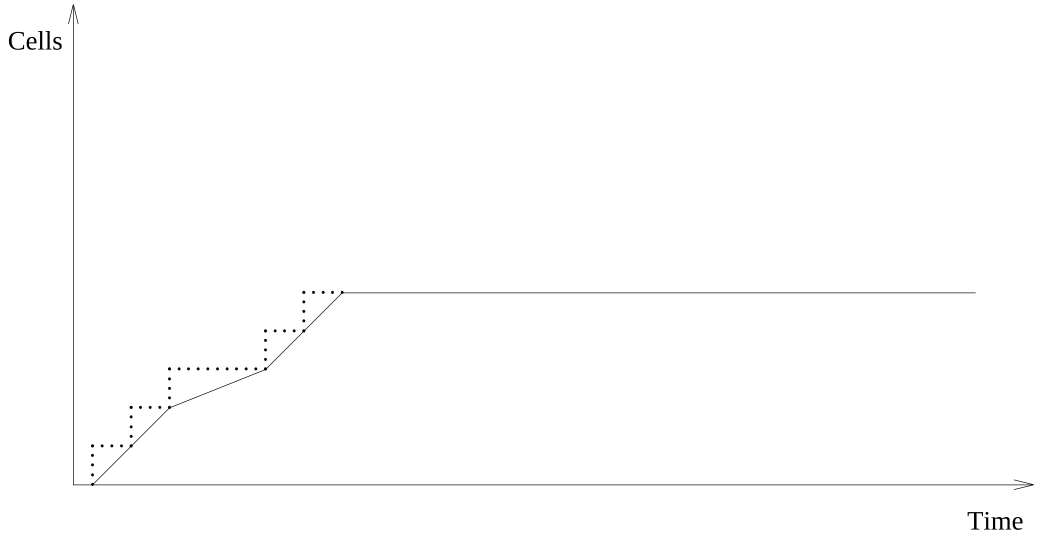

Figure 2. Actual Departure of Traffic vs. Fluid Model

graphical analysis technique can be viewed as a way to combine an input arrival curve and a service curve to produce the output curve for a rate-controlled server. The technique is explained in detail in § 5.

The output graph for $node_i$ encodes information about all the nodes "of interest" up to and including $node_i$. To compute the worst case delay at the next node, we do not need to keep information about all the previous nodes along the path. In the inductive process some nodes cease to be of interest because their effect on the output of the channel is completely masked by a node somewhere downstream of them. For example, if a $node_i$ has a $node_j$ downstream of it, such that $B_j < B_i$, then $node_j$ receives cells from $node_i$ at the higher rate B_i, and transmits them at its own lower rate B_j. So nodes downstream of $node_j$ will not be affected by the output curve of $node_i$ directly.

We use the term *visible* to refer to the set of nodes which cannot be neglected at any given point in the channel. This set changes as we move downstream, since new nodes which come into consideration make some of the earlier ones *invisible*.

We define the *visible envelope* at $node_i$ as the worst case traffic that could come out $channel_i$ (the portion of the channel consisting of the first i nodes) as a function of time. It is represented by the output graph for $node_i$. It can also be thought of as a list of all the nodes *visible* from this point on the channel, together with their reserved rates (B_j) and the period of time they can send data before they must wait from input from some previous node. The visible envelope for $channel_i$ summarizes the information we need to know about the nodes preceding $node_i$. Given this, and the information about $node_{i+1}$, we can calculate the maximum delay and buffer requirements at $node_{i+1}$ as well as the visible envelope for $channel_{i+1}$.

3.1. Example

A simple example of a visible envelope is the description of the input traffic shown in Figure 3. The traffic comes in the maximum possible rate, with inter-cell gaps equal to $x_{\min}$. When I/x_{ave} cells have entered the channel, the input stops because any more cells would violate the average rate restriction on the traffic. (This is often called on-off traffic.)

We can think of this source to consist of one *visible* node, with a reserved rate of $1/x_{\min}$ which is throttled to rate of 0 after $\dfrac{I * x_{\min}}{x_{ave}}$ time units due to an empty queue.

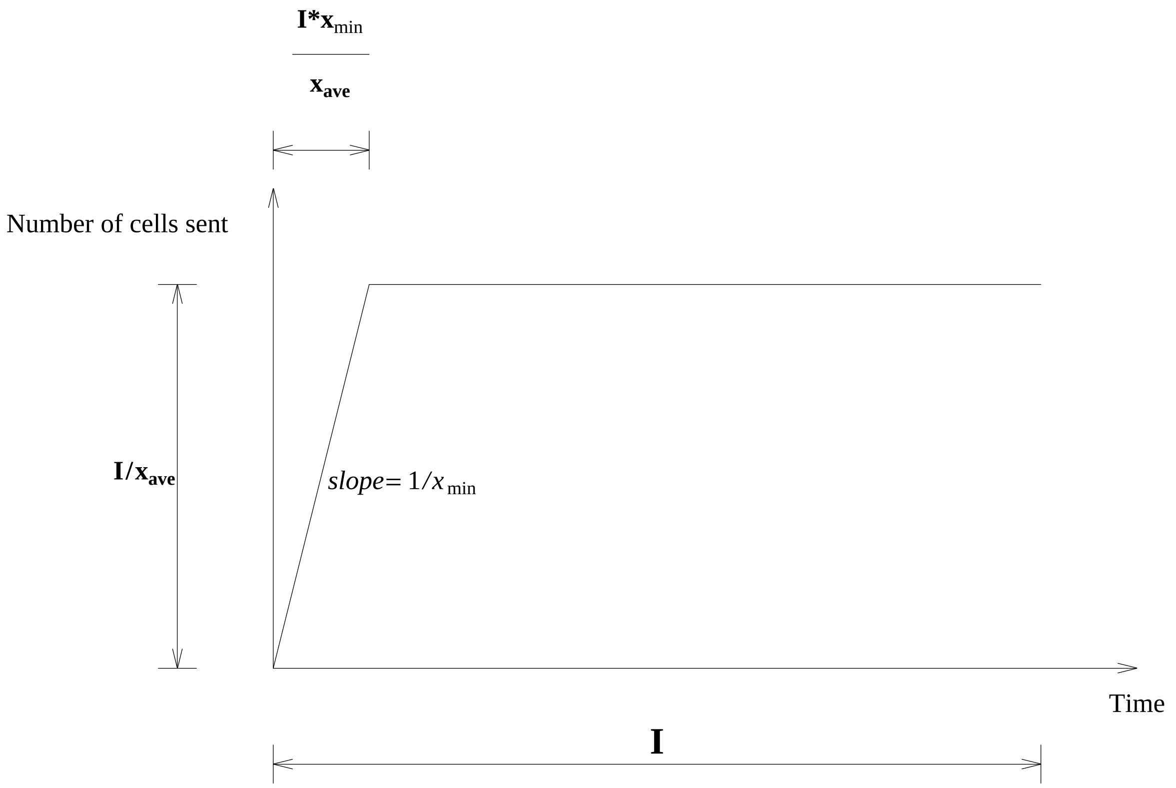

Figure 3. Worst Case Input Pattern

This is the worst case input traffic within the constraints of the input traffic parameters.

The worst case traffic pattern is periodic with period I. To see this we only need note that the restriction on x_{ave} is over **any** interval of length I. Thus if we have a burst of traffic with inter-cell gaps equal to $x_{\min}$ for a time period $\dfrac{I * x_{\min}}{x_{ave}}$ then we must have a silent period of length $I - \dfrac{I * x_{\min}}{x_{ave}}$ immediately following it *and immediately preceding it*. The worst case is to have another burst just before and after the silent period. Thus the sequence of bursts repeat at intervals of length I.

3.2. Slippage

After the first cell of the input traffic arrives at an empty queue for the channel at a given $node_i$, a maximum time F_i can elapse before the next frame begins and the channel becomes *active* at $node_i$. By *active* we mean that the service mechanism on $node_i$ is aware of a non-empty queue for the channel, and by the definition of *framed, non-work-conserving* servers $node_i$ must send out the allocated number of cells in each frame from this point in time until the queue becomes empty again. Then the server may put the channel to a non-active status, and the first cell to arrive after that point may observe the phenomenon of *slippage* again.

We use the term *slippage* to refer to the phenomenon and also to this time interval. During this time, cells build up in the input queue at the rate determined by the graph for the input traffic to $node_i$ (which corresponds to the visible envelope of $node_{i-1}$). After this time, the node starts sending out cells at its reserved rate B_i (or a_i cells in every F_i time units) and the visible envelope of $channel_i$ rises above the x-axis. Due to slippage the x-intercept of the visible envelope of $channel_i$ is different from that of the visible envelope of $channel_{i-1}$.

4. Assumptions

The analysis in the following sections makes some assumptions which we shall state and justify in this section. These are:

1. The reserved service rate at each $node_i$ (B_i) is not less than the average input rate ($1/x_{ave}$).
2. The input pattern is the burstiest possible one, given the values of $x_{min,}$ x_{ave} and I.
3. The slippage at each node is the maximum possible, F_i.
4. The effect of cells from previous intervals of time can be ignored in analyzing the current interval of duration I.

The first assumption is required, since otherwise the defaulting node would quickly overflow its buffers.

As the input pattern becomes more bursty, the queueing delays in the system become larger. The second assumption means that we will do the analysis for the worst case possible, given the input traffic specification. The burstiest possible input pattern is shown in Figure 3. As described before it is periodic with period I. In § 7 we rigorously prove that this pattern is indeed the burstiest possible.

Slippage occurs for the first cell sent by a channel. We assume the largest possible slippage at each node. This allows the most initial build-up at this node, and consequently the highest queue build-ups. It also introduces the highest end-to-end delay. Further, this node will send out cells at its maximum rate B_i for the longest possible time (because it had the largest possible queue build-up). This makes the situation the worst for downstream nodes also. Thus we are correct in saying that assuming the largest possible slippage accounts for the worst case.

The fourth assumption is best justified after presenting the details of graphical analysis in § 5. So, this justification is presented in § 7.

With these assumptions in hand, we now present the computation of the visible envelope for a channel.

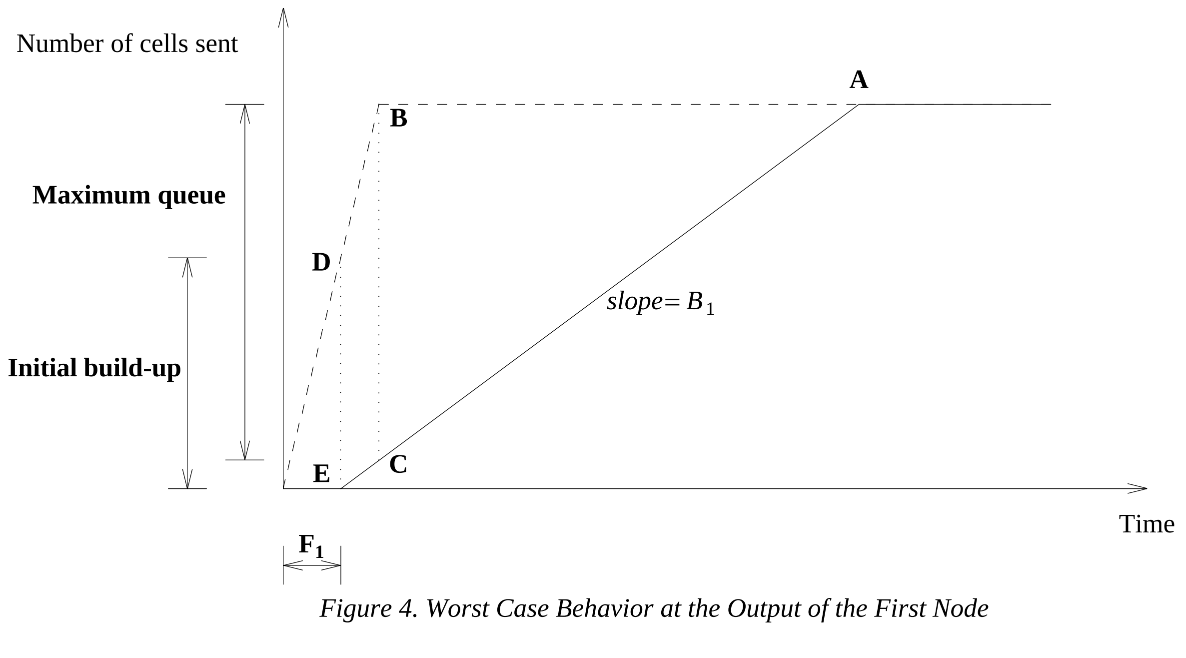

Figure 4. Worst Case Behavior at the Output of the First Node

5. Calculation of the visible envelope

Under the assumptions of the previous section, we shall present the analysis of the queueing delays resulting from the input pattern of Figure 3, as it passes through the nodes in the channel. We first consider the situation at the first node along the path.

5.1. Analysis of the first node

The worst case situation in the first node can be seen in Figure 4. The worst case input pattern is shown as the input to $node_1$ as a dashed curve. After the first cell arrives at the node, a maximum slippage time F_1 passes before the channel becomes active in $node_1$. During this time the cells build up in the queue at the channel's rate of input to the node. Then the node sends out cells at the rate B_1 until its queue becomes empty at point **A**. From this point onwards the output rate from this queue is limited by input rate to this queue. In this case, no more cells are output until the end of the interval I. The solid line shows the visible envelope of $channel_1$. It summarizes the worst case traffic on the link downstream of $node_1$.

The queue length at any time is given by the difference between the input and service curves at that point on the time axis. Of course this is under the assumption of a fluid model, and to get the actual number of cells which could have arrived in that time we need to apply the fluid_correction defined in § 2.3 with $j \equiv 0$*. The buffer requirement is simply the maximum possible queue buildup at this node under worst case assumptions, so we need to find the point on the visible envelope for which the above function is maximized. In this case the maximum buffer required at this node is given by fluid_correction($\overline{BC}$).

The graph also tells us the worst case queueing delay at this node. There is a very simple relationship between the queuelength of any point on the visible envelope and the delay seen by a cell which arrived at that point. Let us say the queuelength is *queuesize*. Now, in each frame time F_1, a_1 cells are transmitted. So, the worst case time to empty this queue is simply $\left\lceil \frac{queue\_size}{a_1} \right\rceil F_1$. The cells which arrive during slippage also suffer an additional delay waiting for the server to become active. The worst slippage is for the first cell from the channel to arrive at the node. For this cell the additional delay is F_1. In general the additional delay of a cell arriving at time t after the arrival of the first cell is given by the time to the end of the slippage interval ($F_1 - t$). Cells arriving after the end of the slippage interval suffer no additional delay.

In this example the worst case delay also corresponds to point **B** and the value is $\left\lceil \frac{\text{fluid\_correction}(\overline{BC})}{a_2} \right\rceil {*F_2}$. In general if we calculate the delay and buffer values for each point on the visible envelope and take the maximum of each over all the points on the visible envelope we should have the worst case delay and buffer requirements.

5.2. Subsequent Nodes

Subsequent nodes can also be analyzed in the same way. For example the situation for the second node is as shown in Figure 5. Data arrives at $node_2$ as described by the graph for $channel_1$ (the dashed line in Figure 5). No service is received till point **C**, which is the end of the slippage period. After that service is received at the rate B_2 until the queue becomes empty at **A**. Here onwards the number of cells going out is limited by the number of cells coming into the node. Thus the output curve for $channel_2$ follows the visible envelope of $channel_1$. The maximum queue build-up is fluid_correction($\overline{BC}$),

* For the purpose of calculating fluid_correction $a_0 \equiv 1$ (and thus $F_0 \equiv x_{\min}$).

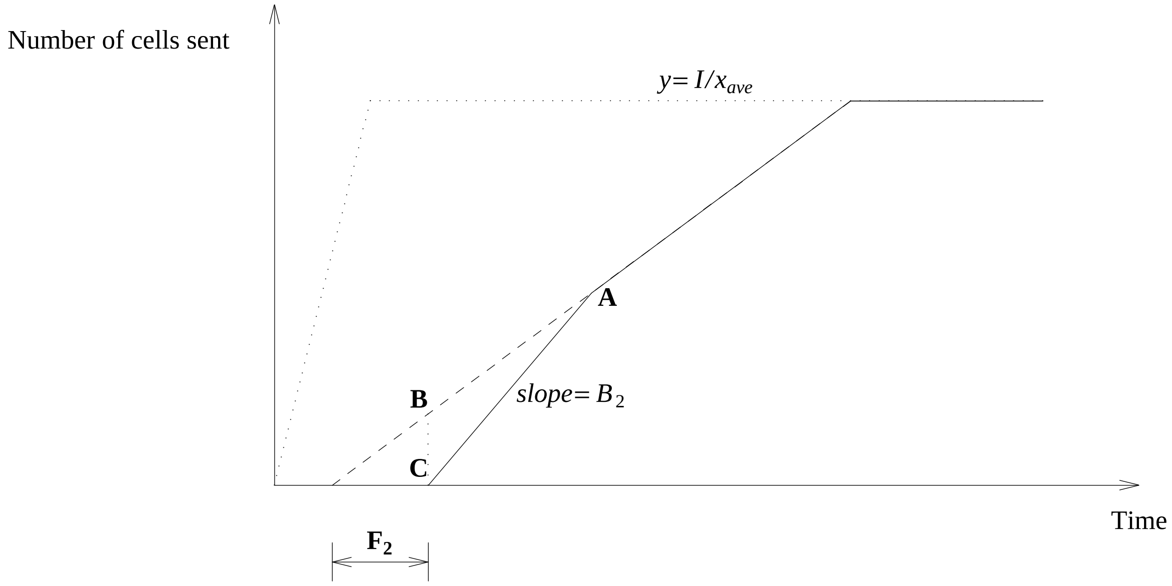

Figure 5. Worst Case Behavior of Second Node

and

the maximum delay is the greater of $\left\lceil \dfrac{\text{fluid\_correction}(\overline{BC})}{a_2} \right\rceil * F_2$ (corresponding to point B) or $\left\lceil \dfrac{\text{fluid\_correction}(0)}{a_2} \right\rceil * F_2 + F_2$ (corresponding to the first cell to arrive). In general, we must compute the delays for all the points separately, and choose the maximum.

Notice that the dotted part of the graph is irrelevant as far as the analysis of $node_2$ goes. The worst case behavior of $channel_1$ (the channel up to and including $node_1$) is summarized by the visible envelope of Figure 4, shown in Figure 5 as a dashed line. The behavior of $channel_2$ is summarized by the visible envelope of Figure 5. This is all that is needed for subsequent nodes, the rest of the information can be discarded.

Notice also that the visible envelope consists, in effect, of a list of all the visible nodes (each segment of the envelope with a constant slope corresponds to a visible node) together with information about the node bandwidth (the slope of the segment) and the length of time the node can transmit at its reserved rate before it must slow down for some previous node.

One point to bear in mind is that the delay over the physical link must also be accounted for while adding up the end to end delays. However, it does not enter the calculation for queue build-ups and queueing delays, if we assume links are behaving as constant delay pipes. The time $node_i$ receives the first cell is Δ_i after the time that $node_{i-1}$ actually sends the cell out, where Δ_i is the link delay over the link from $node_{i-1}$ to $node_i$. However, since the second (and all subsequent) cells also get delayed by the same amount, it does not affect the queue build-ups. In terms of the graphical analysis, the action at the second node occurs shifted to the right on the time axis by Δ_i, but since all events at $node_i$ are shifted identically, we can ignore this effect in calculating the queue build-ups and the queueing delays. We must, of course, add the physical delay in when we are adding up the end to end delay.

6. Formal Algorithm

The visible envelope for $channel_i$ is represented as a list of the points on the envelope (<x,y> coordinate pairs), sorted by y coordinates. The initial envelope describing the worst case input traffic contains three points: (0,0), $(\frac{I\,x_{\min}}{x_{ave}}, \frac{I}{x_{ave}})$, and $(\infty, \frac{I}{x_{ave}})$. To simplify calculations and reduce the number of points we need to carry in the visible envelope, we normalize the envelope on starting the calculations for each node so that the x-intercept of the output envelope is always at the origin. The calculation of the visible envelope for $node_i$ consists of the steps described in the pseudocode of Figure 6.

7. Graphical Justification for Assumptions

Two of the assumptions made in § 4 can be better justified using graphical ideas developed in the preceding discussion. These are:

- The input traffic is the burstiest possible.
- The effect of cells from previous intervals can be ignored.

We first justify our choice of input pattern. In terms of the graphical analysis, we can think of all other input patterns as derivable from the pattern of Figure 3 by shifting the pattern along the x axis, or lowering the envelope of the input pattern by reducing the slope of the input.

Shifting the pattern does not change the behavior of the system because we can shift the origin of the analysis. The restriction on the input traffic, that the average cell interarrival time over *any* interval I must be no less than x_{ave}, implies that no more cells can enter the system after I/x_{ave} have arrived, regardless of where we place the origin.

Changing the slope of the input can only move the curve of the input within the envelope of Figure 3. Consider the effect of this on some $node_i$. If the change moves the input curve above the visible envelope of $channel_{i-1}$ then there is no effect on the behavior of $node_i$. If the new input curve touches the visible envelope of $channel_{i-1}$, lowering it will cause the visible envelope of $node_i$ to lower. This will cause the queue lengths at $node_i$ to go down.

Once the source has sent a burst of $\frac{I}{x_{ave}}$ cells it cannot send any more cells for the remaining portion of the period of length I. We also could not have received any cells in the previous period of the same length. By the arguments of the preceding paragraphs, an identical burst in the next and preceding interval is worst for this channel. This shows that the worst case input is periodic with period I and has the shape shown in Figure 3.

We have also neglected the effect of cells from the previous interval on the current one. Because of the periodic nature of the pattern, we might equally well consider the effect of this interval on the next one. We now show that for the worst case input pattern, cells of the current interval do not interfere with the cells of the next interval.

The last cell of this interval is transmitted from $node_i$ at the time corresponding to the x coordinate of the point where the visible envelope touches the horizontal line $y=I/x_{ave}$. The first cell of the interval is transmitted at the x-intercept of the envelope. These two points are joined by a series of segments each with a slope B_j for each $node_j$. Since each node is assumed to have $B_j > 1/x_{ave}$, the interval between the time the node transmits the first cell of this interval, and when it transmits the last cell of this interval is less than I. Thus by the time the first cell of the next interval arrives, the last cell of this interval must have left. Thus for the worst case input, the cells of one interval do not interfere with the cells of the next interval. Thus neglecting the effect of other intervals does not make our analysis invalid.

#define fluid_correction(y) $\left(\left\lfloor \frac{y}{a_{i-1}} \right\rfloor + 1\right) * a_{i-1}$

max_buffer = max_delay = 0;
for p = each point in the visible_envelope, having coordinates x and y {
  q = previous_point (p)
  $p.x = p.x$ - F_i /* Normalize the point, F_i is the slippage */
  if ($p.x < 0$) {
    /* delay of cells that arrive before start of service */

    $buffer$ = fluid_correction($p.y$)
    $delay = \left\lceil \frac{buffer}{a_i} \right\rceil * F_i - p.x$ /* $-p.x$ adds the remaining slippage */
  }
  else if ($p.x > 0$ and $q.x \le 0$) {
    /* For the point over the origin */
    $buffer$ = fluid_correction($\frac{p.x*q.y - q.x*p.y}{p.x - q.x}$)
    $delay = \left\lceil \frac{buffer}{a_i} \right\rceil * F_i$
  }
  else if ($p.x < \infty$) {
    /* for all other valid points */
    $buffer$ = fluid_correction($p.y$)$- B_i * p.x$
    if ($buffer < 0$)
      break
    $delay = \left\lceil \frac{buffer}{a_i} \right\rceil * F_i$
  }
  max_delay = max(max_delay, $delay$)
  max_buffer = max(max_buffer, $buffer$)
}
/* Throw away all points below the line $y = Bx$ */
$visible\_envelope = p$
/* Find the intersection of the envelope with the line $y = Bx$ */
q = previous_point(p)
$i.x = \frac{p.x*q.y - q.x*p.y}{q.y - p.y + B_i*(p.x - q.x)}$
$i.y = B_i * i.x$
add_to_envelope(i)
add_to_envelope(point(0,0))

Figure 6. Calculations for node$_i$

8. Discussion of Pessimistic Assumptions

The assumptions we have made for our analysis are worst case assumptions but they account for a case which will almost never happen in practice. For one thing, by adding up the worst case delays for each node, we get an upper bound on the worst case end to end delay, but in practice the same cell will not suffer the worst case at each node, so the sum of the per node worst case delays will be a loose upper bound on actual end to end delays.

In addition we have assumed in our analysis that the cells that arrive during the slippage period suffer the maximum slippage (F_i) and *also* get transmitted at the end of their transmission frames, leading to a 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 and additional F_i delay it must get transmitted at the very end of its frame. The only way this could happen in an HRR server is if all the other conversations at that server also have empty input queues, and cells for those conversations arrive before this cell, but after the start of the frame (which means almost simultaneously because we assumed the cell arrived very close to the start of the frame), causing them to be put in the list of conversations to be served ahead of the conversation we are interested in. This is a very unlikely event.

To get feel for what kind of maximum delays would be observed in practice we turned to simulation.

9. Simulation Results

The previous sections have presented a worst case analysis of the queueing delays suffered by the cells of a virtual circuit passing through a series of non-work conserving, rate-controlled switches. In this section, we compare the results of the analysis with simulation results for a conversation that sends data through a series of HRR switches. In addition, simulations allow us to determine the average case delay, which in many cases is more important, as well as the delay jitter introduced by HRR switches (which is not computed by our analysis).

We study three metrics in our simulations:

a) the ratio of the mean delay to the computed delay bound.
b) the ratio of the worst observed delay to to computed delay bound.
c) the delay jitter.

The first metric tells us how pessimistic the worst case computation is. The second metric tells us how tight the analysis is. The delay jitter corresponds to the buffering that is needed at the end system - the larger the delay jitter, the larger the buffering that is needed by a destination that removes data from the network at a constant rate. Since HRR servers do not provide delay jitter guarantees, we would like to see how much delay jitter is introduced by doing HRR instead of, say, Stop-and-Go service.

All simulations were performed using the REAL simulator [Kes88]. The simulation topology is simple - a series of switches from a source to a destination (Figure 7). Sources send rate-controlled traffic, that is, traffic that can be characterized by the parameters $x_{\min}$, x_{ave} and I. Node 1 is a rate-controlled source that sends data on-off, with an adjustable peak rate, average rate and averaging interval. Nodes 2, 3 and 4 are switches that do HRR service. Node 5 is the destination. At each switch, there is cross traffic from sources 6 and 8. These sources are also rate controlled, but within these constraints, they send data at random intervals of time to corresponding destinations 7 and 9.

All cells are 100 bytes, and all lines are 400,000 bits/sec (500 cells/sec). Switches are assumed to have infinite buffering, but the buffers used are traced for comparison against the predicted buffer requirements. Since end-to-end delays and jitters are computed using a histogram with a bucket size of 15ms, the values presented are accurate only to plus or minus 15ms (= 0.015s). The delay values at an individual switch are accurate to 1μs.

There are several factors that can affect the actual end-to-end delay of a cell. These include: the ratio of $x_{\min}$ to x_{ave} at the source (since reservations are made at the average rate), the averaging interval I, the number of hops from the source to the destination, the ratio of the frame times at adjacent switches, and the intensity of cross traffic (which affects the delay within a frame). 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.

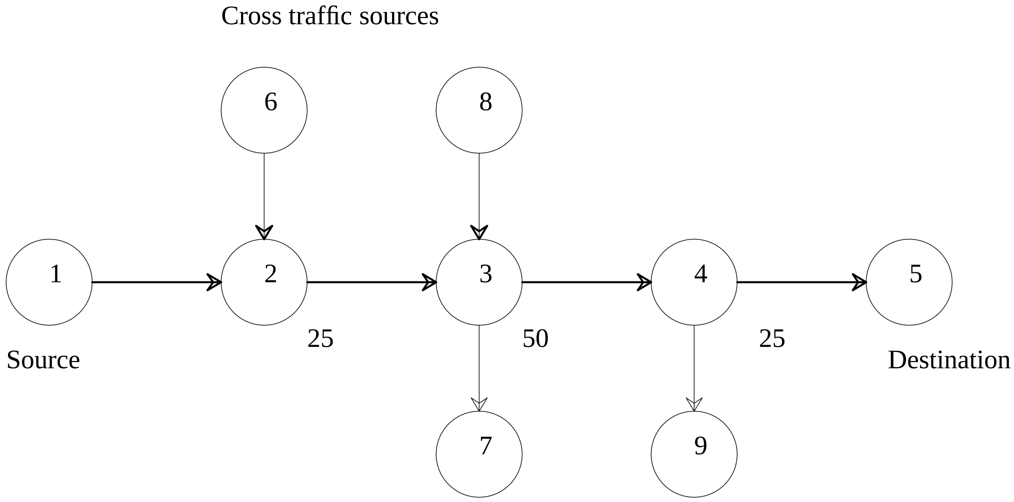

Figure 7. Simulation Scenario

One tricky problem with running a simulation is to know when to stop. We determine the length of the simulation by looking at the end-to-end delay frequency polygons (which are obtained by joining midpoints of histogram bars) over a series of time periods (this is shown in Figure 8 for the base case defined below, and cross traffic intensity of 0.5 of the link bandwidth).

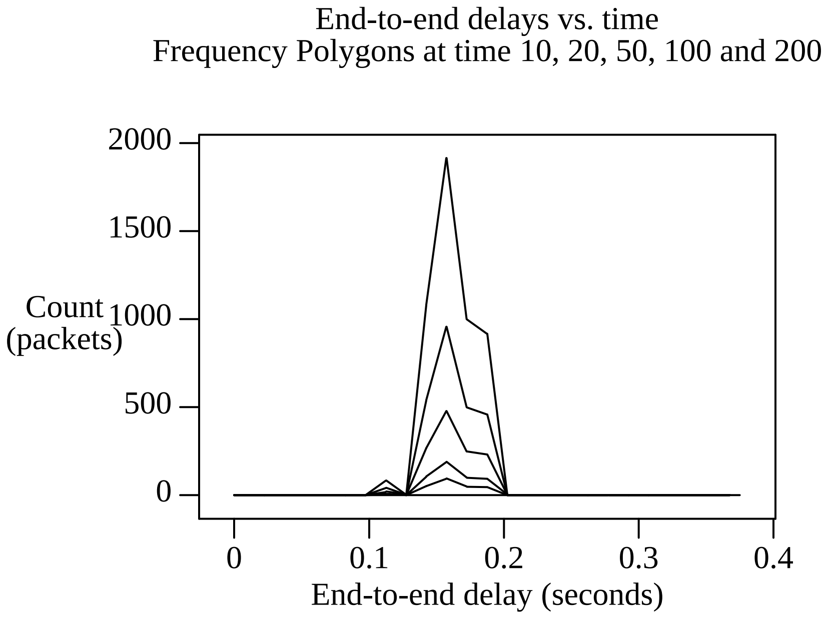

Figure 8. Frequency polygons for end-to-end delays vs. time

Note that a) the polygon does not spread with time, only the central part of the polygon rises as the simulation proceeds and b) the relative heights of each part of the polygon are constant at each time interval. Observation (a) indicates that the delay jitter value stabilizes by 10 seconds, and observation (b) shows that mean delay also stabilizes by this time. For all the scenarios we found that the corresponding polygons show this behavior well before 50 seconds of simulated time, so all the results are shown for simulations that run for 50 seconds. (These observations were also confirmed numerically.)

9.1. 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). 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	Mean (s)	Max. observed (s)	Jitter (s)	Max. computed (s)	Mean/computed (ratio)	Max/computed (ratio)
Base	0.12	0.18	0.09	0.46	0.24	0.39

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

We note that the observed worst case delay is much smaller than the computed worst case 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 (§ 8).
- The worst case computation assumes that the same cell could get the worst possible delay at each switch. In practice, different cells get the largest possible delay, and so the bound is higher than it need be. 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.

Also note that in the base case, the jitter is around 90ms (the accuracy of the histogram used to compute jitter is only ±15 ms). This is much smaller than twice the largest frame time (= 2* 100 = 200ms). So, we find that the delay jitter provided in this case by HRR is comparable to that of Stop-and-Go, which would have provided a tight delay jitter bound of 200ms.

9.2. 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, cells from source 1 are allocated slots after cells from the cross traffic source 6 (since the setup packet from source 6 reaches the switch first). As the volume of cross traffic increases, the number of slots allocated to it increases, and so the cells from

Cross traffic	Mean	Max. observed	Jitter	Max. computed	Mean/computed	Max/computed
0.1	0.11	0.15	0.09	0.46	0.25	0.32
0.2	0.11	0.18	0.12	0.46	0.23	0.39
0.3	0.11	0.18	0.12	0.46	0.23	0.39
0.4	0.11	0.18	0.11	0.46	0.24	0.39
0.5	0.12	0.18	0.09	0.46	0.26	0.39
0.6	0.13	0.18	0.06	0.46	0.27	0.39
0.7	0.13	0.18	0.06	0.46	0.28	0.39
0.8	0.13	0.18	0.06	0.46	0.29	0.39
0.9	0.14	0.18	0.06	0.46	0.29	0.39

Table 3. Effect of cross traffic

source 1 are served later and later in the frame. Clearly, this leads to increased queueing delay at that switch. However, the delay jitter does not appreciably increase, and stays within 120ms.

Since the cells from source 1 are always served after all the cells from the cross traffic source, we are, in some sense, simulating the worst case behavior at the switch. In reality, source 1 cells would be interleaved with the cells from all the other cross traffic sources. However, cross traffic sources would probably send data independently, so the amount of jitter introduced in source 1 cells in these simulations due to cross traffic is roughly the same as it would be in the average case. So, we do not expect appreciably higher delay jitters than shown in these simulations.

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.

9.3. 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. The results are shown in Table 4.

I	Mean	Max. observed	Jitter	Max. computed	Mean/computed	Max/computed
1	0.59	1.00	0.90	1.2	0.49	0.83
2	0.99	1.70	1.60	1.95	0.51	0.87
4	1.79	3.40	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 delays increase. This is due to the smoothing that happens at the first few switches - other switches have roughly the same worst case delay. To see this, compare the maximum observed delays at each switch as I increases in the table below.

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*

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.

In Table 4 we note that a) the prediction accuracy is high for bursty sources and b) the prediction accuracy increases as the input becomes more bursty. The reason for (a) is that the algorithm assumes that the input is the worst possible. In the base case, the source was smooth, and so the observed delay was not as much as it could have been. Here, the source behaves on-off, which is the worst that it could, and so the prediction is correspondingly better.

The reason for (b) is that the computation of the smoothing delays is the most accurate part of the algorithm. As the interval size increases, the smoothing delay dominates all the other delays, and so the algorithm becomes more and more accurate.

Finally, we see that the delay jitter for bursty sources is very large (it is almost as large as the end to end delay). This is because of the effect of traffic smoothing. The first cell that goes through smoothing has no queueing delay, while the last cell has the most, so that the delay jitter and queueing delays are about the same. This jitter can be avoided by reserving bandwidth at the peak rate (which is wasteful). We believe that it is impossible for any realizable scheme to reserve bandwidth at the average rate and still avoid the delay jitter due to smoothing the input (of course, if the input is assumed to be smooth, as in the earlier case, then delay jitter is hardly a problem. So, schemes like Stop-and-Go that postulate smooth input traffic seem to be avoiding the difficult problem of delay jitters due to smoothing!).

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	Mean	Max. observed	Jitter	Max. computed	Mean/computed	Max/computed
2	0.35	0.50	0.35	0.75	0.47	0.67
5	0.53	0.80	0.70	1.05	0.50	0.76
10	0.57	0.90	0.80	1.15	0.50	0.78
20	0.59	1.00	0.90	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 or the jitter. Further, as in the earlier case, as the source becomes more bursty, the prediction accuracy improves. The explanation is exactly as before.

9.4. 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.

Ratio	Mean	Max. observed	Jitter	Max. computed	Mean/computed	Max/computed
1	0.13	0.17	0.09	0.30	0.44	0.55
2	0.12	0.18	0.09	0.46	0.39	0.53
5	0.26	0.36	0.19	0.85	0.42	0.56
10	0.53	0.75	0.40	1.50	0.35	0.50

Table 7. Effect of increasing frame size ratio

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. Further, the delay jitter also rises. Thus, we recommend that conversations be placed at similar frame times at consecutive switches.

The prediction accuracy does not improve much for increasing frame sizes. This is because the source is sending perfectly smooth traffic.

9.5. 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	Mean	Max. observed	Jitter	Max. computed	Mean/computed	Max/computed
4	0.12	0.18	0.09	0.46	0.39	0.53
6	0.32	0.36	0.12	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 and delay jitter increase. The increase in mean delay is because of the additional smoothing delay at the second of the added switches. This smoothing also increases the delay jitter. The conclusion is clear: as the length of a path in a HRR network increases, both the delay and delay jitter are 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.

9.6. 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. For the base case:

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

Table 9. Base case: buffer occupancies

Table 9 shows buffer sizes in cells. For the base case the observed buffer occupancies are rather small so the scope for the algorithm to be wrong is also limited. Therefore we look at the buffer predictions for the bursty input traffic where the queueing delays should cause high buffer occupancies. For the experiment with averaging interval I=4, with $\frac{x_{ave}}{x_{\min}}$ fixed at 20 (section 8.3), we observe:

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

Table 10. Buffer Occupancy for Averaging Interval I=4

Buffer requirement predictions look satisfactory. This reinforces our belief that the smoothing delay and buffering portion of our calculation is the most exact part of our analysis, and the looseness in our calculation comes from the approximation of the slippage delay.

10. Conclusions

We have presented an algorithm to compute worst case queueing delays and buffer requirements for a conversation that sends data through a series of non-work conserving, rate-controlled switches. This requires at most $O(n)$ computation at each node. The actual computation should be less, because many (invisible) nodes need not be considered.

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.

The prediction accuracy improves as the source becomes more bursty, since the algorithm assumes the worst case input. If sources are sending data smoothly within their traffic constraints, the worst case delay they receive can be as little as 40% of the computed worst case. It rises to 70 to 90% for bursty sources. However, prediction of buffer requirements is accurate, for both smooth and bursty sources.

11. Acknowledgments

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

12. References

[Fer90] D. Ferrari, ‘‘Client Requirements for Real Time Communication Services,’’ *IEEE Communications Magazine*, 28(11), November 1990, also RFC 1193.

[FerVer90] 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.

[Gol90] S.J. Golestani, ‘‘A Stop-and-go Queueing Framework for Congestion Management,’’ *Proc. ACM SigComm 1990*,pp 8-18, Philadelphia, Pennsylvania, September 1990.

[KKK90] C. R. Kalmanek, H. Kanakia and S. Keshav, ‘‘Rate Controlled Servers for Very High Speed Network,’’ *Conference Record, GlobeComm ’90,* San Diego, December 1990, pp. 300.3.1-300.3.9.

[Kes88] S. Keshav, ‘‘REAL: A Network Simulator,’’, CS Tech. Report 88/472, December 1988.

[Ver91] D. Verma, ‘‘Guaranteed Performance Communication in High-Speed Networks,’’ *PhD Thesis*, University of California at Berkeley, November 1991.

[ZhaKes91] H. Zhang and S. Keshav, ‘‘Comparison of Rate-Based Service Disciplines,’’ *Proc. ACM SigComm 1991*, September 1991.