Bounded Delay for Weighted Round Robin

Sponsor: Sprint

Kert Mezger
David W. Petr

Technical Report TISL-10230-07

Telecommunications and Information Sciences Laboratory
Department of Electrical Engineering and Computer Sciences
University of Kansas

May 1995

Contents

1 Introduction

Average delay is a common measurement in determining whether or not telecommunication network customers are getting the quality of service they require. This measure is inadequate, however, when a traffic application is time sensitive. In other words, a customer observing an application which has a noticeable quality of service difference above and below a delay bound will not worry as much about the average delay as keeping the delay below the bound. The objective of this investigation is to analyze systems in which bounded or maximum delays can be guaranteed. This allows network providers to guarantee quality of service to a customer without resorting to statistical quality of service descriptions.

1.1 Weighted Round Robin Resource Allocation

The system under analysis is the weighted round robin resource allocation (queuing) method. The algorithm of this system is such that each customer is guaranteed a certain amount of bandwidth. If a customer does not use his bandwidth at a specific time, it is automatically divided up among those customers that need more bandwidth until the original customer needs his allotted bandwidth. Since we are assuming an ATM network, the bandwidth is allocated in a slotted format. We assume each virtual circuit (VC), or customer, has its own FIFO queue in each node as seen in Figure 1. Our goal is to find the maximum delay that an ATM cell will experience through

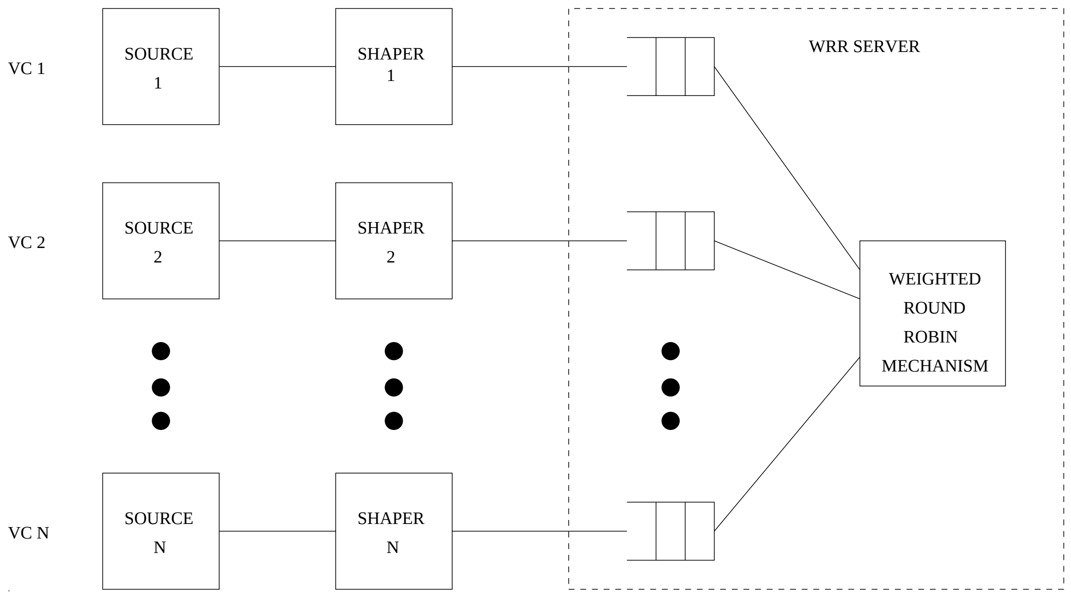

Figure 1: System Configuration

a single WRR server (node) under certain reasonable traffic assumptions. Under these assumptions, the WRR server will then provide *both* minimum bandwidth and maximum delay guarantees.

At the beginning of a slot, a schedule is checked to see which customer is next to be served. If that customer's queue is empty, the next customer in the schedule is checked. If the next customer's queue is also empty, the next in the schedule is checked and so forth. The schedule is set up such that the bandwidth guarantees are preserved (i.e. s = the number of slots in the schedule; t = total slots in the schedule; s/t = guaranteed bandwidth). The schedule is set up in a circular fashion. Once the last customer on the schedule is served, the schedule is reset to the top and the algorithm goes through the schedule again [1].

The format of the schedule can be set up in one of two ways. One way is to give a customer all of their service slots contiguously. In this method, which we call block scheduling, each customer has a block of slots in which to have traffic (ATM cells) served and then waits while all the other customers have a chance to have their traffic served. The second way is to distribute the dedicated slots throughout the schedule. Compared to back-to-back block scheduling, this distributed scheduling serves fewer customer cells back-to-back yet waits a shorter amount of time between servicing times [2].

1.2 System Configuration and Assumptions

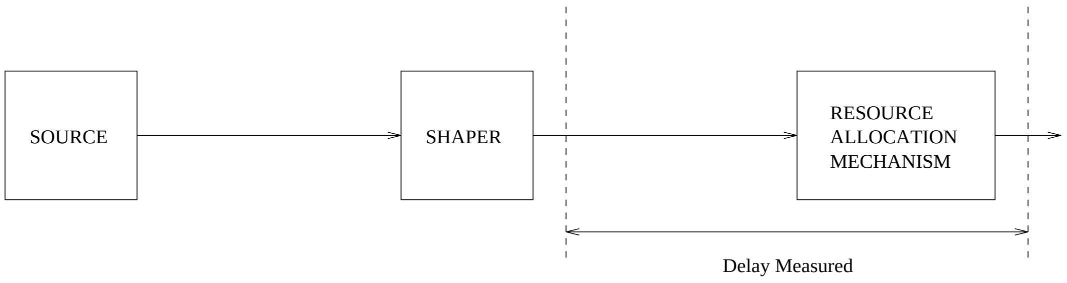

Figure 2: Delay Measurement

An analysis of the weighted round robin resource allocation method would show a maximum delay approaching infinity assuming (1) an infinite buffer for a customer, (2) that the customer offers more traffic than their guaranteed bandwidth allows, and (3) all other customers are using all of their guaranteed bandwidth. For this reason, we will assume that a traffic shaping device on the customer's entrance into the network will control the overloading. This system configuration (Figure 1) will provide a reasonable framework in which to analyze the bounded delay of traffic *after* it enters the network system. In other words, the delay is calculated from the point right after the cell leaves the shaper until the the point when the cell is serviced through the weighted round robin mechanism (see Figure 2).

Two types of shapers are analyzed in this study. One is the sliding window (or i/m) shaper [3]. Within a certain length time window (m, measured in cell slots), up to a specified number of cells (i) are allowed to exit the shaper. As time moves along, the window moves also so that the current slot coincides with the last slot of the window. If the shaper buffer is non-empty, the shaper checks to see if the number of cells transmitted in the last m slots is less than i. If so, one cell will be released from the shaper in the current slot. In our system, we always make the ratio i/m equal to the percent bandwidth guaranteed by the WRR for that customer (s/t).

The second type of shaper is the leaky bucket. It shapes the traffic through a token system. A "bucket" with a specified size (B) holds these tokens. As traffic enters the shaper, it is sent on to the weighted round robin if a token is in the bucket (one cell per slot) and the token is removed from the bucket; otherwise the cell is queued in the shaper until a token is available. Tokens are replenished in the bucket at a constant rate (R). This rate (properly normalized) was always set to be equal to the guaranteed bandwidth of the customer under full load conditions. If a token comes when the bucket is full, that token is dropped. With proper parameter metering (i.e. i/m = s/t and R/C = s/t; C = link rate in cells/second), proper shaping with either shaper is implemented and maximum delay bounds can be calculated based on the parameters of the shaper and the weighted round robin mechanism.

The analysis assumes that there are multiple customers in the system, and all the bandwidth is being used. Hence, the customer under study is only able to be serviced using their guaranteed bandwidth. In order to find a delay bound for a particular system, a worst case traffic condition must be found. It would seem that the worst case condition would be a highly bursty source. In fact, the most bursty source would be a source that is silent for a long period of time and then produces a packet of an infinite length (this burst would be shaped prior to the server as indicated in Figure 1). This traffic situation is assumed to be the worst case and is used to determine the analytical delay bounds in the four cases considered below. The four delay bound cases under study are the combinations of a sliding window or leaky bucket shaper and block or distributed scheduling in the weighted round robin mechanism.

2 Results

2.1 Sliding Window Shaper and Block WRR Scheduler

2.1.1 Analysis

The first configuration of the system is with a sliding window shaper and block scheduling in the weighted round robin. The parameters for the sliding window and the weighted round robin are:

i	$=$	number of cells allowed to be transmitted from the shaper in a given sliding window
m	$=$	number of cell slots in a sliding window
s	$=$	number of slots guaranteed to the customer in the WRR schedule
t	$=$	number of total slots in the WRR schedule

Thus, the maximum normalized average rate out of the shaper is *i*/*m* and guaranteed normalized server access is *s*/*t*. To keep the delay bounded, we must assume *i*/*m* is less than or equal to *s*/*t*. To find the maximum (worst-case) delay, we assume *i*/*m* = *s*/*t*.

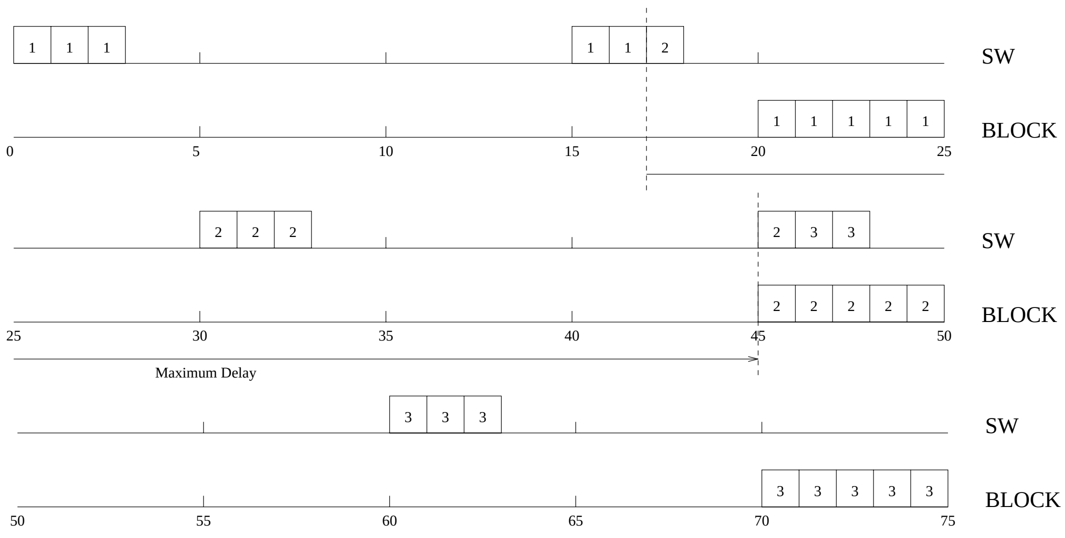

Figure 3: Example for Sliding Window/Block Scheduling Case

The analysis for this case is illustrated by the time diagram shown in Figure 3, where i = 3, m = 15, s = 5, and t = 25. The SW lines show the output of the shaper and the BLOCK lines show the output of the block WRR scheduler. ATM cells are shown as boxes, and the number in each box indicates which WRR block will serve the corresponding cell. Worst-case "phasing" is shown, in which the first cell out of the shaper is at the beginning of the WRR schedule and the service block is at the end of the schedule. Notice that the maximum delay in this case occurs when the number of cells left in the weighted round robin queue is at a minimum (but non-zero) after the customer's service period has ended. The next cell to be serviced will incur the maximum delay. To formalize this, we first note that the number of cells remaining in the WRR queue after the j^{th} WRR service period is:

$$\begin{aligned} k(j) &= \lceil j*s/i \rceil *i - j*s \\ &= [\lceil j*s/i \rceil - (j*s/i)]*i \end{aligned}$$

The ceiling expression in the first form determines the number of shaping periods that have allowed ATM cells to go through the shaper after the j^{th} scheduling window. This value times *i* is the number of cells allowed through the shaper. Subtracted from this is the number of cells remaining to be served by the WRR. Letting *n* be the number of customer service periods before the maximum delay cell is served, we then have:

$$n = \text{arg min } k(j) \text{ subject to } k(j) \text{ not equal to } 0,$$

that is, *n* is the value of *j* that produces the smallest non-zero value of *k(j)*.

Using this value *n* and the parameters of the shaper and weighted round robin, the WRR queue arrival time for the maximum delayed cell can be calculated as well as the time of which it has departed from the WRR queue.

$$\text{Arrival time} = (\lfloor n*t/m \rfloor *m)+i-1$$

$$\text{Departure time} = (n*t)+(t-s)+((\lceil n*s/i \rceil -(n*s/i))*i)-1$$

The first term in the arrival time expression is the number of cells that are served before the maximum delay cell. The second term defines the maximum delay cell to arrive *i* cells later. The first term in the departure time expression is the number of scheduling windows that are served before getting to the departure of the maximum delay cell. The second term is the waiting period between the end end of the current servicing block and the beginning of the next servicing block. The last term is the offset into the next servicing block corresponding to the departure of the maximum delay cell. The maximum delay is just the difference.

$$\text{Maximum delay} = \text{Departure time - Arrival time}$$

This delay only occurs when the parameters are such that *s* is a multiple of the remainder of the offered cells from the shaper not served within $\lceil i/s \rceil$-1 scheduling intervals. When *s* is a multiple value, the maximum delay is bounded to a lower limit defined as the interservice time (*t* - *s*) between $\lceil i/s \rceil$ scheduling intervals. Incorporating this into the previous equations, the maximum delay is as follows.

$$\text{rem} = \frac{s}{s-(\lceil i/s \rceil - i/s)*s} = \frac{1}{1-(\lceil i/s \rceil - i/s)}$$

$$\text{Maximum delay} = \begin{cases} (\lceil i/s \rceil)*(t - s) & \text{if rem is an integer value} \\ \text{Departure - Arrival time} & \text{otherwise} \end{cases}$$

Figures 4 through 12 show the relations between the variables *i*, *t*, and the ratio of *i* to *m* and *s* to *t*, which is *r*. For fixed *r* and *t*, delay tends to increase as *i* increases (larger "bursts"), but Figure 5 shows that this increase is not monotonic. Figure 4 shows that as *t* is held constant then the maximum delay increases as *r* decreases (smaller average rate) for a given *i*. This shows that a user with a higher percentage of the slots will tend to have a lower maximum delay than a user with a lower percentage due to a longer wait for service blocks. In Figure 5, as *r* is held constant, the maximum delay tends to increase as *t* increases for a given *i*. The term "tends to increase" means that, although some of the points from all three cases coincide, the maximum delays are usually greater for greater values of *t*. Figures 6 through 8 show the individual curves from Figure 5. These curves show a unique stair step function bounding the bottom points of the line and a linear slope bounding the top points of each line. The steps tend to increase every *s* slots and the point pattern seems to repeat itself. Turning to Figure 9, this graph illustrates again that maximum delay decreases with increasing *r* and increases as *i* increases for a fixed *t* and a given *r*. An interesting feature occurs at *r* = 0.05 and 0.1 and *i* = 100. This occurs because *s* is a multiple of *i* (which equals 100) at these point. Therefore, the maximum delay is the minimum value of the silence interval between successive service blocks, that is, *t*-*s* (which is 1900 at these points). Figure 10 shows that maximum delay increases as *t* increases for a fixed *i* and given *r* due to a longer wait between service blocks. Similar to Figure 9, this graph shows that the maximum delay values for *t* = 2000 and *t* = 1000 are the same at *r* = 0.05. This is because *i* is an integer multiple of *s* at both of these points. Figure 11 also shows that maximum delay tends to increase with increasing values of *t*. In addition, Figure 11 shows that as *i* increases so does maximum delay given a *t* with *r* fixed. Again, the maximum delay dips down when *s* and *i* are integer multiples of each other. With Figure 12, there is again a trend that shows maximum delay increases with decreased values of *r* with a fixed *i* and given *t*.

A significant conclusion from this is that maximum delay can be minimized by choosing *s* and *i* such that $1/(1-(\lceil i/s \rceil - i/s))$ is an integer. The advantages of such a choice are very significant, as illustrated by the negative-going spikes in Figures 6 through 8.

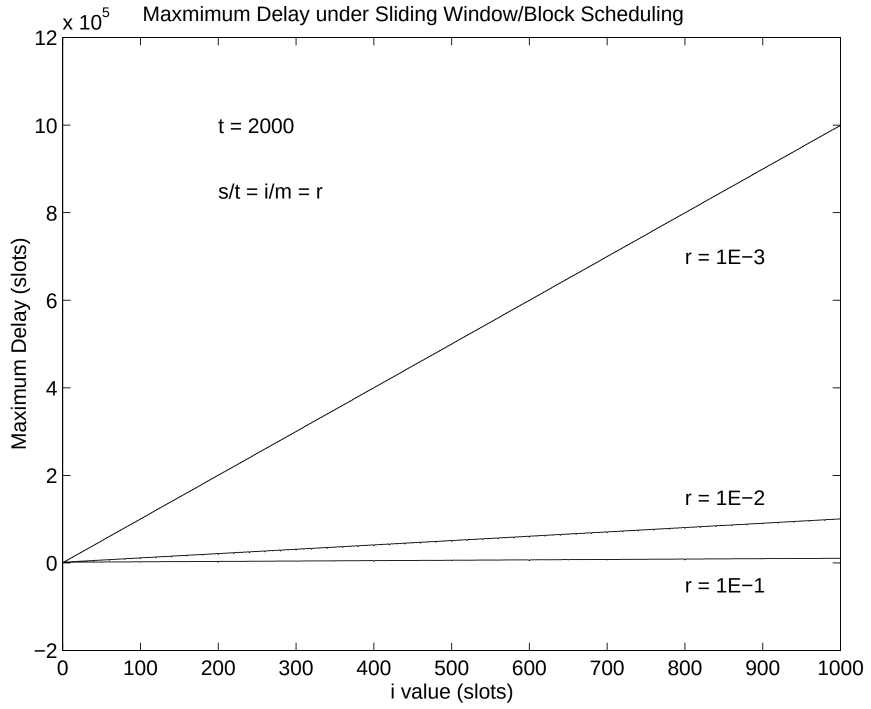

Figure 4: Sliding Window/Block Scheduling (maximum delay vs. i and r)

2.1.2 Validation by Simulation

Simulations were performed to verify the delay bounds. A single source was tracked to find its maximum delay using a C program. This source was set up such that during each slot time the source could either send a burst of cells or send nothing. If the source did send a burst, the burst size was generated randomly based on a geometric distribution. The parameters of this source were calculated such that the load generated from the source matched the guaranteed bandwidth of the source (i.e.

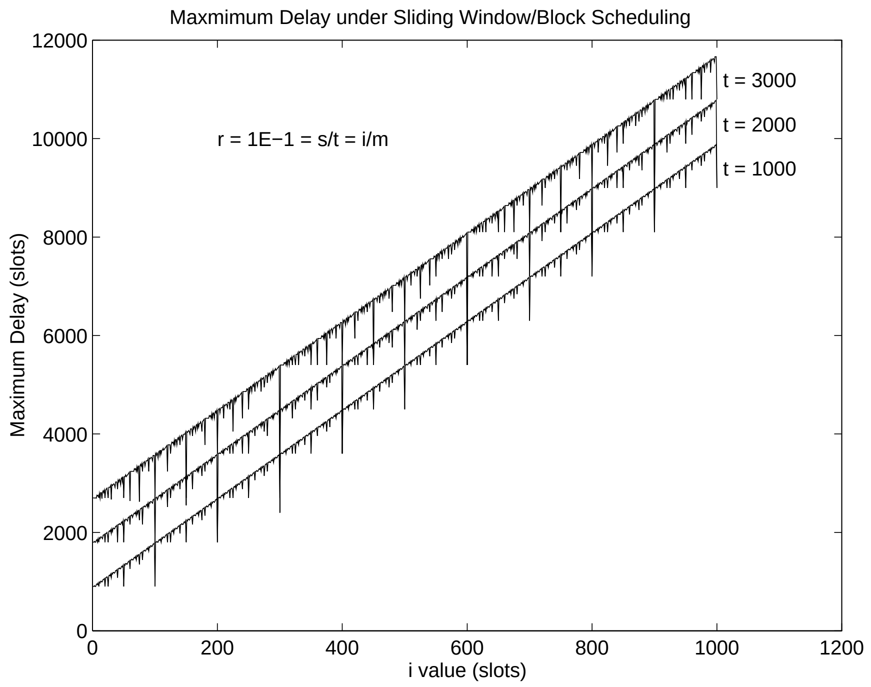

Figure 5: Sliding Window/Block Scheduling (maximum delay vs. i and t)

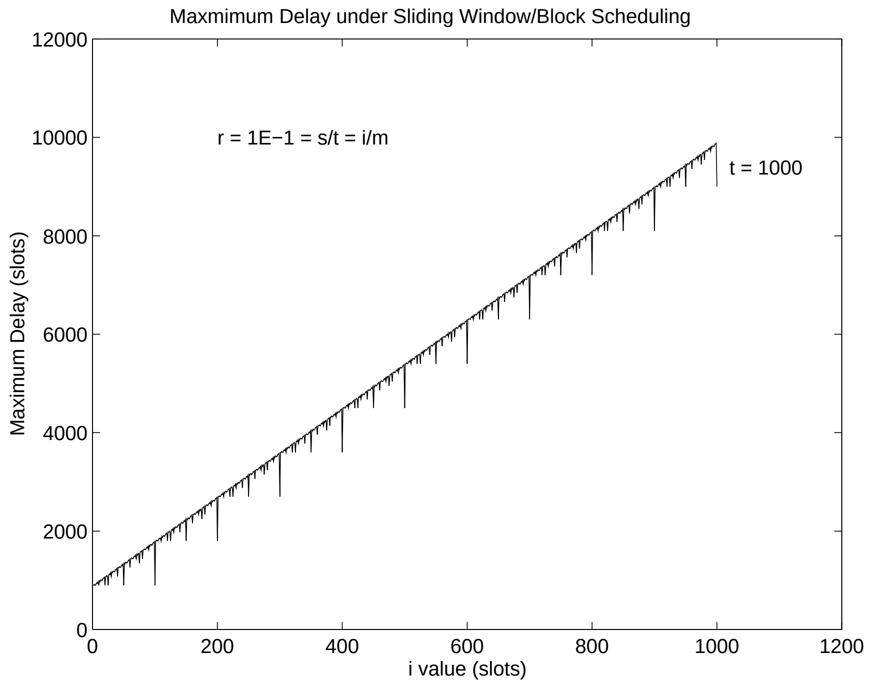

Figure 6: Sliding Window/Block Scheduling (maximum delay vs. i and t) (t = 1000)

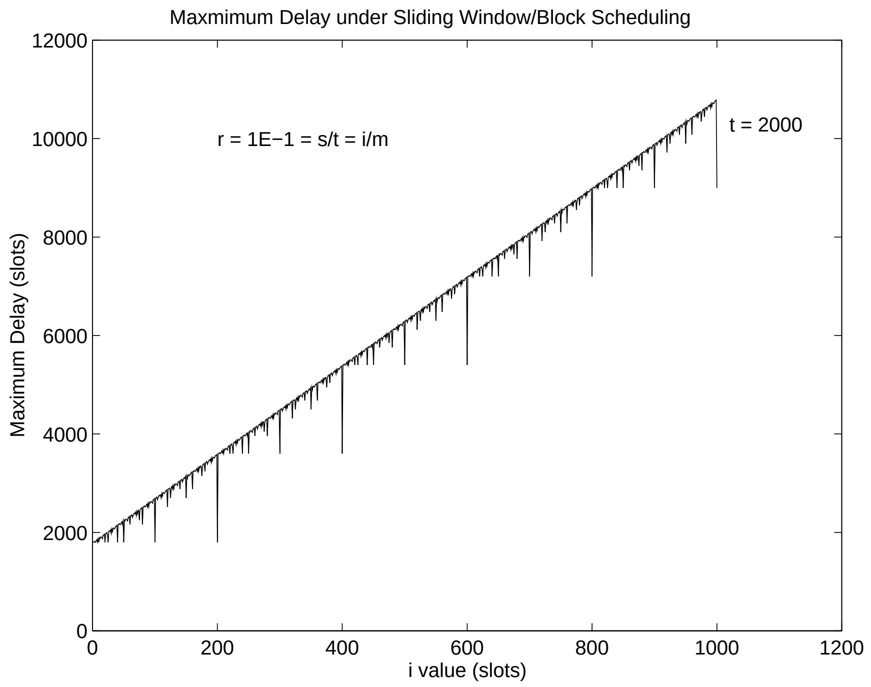

Figure 7: Sliding Window/Block Scheduling (maximum delay vs. i and t) (t = 2000)

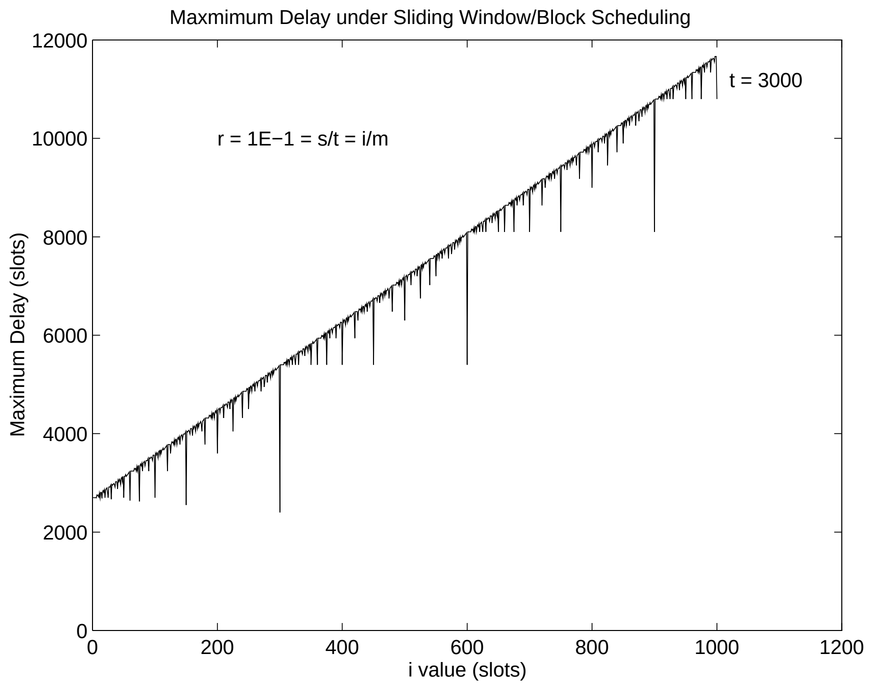

Figure 8: Sliding Window/Block Scheduling (maximum delay vs. i and t) (t = 3000)

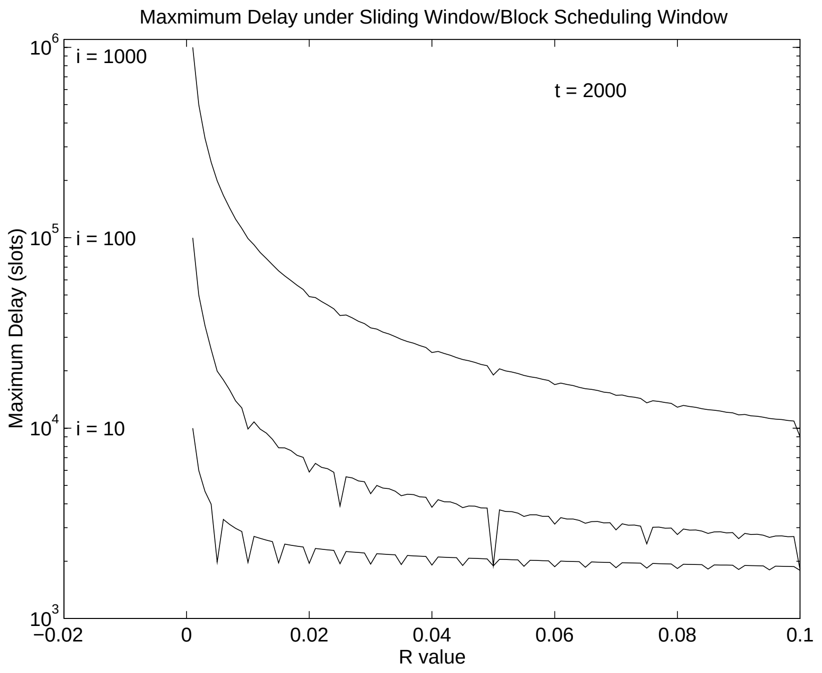

Figure 9: Sliding Window/Block Scheduling (maximum delay vs. r and i)

i/m = s/t = guaranteed bandwidth). In order to get a burstier source, the average size of the burst must be increased. This, in turn, will increase the probability of getting an empty slot from the source in order to maintain a constant load. Figure 13 shows that the bounds are indeed met in some cases despite the burstiness of the source. In this simulation, t = 2000 and s = 200. The more bursty simulation corresponds to a higher load (200%) while the less bursty simulation corresponds to a nominal load (100%). It is important to note that even the minimum points of the upper bounds are met by both simulations. Also, the more bursty the source is then the closer the maximum delay for the simulations will be to the theoretical bound.

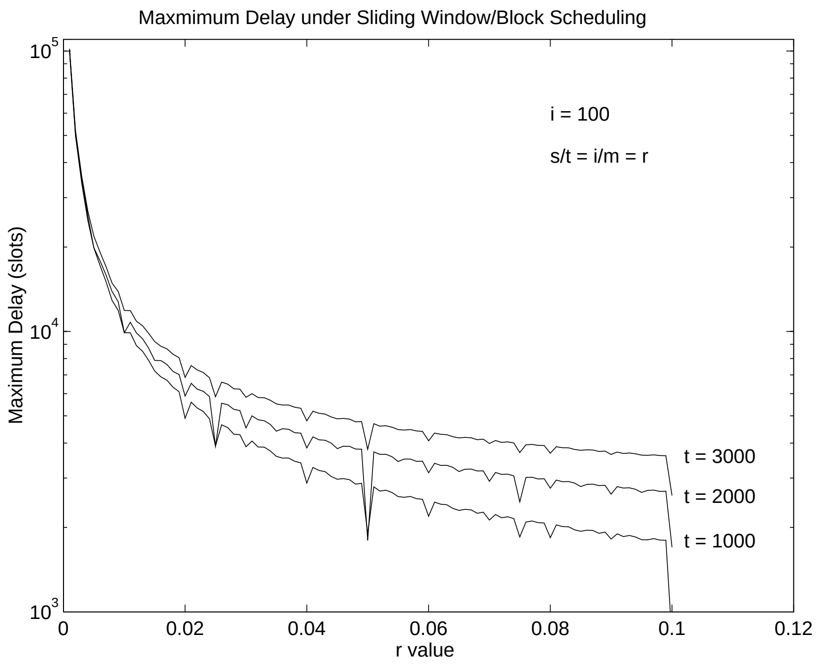

Figure 10: Sliding Window/Block Scheduling (maximum delay vs. r and t)

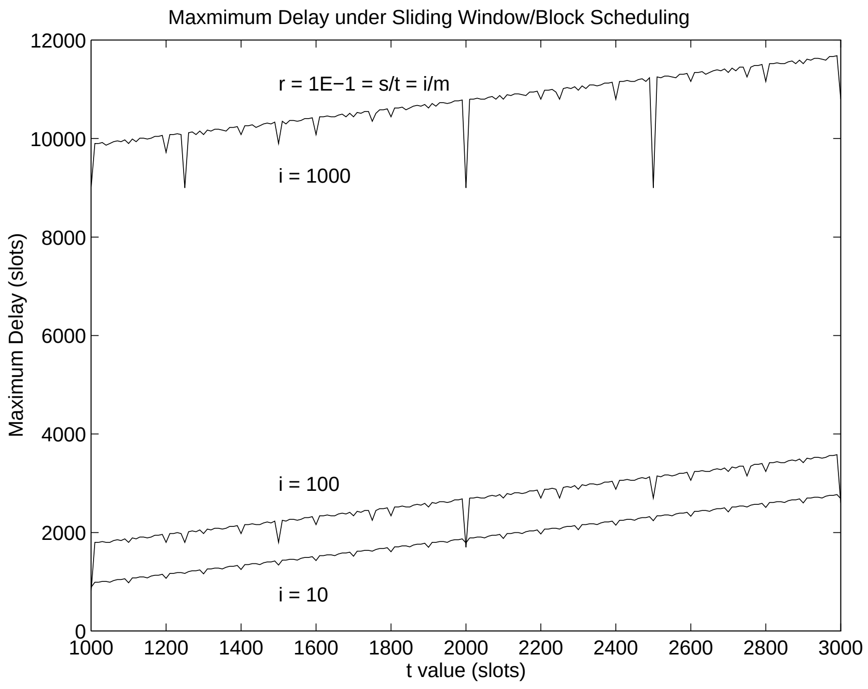

Figure 11: Sliding Window/Block Scheduling (maximum delay vs. t and i)

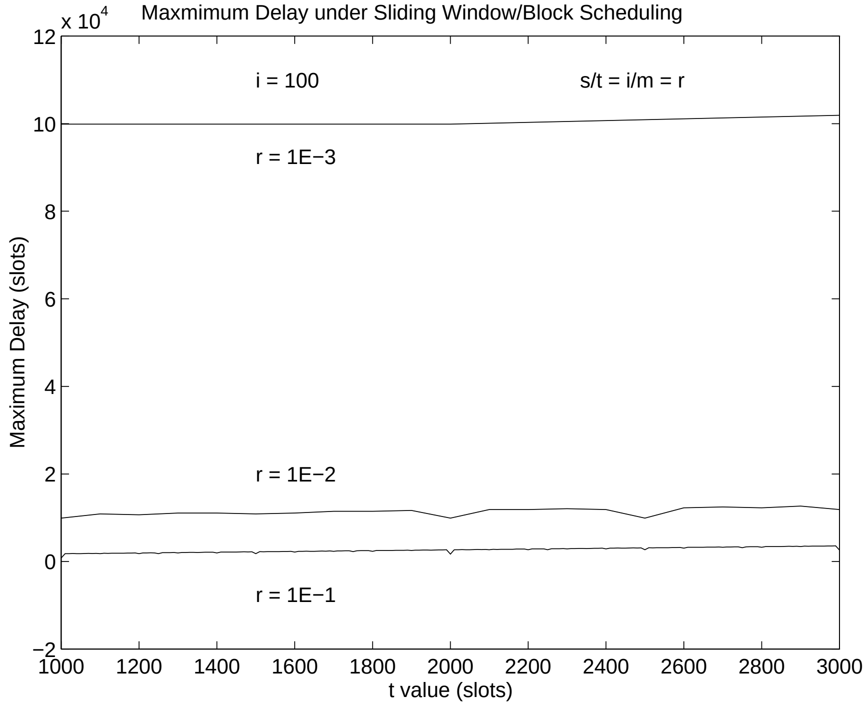

Figure 12: Sliding Window/Block Scheduling (maximum delay vs. t and r)

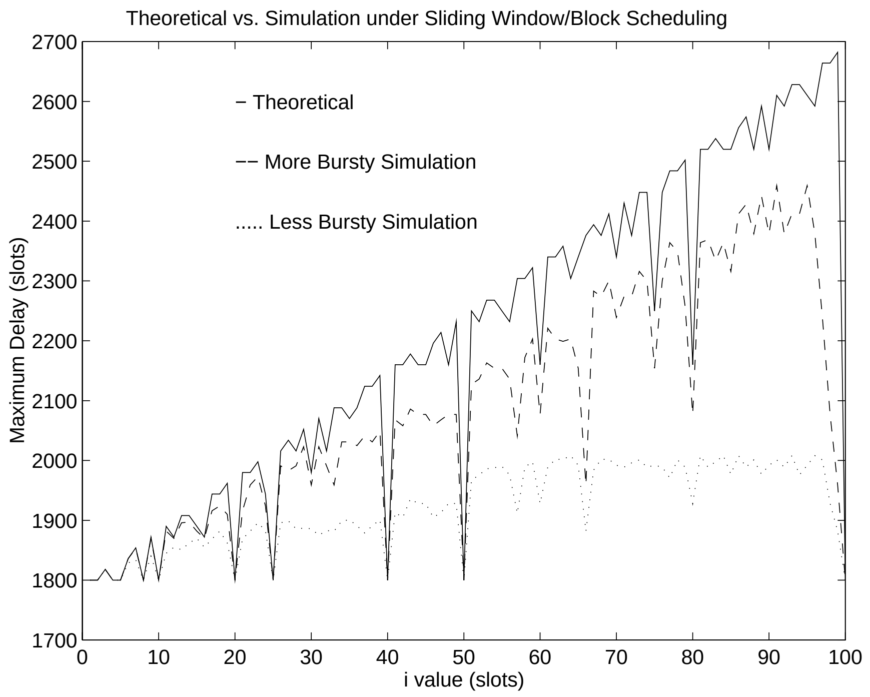

Figure 13: Sliding Window/Block Scheduling (Theoretical vs. Simulation)

2.2 Sliding Window Shaper and Distributed WRR Scheduler

2.2.1 Analysis

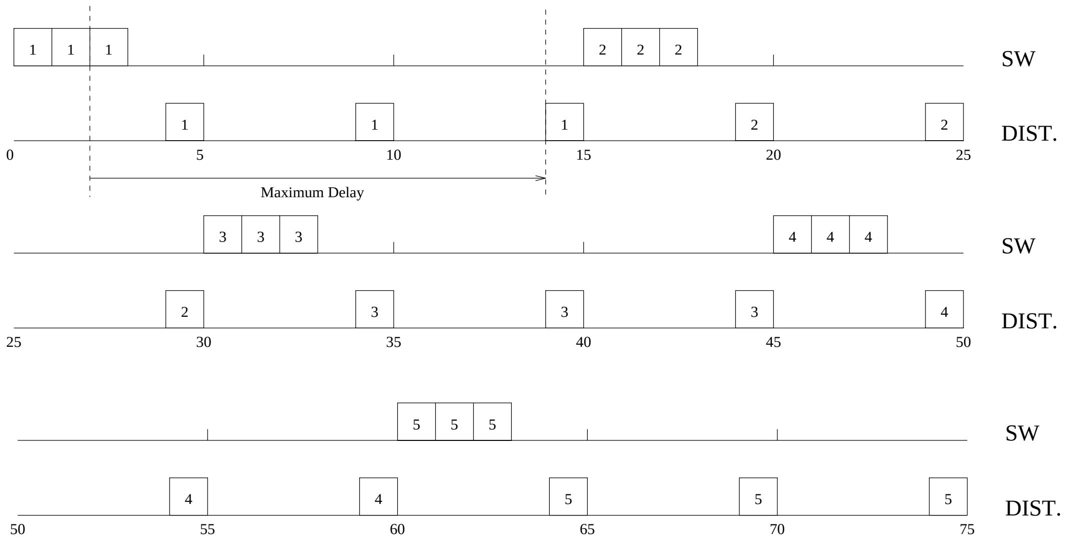

Figure 14: Example for Sliding Window/Distributed Scheduling Case

A time referenced example of the sliding window shaper with distributed scheduling in the weighted round robin under worst case conditions is shown in Figure 14 using the same parameters as in the previous case. It can be seen here that the last cell of a shaper burst is the cell which experiences the maximum delay. It can also be seen that all delays in this example are integer multiples of one less than the scheduled interservice time of the customer. This leads to the following maximum delay formula.

$$\text{Maximum delay} \quad = \quad i*(t/s - 1)$$

This formula assumes an evenly distributed schedule. If the scheduling parameters are such that the servicing slots cannot be evenly distributed, then the maximum delay will occur as one less than the maximum interservice time between each of i+1 servicing slots.

Figures 15 through 17 assume evenly distributed scheduling and show the relation between *i*, *r*, and *t* for the maximum delay of this case. Again, in all cases, we have held *i/m* = *s/t* = *r*. The delay bounds look much more monotonic in the distributed case as compared to the block case. Figure 15 is very similar to Figure 4. Maximum delay increases with *i*, and as *r* decreases, maximum delay increases given *i* and a fixed *t*. Looking to Figure 16 (and the equation above), maximum delay is linear in *i* for fixed *r*. Also, maximum delay is not affected by changes in *t* for a given *i* and fixed *r*. This is due to the nature of the distributed scheduling window. Despite the size of the schedule, the distribution pattern within the schedule is the same. These two relationships illustrated in Figure 16 can be proven by simple examination of the equation above, since fixed *r* implies fixed *t/s*. Figure 17 shows that maximum delay decreases as *r* increases, which matches our expectations.

2.2.2 Validation by Simulation

Figure 18 shows the theoretical maximum delay versus the maximum delay encountered through simulation. The simulation is the same as used in the sliding window/block scheduling case. It is illustrated that the theoretical bound is linear while the simulation obviously is not. As the source in the simulation becomes more

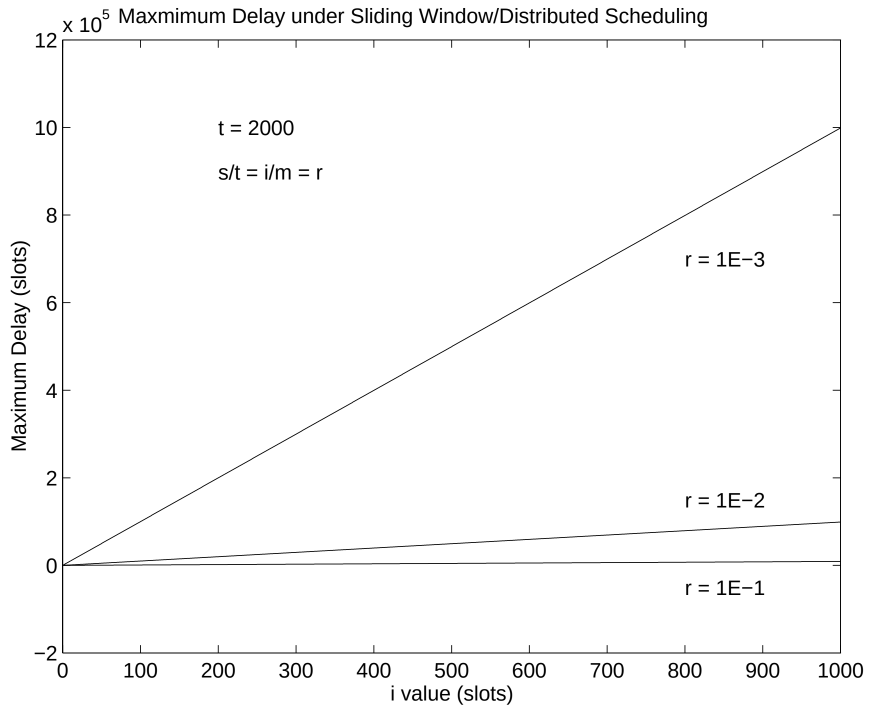

Figure 15: Sliding Window/Distributed Scheduling (maximum delay vs. i and r)

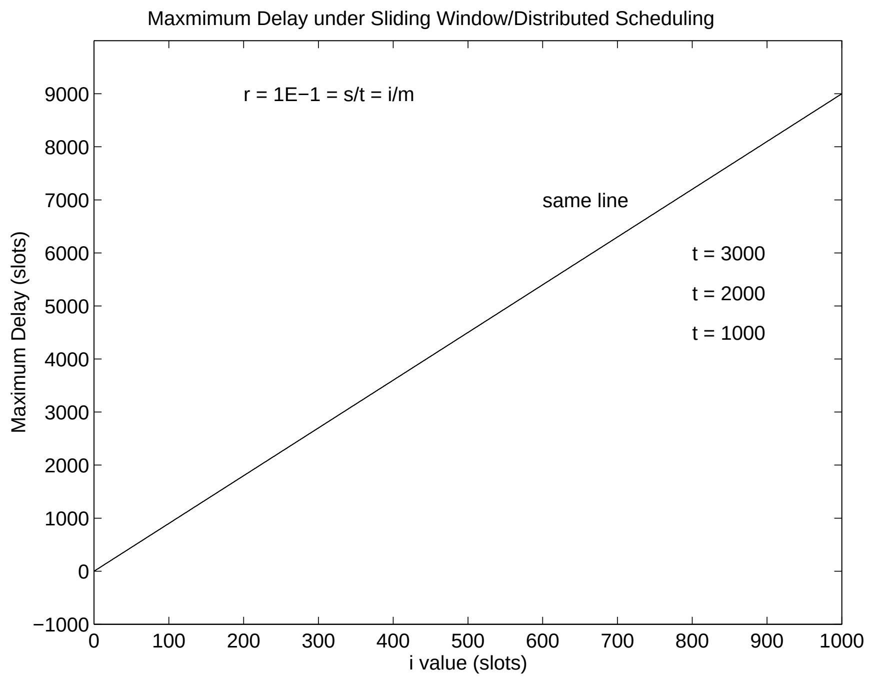

Figure 16: Sliding Window/Distributed Scheduling (maximum delay vs. i and t)

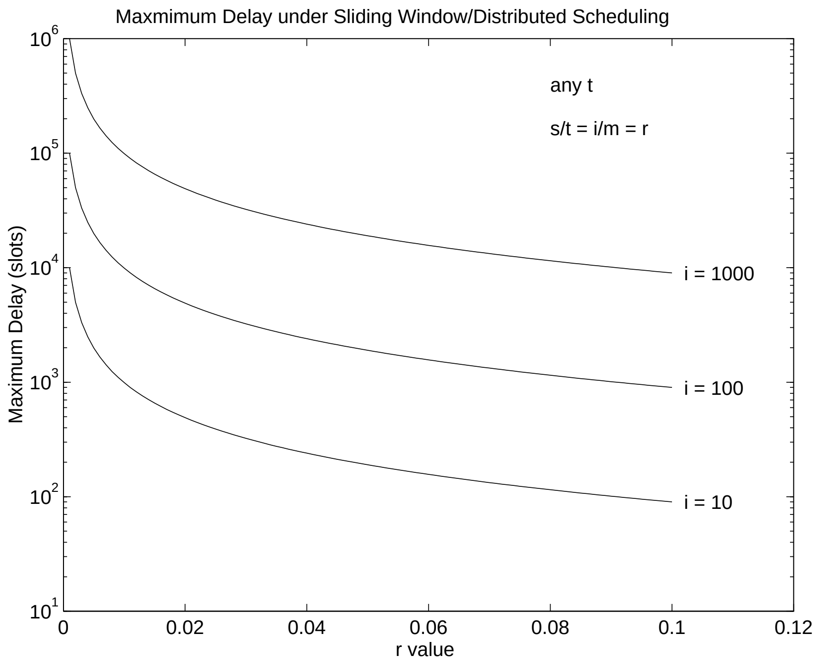

Figure 17: Sliding Window/Distributed Scheduling (maximum delay vs. r and i)

bursty and goes towards infinite burstiness, the delay bound for the simulations will approach the theoretical bound. Again, t = 2000 and s = 200. Also, the more bursty simulation corresponds to a 200% load and the less bursty simulation corresponds to a 100% load.

2.3 Leaky Bucket Shaper and Block WRR Scheduler

2.3.1 Analysis

A time referenced example for this case is shown in Figure 19. This case is very similar to the sliding window worst case scenario for block scheduling. The difference here is that the original burst in the sliding window is of fixed sized equal

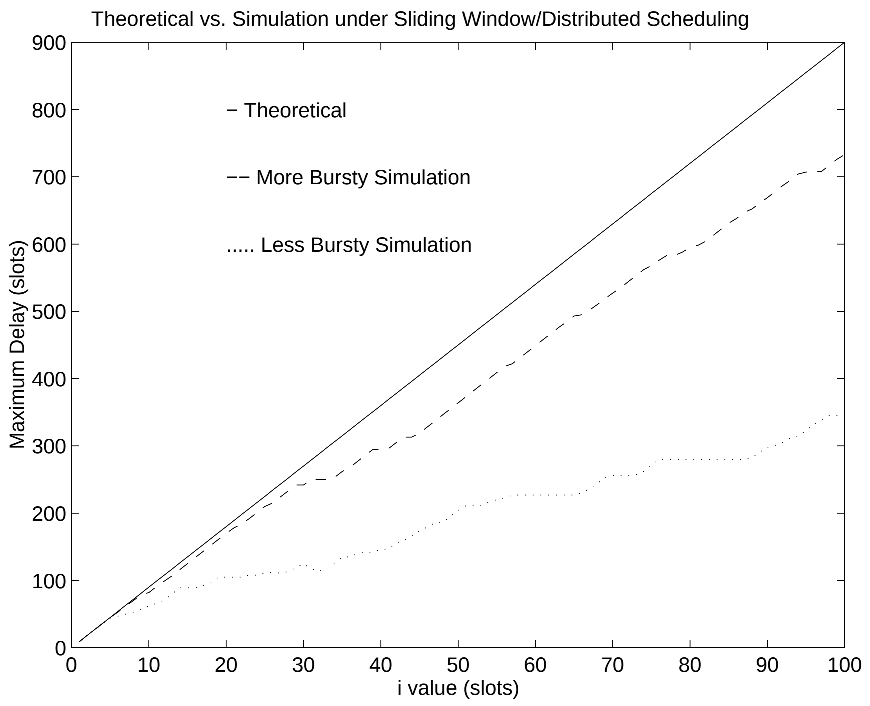

Figure 18: Sliding Window/Distributed Scheduling (Theoretical vs. Simulation)

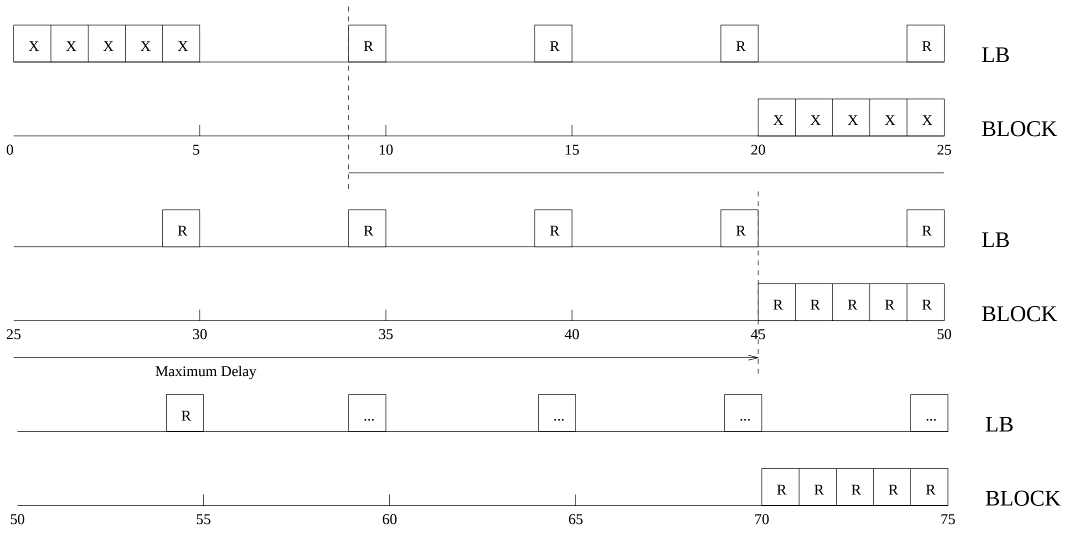

Figure 19: Example for Leaky Bucket/Block Scheduling Case

to the parameter *i*. Here, there are two different parameters.

$$B = \text{the leaky bucket size}$$

$$R = \text{the rate of token replenishment}$$

These parameters replace the parameters i and m. In fact, R is set to the ratio of i over m and B is set to the same value as i. Using this logic, it would seem that the original burst coming from the worst case condition would be equal to B. The fault in this line of reasoning is that the tokens will be replenished during the servicing of the block. So to get the actual burst size, X, the replenishment must be considered as follows.

$$\begin{aligned} X &= \text{B + } \lceil \text{X*R} \rceil \\ &= \lfloor \text{(B/(1-R))+0.5} \rfloor \end{aligned}$$

From this burst size, it can be seen that the cell associated with the maximum cell delay is the same cell that is served first in the next service cycle after the service cycle which serves the last cell of the burst. This gives the following WRR queue arrival and departure times for the first cell served in the next service cycle.

$$\begin{aligned} \text{Arrival time} &= \text{X+(((s* } \lceil \text{X/s} \rceil \text{)-X)*(1/R))} \\ \text{Departure time} &= \text{(} \lceil \text{X/s} \rceil \text{ *t)+(t-s)} \end{aligned}$$

$$\text{Maximum Delay} = \text{Departure time - Arrival time}$$

The first term in the arrival time equation is the expanded number of cells shaped in a burst through the leaky bucket. The first part of the second term is the number of cells needed to make the number of shaped cells be a multiple of *s* plus one. This result is then multiplied by the token replenishment interarrival time to get the arrival time of the maximum delay cell. The first term in the departure time equation is the amount of time needed to serve the *s* multiple number of cells. The second term is the waiting time until the next service block.

When we set the replenishment rate of the tokens (*R*) equal to the guaranteed bandwidth (*s/t*), the leaky bucket shapes the traffic at a rate equal to the guaranteed bandwidth. This leads to the following simplified version of the maximum delay equation.

$$\text{Maximum Delay} = (((s-t)/s)*X)+t-s$$

While the simplification eliminates the nonlinearity associated with the ceiling functions, the nonlinearities in the *X* function still exists. In terms of *s* and *t* for R = *s*/*t*, we have the following.

$$X = \lfloor (B*t/(t-s))+0.5 \rfloor$$

The trends experienced by the maximum delays as functions of *R* and *t* are shown in Figures 20 through 25. Most figures show the same trends as in the sliding window/block scheduling case except without the dips. The absence of dips is due to the smoothing nature of the leaky bucket token replenishment. Figure 20 (similar to Figures 4 and 15) shows that maximum delay increases as B increases. The relationship appears to be linear, but is not quite (see Figure 26). This is only because X is not quite linear in B. Also, maximum delay increases as R decreases for a given B and fixed t. Figure 21 illustrates that as t increases so does the maximum delay for a given B and fixed R. Figure 22 shows the same relationships as Figure 20, but with R on the horizontal axis. It is also interesting to point out that the decrease in maximum delay with increasing R is very sudden for small values of R as can be seen by the sharp drop on the log scale. The drop is also seen in Figure 23. This graph also depicts maximum delay increasing as t increases for a fixed B and given R. Figure 24 shows the maximum delay increasing while B increases for a given t and holding R constant. Figure 25 shows the maximum delay increasing while R increases for a given t and holding B constant.

2.3.2 Validation by Simulation

Figure 26 shows the theoretical delay bounds versus the simulation maximum delays. The simulation delays do fall under the theoretical. In fact, the simulation delays follow the theoretical bounds much more closely under the leaky bucket case than do the sliding window simulations given the same burstiness. Again, this may be

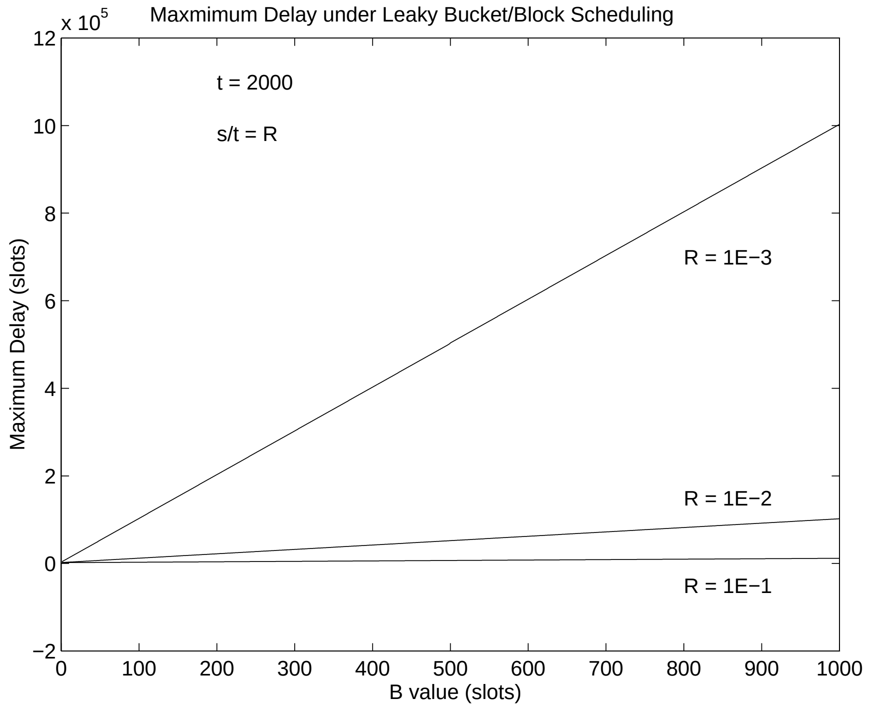

Figure 20: Leaky Bucket/Block Scheduling (maximum delay vs. B and R)

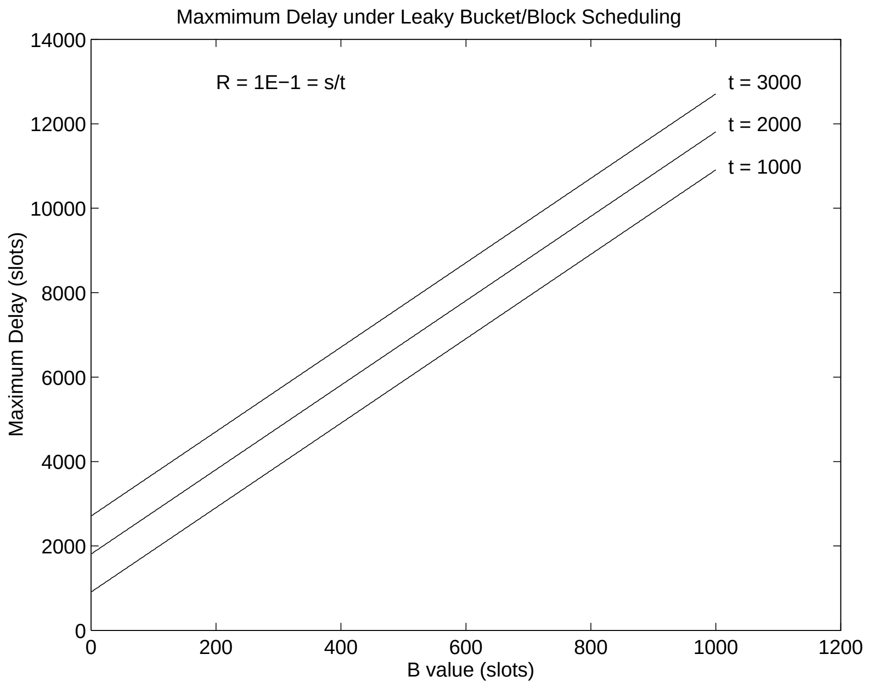

Figure 21: Leaky Bucket/Block Scheduling (maximum delay vs. B and t)

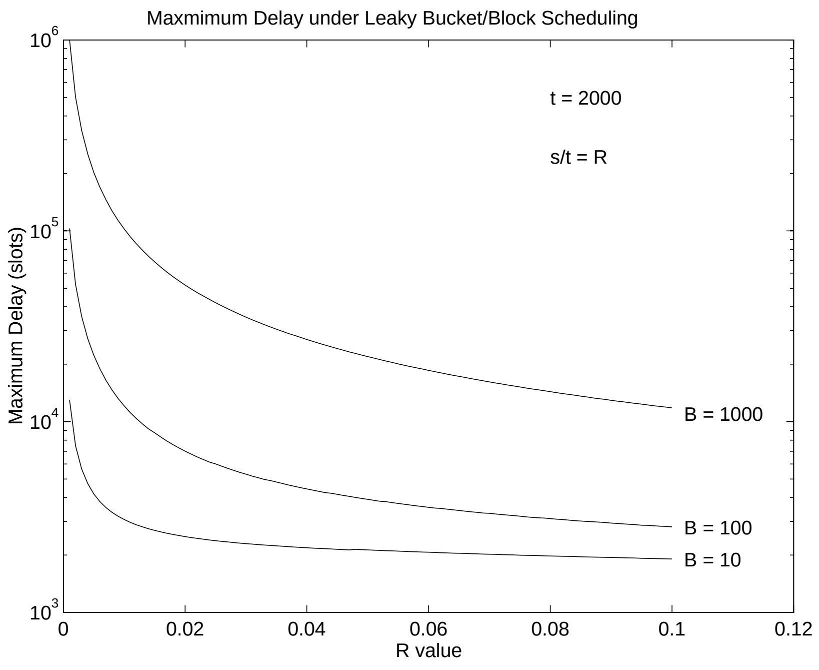

Figure 22: Leaky Bucket/Block Scheduling (maximum delay vs. R and B)

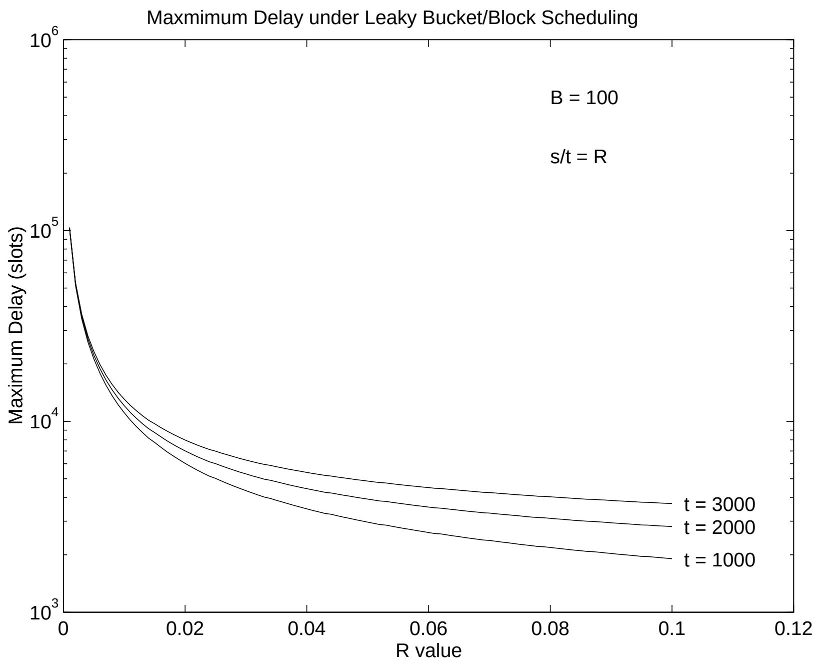

Figure 23: Leaky Bucket/Block Scheduling (maximum delay vs. R vs. t)

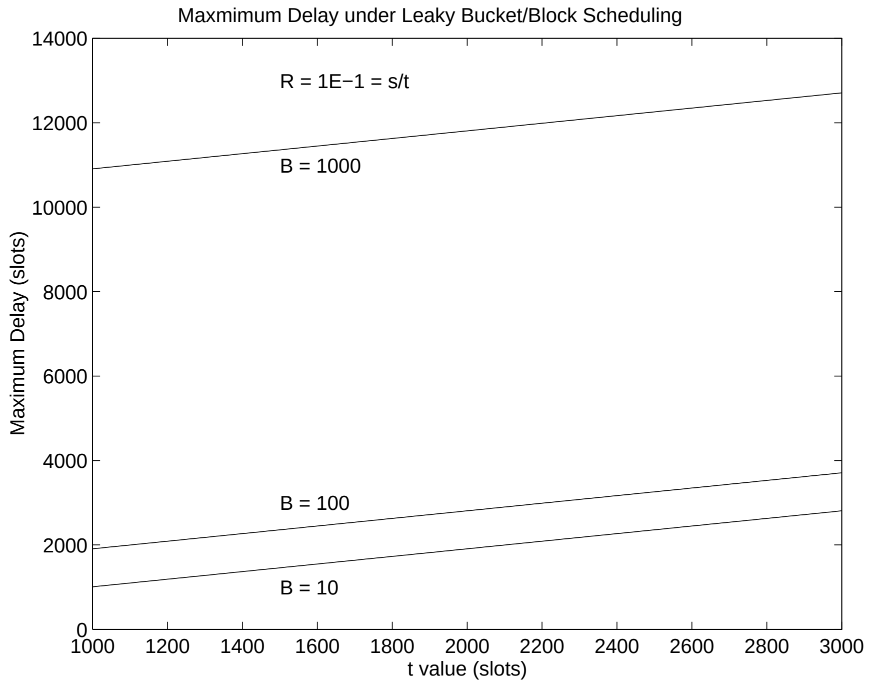

Figure 24: Leaky Bucket/Block Scheduling (maximum delay vs. t vs. B)

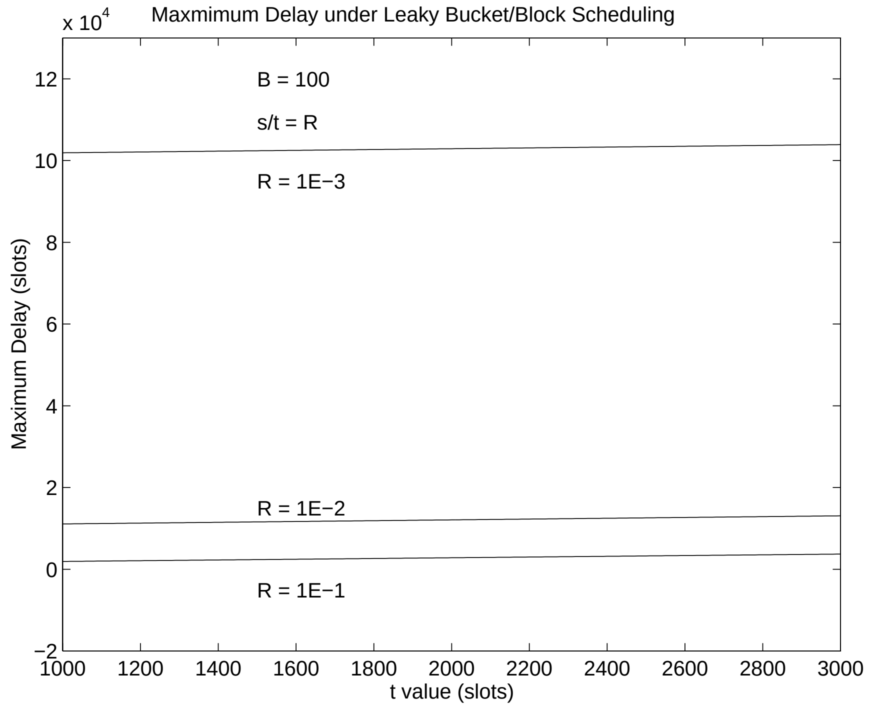

Figure 25: Leaky Bucket/Block Scheduling (maximum delay vs. t vs. R)

due to the fact that the leaky bucket shaper is much less bursty than the sliding window shaper. These specific simulations do not show much difference between more bursty or less bursty sources.

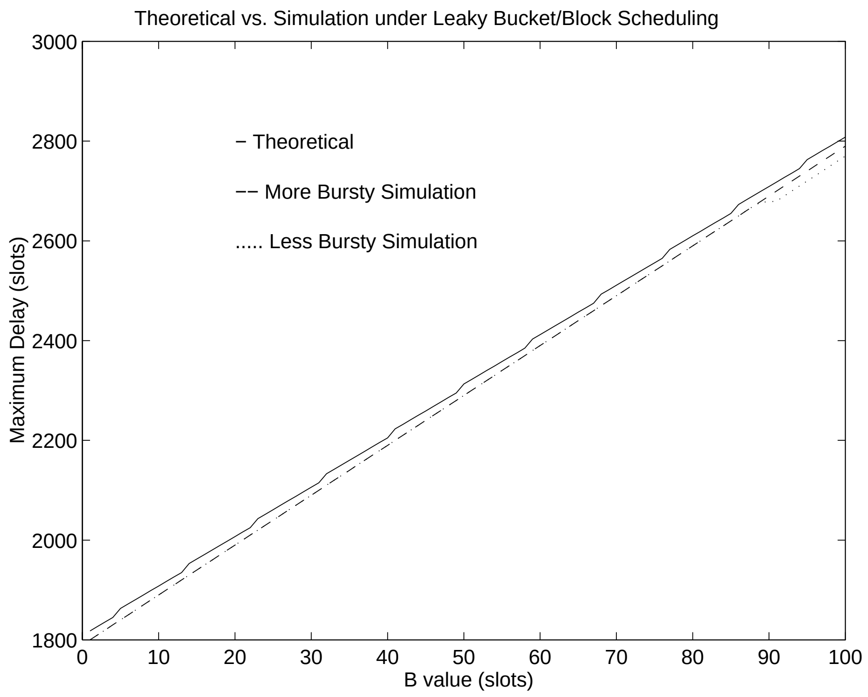

Figure 26: Leaky Bucket/Block Scheduling (Theoretical vs. Simulation)

2.4 Leaky Bucket Shaper and Distributed WRR Scheduling

2.4.1 Analysis

In looking at the time referenced example for this case in Figure 27, the distributed case again proves to be a simpler case to analyze. The original burst must again

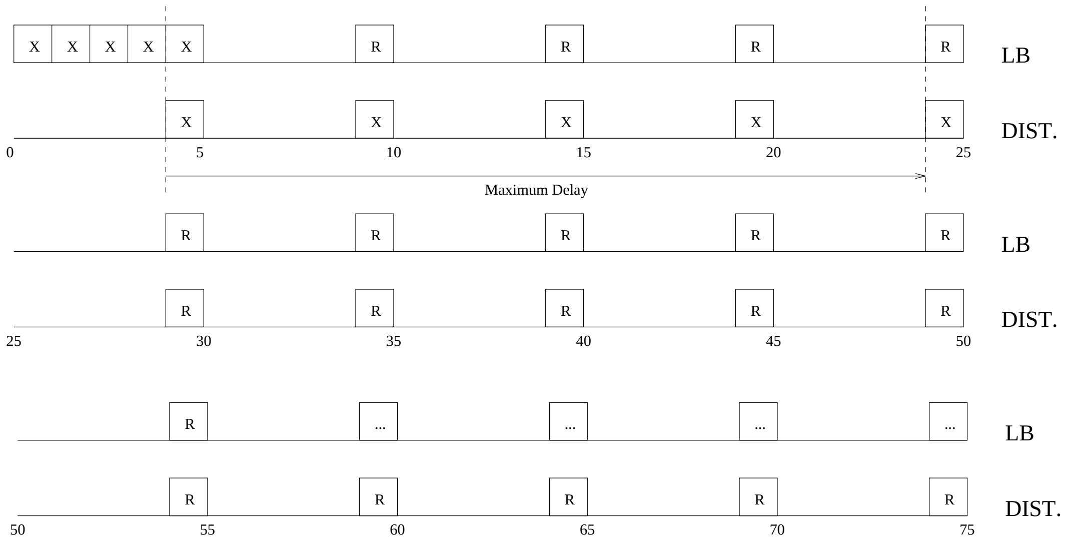

Figure 27: Example for Leaky Bucket/Distributed Scheduling Case

consider replenishment in giving X.

$$X = \lfloor (B*t/(t-s))+0.5 \rfloor$$

Again, in an evenly distributed scheduling window, the maximum delay will be a multiple of the interservice time between service slots (t/s-1) as follows.

$$\text{Maximum delay} = (X+1)*((t/s)-1)$$

If the scheduling window is not evenly distributed, the maximum delay is the maximum interservice time between X+2 service slots.

Figures 28 through 30 show the characteristics of maximum delay compared with the values of *B*, *R*, and *t*. Since the structure of the maximum delay equations are similar between the sliding window/distributed scheduling and leaky bucket/distributed scheduling, many similarities in the graphs between the two cases can be seen. Figure 28 is similar to that of Figure 20, because maximum delay increases also as *B* increases. As with the distributed case in the sliding window case, the delay bounds in Figure 29 shows no variance as the values of *t* are changed due to the distributed nature of service. Figure 30 shows that maximum delay decreases as *R* increases for a given *B* and a fixed *t*. The oddity seen in the curve where *i* = 10 is due to the nonlinearity of the equation used to derive *X*. In Figure 30, the exponential decay on the log scale is also seen again.

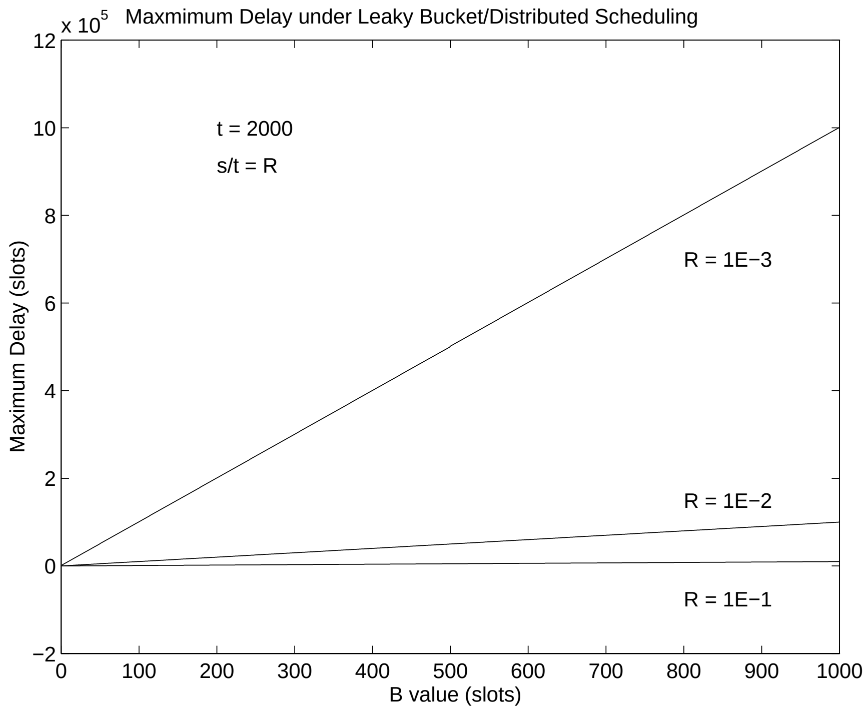

Figure 28: Leaky Bucket/Distributed Scheduling (maximum delay vs. B and R)

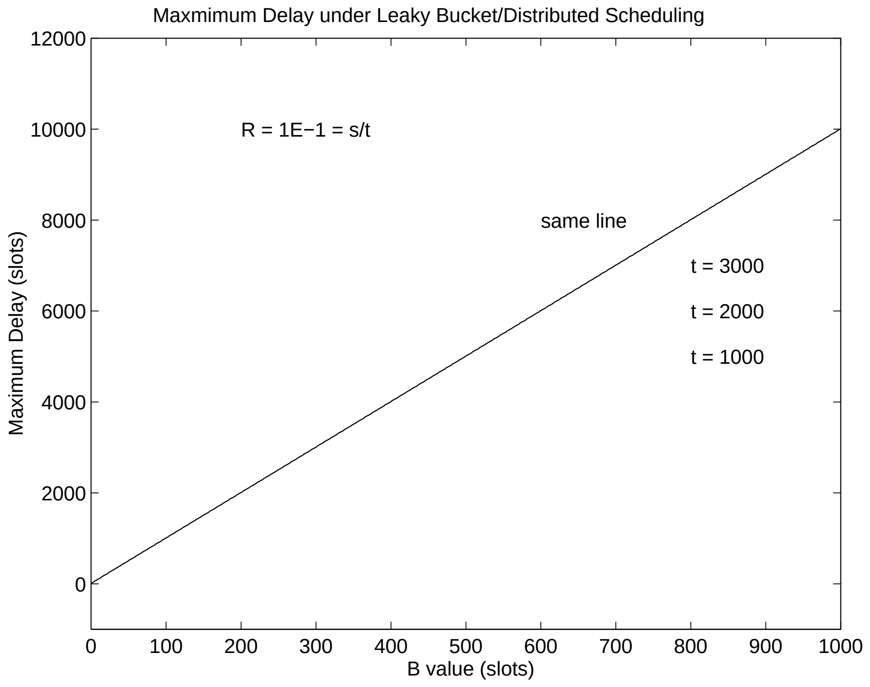

Figure 29: Leaky Bucket/Distributed Scheduling (maximum delay vs. B and t)

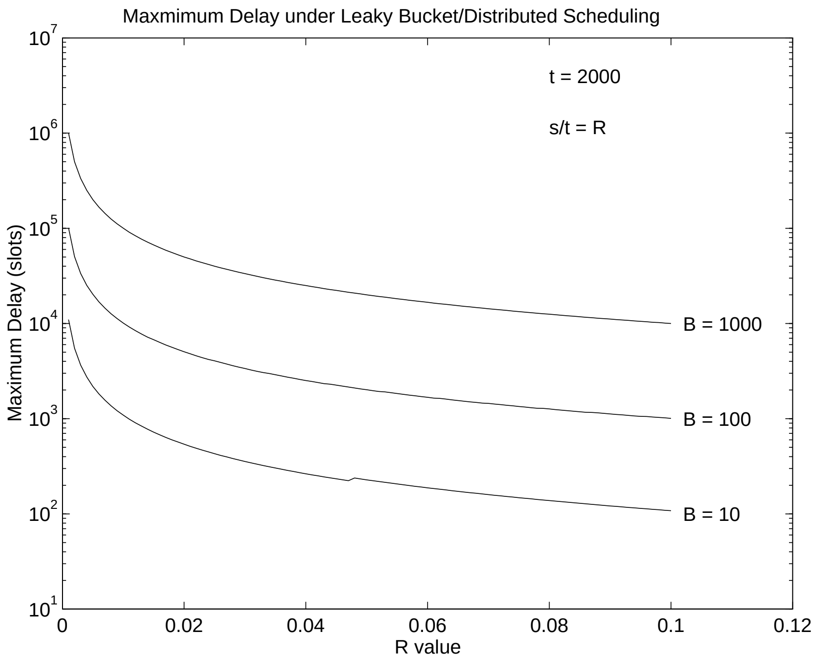

Figure 30: Leaky Bucket/Distributed Scheduling (maximum delay vs. R and B)

2.4.2 Validation by Simulation

Figure 31 shows that the theoretical bounds do hold up under simulation. Again, the simulation maximum delays are much closer to the theoretical bound than under the sliding window case. Also, the burstiness of the source does not affect the simulation maximum delays much. The parameters are t = 2000 and s = 200. The more bursty simulation has a load of 200% and the less bursty simulation has a load of 100%.

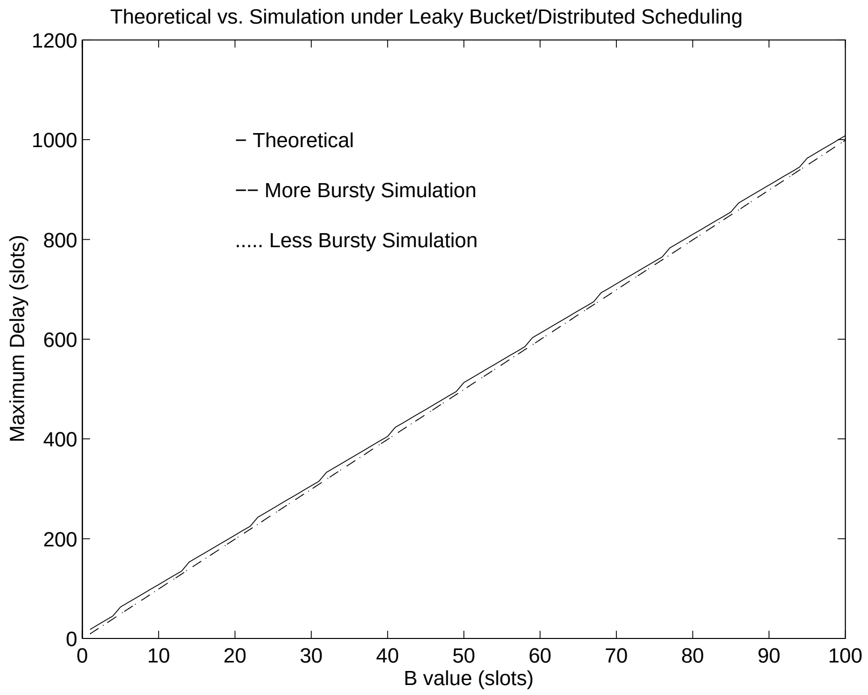

Figure 31: Leaky Bucket/Distributed Scheduling (Theoretical vs. Simulation)

References

[1] Kert Mezger, David W. Petr, Timothy G. Kelley. *Weighted Fair Queuing vs. Weighted Round Robin: A Comparative Analysis*. IEEE Wichita Conference on Communications, Networks and Signal Processing 1994.

[2] Brian Lang, Kert Mezger, David W. Petr. *Delay Performance of Weighted Round Robin Scheduling*. TISL Technical Report TISL-10230-03.

[3] Cameron Braun and David W. Petr. *Performance Evaluation of Dynamic and Static Bandwidth Management Methods for ATM Networks*. TISL Technical Report TISL-10230-05, December 1994.