# Alpha Message Scheduling for Packet-Switched Interconnects

Ludmila Cherkasova and Tomas Rokicki
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94303

**Abstract.**

Evaluation of interconnect performance generally focuses on fixed-size packet latency as a function of traffic load. To an application, however, it is the latency of variable-length messages, rather than the individual packets, that is important. In this report, we discuss how scheduling the packets of messages according to various strategies can lead to effective performance differences of more than a factor of three. We present a new scheduling technique, called *alpha scheduling*, that can combine the bandwidth-fairness of round robin scheduling while attaining close to the optimal performance of shortest-first scheduling. We demonstrate our results with a simple simulation model.

## Contents

# 1 Introduction

Packet-switched interconnect hardware transmits fixed-size packets, while most operating system interfaces provides for the transfer of variable-length messages. Each message is therefore broken down into some number of packets for transmission by the hardware. In this report, we consider scheduling strategies for the insertion into the interconnect hardware of the packets that comprise a message. Our primary result is that using an appropriate strategy to insert the packets comprising messages into an interconnect can have a tremendous impact on the performance of that interconnect. Indeed, suitable selection of such a strategy can increase the effective performance of the interconnect by a factor of two or three over naive FIFO or round robin packet insertion.

Messages, and also packets, are separated into "priority" messages and "non-priority" messages. The former always take precedence over the latter in the hardware, and presumably this will also be the case in the software. The hardware provides non-preemptive packet transfer, possibly with in-order delivery provided by deterministic and possibly with out-of-order delivery using adaptive routing. While packets cannot preempt each other, one packet can overtake another during routing. The in-order delivery, even with deterministic routing, is only guaranteed among packets of the same priority level.

# 2 Assumptions and Investigation

Since priority packets always take precedence over non-priority packets, we need consider scheduling only for a single priority class. We consider the tasks to be a set of messages that arrive to be delivered; the scheduling task is ordering the packets of the messages in such a way that average message latency is reduced while guaranteeing the delivery of all messages. We will assume that there is some finite number of applications sending messages, and each application sends a new message only after the previous message has been delivered. (If an application can send a finite number of messages concurrently, it is easily modeled by that many component applications.) We only consider the queue latency since this is the waste time we can control through scheduling.

Questions we consider are:

- Should packets from multiple messages be interleaved? At what grain should interleaving be performed if so (per packet, every ten packets, etc.)?
- Is there a trade-off between average latency, fairness, and guaranteed delivery? Can this trade-off be controlled?
- How does the resulting scheduling strategy compare with classic scheduling strategies?

# 3 FIFO Scheduling

The simplest scheduling strategy is first-in, first-out. With such a strategy, starvation is impossible; each message from each application will eventually be delivered. The maximum time waiting in the queue is proportional to the sum of the lengths of the messages in the queue. To help keep this within a reasonable bound, messages longer than a particular size (perhaps 100K bytes) can be broken up into smaller messages.

On the other hand, it is not fair; for two fast applications each submitting messages continuously, the application that submits the longer messages will get a proportionately higher share of the bandwidth.

The average latency is also not optimal. If one application submits a long message immediately before another application submits a short message, then the short message will be delayed; the best average latency in this case is to schedule the shorter message first.

FIFO is extremely cheap to implement, requiring the least computation by the host processor or interface board.

Finally, short control messages can be delayed by the entire contents of the message queue at the time they were submitted.

# 4 Round Robin

Another scheduling strategy is to iterate through the messages currently in the message queue, interleaving packets from outstanding messages. If we assume that new packets are inserted at the end of the message queue, then this strategy is maximally fair; each application with an outstanding message will receive the same share of the bandwidth. It also guarantees delivery, since there are a finite number of applications. In this case, the maximum delivery time is proportional to the length of the message multiplied by the number of applications; this is a better result than for FIFO, and there need be no upper limit on the message size.

The average latency, on the other hand, is not optimal; interleaving a short and a long message delays the short message by about a factor of two without changing the latency of the longer message. The worst-case average latency is when the final packets for all messages are sent at approximately the same time; this is possible with round robin scheduling. If all messages are about the same length, the average message latency is twice as bad as the optimal value. The increase in latency for a one-packet control message, on the other hand, is proportional to the number of applications; this is much better than the FIFO scheduling strategy.

Another issue with round robin is that the 'current message' is constantly changing with every packet. Depending on how access to the actual message body is done, this can have negative effects on cache hit rates. Round robin will strain a finite buffer pool used to store the messages on an interface board. Finally, each message has a certain amount of state that will constantly need to be switched. If we are sending more than a million packets a second, these state switches might have a large negative impact.

A minor variant of this round robin strategy is to insert new packets at the front of the message queue. In this case, messages of only one packet go out 'immediately'. Even in this variant, short messages of length two or more suffer in latency. In addition, always inserting the short packets in the front of the queue allows a few applications generating many short messages to indefinitely starve a long message.

# 5 Shortest First

Another scheduling strategy is shortest message first. In this case, shorter messages are always sent before longer messages. In addition, if a message arrives that is shorter than the remaining portion of the message currently being sent, then that latter message is preempted and the shorter message sent instead.

This strategy is optimal with respect to average latency. Given a ordered set of tasks $t_i$ each of length $l_i$ for $0 \le i < n$, the average delay for all tasks is

$$\sum_i (n-i) l_i$$

because each task delays the $(n-i)$ tasks in front of it by the amount of time necessary to finish that task. This weighted sum is minimized if the tasks are sorted by decreasing $l_i$.

Unfortunately, this scheduling strategy is subject to starvation; two applications that constantly schedule short messages can starve an application with a pending longer message. Because of this, the algorithm is also unfair.

These different strategies are summarized below.

| | FIFO | Round Robin | Shortest First |
|---|---|---|---|
| Starvation | No | No | Yes |
| Fair | No | Yes | No |
| Latency | Poor | Moderate | Optimal |

# 6 Alpha Scheduling Strategy

We propose a scheduling strategy that lies between FIFO and shortest-first, based on the value of a coefficient. The messages are stored in a priority queue. Three parameters control the ordering of messages in the queue:

- The node parameter $c$ is a "clock" that starts at zero and increments for each packet inserted into the interconnect through the current node. It is easy to keep this value bounded without changing the scheduling solution as we shall see.
- The message parameter $l$ is the number of packets in the message that have not yet been sent. Initially this is just the length of the message. As each packet is sent out, the message priority is decremented by $\alpha$ to keep the head message priority up to date. Another strategy is to recalculate the head message priority before preempting it during the scan for insertion of a new message.
- The tuning parameter $\alpha$ controls the balance between fairness and latency minimization; it can range from 0 to $\infty$

Messages are inserted into the delivery queue with a priority of

$$c + \alpha l.$$

Messages with the lowest priorities get delivered first. A new message inserted into the queue with a priority lower than that of the sending message preempts the sending message.

If $\alpha = 0$, then this strategy is simply FIFO.

If $\alpha = \infty$, then this strategy is simply shortest-packet first; this is optimal for latency.

If $\alpha = 1$ or some other finite positive value, then the strategy will not allow any single application to be delayed indefinitely by the other applications, no matter what their message stream looks like. Larger $\alpha$ provides better average latency; smaller $\alpha$ provides better fairness.

# 7 Simulation

In this section, we present the results of simulating these different message queue management strategies. Our simulation consists of three main components: a simplified model of the interconnect, an instantiation of the queue and its strategy, and a model to generate messages from a specific traffic pattern. We describe each in turn.

Since we are only interested in the impact of message scheduling, we simplified our model of the interconnect to be a service queue with an average delay of one. This is the default time unit for our simulation. For the probability distribution function, we use the sum of a constant 0.5 plus a negative exponential with an average of 0.5 to reflect the fact that the port has a specific maximum bandwidth, and that the dead time between packets can vary greatly.

Simulating the queue is straightforward for each of the strategies. Since the queue length (in messages) is usually small, we use a simple priority queue based on linear lists. We construct the simulation in such a way that we can use the same message generator and connect it to many different queues and packet acceptor models in order to run many different parameters in parallel. This allows us to amortize the expensive random-number generation over many effective simulation runs.

For a default traffic distribution, we assume 10% of the messages to be long, 20-packet messages, and the remaining 90% are from one to five packets in length. The average message length is therefore 4.7 packets. Given a traffic density $u$ between zero and one, we generate new messages using a negative exponential distribution with an average interarrival time of $4.7/u$.

The final simulator model has three inputs. The first input is the traffic density to use. The second input was a list of strategies to consider; the model included both variants of round robin, and all possible values of $\alpha$. Since FIFO corresponds to an $\alpha$ of zero and shortest-first corresponds to a very large $\alpha$, these two strategies are implicitly included in the possibilities. The third input is the message length distribution to use.

We collect statistics and report several different parameters for each run. We report the average queue length in packets, the total number of packets inserted and removed from the queue, and the latencies for each of the different classes of messages. In addition, we calculate and report the average latency for all of the messages put together.

# 8 Results

Our primary results are summarized here.

- The effects of message scheduling increase with traffic load.
- Round robin and FIFO scheduling can always be out-performed with a judicious selection of the $\alpha$ parameter. A value of 10 will outperform both round robin and FIFO scheduling for traffic loads up to and including 98% of utilization for our traffic load. Other traffic loads show similar results.
- The $\alpha$ parameter trades long-message latency for short-message latency. Higher $\alpha$ gives better short-message latency and better average latency; lower $\alpha$ decreases the worst-case message latency.
- Heavier traffic requires larger $\alpha$ to obtain near-optimal average latency.
- An $\alpha$ of 10 works well over a wide range of workloads and utilizations.

We ran the simulation for the two variants of round robin and for values of $\alpha$ of 0, 1, 2, 3, 6, 10, 20, 30, 60, 100, and 1000000. We investigated traffic loads of 0.01%, 1%, 20%, 50%, 70%, 80%, 90%, 95%, 97%, and 98%. Thus, there more than one hundred simulation runs. In order to attain steady-state at the higher traffic rates, tens or hundreds of millions of packets were simulated.

In addition to the results illustrated here, we also ran many tests with various different message length distributions. While the numbers varied, the conclusions drawn remain the same.

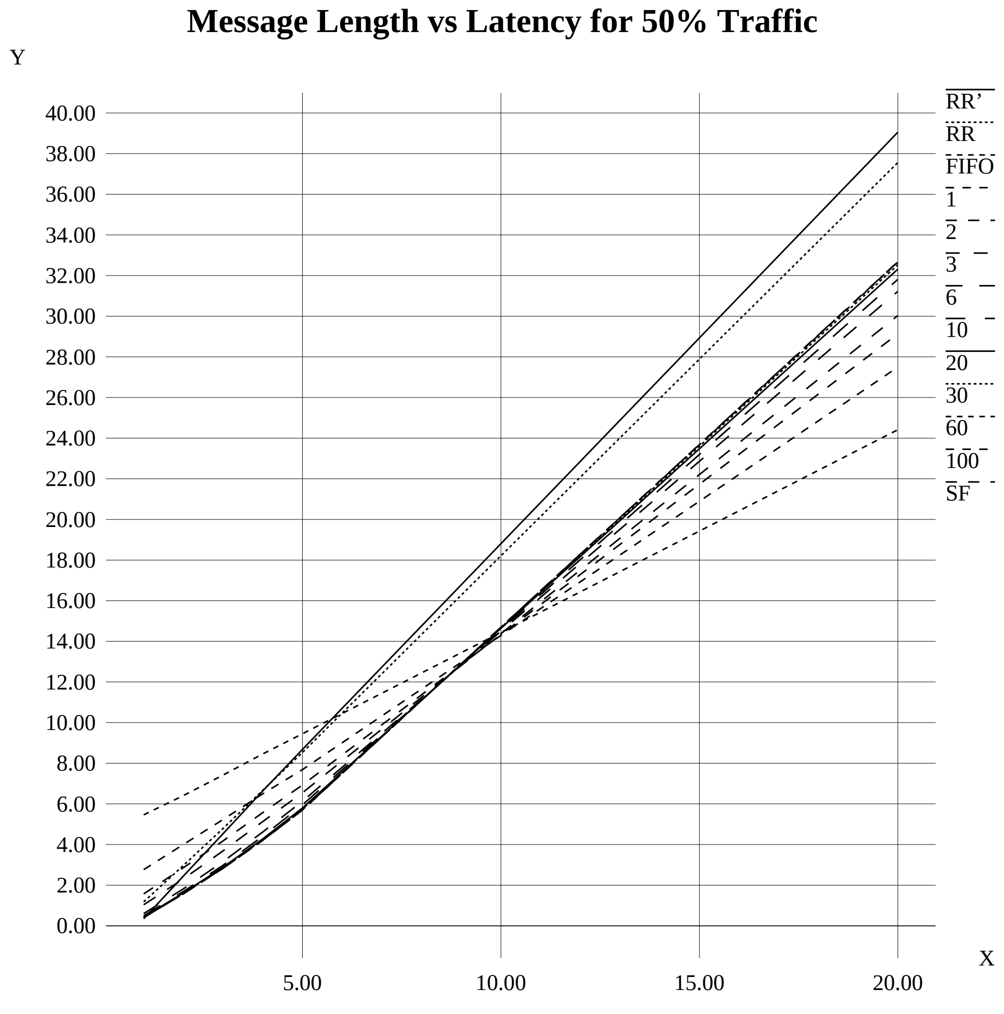


Figure 1: Message length vs latency for 50% traffic. In this and all figures in this report, RR' means the modified round robin, RR is the normal round robin, SF is shortest first, and FIFO is first-in, first-out. The two round robin strategies are the ones that are straight and of the highest slope. The remaining curves are a family for $\alpha$ of 0 through $\alpha$ of $\infty$; the former is nearest to horizontal, and the latter is the curved line nearest to vertical.

The primary effect of increasing $\alpha$ is to insert more short messages before long messages, thus trading off long message latency for short message latency. We assume that short mes-

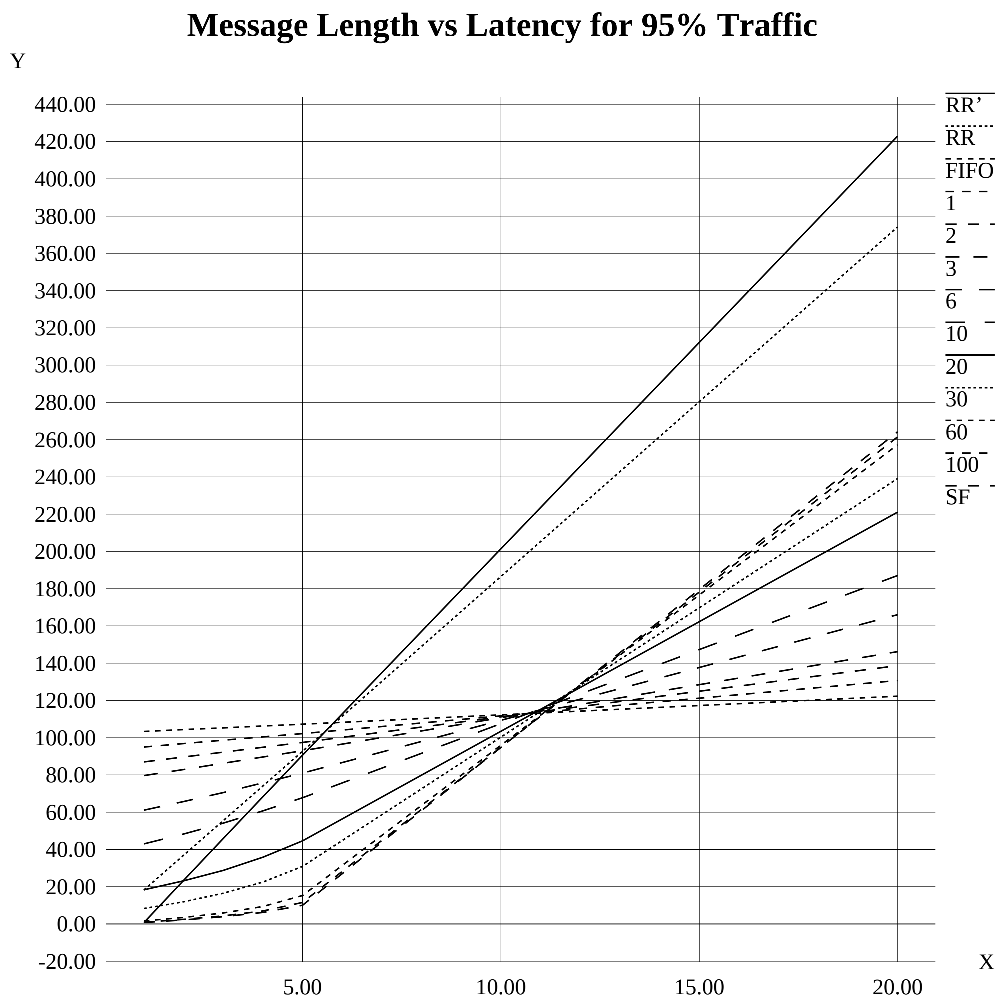


Figure 2: Message length vs latency for 95% traffic.

sage latency is extremely important, and that short messages will outnumber long messages significantly. Yet, long message latency is of some importance and should factor into our calculations. Figure 1 shows the impact of message length on message latency for a traffic rate of 50%; figure 2 shows the same graph for a traffic rate of 95%. Even with traffic as light as 50%, it is obvious that scheduling has a large impact on message latency. For one-packet messages, average in-queue latency varied from an average of 0.35 (for round robin with insertion at the head), to 0.41 (for shortest-first), to 1.18 (for standard round robin), to 5.46 for FIFO. For twenty-packet messages, average in-queue latency varied from 24.41 for FIFO to 39.05 for round robin with insertion at the head of the queue.

At 95% traffic, the impact was much more pronounced. In this case, for one-packet messages, average in-queue latency for standard round robin was 18.25, while for shortest-first it was only 0.79, more than twenty times faster. For twenty-packet messages, standard round robin yielded an average in-queue latency of 374.23; this was much worse than even shortest-first with an average of 264.23. In general, apart from fairness considerations, for all message lengths and traffic densities, standard round robin is always slower than shortest-first, and usually significantly slower.

On the other hand, round robin is always faster than FIFO for short messages, and always slower than FIFO for long messages. We next show how we compare FIFO, and scheduling with other $\alpha$ values, under these circumstances.

To weight things appropriately, we envision a 'typical' application that calculates, sends out a message, waits for it to be received, and then calculates again. We assume that this application generates messages according to the current traffic distribution. While this is not typical of applications (most will generate highly skewed distributions), it is a useful approximation of the entire pool of applications. The primary metric of message queue delay to the application is the total amount of time its messages spend in the queue. If we normalize this over the number of messages generated, we find that the overall average message latency is a good metric of the cost to the application of the message queue delay.

Figure 3 shows how latency changes with traffic load for the various strategies. Because of the large variance in latency for the different traffic loads and strategies, the vertical scaling makes it difficult to distinguish the lines. For this reason, we adopt the shortest-first average latency value as the vertical unit, and plot the other strategies as a fraction of that value, yielding the graph shown in figure 4. In this graph, curves closer to the horizontal line $y = 1$ reflect more desirable latencies.

Consider the strategies under a 50% traffic load. The optimal, shortest-first, gives an overall average queue latency of 5.92 time units. The standard and insert-at-head round robin strategies yield an average of 7.98 and 8.10 time units, respectively, making the delay 35% and 37% higher than optimal. This is a large decrease in performance for such light traffic. The FIFO strategy yields an average of 9.14 time units, 54% worse than shortest-first. Even a low $\alpha$ value such as 1 yields an average time of 7.46, beating round robin and FIFO. An $\alpha$ value of only 10 yields an average time of 5.98, within one percent of optimal.

Now let us consider a traffic load of 95%. The optimal in-queue average latency is 30.59 time units. The standard and insert-at-head round robin strategies yield an average of 87.34 and 83.46 time units, respectively, for an increase in delay of 186% and 173%, respectively. FIFO yields a poor 107.01 time units, more than three times longer than optimal. An $\alpha$ of 10, with an average delay of 68.00, beats FIFO and both round robin strategies handily. An $\alpha$ of 100 yields an average delay of 30.87, within one percent of optimal.

The graph shows that increasing the traffic load requires $\alpha$ to increase in order to stay close to the optimal throughput. At high traffic loads, low $\alpha$ values behave more like FIFO than like shortest-first. We next derive a quantitative analysis of the bad effects of a too-large $\alpha$ that will allow is to better understand the trade-offs associated with this parameter.

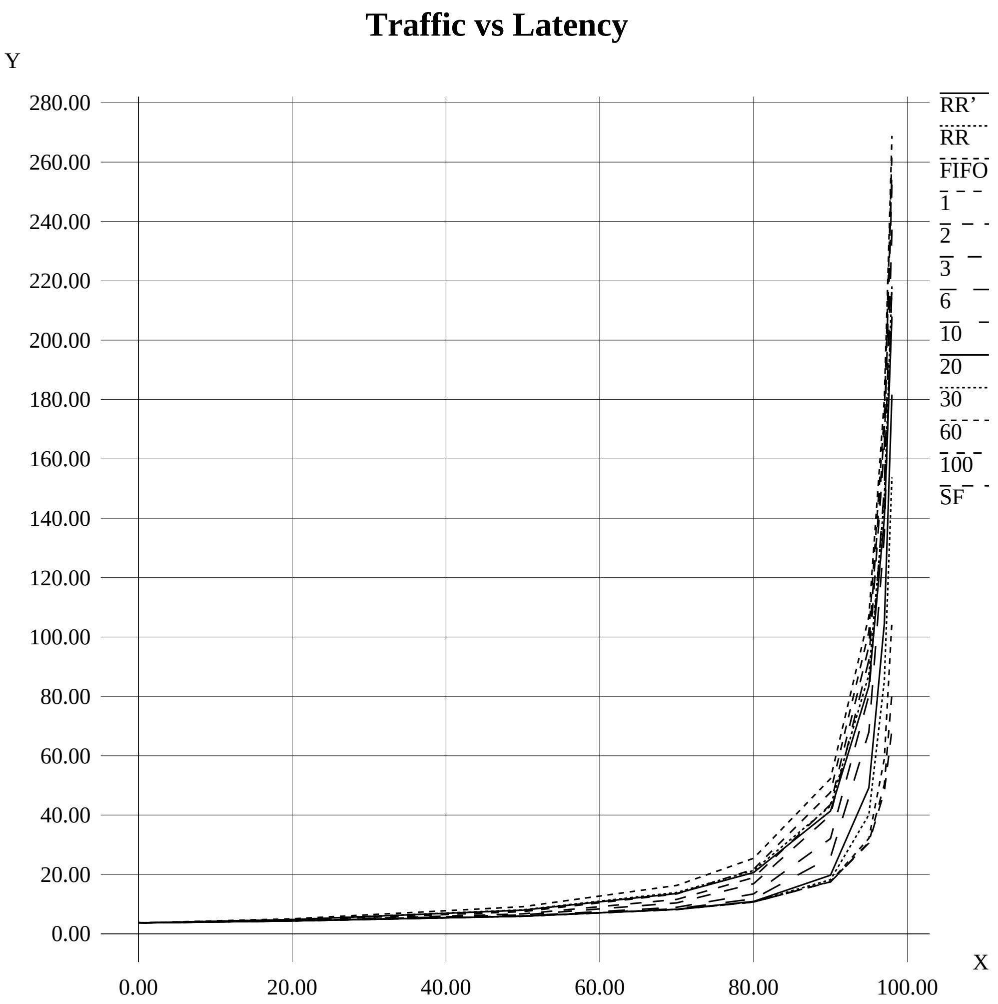


Figure 3: Traffic versus overall latency. This graph is difficult to read because of the large vertical scale; yet, the performance differences for a given utilization are large.

# 9 An Analysis of Large Alpha

Our scheduling algorithm ensures several invariants, independent of $\alpha$.

First, for finite $\alpha$ and message lengths, if the queue does not grow without bound, every message is eventually delivered. That is, starvation cannot occur. This is because $c$ increases with each packet sent, so eventually every new message will be inserted in the priority queue after a given message.

Secondly, if an application submits a message of length $l$ at time $c$, no message of length $l' \geq l$ that is submitted after the first message can slow down its delivery. This axiom does not hold for round robin. This means that for every message size, any $\alpha$ provides FIFO

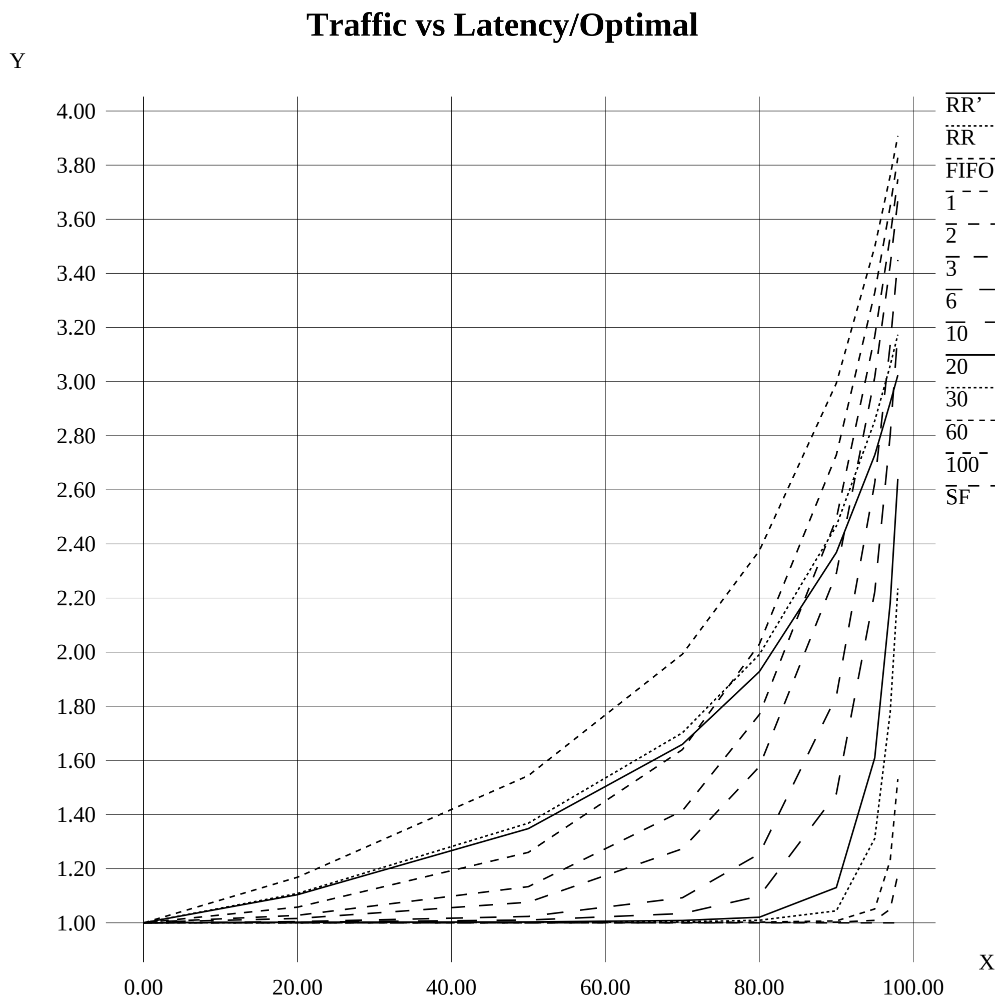


Figure 4: Traffic versus overall latency as a factor of optimal latency.

scheduling among those messages.

More generally, no message of length $l'$ submitted more than $\alpha(l - l')$ time units after $c$ can slow down delivery of the original message. Messages that are much smaller than $l$ have a longer time in which they can be submitted and enter the queue before the original message than do messages that are close to $l$ in size.

If we do not limit the number of applications, or do not limit the number of outstanding messages from any given application, any number of messages might be inserted at a given point in time. Therefore, there is no upper bound on the maximum delay that might be incurred due to optimal scheduling. We can compare the situation to a FIFO queue, however. We can associate a meaning with the value $slop = \alpha(l-1)$. That meaning is 'enter me into a FIFO queue, but you can pretend that I entered as late as the current time plus $slop$ if it will improve the average latency'. Thus, our scheduling algorithm optimizes the average latency such that no ones *slop* constraint is violated. A higher *slop* allows the average latency to be closer to optimal.

This raises the question of how much of *slop* is generally taken advantage of during a run. Our simulation results show that on average only long messages are slowed down, as we would expect. What is surprising is how little long messages are slowed down, even with high $\alpha$ and therefore *slop* values. For instance, at a traffic rate of 95%, an $\alpha$ value of one million (virtually infinity), the average queue wait time of long messages was 264.23 time units, versus 122.30 for FIFO. This is the worst case observed during our entire simulation. While long messages were delayed by a factor of two longer than normal, the overall average message latency decreased by more than a factor of three.

# 10 Using Alpha as a Priority Scheme

The discussion so far has assumed that $\alpha$ is a constant controlled by the queue manager, perhaps varying slowly over time but roughly the same for all messages entering during any short interval. It is also possible that $\alpha$ can be used on a per-message basis as a rough priority indicator; messages with high $\alpha$ can be displaced by messages with low $\alpha$.

# 11 Ensuring In-Order Delivery of Messages

One possible objection to the use of a scheduling algorithm such as alpha scheduling is that messages sent from the same application might arrive out of order. (We assume the interconnect provides some facility for ensuring that the packets from a single message arrive in-order where this is important.) This difficulty is easily resolved by associating with each application a field that stores the priority field of the most recently sent message. With this field, it is a simple matter to ensure that the priorities of successive messages from the same application form an increasing sequence, and thus will be delivered in order. This field can be reset to zero any time that the application submits a message when there is no outstanding message from that application in the queue.

# 12 Adversary Applications

Another consideration in a message scheduling strategy is knowledge of how mean-spirited applications can take advantage of the strategy to maximize their bandwidth. For instance, with the standard Unix system scheduler, the user with the most processes wins, tempting users to spawn many processes in order to maximize their share of computing resources. Yet, the very multiplicity of these processes decreases the efficiency of the system by increasing the cache miss rate, the context switch rate, the page miss rate, and swamping other system resources.

If applications are allowed to queue several concurrent messages, then round robin message scheduling suffers the same fate—the more messages enqueued, the larger share of interconnect performance an application gets. Similarly, for FIFO scheduling, an application that submits a large number of messages at the same time will significantly delay messages submitted by later, more civilized applications.

The $\alpha$ scheduling algorithm suffers the same difficulties under such scenarios. A possible solution is to limit the number of messages a single application may submit at any given time. The computational cost of implementing such a strategy, especially if it involves the generation of CPU interrupts or additional in-line processing at the packet submission point, may overwhelm its advantages.

If we take this situation to the limit, where an application may only submit a single message at a time, the scheduling strategy can still be exploited. In this case, round robin is bandwidth-fair to all applications, regardless of message length, assuming that no extra overhead is incurred between messages from the same application. Any such extra overhead would favor long messages over short messages. FIFO, on the other hand, is message-fair to applications, which implies that long messages will get a correspondingly greater share of available bandwidth. Shortest-first favors short messages, because long messages can be starved indefinitely.

The $\alpha$ scheduling strategy is a continuum between FIFO and shortest-first. An interesting point would be the traffic- and workload-dependent value of $\alpha$ for which fairness is most closely attained. This happens when the message size versus latency curve is most nearly a straight line that passes through the origin. (We shift the line by a one time unit to include the time it takes to send the final packet through the interconnect.) Because we have so few message sizes in our sample workload, and because all of the curves do not fit a straight-line model, we calculate and compare the average latencies divided by the message length (or the effective packet period for messages of a given size) for each of the simulation runs to analyze the fit. We primarily consider the range of these values.

For a given traffic rate and workload, there exists a value of $\alpha$ for which the range between the minimum and the maximum effective packet period is minimized. For instance, at a traffic load of 50%, an $\alpha$ of six yields an effective packet period of between 1.39 (for three-packet messages) and 1.61 (for twenty-packet messages.) For this $\alpha$, overall average latency is within 2% of optimal. Thus, we have the bandwidth-fairness of round robin but a 2% lower average latency.

At the 95% traffic rate, an $\alpha$ of sixty gives a minimum effective packet period for messages of two packets of 2.25, while the maximum effective packet period is for messages of length twenty-five at 12.92. At this point, the average latency is still 5% greater than optimal, and it is 62% less than the average latency for round robin! A bit more bandwidth-fair is an $\alpha$ of thirty, with effective packet periods running from 5.82 (for three-packet messages) to 12.01

(for twenty-packet messages), with an average latency of 8.75, still more than twice as fast as round robin.

Note that these calculations do not take into account the latency due to packet assembly, direct-memory access, or delivery. These factors contribute to raise the latency curve, so a higher $\alpha$ than the ones calculated above would likely be more bandwidth-fair.

# 13 Tracking Alpha

Rather than fixing $\alpha$ at a particular value, it might be useful to have $\alpha$ track the interconnect load as shown by the queue length in some fashion. Some reasonable upper limit (perhaps 100) and lower limit (perhaps 1), along with appropriate weighting of the queue length, would probably yield an overall scheduling strategy with a high degree of bandwidth fairness, no starvation, and a near-optimal average latency—the best features of the round robin, FIFO, and shortest-first strategies.

Depending on how traffic arrives at the node, and how packets are accepted by the interconnect, the dynamic behavior of the queue length over time might exhibit some extreme swings. For this reason, we recommend that if tracking alpha is implemented, no or little hysterisis be used. If the queue is empty for a period of time so $\alpha$ gets very small, a sudden influx of messages should allow $\alpha$ to climb relatively rapidly. This is an interesting avenue for future exploration.

Another avenue of exploration is how different workloads affect the appropriate value of $\alpha$.

# 14 Conclusion

In this report, we have introduced a new scheduling strategy, alpha scheduling, and shown how it can improve the effective performance of an interconnect by simply scheduling the packets within messages appropriately.

# 15 Raw Data

This section gives the raw data from which the conclusions in this report were drawn.

Traffic = 0.01%; packets = 66,479.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| RR | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| FIFO | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| 1 | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| 2 | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| 3 | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| 6 | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| 10 | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| 20 | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| 30 | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| 60 | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| 100 | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |
| SF | 0.00 | 0.00 | 1.02 | 2.02 | 3.01 | 3.99 | 18.97 | 3.74 |

Traffic = 1%; packets = 454,428.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 0.05 | 0.01 | 1.02 | 2.03 | 3.05 | 4.04 | 19.19 | 3.73 |
| RR | 0.05 | 0.02 | 1.03 | 2.03 | 3.05 | 4.03 | 19.18 | 3.73 |
| FIFO | 0.05 | 0.07 | 1.06 | 2.06 | 3.06 | 4.04 | 19.05 | 3.74 |
| 1 | 0.05 | 0.03 | 1.03 | 2.03 | 3.04 | 4.02 | 19.09 | 3.72 |
| 2 | 0.05 | 0.02 | 1.02 | 2.02 | 3.04 | 4.02 | 19.11 | 3.72 |
| 3 | 0.05 | 0.01 | 1.02 | 2.02 | 3.03 | 4.02 | 19.12 | 3.72 |
| 6 | 0.05 | 0.01 | 1.02 | 2.02 | 3.03 | 4.02 | 19.13 | 3.71 |
| 10 | 0.05 | 0.01 | 1.02 | 2.01 | 3.03 | 4.02 | 19.13 | 3.71 |
| 20 | 0.05 | 0.01 | 1.02 | 2.01 | 3.03 | 4.02 | 19.14 | 3.71 |
| 30 | 0.05 | 0.01 | 1.02 | 2.01 | 3.03 | 4.02 | 19.14 | 3.71 |
| 60 | 0.05 | 0.01 | 1.02 | 2.01 | 3.03 | 4.02 | 19.14 | 3.71 |
| 100 | 0.05 | 0.01 | 1.02 | 2.01 | 3.03 | 4.02 | 19.14 | 3.71 |
| SF | 0.05 | 0.01 | 1.02 | 2.01 | 3.03 | 4.02 | 19.14 | 3.71 |

Traffic = 20%; packets = 2,148,042.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 1.23 | 0.13 | 1.40 | 2.65 | 3.93 | 5.16 | 23.96 | 4.78 |
| RR | 1.23 | 0.34 | 1.51 | 2.68 | 3.91 | 5.09 | 23.68 | 4.80 |
| FIFO | 1.23 | 1.38 | 2.36 | 3.33 | 4.38 | 5.34 | 20.36 | 5.06 |
| 1 | 1.23 | 0.61 | 1.67 | 2.71 | 3.80 | 4.82 | 21.34 | 4.58 |
| 2 | 1.23 | 0.37 | 1.45 | 2.51 | 3.62 | 4.70 | 21.74 | 4.45 |
| 3 | 1.23 | 0.28 | 1.35 | 2.44 | 3.56 | 4.63 | 21.95 | 4.40 |
| 6 | 1.23 | 0.20 | 1.27 | 2.36 | 3.48 | 4.57 | 22.21 | 4.35 |
| 10 | 1.23 | 0.18 | 1.24 | 2.33 | 3.44 | 4.53 | 22.36 | 4.34 |
| 20 | 1.23 | 0.17 | 1.23 | 2.31 | 3.41 | 4.50 | 22.50 | 4.34 |
| 30 | 1.23 | 0.17 | 1.22 | 2.30 | 3.41 | 4.49 | 22.53 | 4.33 |
| 60 | 1.23 | 0.16 | 1.22 | 2.30 | 3.41 | 4.49 | 22.53 | 4.33 |
| 100 | 1.23 | 0.16 | 1.22 | 2.30 | 3.41 | 4.49 | 22.53 | 4.33 |
| SF | 1.23 | 0.16 | 1.22 | 2.30 | 3.41 | 4.49 | 22.53 | 4.33 |

Traffic = 50%; packets = 5,928,080.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 5.08 | 0.35 | 2.45 | 4.53 | 6.64 | 8.68 | 39.05 | 7.98 |
| RR | 5.08 | 1.18 | 2.97 | 4.79 | 6.67 | 8.53 | 37.56 | 8.10 |
| FIFO | 5.08 | 5.46 | 6.41 | 7.42 | 8.46 | 9.45 | 24.41 | 9.14 |
| 1 | 5.08 | 2.77 | 4.00 | 5.23 | 6.50 | 7.69 | 27.49 | 7.46 |
| 2 | 5.08 | 1.57 | 2.86 | 4.18 | 5.57 | 6.93 | 29.13 | 6.71 |
| 3 | 5.08 | 1.04 | 2.31 | 3.69 | 5.13 | 6.55 | 30.03 | 6.37 |
| 6 | 5.08 | 0.60 | 1.82 | 3.16 | 4.62 | 6.13 | 31.22 | 6.06 |
| 10 | 5.08 | 0.49 | 1.69 | 2.98 | 4.41 | 5.95 | 31.81 | 5.98 |
| 20 | 5.08 | 0.43 | 1.62 | 2.89 | 4.28 | 5.80 | 32.31 | 5.94 |
| 30 | 5.08 | 0.42 | 1.59 | 2.86 | 4.24 | 5.75 | 32.52 | 5.93 |
| 60 | 5.08 | 0.41 | 1.58 | 2.84 | 4.21 | 5.71 | 32.65 | 5.92 |
| 100 | 5.08 | 0.41 | 1.58 | 2.83 | 4.21 | 5.70 | 32.67 | 5.92 |
| SF | 5.08 | 0.41 | 1.58 | 2.83 | 4.21 | 5.70 | 32.68 | 5.92 |

Traffic = 70%; packets = 16,271,947.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 12.18 | 0.51 | 4.14 | 7.74 | 11.30 | 14.83 | 66.73 | 13.59 |
| RR | 12.18 | 2.51 | 5.52 | 8.55 | 11.59 | 14.67 | 62.44 | 13.94 |
| FIFO | 12.18 | 12.61 | 13.63 | 14.64 | 15.61 | 16.61 | 31.69 | 16.32 |
| 1 | 12.18 | 7.98 | 9.42 | 10.88 | 12.28 | 13.69 | 36.70 | 13.43 |
| 2 | 12.18 | 5.06 | 6.68 | 8.39 | 10.10 | 11.83 | 40.16 | 11.58 |
| 3 | 12.18 | 3.33 | 4.97 | 6.80 | 8.69 | 10.60 | 42.49 | 10.43 |
| 6 | 12.18 | 1.37 | 2.85 | 4.60 | 6.61 | 8.78 | 46.04 | 8.95 |
| 10 | 12.18 | 0.86 | 2.22 | 3.79 | 5.71 | 7.97 | 47.76 | 8.47 |
| 20 | 12.18 | 0.66 | 1.96 | 3.44 | 5.20 | 7.43 | 49.09 | 8.26 |
| 30 | 12.18 | 0.62 | 1.90 | 3.35 | 5.07 | 7.25 | 49.59 | 8.22 |
| 60 | 12.18 | 0.58 | 1.85 | 3.29 | 4.97 | 7.10 | 50.05 | 8.20 |
| 100 | 12.18 | 0.58 | 1.84 | 3.27 | 4.95 | 7.05 | 50.19 | 8.19 |
| SF | 12.18 | 0.57 | 1.84 | 3.27 | 4.94 | 7.04 | 50.24 | 8.19 |

Traffic = 80%; packets = 396,336,637.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 21.24 | 0.60 | 6.15 | 11.66 | 17.13 | 22.56 | 101.96 | 20.65 |
| RR | 21.24 | 4.13 | 8.70 | 13.28 | 17.88 | 22.54 | 93.60 | 21.33 |
| FIFO | 21.24 | 21.74 | 22.74 | 23.74 | 24.73 | 25.75 | 40.74 | 25.44 |
| 1 | 21.24 | 15.80 | 17.38 | 18.95 | 20.50 | 22.06 | 46.96 | 21.74 |
| 2 | 21.24 | 11.35 | 13.28 | 15.25 | 17.27 | 19.30 | 51.86 | 18.95 |
| 3 | 21.24 | 8.21 | 10.26 | 12.48 | 14.81 | 17.17 | 55.62 | 16.89 |
| 6 | 21.24 | 3.43 | 5.35 | 7.65 | 10.31 | 13.19 | 62.51 | 13.44 |
| 10 | 21.24 | 1.58 | 3.19 | 5.16 | 7.71 | 10.75 | 66.57 | 11.77 |
| 20 | 21.24 | 0.87 | 2.26 | 3.91 | 6.04 | 9.00 | 69.58 | 10.93 |
| 30 | 21.24 | 0.76 | 2.12 | 3.71 | 5.72 | 8.56 | 70.51 | 10.81 |
| 60 | 21.24 | 0.69 | 2.02 | 3.57 | 5.50 | 8.20 | 71.41 | 10.74 |
| 100 | 21.24 | 0.67 | 1.99 | 3.53 | 5.44 | 8.08 | 71.74 | 10.72 |
| SF | 21.24 | 0.66 | 1.97 | 3.51 | 5.40 | 8.01 | 71.95 | 10.71 |

Traffic = 90%; packets = 368,394,590.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 48.04 | 0.70 | 11.86 | 22.99 | 34.06 | 45.10 | 207.34 | 41.37 |
| RR | 48.04 | 8.82 | 17.99 | 27.19 | 36.41 | 45.70 | 185.99 | 43.09 |
| FIFO | 48.04 | 48.62 | 49.59 | 50.59 | 51.60 | 52.63 | 67.58 | 52.30 |
| 1 | 48.04 | 41.17 | 42.89 | 44.62 | 46.34 | 48.06 | 75.17 | 47.67 |
| 2 | 48.04 | 34.57 | 36.89 | 39.26 | 41.66 | 44.08 | 82.00 | 43.56 |
| 3 | 48.04 | 29.00 | 31.73 | 34.61 | 37.56 | 40.56 | 87.99 | 40.02 |
| 6 | 48.04 | 17.20 | 20.36 | 23.99 | 27.97 | 32.19 | 101.86 | 32.09 |
| 10 | 48.04 | 8.90 | 11.78 | 15.31 | 19.65 | 24.61 | 113.66 | 25.80 |
| 20 | 48.04 | 2.41 | 4.37 | 6.89 | 10.38 | 15.35 | 126.65 | 19.75 |
| 30 | 48.04 | 1.26 | 2.85 | 4.86 | 7.66 | 12.10 | 130.81 | 18.24 |
| 60 | 48.04 | 0.86 | 2.28 | 4.01 | 6.33 | 10.04 | 133.73 | 17.59 |
| 100 | 48.04 | 0.79 | 2.18 | 3.87 | 6.10 | 9.61 | 134.63 | 17.51 |
| SF | 48.04 | 0.75 | 2.11 | 3.76 | 5.93 | 9.27 | 135.52 | 17.47 |

Traffic = 95%; packets = 353,635,860.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 102.69 | 0.74 | 23.31 | 45.80 | 68.28 | 90.62 | 422.94 | 83.46 |
| RR | 102.69 | 18.25 | 36.85 | 55.44 | 74.09 | 92.71 | 374.23 | 87.34 |
| FIFO | 102.69 | 103.33 | 104.34 | 105.29 | 106.31 | 107.26 | 122.30 | 107.01 |
| 1 | 102.69 | 95.00 | 96.86 | 98.64 | 100.46 | 102.19 | 130.66 | 101.83 |
| 2 | 102.69 | 87.00 | 89.64 | 92.23 | 94.87 | 97.44 | 138.68 | 96.88 |
| 3 | 102.69 | 79.64 | 82.93 | 86.24 | 89.63 | 92.97 | 146.20 | 92.27 |
| 6 | 102.69 | 61.06 | 65.68 | 70.54 | 75.73 | 80.99 | 166.03 | 80.32 |
| 10 | 102.69 | 43.00 | 48.27 | 54.09 | 60.71 | 67.75 | 187.10 | 68.00 |
| 20 | 102.69 | 18.36 | 23.13 | 28.76 | 35.85 | 44.60 | 221.10 | 49.24 |
| 30 | 102.69 | 8.33 | 11.92 | 16.45 | 22.49 | 30.90 | 239.12 | 40.13 |
| 60 | 102.69 | 1.65 | 3.49 | 5.91 | 9.41 | 15.28 | 257.30 | 32.16 |
| 100 | 102.69 | 0.98 | 2.46 | 4.33 | 6.97 | 11.52 | 261.50 | 30.87 |
| SF | 102.69 | 0.79 | 2.19 | 3.90 | 6.22 | 10.06 | 264.23 | 30.59 |

Traffic = 97%; packets = 2,209,436,237.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 175.86 | 1.64 | 39.22 | 76.75 | 114.23 | 151.67 | 709.38 | 139.97 |
| RR | 175.86 | 30.85 | 62.04 | 93.22 | 124.43 | 155.71 | 626.17 | 146.55 |
| FIFO | 175.86 | 176.45 | 177.47 | 178.44 | 179.42 | 180.47 | 195.49 | 180.15 |
| 1 | 175.86 | 167.76 | 169.66 | 171.51 | 173.32 | 175.19 | 204.17 | 174.76 |
| 2 | 175.86 | 159.15 | 161.91 | 164.63 | 167.34 | 170.10 | 212.72 | 169.44 |
| 3 | 175.86 | 150.95 | 154.49 | 158.03 | 161.59 | 165.20 | 220.94 | 164.34 |
| 6 | 175.86 | 128.76 | 134.16 | 139.75 | 145.52 | 151.43 | 243.77 | 150.31 |
| 10 | 175.86 | 104.22 | 111.19 | 118.64 | 126.62 | 134.99 | 270.36 | 134.26 |
| 20 | 175.86 | 61.65 | 69.89 | 79.13 | 89.68 | 101.74 | 321.34 | 104.51 |
| 30 | 175.86 | 36.68 | 44.33 | 53.31 | 64.03 | 77.25 | 356.07 | 85.22 |
| 60 | 175.86 | 8.33 | 12.56 | 18.14 | 25.67 | 36.60 | 407.35 | 58.97 |
| 100 | 175.86 | 1.93 | 4.01 | 6.87 | 11.11 | 18.28 | 427.12 | 50.31 |
| SF | 175.86 | 0.81 | 2.22 | 3.96 | 6.35 | 10.41 | 436.28 | 47.90 |

Traffic = 98%; packets = 782,215,691.

| $\alpha$ | qlen | lat1 | lat2 | lat3 | lat4 | lat5 | lat20 | avlat |
|---|---|---|---|---|---|---|---|---|
| RR' | 264.51 | 0.77 | 57.01 | 113.17 | 169.27 | 225.32 | 1061.99 | 208.00 |
| RR | 264.51 | 46.13 | 92.59 | 139.04 | 185.50 | 232.04 | 931.31 | 218.29 |
| FIFO | 264.51 | 265.17 | 266.16 | 267.11 | 268.08 | 269.10 | 284.09 | 268.82 |
| 1 | 264.51 | 256.29 | 258.20 | 260.04 | 261.86 | 263.72 | 292.93 | 263.31 |
| 2 | 264.51 | 247.36 | 250.16 | 252.93 | 255.67 | 258.46 | 301.74 | 257.80 |
| 3 | 264.51 | 238.71 | 242.36 | 246.00 | 249.64 | 253.33 | 310.34 | 252.44 |
| 6 | 264.51 | 214.50 | 220.34 | 226.31 | 232.40 | 238.60 | 334.82 | 237.27 |
| 10 | 264.51 | 186.00 | 194.06 | 202.45 | 211.25 | 220.36 | 364.63 | 219.00 |
| 20 | 264.51 | 130.29 | 141.39 | 153.32 | 166.42 | 180.73 | 426.94 | 181.68 |
| 30 | 264.51 | 91.26 | 103.14 | 116.31 | 131.23 | 148.36 | 475.08 | 153.76 |
| 60 | 264.51 | 31.53 | 40.54 | 51.60 | 65.35 | 83.00 | 563.56 | 105.32 |
| 100 | 264.51 | 8.30 | 12.96 | 19.30 | 28.11 | 41.04 | 612.74 | 81.02 |
| SF | 264.51 | 0.81 | 2.23 | 3.99 | 6.41 | 10.60 | 644.64 | 68.79 |

# 16 Simulation Program

This section is a listing of the simulation program, written in CSIM.

```
/*
 *   A simple scheduling test for csim.
 */

#include <cpp.h>
#include <math.h>
```

```
const int MAXMESLENGTH = 100 ;

double traffic = 90.0, queuedelay = 200.0, galpha, avgmeslen = 0.0 ;
long packsin ;

struct args {
   char arg ;
   double *param ;
} options[] = { { 'a', &galpha }, { 't', &traffic },
   { 'q', &queuedelay }, { 'm', 0 }, { 0, 0 } } ;

double defaultalphas[] =
   { -2, -1, 0, 1, 2, 3, 6, 10, 20, 30, 60, 100, 1e6 } ;

struct mesdis {
   int low, high ;
   double freq ;
} mesdis[10] = { { 1, 5, 0.9 }, { 20, 20, 0.1 }, { 0, 0, 0.0 },
   { 0, 0, 0.0 }, { 0, 0, 0.0 }, { 0, 0, 0.0 }, { 0, 0, 0.0 },
   { 0, 0, 0.0 }, { 0, 0, 0.0 }, { 0, 0, 0.0 }} ;
int nummesdis = 0 ;

event newelement("newelement") ;

class Message {
 public:
   void sendpack(void) { numleft-- ; }
   int done(void) { return (numleft == 0) ; }
   TIME entry(void) { return entrytime ; }
   int meslen(void) { return len ; }
   Message(int ilen) { numleft = len = ilen ; entrytime = simtime() ; }
 private:
   int len, numleft ;
   TIME entrytime ;
};

class Average {
 public:
   double nval(void) { return n ; }
   double average(void) { return (n ? sum / n : 0) ; }
   int centaver(void) { return (int)floor(100.0*average()+0.5) ; }
   void newval(double v) { sum += v ; n++ ; }
   Average(void) { n = sum = 0 ; }
 private:
   double n, sum ;
```

```
} ;

/*
 *   The schedule queue is implemented as a linked list of void *
 *   based on the current time and alpha.
 */

class schedlink {
 public:
   schedlink(void *el, double pr) { data = el ; next = 0 ; pri = pr ; }
   void *data ;
   double pri ;
   schedlink *next ;
};

class ScheduleQueue *sqs ;

class ScheduleQueue {
 public:
   int empty(void) { return (head == 0) ; }
   void *dequeue(void) ;
   void *firsttick(void) { tick++ ; return head->data ; }
   void insert(void *el, int len) ;
   void toend(void) ;
   int packet_acceptor(void) ;
   void reportvals(void) ;
   void newqpoint(void)
      { queuelength.newval((double)(packsin-packsout)) ; }
   void updatehdpr(void) { if (alpha > 0.0) head->pri -= alpha ; }
   ScheduleQueue(double al) { alpha = al ; head = 0 ; tick = 0 ;
                              next = sqs ; sqs = this ; }
   class ScheduleQueue *next ;
 private:
   double alpha ;
   schedlink *head ;
   int tick ;
   long packsout ;
   class Average mesrep[MAXMESLENGTH+1] ;
   class Average queuelength ;
};

void *ScheduleQueue::dequeue(void) {
   schedlink *sl = head ;
   void *t = sl->data ;
   double latency = simtime() - ((Message *)t)->entry() ;
```

```
   int len = ((Message *)t)->meslen() ;

   mesrep[len].newval(latency) ;
   mesrep[0].newval(latency) ;
   head = head->next ;
   delete sl ;
   return t ;
}

void ScheduleQueue::insert(void *el, int len) {
   double pr = (alpha <= 0.0) ? ((alpha < -1.5) ? 0.0 : tick)
                              : (alpha * len + tick) ;
   schedlink *sl = new schedlink(el, pr) ;
   if (head == 0) {
      head = sl ;
   } else if (head->pri > pr) {
      sl->next = head ;
      head = sl ;
   } else {
      schedlink *hsl = head ;
      while (hsl->next && hsl->next->pri <= pr)
         hsl = hsl->next ;
      sl->next = hsl->next ;
      hsl->next = sl ;
   }
}

void ScheduleQueue::toend(void) {
   schedlink *sl = head ;
   sl->pri = tick ;
   head = head->next ;
   if (head == 0) {
      head = sl ;
   } else {
      schedlink *hsl = head ;
      while (hsl->next)
         hsl = hsl->next ;
      sl->next = hsl->next ;
      hsl->next = sl ;
   }
}

int ScheduleQueue::packet_acceptor(void) {
   Message *mess ;
   if (!empty()) {
```

```
      packsout++ ;
      mess = (Message *)firsttick() ;
      mess->sendpack() ;
      updatehdpr() ; /* reflect that we just shipped a packet */
      if (mess->done()) {
         dequeue() ;
         delete mess ;
      } else if (alpha < 0.0) {
 toend() ;
      }
      return 1 ;
   } else
      return 0 ;
}

void overall_acceptor() {
   create("packet_acceptor") ;
   while (1) {
      int s=0 ;
      for (ScheduleQueue *sq = sqs; sq; sq = sq->next)
         s += sq->packet_acceptor() ;
      if (s)
         hold(0.5+expntl(0.5)) ;
      else
 newelement.wait() ;
   }
}

void message_generator() {
   create("message_generator") ;
   while (1) {
      double lenr = uniform(0.0, 0.99999999) ;
      int len ;
      ScheduleQueue *sq ;
      struct mesdis *md = mesdis ;

      while (md->freq <= lenr) {
 lenr -= md->freq ;
 md++ ;
      }
      if (md->freq > 1.0) {
 fprintf(stderr, "bad distribution\n") ;
 exit(10) ;
      }
      if (md->low != md->high)
```

```
        len = md->low +
           (int)floor(lenr/md->freq*(md->high - md->low + 1)) ;
      else
 len = md->low ;
      packsin += len ;
      for (sq = sqs; sq; sq = sq->next)
         sq->insert(new Message(len), len) ;
      newelement.set() ;
      hold(expntl(100.0 * avgmeslen / traffic)) ;
   }
}

void queue_size() {
   create("queue monitor") ;
   while (1) {
      ScheduleQueue *sq ;

      hold(queuedelay) ;
      for (sq = sqs; sq; sq = sq->next)
         sq->newqpoint() ;
   }
}

void ScheduleQueue::reportvals(void) {
   printf(">> %g %g %d %d %d", traffic, alpha, queuelength.centaver(),
      packsin, packsout) ;
   for (int i = 1; i<=MAXMESLENGTH; i++)
      if (mesrep[i].nval())
 printf(" %d", mesrep[i].centaver()) ;
   printf(" %d\n", mesrep[0].centaver()) ;
   fflush(stdout) ;
}

extern "C" void sim(int argc, char *argv[]) {
   double statdelay = 20000.0 ;
   struct mesdis *md ;

   create("sim") ;
   while (argc > 2 && argv[1][0] == '-') {
      struct args *argp = options ;
      argc-- ;
      argv++ ;
      while (argp->arg)
 if (argp->arg == argv[0][1]) {
    if (argp->param == 0) {
```

```
      switch(argp->arg) {
      case 'm':
                 md = &(mesdis[nummesdis]) ;
 if (sscanf(argv[1], "%d", &(md->low)) != 1 ||
     sscanf(argv[2], "%d", &(md->high)) != 1 ||
     sscanf(argv[3], "%lg", &(md->freq)) != 1) {
            (void)fprintf(stderr, "Oops; need three numbers.\n") ;
            exit(10) ;
                 }
 md->freq /= 100.0 ;
 nummesdis++ ;
 argv += 2 ;
 argc -= 2 ;
      }
   } else if (sscanf(argv[1], "%lg", argp->param) != 1) {
      (void)fprintf(stderr, "Oops; need a float param.\n") ;
      exit(10) ;
           }
   break ;
} else
           argp++ ;
     if (argp->arg == 0) {
(void)fprintf(stderr, "Bad argument.\n") ;
exit(10) ;
     } else if (argp->arg == 'a') {
        (void)(new ScheduleQueue(galpha)) ;
     }
     argc-- ;
     argv++ ;
  }
  if (nummesdis == 0)
     nummesdis = 2 ;
  mesdis[nummesdis].freq = 1000.0 ;
  for (int i=0; i<nummesdis; i++)
     avgmeslen += mesdis[i].freq *
        (mesdis[i].low + mesdis[i].high) / 2.0 ;
  if (sqs == 0) {
     for (int i=0; i<sizeof(defaultalphas)/sizeof(double); i++)
(void)(new ScheduleQueue(defaultalphas[i])) ;
  }
  {
     ScheduleQueue *nsq = 0, *t ;

     while (sqs) {
t = sqs ;
```

```
 sqs = sqs->next ;
 t->next = nsq ;
 nsq = t ;
      }
      sqs = nsq ;
   }
   message_generator() ;
   overall_acceptor() ;
   queue_size() ;
   while (1) {
      ScheduleQueue *sq ;

      hold(statdelay) ;
      printf("\n") ;
      for (sq = sqs; sq; sq = sq->next)
         sq->reportvals() ;
      statdelay = statdelay * 1.1 ;
   }
}
```