Distributed Round-Robin and First-Come First-Serve Protocols and Their Application to Multiprocessor Bus Arbitration †

Mary K. Vernon
Udi Manber

Computer Sciences Technical Report #745

February 1988

† To appear in *Proc. 15th Ann. Int'l. Symp. on Computer Architecture,* Honolulu, Hawaii, May 30 - June 2, 1988.

Distributed Round-Robin and First-Come First-Serve Protocols and Their Application to Multiprocessor Bus Arbitration

Mary K. Vernon

Computer Science Department
University of Wisconsin–Madison
Madison, WI 53706

Udi Manber

Department of Computer Science
University of Arizona
Tucson, AZ 85721

Abstract

Two new distributed protocols for fair and efficient bus arbitration are presented. The protocols implement round-robin (RR) and first-come first-serve (FCFS) scheduling, respectively. Both protocols use relatively few control lines on the bus, and their logic is simple. The round-robin protocol, which uses *statically assigned* arbitration numbers to resolve conflict during an arbitration, is more robust and simpler to implement than previous distributed RR protocols that are based on rotating agent priorities. The proposed FCFS protocol uses partly static arbitration numbers, and is the first practical proposal for a FCFS arbiter known to the authors. The proposed protocols thus have a better combination of efficiency, cost, and fairness characteristics than existing multiprocessor bus arbitration algorithms.

Three implementations of our RR protocol, and two implementations of our FCFS protocol, are discussed. Simulation results are presented that address: 1) the practical potential for unfairness in the simpler implementation of the FCFS protocol, 2) the practical implications of the higher waiting time variance in the RR protocol, and 3) the allocation of bus bandwidth among agents with unequal request rates in each protocol. The simulation results indicate that there is very little practical difference in the performance of the two protocols.

1. Introduction

A parallel contention arbiter, invented at Computing Devices of Canada [UKPS66] and by D. M. Taub [Taub84], is a popular method for multiprocessor bus arbitration. For example, it has been adopted in nearly all multiprocessor bus standards, including Futurebus, Fastbus, NuBus, and Multibus II [Taub84, Gust86, PWGr86, IMSC86].

The parallel contention arbiter is very efficient. Selection among up to 2^k competing devices can be carried out in approximately $k/2$ end-to-end bus propagation delays, plus a small amount of delay for the arbiter logic. Thus, the overhead of this distributed arbiter is comparable to the overhead of efficient central arbiters. However, there are at least three reasons for the popularity of this arbiter over the central arbiters. First, it requires very few wires on the bus to carry out the arbitration algorithm. The number of wires required is $\lceil \log_2 (N+1) \rceil$, where N is

This research was partially supported by the National Science Foundation under Presidential Young Investigator Awards DCR-8451397 and DCR-8451405, and by grants from IBM and Cray Research.

the number of devices that may compete for control of the bus, plus a few control lines. Second, priority scheduling of urgent requests is easily integrated with mechanisms for fair scheduling of non-priority requests in this arbiter. Third, the state of the arbiter is available and can be monitored on the bus. This is useful for software initialization of the system and for diagnosing system failures.

Assured access protocols have been designed to provide fairness in the parallel contention arbiter [Gust84]. These protocols are widely regarded as being fair. That is, it is widely believed that these protocols provide all devices on the bus with equal access to the bus [Gust84, Taub84, IMSC86, PWGr86, Hawl87]. However, recent modeling studies [KlCa86, VeLe88], and a recent trace simulation study [EgGi87], show that existing assured access protocols are actually quite unfair. Under important system assumptions, these studies show that the amount of bus bandwidth allocated to each device in a set of purportedly equal devices is a continuum, in which the most favorably treated device receives perhaps typically 10%, but in the worst case 100% more bandwidth than the least favorably treated device. The relative bus bandwidth allocated to each processor in a multiprocessor translates directly to the relative speeds at which application processes run on the processors.

This paper proposes two new protocols that can be implemented in the parallel contention arbiter. The protocols implement round-robin (RR) and first-come first-serve (FCFS) scheduling, respectively. The RR protocol implements *true* round-robin scheduling, identical to the central round-robin arbiter, yet retains the advantages of the parallel contention arbiter outlined above. The FCFS protocol implements scheduling that is *very close* to true first-come first-serve scheduling, and is the first practical proposal for a FCFS arbiter known to the authors. The RR and FCFS protocols are perfectly fair. Furthermore, the logic required to implement the RR and FCFS protocols is as simple as the logic required for the assured access protocols.

We wish to emphasize that the RR and FCFS protocols are very general and have the potential for wider applicability than bus arbitration. The protocols implement arbitration among a set of agents, each of which is identified by a unique, statically-assigned number. The only requirements are: 1) that there is an efficient mechanism for selecting the maximum out of any subset of agent identities, and 2) for the RR protocol, that each agent is able to know the identity of the winner at the end of each arbitration. It is the application of the protocols to bus arbitration that we focus on in this paper.

The organization of this paper is as follows. In section 2 we provide a review of the of the parallel contention arbiter, including a description of the information that is available to all devices on the bus. Section 3 describes our

new protocols which make use of this information to implement RR and FCFS scheduling. Section 4 presents a comparative analysis of the RR and FCFS protocols. The practical implications of the theoretical properties that distinguish the RR and FCFS algorithms are explored. The results of this section indicate that the advantages of either protocol relative to the other are very small. Section 5 contains the conclusions of this study.

2. Background

In this section we review the parallel contention arbiter and the existing assured access protocols for this arbiter. We then comment briefly on the fairness of the assured access protocols, and on how priority service of urgent requests is integrated with the assured access protocols. Throughout the discussion, we use the term *agent* to refer to any device that can request control of the bus to initiate a data transfer. All other devices on the bus can act only as slaves. The agent that has control of the bus will be called the current *bus master*.

Each bus line used by the arbiter has the important property that it carries the "wired-OR" of the signals applied by all agents. That is, the line is tied to a voltage source and each agent either lets the line float (logical "0"), or forces it to another particular voltage level (logical "1"). Applying a "1" to a line will be called *asserting* the line; allowing it to float back to "0" is *releasing* it.

2.1. The Parallel Contention Arbiter

The parallel contention arbiter is based on assigning each agent a unique k-bit arbitration number, which we will call its "identity". The value of k is $\lceil \log_2(N+1) \rceil$, where N is the maximum number of agents that can be attached to the bus. (No agent is assigned the identity "0".) For example, in the Futurebus standard, $k=6$.

An agent that wants control of the bus asserts a shared bus *request* line, and waits for a signal to start arbitration. The signal to start arbitration may be generated by the current bus master, a central timing controller, or any agent on the bus. The details and alternative possibilities for this mechanism are not important for the current study.

At the start of an arbitration, the agent applies its arbitration number to a set of arbitration lines provided on the bus for this purpose. The agent then monitors each of the arbitration lines, in parallel. If the value carried by line i is "1", but the agent is applying "0" to it, then the agent removes the lower-order $i-1$ bits of its identity. If line i drops back to "0", the agent reapplies the lower-order bits it removed before. For example, consider the case where two agents with identities 1010101 and 0011100, respectively, are requesting the bus. The first agent will remove its three lowest order bits, leaving 1010000, and the second agent will remove all its bits. Next, the first

agent will reapply its three lowest order bits, and the second agent will do nothing, since the most significant bit still remains. It is easy to see that after some period of time the system reaches steady state, in which the lines carry the maximum identity over all competing agents. The agent whose arbitration number matches the winning number becomes the next bus master. Note that at the end of the arbitration, each agent knows the identity of the winner, as well as whether it has won or lost.

The arbitration algorithm is cleverly designed so that the delay to reach steady state is very small. In fact, Taub has proven that the maximum delay to reach steady state is $\frac{k}{2}$ end-to-end bus propagation delays, plus a small amount of delay for the monitoring logic [Taub84]. His proof is based on a worst-case physical assignment of identities along the bus. In practical systems, the delay for arbitration safely occurs within one hundred nanoseconds. Furthermore, a scheme for *binary patterned* arbitration lines [John83] reduces the arbitration overhead to a single end-to-end bus propagation time, plus some overhead for comparison logic. However, the identity of the winning agent is not available to all agents on the bus in the binary patterned scheme.

The overall control of the arbitration, including starting an arbitration and handing over control to the winner, is synchronized by the clock in synchronous buses, or occurs in a self-timed fashion in asynchronous buses. As noted above, the details of this control are not important for the protocols studied in this paper.

2.2. Assured Access Protocols for the Parallel Contention Arbiter

The parallel contention arbiter reviewed above implements fixed priority service, in which an agent's priority is defined by its assigned arbitration number. Round-robin scheduling, implemented using a dynamic assignment of arbitration numbers, has been proposed. However, this scheme is less robust and more complex to implement than schemes that are based on static identities. The two fairness protocols discussed below use the static identities, and have been adopted in the bus standards that employ the parallel contention arbiter.

For non-priority requests, arbitration numbers in the parallel contention arbiter are used as a means for efficient distributed agreement on who will become the next bus master. To overcome the unfairness inherent in the basic priority selection mechanism, *assured access protocols* have been designed to provide all agents with equal access to the bus. These protocols are based on *batching* requests, such that all requests in a batch are served before any new requests can be made. In particular, requests in the batch from agents with low assigned identities will receive service before new requests can be made by agents with high assigned identities.

There are two distinct assured access protocols that implement slightly different batching rules. One protocol has been adopted in the Fastbus, NuBus, and Multibus II standards [Gust86, PWGr86, IMSC86]; the other has been adopted in the Futurebus standard.

In the first protocol, all requests that arrive to an idle bus assert the bus request line and form a batch. An agent in the batch competes during each arbitration until it has been granted ownership of the bus. An agent that generates a new request while a batch is in progress must wait for the batch to end before asserting the request line and competing for access. The end of the batch is generally signalled by a logical "0" on the request line, since each agent in the batch releases the request line at the start of its bus tenure. All requests that are waiting at the end of a batch assert the shared request line and form a new batch. Agents in a batch receive service in order of their assigned identities, according to the parallel contention arbitration.

In the second assured access protocol, an agent with a request asserts the request line and competes in successive arbitrations until acquiring the bus ownership status. At the completion of its bus tenure, the agent marks itself as "inhibited", and won't assert the request line or compete for bus ownership until a fairness release operation takes place. The fairness release operation is an arbitration cycle in which no agents assert the request line. In other words, either there are no outstanding requests, or all agents with outstanding requests are inhibited.

The second protocol implements a batching algorithm similar to the first protocol. A batch starts and ends with a fairness release cycle. No agent is bus master more than once in a batch, but an agent with a request that is generated during a batch is allowed to join the batch if the agent has not previously received service in the batch.

2.3. Fairness of the Assured Access Protocols

There is a significant source of unfairness in the above assured access protocols. In every batch, an agent always receives service *after* all agents in the batch that have higher identities. For multiprocessor systems in which the processors do not continue executing while waiting for a memory request to be satisfied, this means that the lower-identity processors execute at a slower rate. The difference in throughput between the most favorably treated agent (i.e. the agent with the highest assigned identity) and the least favorably treated agent is perhaps typically about 10%, and can be as high as 100% for each of the protocols described above [VeLe88]. A slightly modified version of the second protocol has a maximum of 10-15% difference in throughput. This type of unfairness that can lead to undesirable results when discovered by the users of the system. Furthermore, tightly coupled parallel algo-

rithms are often sensitive to the speed of the slowest processor. In this case, the unfairness can affect total system performance. The unfairness is completely eliminated in the RR and FCFS protocols we propose in Section 3.

2.4. Integration of Priority and Fairness in the Parallel Contention Arbiter

Priority scheduling of urgent requests is easily integrated with the assured access protocols in the parallel contention arbiter. In this case, agents follow the assured access protocol for non-priority requests, but ignore the protocol and compete in every arbitration for priority requests. Furthermore, an extra line can be provided on the bus, to be treated as the most significant bit of the agent's identity. Agents with priority requests assert this line during arbitration; agents with non-priority requests do not. This guarantees that all priority requests will be served before non-priority requests.

The integration of priority requests is also straightforward in the RR and FCFS protocols we propose in Section 3.

3. Two New Distributed Protocols

The key idea in designing distributed arbitration protocols is that some information about the status of all agents can be obtained very efficiently. The information that can be obtained efficiently is usually very partial (e.g., is anyone requesting the bus?), and consists of a set of bits that can be set and/or sensed simultaneously by all agents. In the parallel contention arbiter, the shared information consists of the following: 1) the wired-OR of whether or not each agent is requesting the bus, 2) control signals for starting an arbitration and handing over control to the winner, and 3) the wired-OR of each bit of the competing arbitration numbers, (or a subset of these numbers in the case of binary patterned arbitration lines). It seems at first that this kind of distributed information is very limited, but in fact it can be used as a basis for simple yet powerful algorithms.[1] Below we present two new protocols which show that it is possible to manipulate the given partial information to obtain fair agreement on the next agent to control the bus. Both protocols rely on an efficient *maximum finding* algorithm, such as the algorithm implemented by the parallel contention arbiter. One protocol enforces a round robin scheduling policy; the other enforces a first-come first-serve policy. In both cases, *statically assigned* identities are used in resolving conflict during an arbitration. We believe that these protocols are more robust and simpler to implement than the round-

[1] The general idea can be compared with the design of the Ethernet and other multiple access channel protocols. However, the possibilities are very different, since the propagation of signals is much faster in the current setting. There is no need for "backoff," nor probabilistic choices. Also, we do not assume to be able to detect collisions. We can only tell whether the line is busy or not.

robin protocol that can be implemented using a dynamic assignment of identities.

3.1. The Round Robin Protocol

The most obvious way to implement round-robin scheduling in the parallel contention arbiter is to use a dynamic assignment of identities for each agent. However, more careful thought reveals that a simple protocol which uses the statically assigned identities is possible. To see this, note that if an agent with assigned identity j is the winner of a given arbitration, then the round-robin algorithm will scan agents with assigned identities $j-1$ through 1 and then agents with identities N through j in the next arbitration. Our round-robin protocol is based on the important observation that the maximum finding algorithm will implement this round-robin scan if we provide a mechanism to specify that agents with identities $j-1$ through 1 have priority over agents with identities N through j, in each arbitration. Below we outline three possible implementations of the round-robin protocol that are based on this idea. All three implementations require the identity of the winning agent to be available to all agents. In other words, binary patterned arbitration lines cannot be used easily for the protocol. This is not a serious drawback, since the parallel contention arbiter with full arbitration lines is highly efficient.[2]

The first, and probably simplest implementation of the round-robin protocol requires an extra bit of shared information (i.e. an extra line on the bus). We call this bit the round-robin priority bit. The round-robin priority bit is treated as the most significant bit of the agent's identity. Each agent records the identity of the winning agent at the end of every arbitration, excluding the round-robin priority bit. An agent asserts the shared bus request line and competes in the next arbitration whenever it desires control of the bus. When an agent competes in an arbitration, the agent sets the round-robin priority bit to "1" if its static identity is smaller than the recorded identity of the winner of the previous arbitration. The logic needed to implement this protocol thus primarily consists of a register to store the winning identity, and a comparator to determine if the agent's assigned arbitration number is smaller than the recorded value. The output of the comparator is input to the round-robin priority bit of the agent's arbitration number. Note that this logic replaces the logic required to implement either of the assured access protocols described in Section 2.2.

[2] Note that if the efficiency of the binary patterned arbitration lines is required, the identity of the winning agent could be broadcast on an extra set of k lines that would have to be provided for this purpose.

The second implementation of the round-robin protocol also requires the round-robin priority line, but uses it in a different way. To avoid confusion, we re-name the round-robin priority line the *low-request* line. Any agent that wants control of the bus asserts the shared bus request line. In addition, an agent requesting control of the bus asserts the low-request line if its identity is lower than the recorded identity of the most recent winner of an arbitration. An arbitration is only started if the bus request line is asserted. If low-request is also asserted at the start of an arbitration, only agents with identities lower than the winner of the previous arbitration compete in the arbitration. The agent with the highest identity among this group of competitors becomes the next bus master, and releases the shared request line at the start of its bus tenure. The logic required for the second implementation is similar to the logic for the first implementation. Each agent requires a register to record the winning identity at the end of each arbitration, and a comparator to determine if it's statically assigned identity is smaller than the recorded identity.

The third implementation of the round-robin protocol is somewhat less efficient than the first two implementations, but does not require the extra line on the bus. As in the first two implementations, an agent asserts the shared bus request line whenever it wants control of the bus, and each agent records the identity of the winning agent at the end of every arbitration. An arbitration is only started if the shared bus request line is asserted. Only agents with assigned identities lower than the recorded identity of the previous winner compete in an arbitration. Since no agent has an assigned identity of zero, a winning identity of zero indicates that no agent participated in the arbitration. In this case, the value $N+1$ is recorded as the winning value and a new arbitration is started immediately. Note that no agents will be inhibited in the second arbitration.

The above protocols implement non-preemptive round-robin scheduling, each with approximately the same complexity. The round-robin scan is conceptually implemented in two parts. The first part scans all agents with identities lower than the previous winner, and the second part scans all agents with higher identities. Detailed timing considerations, currently under study, will reveal which of the first two implementations is best. In any case, the implementation is expected to be as simple and efficient as existing assured access protocols.

Priority scheduling of urgent requests is easily integrated with round-robin scheduling of non-priority requests in any of the implementations of the round-robin protocol. The first implementation has the further advantage that round-robin scheduling can be easily implemented within the priority class, if desired. Mechanisms for round-robin scheduling of priority requests are more complex in the other two implementations. The integration of priority service in the first implementation of the round-robin protocol works as follows. The round-robin priority bit is treated

as the second most significant bit of the arbitration identity, a new most significant bit is used for true priority requests, and the remaining bits carry the agent's static identity. Agents may ignore the round-robin protocol for priority requests by always setting the round-robin priority bit to "1" for these requests. Alternatively, agents can follow the protocol to implement round-robin scheduling within the priority class.

3.2. The First-Come First-Serve Protocol

The key idea in our first-come first-serve (FCFS) algorithm is that each agent's identity is the concatenation of two parts. The first and least significant part is the statically assigned arbitration number, as in the standard parallel contention arbiter. The second and more significant part is a counter that indicates how long a request has been waiting relative to other requests. The counter is set to "0" when the agent has a new request for the bus, and is incremented upon some predefined global events that occur while the agent waits for bus ownership. The idea is similar to the idea behind proposals for rotating assigned arbitration numbers in the parallel contention arbiter, but is more robust since part of each identity is statically assigned.

An agent wanting control of the bus asserts the shared bus request line immediately, and competes in the next arbitration using its composite identity. The trick is to use the counter as the most significant bits of the full identity. The maximum finding algorithm will select the agent with the highest counter. This algorithm gives priority to the agents that have waited longer, which is exactly the goal in the FCFS policy.

The waiting time counters cannot implement the global FCFS order of the requests in the bus queue perfectly, since there will always be some chance that two requests will arrive at different instants, but within the same interval between two events that cause the counters to be incremented. In general, there is a trade-off between how accurately the counters implement the FCFS ordering, and the simplicity of the hardware needed to implement the counting mechanism. Below we suggest two alternative strategies for incrementing the counters which have different characteristics in each of these two dimensions. We first discuss the strategies assuming priority requests are not supported in the implementation, and then discuss how the implementations change to integrate priority requests.

The simpler, but less accurate strategy for incrementing the counter, is to increment the counter each time the agent loses in an arbitration. In this case, two agents that generate requests during the the same interval between two successive arbitrations will have the same value of the waiting time counter, and will be served in order of their

statically assigned identities. Otherwise, agents will be served in FCFS order. If each agent can have a maximum of one outstanding request, then N is the maximum number of requests that can received service while the agent waits for bus ownership. Thus, the logic needed to implement this strategy is a modulo-N counter that is incremented by the arbitration result: "lose", and reset by the arbitration result: "win".

The second strategy for incrementing the counter requires an extra line on the bus. Call this line *a_incr*. An agent asserts this line for a very short period of time (e.g., two to four end-to-end bus propagation delays), when it senses a "0" on the *a_incr* line and it has a new request for the bus. A waiting agent increments its counter each time the *a_incr* line is asserted. It is possible to have two agents asserting the *a_incr* line at the same time, in which case these two agents will have the same values for their waiting time counters, and will be served in order order of their statically assigned identities. However, the length of the interval in which this can occur is defined by the time it takes to sense the *a_incr* line, generate the *a_incr* signal, and propagate the signal along the bus. This interval is certainly much smaller than the time between successive arbitrations. Thus, this second strategy implements the FCFS policy more precisely than the first strategy. The second strategy also requires the same number of bits to implement the counter. However, the logic to implement the second strategy is slightly more complicated, due to the need to generate a signal on the *a_incr* line when a new request arrives.

Both implementations clearly implement a first-come first-serve policy, except for the case when more than one agent requests the bus in the same interval between counter updates. These reqeusts are served in a fixed priority order. We expect the inaccuracy and unfairness due to this effect to be negligible for the second method of incrementing the counters. In Section 4, we evaluate the unfairness in the protocol due to this effect for the first method of incrementing the counters.

One nice property of the FCFS algorithm is that it can easily be modified to allow each agent to have more than one active request, yet still serve all requests in FCFS order. If the maximum number of outstanding requests from each agent is r, then only $\lceil \log_2 r \rceil$ more bits are needed for the waiting time counters. For example, if one allows each agent to have up to 8 requests outstanding, first come first serve can still be implemented with only 3 more lines, and a small increase in arbitration delay.

Priority requests can be integrated in the FCFS arbiter by adding a third part to each agent's composite identity. This third part is a most significant bit that is set to "1" for priority requests. Note that in the case of priority requests, agents can use the counting mechanism that implements FCFS scheduling, or some other value can be

used for the second part of the identity. However, priority requests introduce some complexity in the strategies for updating the waiting time counters for non-priority requests in the FCFS arbiter. The complexity arises because an arbitrary number of priority requests can increment the counter for any given non-priority request. One solution is to ignore this problem and increment the counters exactly as specified in either of the strategies above. In this case, the waiting time counters for non-priority requests may overflow and be reset to zero. This may be the right approach if the likelihood of overflow is small. Another option, using the first strategy for incrementing the waiting time counters, is to update the counter only if the most significant bit of the winning identity matches the value of the priority bit for the agent's request. In this case, the value of the waiting time counter for a non-priority request correctly specifies the number of intervals the request has been waiting, but the size of the interval during which two new requests might arrive is larger, since it is measured by the time between two successive arbitrations in which no priority requests compete. The relative merit of this approach compared with the strategy that allows counter overflow is highly dependent on the characteristics of the bus workload, and is beyond the scope of this paper. A third option can be implemented using the second strategy for incrementing the counters. In this case, two extra lines are provided on the bus: *a_incr*, and *a_incr_priority*. An agent's waiting time counter is updated only if the priority of its request matches the priority of the asserted *a_incr* line. In this case, the waiting time counters work as well as in the original scheme, at the expense of some complexity in the logic.

The distributed first-come first-serve algorithm is efficient, although not as efficient as the round robin algorithm. The main difference is due to the larger identities in the FCFS protocol. This implies a larger overhead for the maximum finding algorithm (i.e., for the arbitration), and requires more lines on the bus. However, only $\lceil \log_2 N \rceil$ bits are required for the dynamic portion of the identity. That is, at most we need to double the the size of the identities. It is possible that binary patterned arbitration lines [John83] can be used for the lines that carry the static portion of the agent identities, to make up for the higher overhead.[3] Alternatively, fewer bits in the dynamic portion should implement nearly ideal FCFS scheduling when the bus is not saturated.

[3] The description of how this works is beyond the scope of this paper; however, in this case, the arbitration overhead for the FCFS protocol would be nearly identical to the overhead for the RR protocol, since the delay for the dynamic portion of the identity would be $k/2$ end-to-end bus propagation delays, and the delay for the static portion of the identity would be a single end-to-end bus propagation delay. Recall that in the round-robin arbiter, binary patterned arbitration lines require an extra k lines on the bus, plus overhead for broadcasting the winning identity on the extra lines.

4. Comparative Evaluation of the Proposed Protocols

Given the above proposals for round-robin and first-come first-serve arbitration, we might ask which of the two protocols is the preferred method? The RR protocol is simpler to implement, somewhat more efficient unless binary patterned arbitration lines are used in the FCFS scheme, and is perfectly fair. On the other hand, the FCFS protocol is also simple and efficient, and has the nice theoretical property of minimum waiting time variance [ShAh81].[4] The simpler implementation strategy for the FCFS protocol has some potential for unfairness due to the coarse resolution of the waiting time counters; however, the second implementation strategy we proposed should be nearly perfectly fair.

In this section we report the results of some preliminary simulation studies that explore the practical potential for unfairness in the simpler FCFS implementation, and the practical implications of the lower waiting time variance for the FCFS discipline. We also explore how agents with unequal mean request rates are treated in each of the arbitration protocols, and we examine the performance of the RR protocol for a contrived "worst case" workload scenario. In all experiments, we assume one of the first two implementations of the RR protocol, and the first implementation of the FCFS protocol. Section 4.1 discusses the assumptions made in our simulation experiments; Sections 4.2 - 4.5 present the results.

4.1. Assumptions in Our Simulation Experiments

We outline the assumptions made in our simulation studies in this subsection. We believe that these assumptions represent one reasonable and important set of assumptions that can be used for comparing the protocols.

To begin with, we assume that bus transaction times are deterministic, as would be the case if the transfers were cache block transfers or I/O block transfers in a multiprocessor. We let the bus transaction time define the unit of time in our simulations. We allow inter-request times to vary, and specify only the mean and the coefficient of variation (CV) of the interrequest time distribution.[5] We vary the CV between 0 (i.e. deterministic) and 1 (i.e. the exponential distribution). The exponential distribution yields the highest contention for the bus, and thus the largest value for metrics such as waiting time variance. In the presentation of the results, values of CV are only explicitly

[4] Note that the mean waiting time for bus requests is the same for both the RR and FCFS protocols, as well as for the existing assured access protocols reviewed in Section 2.2. This is true according to a well-established conservation law for work-conserving systems with non-preemptive service disciplines, where the order-of-service is not determined by the service times of the individual requests [Klei76].

[5] The CV is defined as the standard deviation divided by the mean. For $0<CV<1$, we use the k -stage Erlang distribution with the specified mean and $k=CV^{-2}$.

given if they are other than 1.0. We further assume that the bus arbitration overhead is 0.5 units of time. That is, arbitration overhead is half of a bus transaction time. (Generally, this last assumption implies small block transfers, on the order of four bus cycles for 10 MHz buses; however, the primary significance of the assumption is that arbitration is completely overlapped with bus service whenever requests are waiting.) Finally, we assume that arbitration for the next master starts at the beginning of a bus transaction whenever requests are waiting.

Because the input bus interrequest times in our simulations are samples from a specified probability distribution, obtained by using a pseudo-random number generator, it is important that we compute confidence intervals for our output measures. We do so using the method of Batch Means [Lave83]. All of our simulations were run for 10 batches, with 8000 sample outputs in a batch. We have computed 90% confidence intervals, which are generally within 5% of the reported measures.

In most of the studies below, we considered systems with 10, 30, and 64 agents. The *offered load* of an individual agent is defined as its bus transaction time divided by the sum of its bus transaction time and mean interrequest time. This is the fraction of time the agent would use the bus if there were no interference from other agents. The total offered load is the sum of the individual loads over all agents. Total offered load is referred to as "load", and the offered load of a agent is referred to as "load" with a subscript for the static identity of the agent.

We report performance measures as a function of total offered load, which we typically varied from 0.25 to 7.5 for each system. We note that a total offered load of 1.5-2.0 is sufficient to keep the bus 100% utilized, even with variable interrequest times. Thus, the loads we studied above this value represent peak demand for the system, and are useful for looking at the asymptotic behavior of the protocols.

4.2. Fairness of the Simple FCFS Implementation

Our first simulation experiments were performed to investigate the practical potential for unfairness in the simple implementation of the FCFS protocol. We investigate how equitably this implementation allocates bus bandwidth to a set of agents with identical bus interrequest time (and service time) distributions. We have noted earlier that the batching assured access protocols can be highly unfair in this regard. In contrast, the RR protocol is perfectly fair. The second implementation of the FCFS protocol proposed in Section 3.2 can also be used to achieve nearly perfect fairness. The extent to which this is necessary is the subject of our first simulation experiments.

The results for a bus with 10 agents are shown in Table 4.1(a). The first column in the table contains the total offered load. The second column indicates the total system throughput, in bus requests per unit of time. This value is also the bus utilization, since the unit of time is defined by the bus transaction time. The third and fourth columns of the table give the ratio of the throughput for the agent with the highest static identity to the throughput of the agent with the lowest static identity, for the RR and FCFS protocols, respectively. The RR results are included to illustrate the statistical variations that occur in the simulation outputs, since the throughput ratio for this protocol should be precisely 1.0. Tables 4.1(b) and (c) contain the results for 30 and 64 agent systems, respectively.

We observe that the fairness of the simple implementation of the FCFS protocol is generally quite good. The maximum difference in throughput appears to be about 6-8%, which occurs near the point at which the bus becomes saturated. This compares favorably with the results for the assured access protocols at high load [VeLe88]. Results for the first assured access protocol described in Section 2.2 are given in the last column of Table 4.1(b) to illustrate the comparison. If more perfect fairness is desired in the FCFS protocol, the second proposed implementation method should be used.

4.3. Significance of the Waiting Time Variance for RR and FCFS

In our next set of experiments, we investigate the practical significance of the difference in waiting time variance between the RR and FCFS protocols. Sharma and Ahuja have shown that the FCFS protocol has the minimum waiting time variance [ShAh81]. Bain and Ahuja have presented some results that indicate the difference in waiting time variance may be substantial at high load [BaAh81]. However, the practical significance of the difference in waiting time variance is not clear. We have measured the standard deviation of the bus waiting time in several simulation experiments. We also speculate that due to the reduced waiting time variance, FCFS will have better performance if useful execution can be partially overlapped with bus delays. We have studied a hypothetical system with this property. These results are reported below.

In Table 4.2, we present the mean bus waiting time (for both protocols), the waiting time standard deviation for each protocol (σ_X, X=RR,FCFS), and the ratio of the waiting time standard deviations, as a function of offered load for each of three system sizes. We find that the standard deviation in RR is as much as 60% higher for 10 agents, 195% higher for 30 agents, and 350% higher for 64 agents, than in the FCFS arbiter.

Table 4.1: Allocation of Bus Bandwidth Among Agents with Equal Request Rates

(a) 10 Agents

Load	Λ	$\frac{\Lambda_N}{\Lambda_1}$ RR	$\frac{\Lambda_N}{\Lambda_1}$ FCFS
0.25	0.25	0.99 ± 0.04	1.00 ± 0.03
0.50	0.48	0.96 ± 0.03	1.03 ± 0.05
1.00	0.85	1.02 ± 0.04	1.04 ± 0.04
1.50	0.99	0.98 ± 0.02	1.08 ± 0.03
2.00	1.00	1.00 ± 0.02	1.09 ± 0.02
2.50	1.00	1.00 ± 0.01	1.09 ± 0.01
5.00	1.00	1.00 ± 0.00	1.05 ± 0.00
7.52	1.00	1.00 ± 0.00	1.01 ± 0.00

(b) 30 Agents

Load	Λ	$\frac{\Lambda_N}{\Lambda_1}$ RR	$\frac{\Lambda_N}{\Lambda_1}$ FCFS	$\frac{\Lambda_N}{\Lambda_1}$ AAP
0.25	0.25	1.01 ± 0.09	1.00 ± 0.08	0.98 ± 0.09
0.50	0.49	1.00 ± 0.06	0.98 ± 0.07	0.99 ± 0.07
1.00	0.91	1.02 ± 0.07	1.05 ± 0.06	1.07 ± 0.07
1.50	1.00	1.00 ± 0.05	1.06 ± 0.04	1.27 ± 0.04
2.00	1.00	1.00 ± 0.03	1.06 ± 0.05	1.53 ± 0.05
2.50	1.00	0.99 ± 0.02	1.03 ± 0.03	1.68 ± 0.04
5.00	1.00	1.00 ± 0.01	1.04 ± 0.01	1.96 ± 0.02
7.50	1.00	1.00 ± 0.00	1.03 ± 0.01	1.99 ± 0.02

(c) 64 Agents

Load	Λ	$\frac{\Lambda_N}{\Lambda_1}$ RR	$\frac{\Lambda_N}{\Lambda_1}$ FCFS
0.25	0.25	1.00 ± 0.14	1.05 ± 0.15
0.50	0.50	1.05 ± 0.11	1.01 ± 0.08
1.00	0.93	0.97 ± 0.09	1.07 ± 0.12
1.50	1.00	0.99 ± 0.04	1.01 ± 0.06
2.00	1.00	0.99 ± 0.06	1.00 ± 0.05
2.50	1.00	0.98 ± 0.03	1.02 ± 0.04
5.00	1.00	1.00 ± 0.01	1.01 ± 0.01
7.50	1.00	1.00 ± 0.00	1.01 ± 0.01

Table 4.2: Standard Deviation of the Waiting Time for FCFS and RR

(a) 10 Agents

Load	Λ	W	σ_W FCFS	σ_W RR	$\frac{\sigma_W RR}{\sigma_W FCFS}$
0.25	0.25	1.64	0.33	0.33	1.01 ± 0.03
0.50	0.48	1.85	0.56	0.58	1.02 ± 0.03
1.00	0.85	2.77	1.18	1.30	1.10 ± 0.03
1.50	0.99	4.47	1.54	1.94	1.26 ± 0.02
2.00	1.00	6.00	1.43	2.09	1.47 ± 0.02
2.50	1.00	7.00	1.25	2.02	1.61 ± 0.01
5.00	1.00	9.00	0.71	0.99	1.40 ± 0.02
7.52	1.00	9.67	0.32	0.33	1.03 ± 0.01

(b) 30 Agents

Load	Λ	W	σ_W FCFS	σ_W RR	$\frac{\sigma_W RR}{\sigma_W FCFS}$
0.25	0.25	1.66	0.36	0.36	1.01 ± 0.04
0.50	0.49	1.94	0.68	0.71	1.04 ± 0.04
1.00	0.91	4.11	2.18	2.63	1.21 ± 0.07
1.50	1.00	11.02	3.06	5.39	1.76 ± 0.04
2.00	1.00	16.00	2.67	6.42	2.41 ± 0.00
2.50	1.00	19.00	2.35	6.62	2.81 ± 0.04
5.00	1.00	25.00	1.60	4.71	2.94 ± 0.03
7.50	1.00	27.00	1.25	2.99	2.38 ± 0.03

(c) 64 Agents

Load	Λ	W	σ_W FCFS	σ_W RR	$\frac{\sigma_W RR}{\sigma_W FCFS}$
0.25	0.25	1.66	0.37	0.37	1.01 ± 0.04
0.50	0.50	1.96	0.72	0.76	1.05 ± 0.05
1.00	0.93	5.52	3.23	4.06	1.26 ± 0.10
1.50	1.00	22.32	4.54	10.99	2.42 ± 0.07
2.00	1.00	32.99	3.93	13.78	3.51 ± 0.07
2.50	1.00	39.39	3.51	14.45	4.12 ± 0.02
5.00	1.00	52.20	2.44	10.89	4.46 ± 0.00
7.50	1.00	56.46	1.95	7.46	3.82 ± 0.08

To illustrate the difference in waiting time variance further, we have plotted the cumulative distribution function (CDF) of the waiting time for both protocols in Figure 4.1, for a "typical" set of parameter values. Note how sharply the CDF rises near the mean waiting time for the FCFS protocol.

One advantage of a smaller waiting time variance may occur when useful execution can be overlapped with bus waiting times. In this case, for certain values of the amount of time the agent can execute productively while

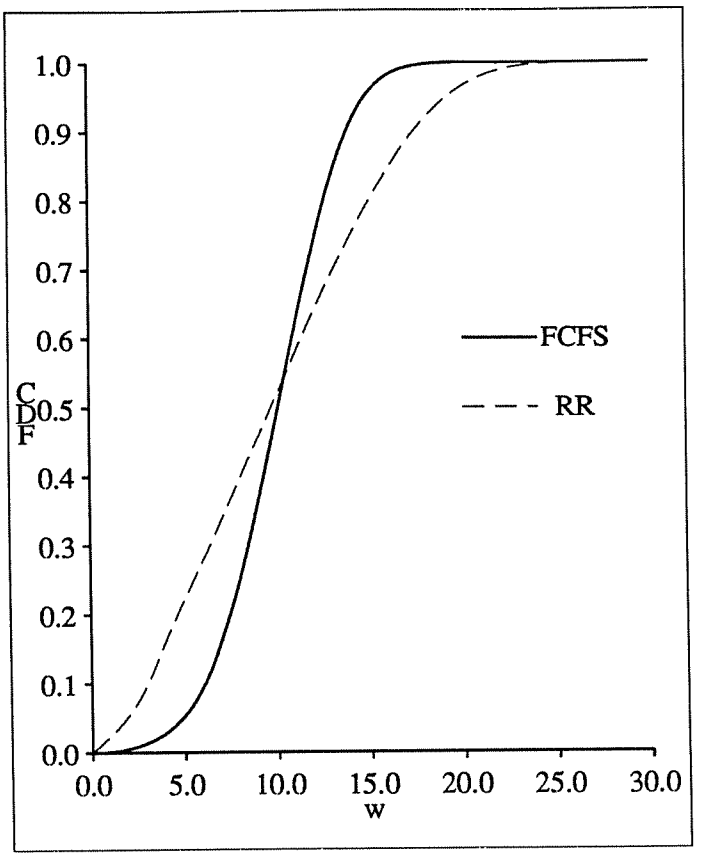

Figure 4.1: CDF of the Bus Waiting Time for RR and FCFS (30 Agents; Load = 1.5)

waiting for bus ownership, the agents in the FCFS system may be able to precisely overlap productive execution with waiting time more often. Note that the "optimal" overlap for differentiating between the protocols will have some value approximately equal to the mean waiting time. This value must be selected carefully in order for the differences between the two protocols to be observable. In this sense, the experiments which yield observable differences between the protocols are very contrived.

We have investigated one hypothetical case that maximizes the advantages of the FCFS protocol. In this experiment, we set the amount of useful execution that *may be* overlapped with bus waiting to be fixed and equal to the minimum integer value at which the CDF for RR is less than the CDF for FCFS. The actual amount of useful execution that *is* overlapped with each request waiting time, is the minimum of this value and the request waiting time. Inter-request times remain unchanged in this experiment. This implies that the overlapped execution is some "extra work" that the processor performs. Another way to look at this assumption is that the extra work, when

added to the inter-request time, approximately represents a longer inter-request time with data pre-fetching. The model is not realistic, since some of the useful work is ignored (when bus waiting times are less than the overlap value.) More realistic assumptions lead to much smaller differences between the protocols.

Table 4.3 contains the data for the overlap experiments. For each offered load, we give the total mean waiting time including the overlapped execution, the mean bus waiting time after the overlapped execution is subtracted for the RR and FCFS systems, a measure of the "productivity" of the agents in the system for both RR and FCFS, and the value that was used for the execution overlap. The agent productivity is defined to be the mean time it spends executing productively between bus requests, divided by the mean time between bus requests. We note that the confidence intervals for the productivity measures are so large that they are no longer useful; thus they are omitted from the table. However, it appears that under the assumptions stated above, productivity in the FCFS system is somewhat higher than in the RR system.

We conclude that further study of this aspect of the two protocols, based on data for real systems, is warranted. However, we emphasize that minor changes in the assumptions, such as specifying a variable amount of overlap with the same mean, result in productivity measures which are nearly equal in the two systems. Furthermore, the waiting time standard deviations decrease, and become closer in value, as the CV of the interrequest times in reduced. Our results lead us to predict that significant differences in the protocols will not be observable.

4.4. Allocation of Bus Bandwidth Among Agents with Unequal Loads

A third difference between the protocols that we have explored concerns the allocation of bus bandwidth to agents with unequal request rates. At low bus utilizations, (i.e. when there is plenty of bandwidth for all agents), both protocols should allocate bus bandwidth in proportion to the agent request rates. However, at high bus utilizations, the RR scheduling discipline will tend to even out the allocation of bus bandwidth, since requests are served in a fixed order, regardless of arrival times. On the other hand, the FCFS protocol will potentially continue to allocate bandwidth more in proportion to each agent's request rate, since scheduling is based on request arrival times. The first question we might ask is which of these two properties is more desirable. This question is difficult to answer, and appears to be dependent on system implementation goals. Instead, we report the results of simulation experiments which investigate when and by how much the two schemes differ in this regard.

Table 4.3: Performance Comparison for Execution Overlapped with Bus Waiting Times

(a) 10 Agents

Load	W	$W_{overlap}$		Productivity		Overlap
		RR	FCFS	RR	FCFS	
0.50	1.64	0.01	0.00	1.00	1.00	4.0
1.00	1.85	0.08	0.04	0.96	0.98	5.0
1.50	2.77	0.30	0.14	0.89	0.95	6.0
2.00	4.47	0.45	0.19	0.90	0.96	7.0
2.50	6.00	0.84	0.50	0.86	0.92	7.0
5.00	7.00	0.37	0.29	0.95	0.96	9.0
7.52	9.00	0.68	0.68	0.92	0.92	9.0

(b) 30 Agents

Load	W	$W_{overlap}$		Productivity		Overlap
		RR	FCFS	RR	FCFS	
0.25	1.66	0.00	0.00	1.00	1.00	4.0
0.50	1.94	0.02	0.01	0.99	0.99	4.0
1.00	4.11	0.49	0.31	0.88	0.92	6.0
1.50	11.02	1.82	0.79	0.83	0.93	12.0
2.00	16.00	2.69	1.06	0.83	0.93	16.0
2.50	19.00	2.73	0.94	0.86	0.95	19.0
5.00	25.00	1.78	0.64	0.93	0.97	25.0
7.50	27.00	1.10	0.50	0.96	0.98	27.0

(c) 64 Agents

Load	W	$W_{overlap}$		Productivity		Overlap
		RR	FCFS	RR	FCFS	
0.25	1.66	0.00	0.00	1.00	1.00	4.0
0.50	1.96	0.03	0.02	0.99	0.99	4.0
1.00	5.52	0.67	0.36	0.88	0.94	9.0
1.50	22.32	4.30	1.49	0.81	0.93	23.0
2.00	32.99	5.80	1.57	0.82	0.95	33.0
2.50	39.39	6.21	1.61	0.84	0.96	39.0
5.00	52.20	4.28	1.08	0.92	0.98	52.0
7.50	56.46	3.06	1.04	0.95	0.98	56.0

Our results, given in Tables 4.4(a) and (b), indicate how much difference we can expect from the protocols when the offered load of agent 1 is twice and four times the offered load of agent 2, respectively. All other agents have offered loads equal to the offered load of agent 2 in this experiment. The results are only shown for the 30-agent system, since the results for 10 and 64 agents are very similar. The first column indicates the total offered load, which is slightly higher in each case than in Table 4.1, due to the one higher-rate requester. The second

column shows the bus utilization, which is similarly slightly higher than in the previous tables, and the third column shows the ratio of agent 1's offered load to agent 2's offered load. Columns 4 and 5 give the ratio of the agent 1's throughput to agent 2's throughput.

As expected, both protocols allocate bus bandwidth in proportion to agent request rates at low load. As load increases, throughput ratios tend toward 1.0 in both systems, due to the effects of bus waiting times. However, we do observe some evidence that the FCFS protocol allocates bandwidth more in proportion to agent demands, (e.g. at offered loads of 2.07 and 2.58 for the double-rate experiments). The difference is very slight. Based on these preliminary results, it appears unlikely that this characteristic will be a major deciding factor in selecting one of the two protocols over the other. However, it may be worth considering this dimension as a secondary consideration.

Table 4.4: Allocation of Bus Bandwidth Among Agents with Unequal Request Rates

(a) 30 Agents, One Request Rate Doubled

Load	Λ	$\frac{Load_1}{Load_2}$	$\frac{\Lambda_1}{\Lambda_2}$ RR	$\frac{\Lambda_1}{\Lambda_2}$ FCFS
0.26	0.26	2.00	2.00 ± 0.14	1.95 ± 0.14
0.52	0.51	2.00	1.99 ± 0.09	2.08 ± 0.12
1.03	0.92	2.00	1.85 ± 0.12	1.80 ± 0.08
1.55	1.00	2.00	1.42 ± 0.06	1.47 ± 0.06
2.07	1.00	2.00	1.22 ± 0.03	1.31 ± 0.04
2.58	1.00	2.00	1.10 ± 0.02	1.26 ± 0.03
5.17	1.00	2.00	1.01 ± 0.00	1.10 ± 0.01

(b) 30 Agents, One Quadruple Request Rate

Load	Λ	$\frac{Load_1}{Load_2}$	$\frac{\Lambda_1}{\Lambda_2}$ RR	$\frac{\Lambda_1}{\Lambda_2}$ FCFS
0.28	0.27	4.00	3.99 ± 0.23	3.85 ± 0.29
0.55	0.54	4.00	3.92 ± 0.21	3.83 ± 0.23
1.10	0.95	4.00	3.03 ± 0.14	2.99 ± 0.11
1.65	1.00	4.00	1.70 ± 0.04	1.94 ± 0.06
2.20	1.00	4.00	1.28 ± 0.03	1.59 ± 0.04
2.75	1.00	4.00	1.10 ± 0.02	1.41 ± 0.03
5.50	1.00	4.00	1.01 ± 0.00	1.16 ± 0.01

4.5. Worst-Case Analysis of the Protocols

In our final experiment, we consider a worst-case scenario for the RR protocol. This model is again very contrived, but also serves to point out a potential drawback to the protocol and its practical significance. Here we assume that one requester repeatedly "just misses" its turn in the RR order. This can occur reliably only with deterministic interrequest times and high bus utilizations. For example, it will occur in the following case. Let the interrequest time for the "slow" agent be deterministically n-0.5, where n is the number of agents on the bus, and let the interrequest times of all other agents be deterministically n-3.6. The slow agent will just miss it's turn, and will have a mean waiting time of $n-0.5$, whereas all other agents will have mean waiting times of approximately 3.5 units. In Table 4.5 we present the ratio of the offered loads of the slow and regular agent, and the ratio of the throughputs of these agents measured in the simulation runs, for various values of the coefficient of variation of the interrequest time. We note that the effect of "just missing" a turn at the bus is manifested in reduced throughput for the slow agent. This effect is only observable for CV=0. Just a small amount of variability in the inter-request times is sufficient to render this characteristic unimportant. The intuitive explanation for this is that a small amount of variability in the interrequest times allows a processor to "sneak in" ahead of other processors that have been waiting longer about as often as the agent "just misses" its turn.

We could similarly devise a worst-case model for FCFS, in which all agents generate a request for the bus within the same interval defined by the waiting time counters, each time they make a request. This situation would be equally as contrived, if not more so, than the previous model. Thus, we choose not to pursue this issue further.

5. Conclusions and Further Research

We have presented two new distributed arbitration protocols which can be implemented in the parallel contention arbiter for multiprocessor system buses. The protocols implement the round-robin (RR) and first-come-first-serve (FCFS) scheduling policies. The RR protocol is simple, efficient, and perfectly fair. The FCFS protocol is also simple and efficient and can be nearly perfectly fair. Both protocols are significantly more fair than existing assured access protocols for the parallel contention arbiter. They also have a better combination of efficiency, cost, and fairness characteristics than existing multiprocessor bus arbitration algorithms in general.

Our initial simulation studies show that the practical differences in performance between the RR and FCFS protocols are relatively minor. One potential advantage of the FCFS arbiter is due to its lower variance in bus

Table 4.5: Worst Case Bus Allocation for RR

(a) 10 Agents

CV	$\frac{Load_{slow}}{Load_{other}}$	$\frac{\Lambda_{slow}}{\Lambda_{other}}$ FCFS	$\frac{\Lambda_{slow}}{\Lambda_{other}}$ RR
0.00	0.70	0.75 ± 0.00	0.50 ± 0.00
0.25	0.70	0.77 ± 0.01	0.76 ± 0.01
0.33	0.70	0.76 ± 0.01	0.76 ± 0.01
0.50	0.70	0.76 ± 0.01	0.76 ± 0.01
1.00	0.70	0.76 ± 0.02	0.76 ± 0.02

(b) 30 Agents

t[slow]	t[other]	CV	$\frac{Load_{slow}}{Load_{other}}$
0.00	0.90	0.91 ± 0.00	0.50 ± 0.00
0.25	0.90	0.90 ± 0.02	0.92 ± 0.01
0.33	0.90	0.91 ± 0.02	0.90 ± 0.02
0.50	0.90	0.91 ± 0.04	0.92 ± 0.03
1.00	0.90	0.88 ± 0.06	0.93 ± 0.09

(c) 64 Agents

t[slow]	t[other]	CV	$\frac{Load_{slow}}{Load_{other}}$
0.00	0.95	0.96 ± 0.00	0.50 ± 0.00
0.25	0.95	0.96 ± 0.03	0.95 ± 0.03
0.33	0.95	0.95 ± 0.02	0.96 ± 0.03
0.50	0.95	0.97 ± 0.05	1.00 ± 0.04
1.00	0.95	0.90 ± 0.10	0.93 ± 0.09

waiting times. A case where this could be significant is in systems where bus requests can be made in advance of when the response is actually needed. Our simulation results show a somewhat higher productivity for the FCFS system in a contrived best possible case. Another observable difference between the protocols is the way bus bandwidth is allocated to agents with unequal request rates at high bus loads. The RR protocol tends to even out the allocation of bus bandwidth among the agents in this case, whereas the FCFS protocol gives service more in proportion to the actual request rates. Again, the differences are relatively small, and are negligible at bus loads below saturation. Also, which of these properties is more desirable depends on system implementation goals.

It may be possible to combine both protocols into one hybrid protocol. For example, the round robin protocol might be used only for requests that arrive at the same time, while the FCFS protocol is used for other requests. It

may also be possible to design an adaptive scheme that uses the history of request patterns to optimize its behavior. Further investigation into these possibilities is in progress.

Acknowledgments

The authors wish to acknowledge several fruitful discussions regarding the work reported in this paper with Jim Goodman, Rae McLellan, and Mark Hill. We also wish to thank two anonymous referees for helpful comments on an earlier draft of the paper.

References

[BaAh81] Bain, W. L., and S. R. Ahuja, "Performance Analysis of High-Speed Digital Buses for Multiprocessing Systems," *Proc. 8th Int'l. Symp. on Computer Architecture*, Minneapolis, Minn., May 12-14, 1981, pp. 107-133.

[EgGi87] Eggers, S., and G. Gibson, private communication.

[Gust84] Gustavson, D. B., "Computer Buses - A Tutorial," *IEEE Micro*, Vol. 4, No. 4, August 1984, pp. 7-22.

[Gust86] Gustavson, D. B., "Introduction to the Fastbus," *(Butterworth & Co. publication)*, Vol. 10, No. 2, March 1986, pp. 77-85.

[Hawl87] Hawley, D., "Control Acquisition and Arbitration," *The IEEE 896 Futurebus Tutorial*, IEEE 896 Futurebus Tutorial Working Group, February 4, 1987.

[IMSC86] IEEE Microprocessor Standards Committee P1296 Working Group, "High Performance 32-bit Bus Standard P1296," Unapproved Draft D2.0, June 20, 1986.

[John83] Johnson, M., Jr., "Synchronous Bus Arbiter", United States Patent #4,375,639, March 1, 1983.

[KlCa86] Kleeman, L., and A. Cantoni, "The Analysis and Performance of Batching Arbiters," *Proc. of Performance '86 and ACM SIGMETRICS 1986 Joint Conf. on Computer Performance Modeling, Measurement and Evaluation*, Raleigh, NC, May 27-30, 1986, pp. 35-43.

[Klei76] Kleinrock, L., *Queueing Systems, Volume II: Computer Applications*, Wiley, 1976.

[Lave83] Lavenberg, S. S., *Computer Performance Modeling Handbook*, Academic Press, 1983.

[PWGr86] P1196 Working Group of the Microprocessor Standards Committee, *NuBus - a Simple 32-Bit Backplane Bus, P1196 Specification", Draft 2.0, IEEE, Inc., 1986.*

[ShAh81] Sharma, D. K., and S. R. Ahuja, "A First-Come-First-Serve Bus Allocation Scheme Using Ticket Assignments," *The Bell System Technical Journal*, Vol. 60, No. 7, September 1981, pp. 1257-1269.

[Taub84] Taub, D. M., "Arbitrations and Control Acquisition in the Proposed IEEE 896 Futurebus," *IEEE Micro*, Vol. 4, No. 4, August 1984, pp. 28-41.

[Taub87] Taub, D. M., "Improved Control Acquisition Scheme for the IEEE 896 Futurebus," *IEEE Micro*, Vol. 7, No. 3, June 1984, pp. 52-62.

[UKPS66] UK Patent Specification No. 1099575, Jan. 17, 1966, (no inventor specified).

[VeLe88] Vernon, M. K., and S. T. Leutenegger, "Performance and Fairness of Multiprocessor Bus Arbitration Protocols", submitted for publication.