Approximate Mean Value Analysis of Deflection-routed Shuffle-loop Networks

David B. Wagner

Department of Computer Science
Campus Box #430
University of Colorado, Boulder 80309–0430

CU-CS-608-92 August 1992

University of Colorado at Boulder

Technical Report CU-CS-608-92
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

Department of Computer Science
Campus Box #430
University of Colorado, Boulder 80309–0430

Approximate Mean Value Analysis of Deflection-routed Shuffle-loop Networks

David B. Wagner

Department of Computer Science
Campus Box #430
University of Colorado, Boulder 80309–0430

August 1992

Abstract

This paper presents an Approximate Mean Value Analysis model of deflection routing in Shuffle-Loop Networks. In contrast to most previous work on deflection routing, the model makes no assumptions about traffic patterns, nor does it assume that messages that cannot be admitted to the network are lost. The technique allows the network to be modeled in its entirety: all processors, switches, and memory modules, and their steady-state interactions, are modeled explicitly. The results of the model are found to be in close agreement with the results of simulation experiments.

1 Introduction

Advances in opto-electronic computing technology have focussed a great deal of attention on the the design and analysis of opto-electronic processor interconnection networks. Opto-electronic interconnection networks promise to alleviate memory latency, allowing multiprocessor designs to scale up to very large numbers of processors.

However, opto-electronic networks have some unusual characteristics. Foremost among these is that it is infeasible to buffer messages in the optical domain. An alternative to message buffering is *deflection routing* [3, 9, 14, 18, 25]. In a deflection routed network, switches attempt to route each message along the shortest route to its destination. However, if that route is unavailable, the message is temporarily *misrouted*. Furthermore, messages trying to enter the network must defer to messages that are already in the network. These characteristics have led to this technique also being referred to as "hot potato routing."

Because deflection routing eliminates the need for switch buffering, investigation of its performance is currently receiving widespread attention [4, 6, 7, 8, 11, 12, 19, 21, 26, 30]. The typical strategy of these analyses is to make uniformity assumptions about the network that allow it to be decomposed, reducing the problem from studying the entire network to studying only a single switch. Another assumption that is commonly made is that messages unable to enter the network when they are generated are dropped (presumably to be regenerated at some later time by some higher-level protocol).

In this paper we analyze deflection-routed multistage interconnection networks without resorting to any of the assumptions just mentioned. Our methodology is to apply Approximate Mean Value Analysis (AMVA) [20, 22] with the appropriate residence time equation heuristics. To the best of our knowledge, this technique has not been applied to this problem until now.

We chose to use AMVA for several reasons:

- It allows the network to be modeled in its entirety: all processors, switches, and memory modules, and their steady-state interactions, are modeled explicitly.
- It allows the specification of non-uniform traffic patterns.
- It has been applied with great success to model buffered interconnection networks [28, 29].
- The approach lends itself well to heuristic approximations, and appears to be very robust in the presence of such approximations [1, 16, 27].

Because of these factors, the model combines a high degree of realism with a high degree of accuracy.

The remainder of this paper is organized as follows. Section 2 describes our architectural assumptions, the generic AMVA technique, and the specialization of this technique to our model. This section includes detailed descriptions of all model equations. In Section 3 we present predictions of the model, which are validated via simulation. We also compare the model to a simpler model in which deflections are not taken into account. Finally, Section 4 presents our conclusions and a discussion of future directions, of which there are many.

2 The Model

2.1 Architectural Assumptions

The architectures we are considering are clocked, packet-switched multistage interconnection networks with *no* ability to buffer in-transit messages. This is characteristic of opto-electronic networks. Routing of messages is accomplished by *deflection:* if more than one packet entering a switch on the same clock cycle wishes to exit the switch on the same output port, only one is routed to that port and the others are misrouted (deflected) to other, possibly less-preferred output ports. Topology is arbitrary except for a single constraint: since messages are not guaranteed to be routed along their optimal path, the network must contain a path from every switch to every other switch. Therefore, there is no concept of separate "forward" and "return" networks, as described in [29].

Examples of such network topologies are meshes, hypercubes, and recirculating shuffle-exchange networks. The particular network that we will analyze in this paper is the Shufflenet [10], which is simply any recirculating shuffle-exchange network in which the number of network stages is equal to the logarithm of the number of switches in each stage (where the base of the logarithm is the fanin/fanout of the switches). Extension to other topologies is straightforward.

We assume that each switch node contains a processor and a memory module. The (electronic) buffers associated with each of these elements are infinite in size. Requests from a processor to its associated memory module do not use the network. Memory access patterns can be arbitrary.

Memory requests/replies are assumed to fit into a single packet, which is not unrealistic if wavelength division multiplexing is employed. We assume that there is some fixed limit on the number of requests that any particular processor may have outstanding. This is arguably more

realistic than having processors issue requests at some constant rate from a potentially inexhaustible supply. Furthermore, it maps nicely onto multi-stream processor architectures.

Processor service times (the time between generation of consecutive memory requests by any given processor) are geometrically distributed. Memory service times are assumed to be deterministic.

In a single clock cycle, a switch can route a different message to every one of its output ports. These messages may come from input ports, the processor, or the memory module. However, traffic from input ports, which is in the optical domain, always has priority over traffic generated locally, which can be buffered in the electronic domain. (Messages from the memory module and the PE have equal priority.) Locally generated messages always take a free output port if there is one, rather than waiting for an optimal port. Locally generated messages that are unable to enter the network immediately wait in a special FIFO buffer called the *injection queue*.

2.2 Methodology

Inspired by [29], we model the architecture as a multiclass closed queueing network.

Each processor, memory module, switch output port, and injection queue is represented as a queueing service center. Each request is represented as a customer. The requests generated by the different processors are assigned to different customers classes (i.e., processor p generates customers of class p). The topology of the network is represented by the visit counts of the queueing network model. The queueing network model is solved using Approximate Mean Value Analysis (AMVA).

The significant characteristic of this problem that sets it apart from previous applications of AMVA is the deflection routing. From a modeling standpoint, deflection routing implies that

1. There is no queueing at the switch output ports.

2. The visit counts to the switch output ports are not known in advance.

At first glance, these conditions seem to imply that it would be impossible to apply AMVA to this problem, since, first, AMVA is all about queueing, and second, visit counts are an input to the AMVA algorithm. However, although there is no queueing of messages *in the network*, there is a lot of interesting queueing behavior at the memory modules and the injection queues that can be adequately captured only with a queueing model. The technique works well despite these obstacles, and is excellent evidence of the robustness of AMVA.

Our strategy for dealing with the first problem is to use a switch response time equation $R = 1$. Although this seems absurdly obvious, there are some very non-obvious ramifications of this that we will discuss in Section 2.8.

Our strategy for dealing with the visit counts is as follows:

1. Calculate visit counts assuming that no deflections occur. In this case, visit counts are functions of the static branching probabilities (which are derived from the network topology and the access distribution), and the calculation is fairly straightforward.

2. Solve the system using the AMVA equations given in the next section.

3. Calculate dynamic branching probabilities as functions not only of the static branching probabilities, but also of the utilizations of the switch output ports obtained from the previous step.

4. Calculate new visit counts based on the dynamic branching probabilities.

5. Repeat steps 2–4 until the per-request response times converge.

2.3 Model inputs

Our model parameterization is very similar to that of Willick and Eager's model [29].

There are two classes of inputs to the model: the hardware configuration and the workload characterization. The hardware configuration is given by:

F — the fanin/fanout of the switch modules. Although it would be possible to allow fanin to differ from fanout, it complicates the notation and will not be considered here.

S — the number of stages in the network.

N — the number of processor/memory/switch modules per network stage. In contrast to the notation used in [29], the total number of processors in our model is $N * S$, rather than N.

Note that for the Shufflenet[10], $S = \log_F N$, but our technique imposes no restrictions on N or S except that N must be power of F.

The network so described contains $npe = N * S$ processing elements, npe memory modules, npe switches, and $nsp = F * npe$ switch output ports. Each type of queueing center is numbered consecutively from 0. Switch module s contains processor s, memory s, injection queue s, and output (input) ports $Fs \ldots Fs + F - 1$. The output (input) ports within each switch may also be referred to using switch-relative numbering $(0 \ldots F - 1)$ when it is clear to which switch we are referring.

The workload characterization is given by:

NC — the maximum number of requests a processor may have outstanding before it must block to await a reply.

S_p — the average processor interrequest time when not blocked, in clock cycles. Interrequest times are assumed to be geometrically distributed.

S_m — the memory service time, assumed to be deterministic.

P_{cm} — the probability that a request generated by processor c will have memory module m as its destination.

Although the AMVA technique allows NC, S_p, and S_m to depend on the identity of the processor that originates the request (and in the case of S_m upon the identity of the destination memory module as well), we will assume that these quantities are independent of the originating processor (or destination memory module) for the sake of clarity of notation.

2.4 Model outputs

The following quantities are outputs of the model. Note that in all cases the subscript x refers to an arbitrary queueing center, which may be a processor, a switch port, a memory module, or an injection queue. Since our numbering scheme allows different types of queueing centers to have the

same index, we will rely on context to make clear which type of queueing center is being discussed, rather than create additional notation to distinguish each of these cases. We do, however, adopt the following subscript conventions: s: switch; p: processor; m: memory module; i: input port; o: output port; q: injection queue.

R_{cx} — the average residence time of a class c customer at center x.

R_c — the average total response time of a class c request. $R_c = \sum_x V_{cx} R_{cx}$, where the sum is taken over all switch outputs, memory modules, and injection queues (but *not* processor c).

X_{cx} — the throughput of class c customers at center x.

X_c — the system throughput of class c customers.

Q_{cx} — the average number of class c customers at center x (including both queued and in-service customers).

U_{cx} — the average utilization of center x by class c customers.

In addition to these outputs, there are many intermediate results calculated by the algorithm that are important enough to our discussion to merit their own notation:

V_{cx} — the visit count of class c customers at center x, that is, the average number of visits made by a class c customer to center x.[1] Note that for processors, $V_{cx} = 1$ if $c = x$, and 0 otherwise. For memory modules, $V_{cx} = P_{cx}$.

V_{cx}^i — the portion of V_{cx} that is attributable to customers arriving on switch(x)'s input port $i \in [0..F-1]$.[2] In this context x may be a processor, switch output port, or memory module.

V_{cx}^* — the portion of V_{cx} that is attributable to customers being sourced at switch(x). If x is a memory module, this means customers coming from the corresponding processor, and vice-versa.

$b_{csio}(d)$ — the probability that a class c customer arriving on switch s's input port $i \in [0..F-1]$ will exit switch s on output port $o \in [0..F-1]$, as a function of the customer's destination d. (b is intended to be a mnemonic for "branch probability.") Note that this probability is not a static property of the network; it depends on the probability of being deflected away from the customer's preferred output port, which is an output of the model.

$b_{cs*o}(d)$ — Similar to above, except that it refers to customers originated at switch s (by either processor s or memory module s).

[1] Note that the interpretation given to V_{cx} by [29], which is the probability that a class c customer visits center x, is not correct in this context, since if the service center is a switch output port, deflection routing implies that a given customer may visit that center more than once.

[2] For readers familiar with the notation used in [29], note that the quantity that they call p_{ikj}, which is the probability that a class i customer at switch output port j arrived on switch input port k, can be calculated quite easily as V_{ij}^k / V_{ij}.

A brief explanation of the branch probabilities b is in order at this point. b is clearly a function of the customer's current location s and eventual destination d, since these will affect the customer's *desired* route. What is not so clear is why b is a function of the input port i, or on the fact that the message originated at this switch module. The reason for the former is that the probability of deflection depends on the traffic coming in on the *other* input port(s) to switch s. The reason for the latter is that we assume that messages being injected into the network at a given switch must defer to all traffic passing through that switch. This is in accordance with real-world implementation considerations on optical switch fabrics.

2.5 Residence time equations

In this section we develop the residence time equations for a class c customer at each type of queueing center (in increasing order of complexity): switch output ports, processors, memory modules, and injection queues.

Before proceeding, we illustrate some of the basic analytical idioms of AMVA, so that they will not have to be re-explained each time they appear.

2.5.1 MVA basics

A simple exact MVA algorithm is shown in Figure 1. The main body of the algorithm consists of three steps:

1. Calculate residence times at each queueing center. The residence time of a customer at center q when there are a total of n customers in the system is the sum of that customer's service time plus the service time of all customers already at center q when this customer arrives, all multiplied by the visit count for this center. By the Mean Value Theorem [13, 24], under certain assumptions the arrival-instant queue length at center q in a system with n customers is equal to the steady-state average queue length in a system with $n-1$ customers.

2. Use Little's Law [17] to calculate system throughput from the residence times.

3. Use Little's Law to calculate new queue lengths from the system throughput and queue residence times.

Since $R_q(n)$ depends upon $Q_q(n-1)$, the exact strategy builds up to the desired system population one customer at a time, as indicated by the outer loop in Figure 1.

An advantage to the exact MVA algorithm is that it yields performance results for all populations up to and including the one of interest. This advantage becomes a disadvantage when multiple customer classes are involved, however, since the number of possible customer populations explodes combinatorially[15]. To overcome this problem, a popular approximation (introduced by Schweitzer [22]) is to assume that steady-state queue lengths are linear functions of system population, so that rather than using $Q_q(n-1)$ to compute $R_q(n)$, we instead use the quantity $\frac{n-1}{n}Q_q(n)$. For multiple customer classes, a further assumption that is made is that the arrival-instant queue lengths of classes other than the class of customer under consideration are unaffected by that customer's absence from the queue. The algorithm (Figure 2) becomes iterative in nature, continuing until the queue lengths converge. Of course, this technique yields results that are only approximate, and only for a single population at a time. Note that the dependence of R, Q, and X on n has been dropped from the notation in Figure 2.

for $n = 1$ to N do
    for each queue q do
        $R_q(n) = V_q S_q(1 + Q_q(n-1))$
    end
    $X(n) = \dfrac{n}{\sum_q R_q(n)}$
    for each queue q do
        $Q_q(n) = X(n) R_q(n)$
    end
end

Figure 1: The exact MVA algorithm

do
    for each queue q do
        for each customer class c do
            $$R_{cq} = V_{cq} S_q \left(1 + \frac{n_c - 1}{n_c} Q_{cq} + \sum_{k \neq c} Q_{kq}\right)$$
    end
    for each customer class c do
        $X_c = \dfrac{n_c}{\sum_q R_{cq}}$
    end
    for each queue q do
        for each customer class c do
            $Q_{cq} = X_c R_{cq}$
        end
    end
until convergence

Figure 2: The multi-class approximate MVA algorithm

The residence time equation now consists of three terms: the customer's own service time, the service time of queued customers of the same class, and the service times of queued customers of *other* classes.

Further extensions to MVA rely almost exclusively on modifications to the residence time equations. For example, if different classes of customers have different service times at center q, the residence time equation for a class c customer becomes

$$R_{cq} = V_{cq}\left(S_{cq} + S_{cq}Q_{cq}\frac{n_c - 1}{n_c} + \sum_{k \neq c} S_{kq}Q_{kq}\right) \tag{1}$$

If service times for certain c, q combinations are not memoryless then Q_{cq} is split into two parts: $(Q_{cq} - U_{cq})$, representing the average number of class c customers that are queued *but not in service* at center q, and U_{cq}, representing the average number of class c customers that are in service at center q. The service time of each of the former customers is still S_{cq}, but the service time of the latter is $\bar{r}_{cq}$, the mean residual, or excess, life of the service time distribution for class c at center q. In the most general case this yields:

$$R_{cq} = V_{cq}\left(S_{cq} + S_{cq}(Q_{cq} - U_{cq})\frac{n_c - 1}{n_c} + \sum_{k \neq c} S_{kq}(Q_{kq} - U_{kq}) + \bar{r}_{cq}U_{cq}\frac{n_c - 1}{n_c} + \sum_{k \neq c} \bar{r}_{kq}U_{kq}\right) \tag{2}$$

Finally, there are two complications introduced by the synchronous nature of the system being modeled. The first is that, since arrivals can occur only at discrete points in time, the mean residual service time of an in-service customer is smaller than it would be if arrivals could occur at continuous points in time. This reduces the residence times relative to a non-synchronous system. Willick derived the discretized mean residual life for certain special cases in his M.S. thesis [28]; we derive a formula for the general case in the appendix.

The second complication is that there is a non-zero probability of *simultaneous* arrivals to a service center. The standard assumptions on continuous models (which are based on Poisson processes) essentially rule out this possibility. Thus, it may be necessary to introduce an extra term into the residence time equations to account for this. This factor tends to increase the residence times relative to a non-synchronous system, although the net result of the combination of these two factors is by no means obvious.

Before leaving this topic, we emphasize that each of these modifications is only approximate, as each one violates one or more assumptions on which the Mean Value Theorem is based. One of the great strengths of mean value analysis is its robustness, by which we mean the accuracy of the results produced in the presence of such approximations.

2.5.2 Switch output port residence times

In a network with buffered output ports, the cost of contention for a port is reflected by the queueing terms in the residence time equation (i.e., all terms other than S_{cq} in Equation (2)). This leads to a per-visit response time that is larger than the service time. In our model, there is no buffering capacity on the output ports, and hence the per-visit response time must be equal to the service time; the residence time of a class c customer on output port o is therefore

$$R_{co} = V_{co} \tag{3}$$

(Note that $S_{co} = 1$ for all c, o.) The analogue to contention in our model is deflection. Analytically, deflection manifests itself as an increase in the visit counts to the switch output ports, which is discussed in Section 2.6, and as an increase in the queue lengths at the injection queues, discussed in Section 2.5.5.

2.5.3 Processor residence times

Residence time equations for processors are the next simplest, since the only class of customer that visits processor p is class p. Since there is queueing of customers at processors (queued customers represent "potential future requests" to be made by this processor), the residence time equation takes on a form that is more typical of AMVA:

$$R_{pp} = S_p + S_p(Q_{pp} - U_{pp}) \left(\frac{NC-1}{NC}\right) + (S_p - 1)U_{pp}\frac{NC-1}{NC} \tag{4}$$

Note the following:

- The visit count to a processor is by definition equal to 1.
- The average remaining service time of an in-service customer, despite the fact that service times have a geometric distribution, is only $S_p - 1$. This is less than expected due to the discrete nature of the arrival instants.[3]

If $S_p = 1$, Equation (4) reduces to

$$\begin{aligned} R_{pp} &= 1 + (Q_{pp} - U_{pp}) \left(\frac{NC-1}{NC}\right) \\ &= 1 + X_{pp}(R_{pp} - 1) \left(\frac{NC-1}{NC}\right) \\ R_{pp} - 1 &= \gamma(R_{pp} - 1) \end{aligned} \tag{5}$$

Note that since γ is strictly less than 1, the only solution to this equation is $R_{pp} = 1$. This seems counter-intuitive, until one realizes that the only way queueing can occur in such a situation is if more than one customer arrives in the same clock cycle. Thus, Equation (4) ought to contain a term to account for the possibility of simultaneous arrivals. This term is

$$\frac{S_p}{2} \sum_i \frac{V_{pp}^i}{V_{pp}} \left[\left(1 - \frac{V_{pp}^i}{V_{pp}}\right) X_{pp} \frac{NC-1}{NC} \right] \tag{6}$$

Here we have used the technique of [29]. $\frac{V_{pp}^i}{V_{pp}}$ is the probability that a class p customer visiting processor p came from source $i \in \{0, \ldots, F-1\} \cup \{*\}$.[4] Likewise, $1 - \frac{V_{pp}^i}{V_{pp}}$ is the probability that

[3] See the appendix for a formal derivation. To gain some insight into this expression, note the following. If $S_p = 1$, then clearly an arriving customer will never be delayed by a customer already in service, since that customer would necessarily be leaving at the end of the same clock cycle in which the arriving customer arrived. At the other extreme, if S_p is very large, then arrivals "appear" to be continuous to the customer in service, and so the effect of the discretization of arrival times on that customer's expected remaining service time is negligible.

[4] V_{pp}^* is the number of visits to processor p by customers returning from memory module p, which is simply P_{pp}.

a class p customer visiting processor p did *not* come from source i. $X_{pp}\frac{NC-1}{NC}$ is the steady-state probability of some *other* class p customer arriving at processor p, as observed by the current class p customer. Thus, the expression inside the square brackets is the conditional expectation of the number of other customers arriving at processor p, given that they did *not* arrive on input i. The sum unconditions this expression with respect to the source (i) of the class p customer under consideration. Finally, the factor of $1/2$ accounts for the random resolution of such conflicts by the switch logic.

The results presented in Section 3 were obtained by including this term in R_{pp}; however, the magnitude of this correction is negligible for realistically large network sizes with uniform memory access patterns. For small NC, it is clearly unlikely for two class p customers to arrive at any switch simultaneously; for large NC, the network becomes very busy, and the fact that each network link can hold at most a single customer implies that there are on average only F customers of each class anywhere in the network. In such a situation, most customers are buffered either at the processors, the memories, or are waiting to enter the network. Which of these scenarios is most likely depends, of course, on the relative service times of the processors and memories.

2.5.4 Memory module residence times

Residence time equations for memory modules are more complicated than those for processors in two respects. First, all customer classes can potentially visit a memory module. The second complication is a consequence of the first: it is no longer acceptable to ignore the possibility of simultaneous arrivals. This leads to the following expression:

$$\begin{aligned} R_{cm} = \; V_{cm} \Bigg(& \; S_m \left(1 + (Q_{cm} - U_{cm}) \frac{NC-1}{NC} + \sum_{k \neq c} (Q_{km} - U_{km}) \right) \\ & + \frac{S_m - 1}{2} \left(U_{cm} \frac{NC-1}{NC} + \sum_{k \neq c} U_{km} \right) \\ & + \frac{S_m}{2} \sum_i \left(\frac{V_{cm}^i}{V_{cm}} \left[\left(1 - \frac{V_{cm}^i}{V_{cm}} \right) X_{cm} \frac{NC-1}{NC} + \sum_{k \neq c} \left(1 - \frac{V_{km}^i}{V_{km}} \right) X_{km} \right] \right) \Bigg) \end{aligned} \quad (7)$$

The terms of this equation have interpretations identical to those in the previous section, with the proviso that we must now include customers of other classes in each term. One additional difference is that, since service times are deterministic, $\bar{r}_{cm} = \frac{S_m - 1}{2}$.

2.5.5 Injection queue residence times

Because of the fact that there is no queueing allowed on switch output ports, customers that wish to enter the network at a switch (whether generated by the processor or the memory module) must defer to all traffic that is passing through that switch. Such customers wait at a special queueing center at that switch called the *injection queue*.

In reality, the injection queue is a pure queueing center. That is, it has no server of its own: the customer at the head of the injection queue waits for a free cycle at one of the output ports.

Because of this unusual characteristic, the injection queue is very difficult to model accurately. (See Sections 2.8 and 3.)

Our strategy for modeling this phenomenon is to pretend that the injection queue has a server, and set the average service time at this server is equal to the average time spent waiting for a free output port, which we can calculate from the utilizations of those output ports. This is similar to the so-called *shadow server* technique for modeling the delays experienced by a low-priority customer at a non-preemptive priority-scheduled FIFO server [15].

For any class c and output port o, let $u_{\bar{c}o}$ be defined at follows:

$$u_{\bar{c}o} = U_{co}\left(1 - \frac{V^*_{co}}{V_{co}}\right)\frac{NC-1}{NC} + \sum_{k \neq c} U_{ko}\left(1 - \frac{V^*_{ko}}{V_{ko}}\right) \tag{8}$$

$u_{\bar{c}o}$ is the utilization of output port o by customers that do *not* originate at (i.e., pass through) switch $\lfloor o/F \rfloor$, as seen by a customer of class c. Assuming that the $u_{\bar{c}o}$ are independent,[5] the probability of a class c customer at the head of injection queue q being unable to leave the queue in any given clock cycle is

$$f_{cq} = \prod_{o=Fq}^{Fq+F-1} u_{\bar{c}o} \tag{9}$$

If we model the clock cycles a class c customer spends at the head of the injection queue as a sequence of Bernoulli trials with probability of failure f_{cq}, then the number of such such clock cycles has a negative binomial distribution. The mean shadow service time for a class c customer is thus

$$S_{cq} = \frac{f_{cq}}{1 - f_{cq}} \tag{10}$$

Note that service times at the injection queue are class dependent, and that the service time distribution is memoryless (under the assumption of discretized arrival instants).

The visit count at the injection queue is

$$V_{cq} = \begin{cases} 1 - P_{cc} & \text{if } c = q \\ P_{cq} & \text{if } c \neq q \end{cases} \tag{11}$$

This follows immediately from the following facts: (a) the only visits to injection queue c by class c are requests from processor c to some memory module *other than* c,[6] (b) the only visits to injection queue $q \neq c$ by class c are replies from memory module q to processor c.

The term representing simultaneous arrivals depends on whether or not $c = q$. If $c = q$ then the customer under consideration originated from processor q. In this case, the only possible simultaneous arrival must come from the local memory module, which implies that the class of said arrival is different from c. On the other hand, if $c \neq q$ then the customer under consideration originated from memory module q. In this case, the only possible simultaneous arrival must come from the local processor, which implies that the class of said arrival is equal to q. Using these observations, we can write down an expression for the expected service time of all simultaneous arrivals:

$$\psi_{cq} = \begin{cases} \sum_{k \neq c} S_{kq} X_{kq} & \text{if } c = q \\ S_{qq} X_{qq} & \text{if } c \neq q \end{cases} \tag{12}$$

[5] N.b.: this assumption may be substantially inaccurate.

[6] This is because traffic between a processor and its local memory does not utilize the network.

This quantity will, of course, be multiplied by 1/2 to reflect the random ordering of these arrivals in the queue.

Given V_{cq}, S_{cq}, ψ_{cq}, and the fact that $\bar{r}_{cq} = S_{cq}$, the residence time equation for a class c customer at injection queue q can now be written as:

$$R_{cq} = V_{cq}\left(S_{cq} + S_{cq}Q_{cq}\frac{NC-1}{NC} + \sum_{k\neq c} S_{kq}Q_{kq} + \frac{1}{2}\psi_{cq}\right) \tag{13}$$

The astute reader might have noticed that since the clock cycle during which the departing customer is using the output port is counted in the output port residence time equation, rather than in the injection queue residence time equation, the lower bound on the shadow service time is 0. Thus, the model theoretically allows arbitrarily many customers to leave the queue on the same clock cycle, which is clearly an architectural impossibility. It would seem that the potential loss of accuracy caused by ignoring this feature is small. The reasoning: unless the network is heavily loaded, the probability of there being more than one customer in the injection queue at any time is very small. On the other hand, if the network is heavily loaded, then the probability that the shadow service time is 0 is very small, and thus the probability of more than one customer being able to leave the queue on the same cycle is "very small." Although this reasoning may seem quite plausible, there are in fact cases in which it breaks down rather badly; we discuss this further in Section 3.

2.6 Visit count computation

After each evaluation of the model using AMVA, new visit counts are calculated based on performance metrics just obtained. This section describes our method for accomplishing this.

The hardest part, conceptually speaking, in the calculation of visit counts is the computation of the branching probabilities $b_{csio}(d)$ and $b_{cs*o}(d)$. Given these probabilities, visit counts can be computed by solving several sparse systems of linear equations whose non-zero coefficients are the branching probabilities.

The branching probabilities depend on the probability of encountering a conflicting customer from another source at each output port; these latter probabilities are in turn estimated from performance metrics of the previous AMVA iteration. However, the branching probabilities also depend on the the topology of the network (i.e., the fanin/fanout of the switches, whether or not there is a concept of a preferred path to a given destination, and whether or not there is more than one such path) and the contention-resolution logic of the switches. For this reason, it is impossible to write down a general expression for the branching probabilities as a function of the conflict probabilities. We illustrate the computations by assuming a shuffle-loop network with a switch fanin/fanout $F = 2$.

We first describe the computation of $b_{cs*o}(d)$, the probability that a class c customer being injected at switch s, bound for switch d, will depart on output port o. Since this customer must defer to all traffic through switch s, the probability of being unable to use output port o on any given clock cycle is $u_{\bar{c}o}$, the total utilization of that output port by customers passing through the switch, as seen by an arriving customer of class c (Equation (8)).

One of the interesting things about the shuffle-loop topology is the fact that if the length of the shortest path from a given source to a given destination is less than $\log_F N$, then that path is unique. A consequence of this is that the message has only one choice among the output ports at

each switch in order to remain on this path; we call such an output port the *preferred output port*. On the other hand, if the shortest path is longer than $\log_F N$, then there are no preferred output ports until the message has traveled far enough to reduce the distance to $\log_F N$. The existence of a preferred output port for a given class at a given switch is a static property of the network.

Thus there are two cases in computing the branching probabilities, depending on whether or not a customer at switch s bound for switch d has a preferred output port or not. Consider the former case, and suppose the preferred output port is p, and call the other output port o. Then

$$\begin{aligned} b_{cs*p}(d) &= \frac{(1-u_{\bar{c}p})}{1-u_{\bar{c}p}u_{\bar{c}o}} \\ b_{cs*o}(d) &= \frac{u_{\bar{c}p}(1-u_{\bar{c}o})}{1-u_{\bar{c}p}u_{\bar{c}o}} \end{aligned} \tag{14}$$

The first expression is simply the conditional probability that the preferred output port is not busy, given that it is not the case that both output ports are busy. The second expression is the conditional probability that the preferred output port *is* busy but the other output port is not, given that they are not both busy.

In case there is no preferred output, the branching probabilities become

$$\begin{aligned} b_{cs*0}(d) &= \frac{(1-u_0(c))\big(u_1(c)+\frac{1}{2}(1-u_1(c))\big)}{1-u_0(c)u_1(c)} \\ b_{cs*1}(d) &= \frac{(1-u_1(c))\big(u_0(c)+\frac{1}{2}(1-u_0(c))\big)}{1-u_0(c)u_1(c)} \end{aligned} \tag{15}$$

(Note that we have labeled the ports simply as 0 and 1 in this case because neither port is preferred.) Each of these expressions reflect that the given port is not busy, and that either (a) the other port is busy, or (b) the other port is not, in which case the customer chooses each port with probability $1/2$.

We next describe the computation of $b_{csio}(d)$, the probability that a customer arriving on input port i will depart on output port o. In this case we need to estimate from the AMVA outputs the probability that a customer from the other input port j wishes to use each of the possible output ports.

A simple (and incorrect) way to estimate this probability is by analogy with the b_{cs*o}: simply compute the utilization of each output port by customers from input port j. The reason that this is incorrect is very subtle: the utilizations do not represent the probability that a customer actually *wanted* to use a given output port, only that the customer *ended up doing so* (in the previous AMVA iteration).[7]

A better estimate is to base the conflict probability for output port o on the throughput of input port j and the probability that a customer from j "wants" to take output port o. The latter probability is difficult to compute, but we can estimate it from $V0_{co}$, the (static) visit counts from an identical network *without* deflections. (The $V0$ are computed at the very beginning of the model evaluation as inputs to the first AMVA iteration, so no extra effort is required.) Then the probability that a class c customer arriving on input port $i \neq j$ "sees" a customer from input port j trying to use output port o, which we will call t_{cjo}, is

$$t_{cjo} = X[c][j]\frac{\sum_p V^j_{cp}}{V_{cj}}\frac{V0^j_{co}}{\sum_p V0^j_{cp}}\frac{NC-1}{NC} + \sum_{k\neq c} X[k][j]\frac{\sum_p V^j_{kp}}{V_{kj}}\frac{V0^j_{ko}}{\sum_p V0^j_{kp}} \tag{16}$$

[7] Note that this was not a concern in the case of b_{cs*o} because it was assumed that a customer being injected into the network cannot affect the routing of customers passing through the switch.

All summations are over all outputs ports p at this switch. The first quotient in each term, based on the visit counts from the previous AMVA iteration, is the probability that a customer arriving on input port j actually passes through the switch (rather than sinking there). The second quotient, based on the static visit counts, is the probability that said customer chooses output port o from the choices available to it.[8]

The t_{cjo} can be used to compute the branching probabilities as follows: for the asymmetric case (preferred output p and other output port o),

$$\begin{aligned} b_{csip}(d) &= 1 - \frac{t_{cjp}}{2} \\ b_{csio}(d) &= \frac{t_{cjp}}{2} \end{aligned} \tag{17}$$

and for the symmetric case (no preferred output port, ports labeled simply 0 and 1),

$$\begin{aligned} b_{csi0}(d) &= t_{cj1} + \tfrac{1}{2}(1 - t_{cj0} - t_{cj1}) \\ b_{csi1}(d) &= t_{cj0} + \tfrac{1}{2}(1 - t_{cj0} - t_{cj1}) \end{aligned} \tag{18}$$

With the dynamic branching probabilities in hand, the visit count for class c customers *that are bound for destination* d at output port o of switch s can be computed from the following $nsp \times nsp$ system of linear equations:

$$V_{c,Fs+o}(d) = \begin{cases} 0 & \text{if } s = d \\ \sum_i b_{csio}(d) V_{ci}(d) + b_{cs*o}(d) P_{cd} & \text{if } s = c \neq d \\ \sum_i b_{csio}(d) V_{ci}(d) + b_{cs*o}(d) P_{cs} & \text{if } s \neq c = d \\ \sum_i b_{csio}(d) V_{ci}(d) & \text{otherwise} \end{cases} \tag{19}$$

In all cases o takes on the values $0, \ldots F-1$, and in the last three cases the summation is over all input ports to switch s.

The first formula corresponds to the case in which the output port is at the destination switch. Since we are restricting our attention to traffic that is bound for this switch, the output port visit counts in this case clearly are 0.

The next three formulas all contain an identical linear combination of the visit counts at the *input* ports of switch s (which are, of course, the same as the visit counts at the output ports of the switches that feed switch s). The coefficient of $V_{ci}(d)$ is just the probability that a message coming in on port i will be routed to output o. In "most" cases this is the entire right-hand side of the equation. However, for two special sub-cases there is an additional constant term.

In the first of these special sub-cases ($s = c \neq d$), the switch under consideration is the source of all class c requests. Therefore, the switch is adding a total of P_{cd} to the visit counts of its output ports, where the allocation of these customers is determined by the branching probabilities of injected messages.

In the second of these special sub-cases ($s \neq c = d$), the destination node is the home node for class c. In this case, the traffic being analyzed consists of replies to processor c. Thus, each switch $s \neq d$ is adding a total of P_{cs} to the visit counts of its output ports.

Once the visit counts as a function of destination d have been computed, the total visit counts for each output port follow simply by summing over d. To obtain the partial visit counts, sum over

[8] Unfortunately, in the statically routed network there may be some ports j that a customer of class k never visits, causing the denominator of this quotient to be 0. In such cases we substitute the quotient $\frac{V0_{co}}{\sum_p V0_{cp}}$.

the individual terms in Equation (19). For example, in the case $s = c \neq d$,

$$\begin{array}{rcl} V^i_{c,Fs+o} & = & \sum_d b_{csio}(d) V_{ci}(d) \\ V^*_{c,Fs+o} & = & \sum_d b_{cs*o}(d) P_{cd} \end{array} \tag{20}$$

The other cases are analogous.

The total visit counts for the memory modules, as well as the partial visit counts from the local processor, never change. To obtain the partial visit counts to memory module m from any input i of switch m,

$$V^i_{cm} = \begin{cases} 0 & \text{if } c = m \\ V_{ci}(m) & \text{if } c \neq m \end{cases} \tag{21}$$

A similar technique can be used to obtain the partial visit counts to the processors (cf. Equation 6). This completes the visit count calculations.

Solving for the visit counts is the most computationally expensive step in the entire model. This is because we must solve a $nsp \times nsp$ linear system for *each* (class, destination) pair. The time complexity of the visit count calculation is thus $O(npe^2 nsp^3) = O((NS)^5 F^3)$. Because of this, a customized Gaussian elimination algorithm was created to exploit the special structure of the sparse coefficient matrix. The complexity of this algorithm is $O((NF)^3)$ rather than $O((NSF)^3)$, reducing the total complexity to $O(N^5 S^2 F^3)$. Nevertheless, this step is obviously still the chief limitation on the size of the network that can be analyzed with this model. Approximations that will result in a reduction of this complexity are the subject of ongoing research; see Section 4.

2.7 Initialization

Initialization of AMVA variables is done only once, before the first AMVA computation. For subsequent outer iterations of the model, the last values obtained from the previous AMVA computation are used as the initial values for the next AMVA computation.

Proper initialization is important since some of the response time equations depend not only on queue lengths but also on throughputs and utilizations. Failure to initialize these variables in a consistent manner was found to lead to negative response times in some instances.

Our framework for a consistent initialization assumes that the memory modules are the bottleneck servers. (For realistic systems, this is probably not too far from the truth.) Thus we initialize memory queue lengths as though all customers were queued at the memories: for each customer class c and memory module m, $Q_{cm} \leftarrow P_{cm} NC$.

A consequence of the memories being bottlenecks is that initial system throughput for class c is limited to $X_c = 1/S_{mm}$. Initial throughputs for memory modules are thus $X_{cm} \leftarrow P_{ij}/S_{mm}$. Since we do not know output port visit counts, we assume that initial output port throughputs are $X_{co} \leftarrow 1/nsp \cdot 1/S_{mm}$.

Similarly, assuming that there is no queueing anywhere except the memory modules implies that response time at any processor p will be approximately S_{pe}. This implies that for any processor p we must initialize $Q_p \leftarrow S_{pe}/S_{mm}$.[9]

Initialization of injection queue measures is not critical, as response time for an injection queue depends mainly on the utilizations of its output ports.

[9] Note that this results in total queue lengths for each class initially adding up to slightly more than NC; this has not been found to be a problem.

2.8 Convergence

Due to the nature of several of the approximations used in the model, it is often the case that the throughput for a customer class will exceed one, which is, of course, impossible. The two main causes of this are:

- the assumption that response time at a switch output port is exactly one, and
- the reduction in time spent waiting for an in-service customer because of the synchronous nature of arrivals.

When a class throughput exceeds one, most or all switch output port throughputs will also exceed one. This is a serious problem, since it may lead to negative shadow service times at the injection queues (refer to Equations (8–10)). Failure to address this problem results in a catastrophic breakdown of the model.

We address this problem in the following way. After calculation of the switch output port throughputs, if any of them exceeds one then we scale down all of them by a factor that is equal to the maximum such throughput. This results in a new set of switch output port throughputs whose maximum is exactly one.[10]

Obviously, in order to reduce throughput we must increase response times somewhere. We choose the injection queues, for two reasons:

- The injection queue is the feature of the architecture that is supposed to provide the buffering capacity that the network is lacking.
- Since injection queue service times are not fixed, but are actually functions of other model variables, we reason that the "lion's share" of the inaccuracy must be in the injection queue response times. In fact, comparison with simulation results confirmed this hypothesis.

Therefore, the technique we propose is to scale up the injection queue response times enough to bring the maximum switch port throughput down to one.

Let $\widehat{X} > 1$ be the maximum switch output port throughput. We wish to obtain new per-class throughputs $X'_c = X_c/\widehat{X}$. Let P_c, M_c, S_c, and I_c be, respectively, the total of all processor, memory module, switch port, and injection queue response times for an arbitrary class c, and let T_c be their sum. Then $X'_c = X_c/\widehat{X}$ implies that

$$\begin{aligned} P_c + M_c + S_c + I'_c &= \widehat{X}(P_c + M_c + S_c + I_c) \\ I'_c &= I_c + (\widehat{X} - 1)(P_c + M_c + S_c + I_c) \\ I'_c &= I_c + (\widehat{X} - 1)T_c \end{aligned}$$

Therefore, for each class c and injection queue q, we multiply the injection queue residence time R_{cq} by the following scale factor:

$$\alpha_{cq} = 1 + \frac{(\widehat{X} - 1)T_c}{\sum_q R_{cq}} \tag{22}$$

Empirical evidence (Section 3) suggests that this works quite well.

[10] A switch output port throughput exactly equal to one does not pose a problem, because of the factor of $\frac{NC-1}{NC}$ in Equation 8.

At this point one might question the utility of the injection queue response time equations (Equations 8–13)! After all, it seems that we must "reverse engineer" the injection queue response times using the method outlined in this section. However, in any model evaluation for which the output port throughputs do not exceed one, this method is inapplicable. In such cases, the injection queue response time equations work quite well.

Despite these efforts (or perhaps *because* of them), in some cases the model will not converge to a unique set of values, but will oscillate back and forth between two sets of closely-spaced values. Typically this happens only in cases in which the switch output ports are almost but not quite saturated, and then only rarely. Such cases will be discussed further in the next section.

3 Results

In this section we present data to validate the model. All results in this section, unless noted otherwise, pertain to a Shufflenet topology with 3 stages of 8 processors each. Every processor references each memory with equal probability.

Unless explicitly stated otherwise, the following conventions should be assumed for each of the plots presented here: the solid curves were generated by the model developed in this paper; the dashed curves were generated by a simulation; and the dotted curves were generated by a model of an identical topology network with infinite output buffers at each switch output port (and hence no deflections).[11] All simulations results are the average of three simulation runs; each run was 36,000 clock cycles in length, with the first 4,000 clock cycles deleted from the performance measures.

Figures 3 and 4 show the average total response time and total network throughput, respectively, for a network with $S_p = 1$ and various values of S_m. Although $S_p = S_m = 1$ is not very realistic, it represents a "stress case" for the model, since it causes very high utilization of the network, and hence deflections play a major role in performance. As the figures show, the model is quite accurate, and is a dramatic improvement over not modeling deflections at all.

These figures also demonstrate that S_m, within the range shown here, has no effect on performance for $NC \geq 4$. This shows that at larger customer populations, the network itself becomes the bottleneck. Note that we did obtain results for larger customer populations (up to 24 as of this writing), but these are not shown because (a) the graphs are almost perfectly linear outside of the range shown, and (b) expanding the graphs would obscure the detailed behavior of the model at the lower customer populations.

Figures 5 shows the average injection queue response time. Agreement with the simulation results is excellent, especially when one considers all of the approximations made in the injection queue response time equation.

Figure 6 shows the average memory module response time. There is some disagreement here between the model and the simulation at $S_m = 4$.

Figure 7 shows the average response time at the processors. We have shown the line $R_{pp} = 1$ for visual reference. Note that processor response time never gets much larger than 1, which corroborates the reasoning of Section 2.5.3. Had we not included the term (6) in the response time equation, the error (relative to the simulation) would have been no greater than about 6%.

[11] The no-deflection model is a straightforward adaptation of the model of [29] to the Shufflenet topology. It was separately validated by another simulation, the results of which are not shown here.

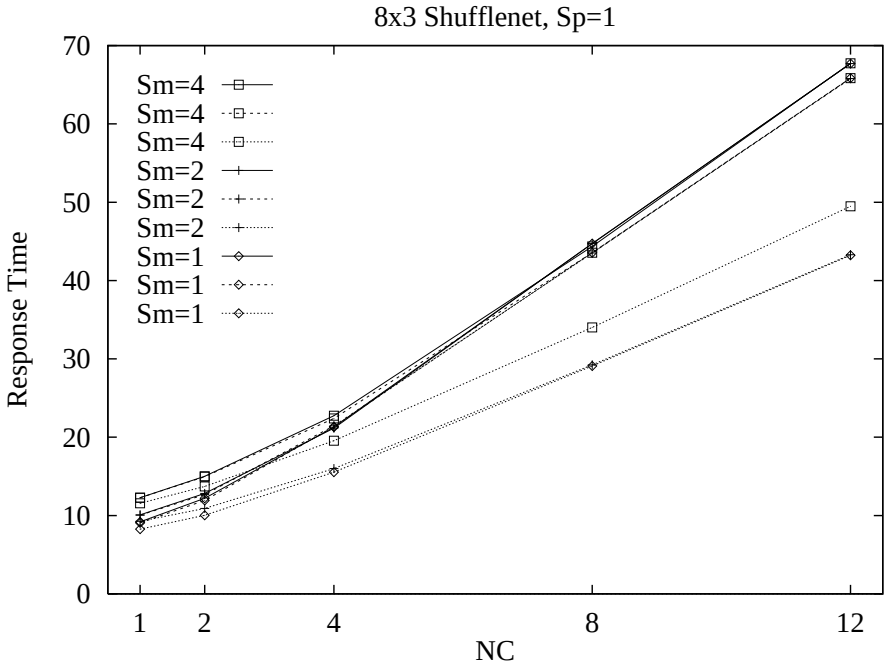

Figure 3: Average per-request total response time, $S_p = 1$.

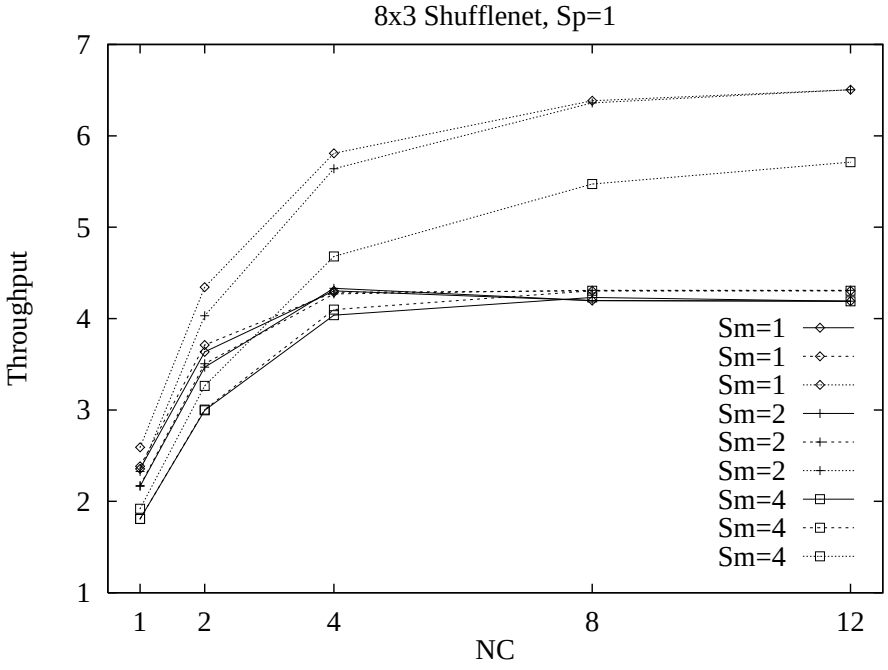

Figure 4: Average total network throughput, $S_p = 1$.

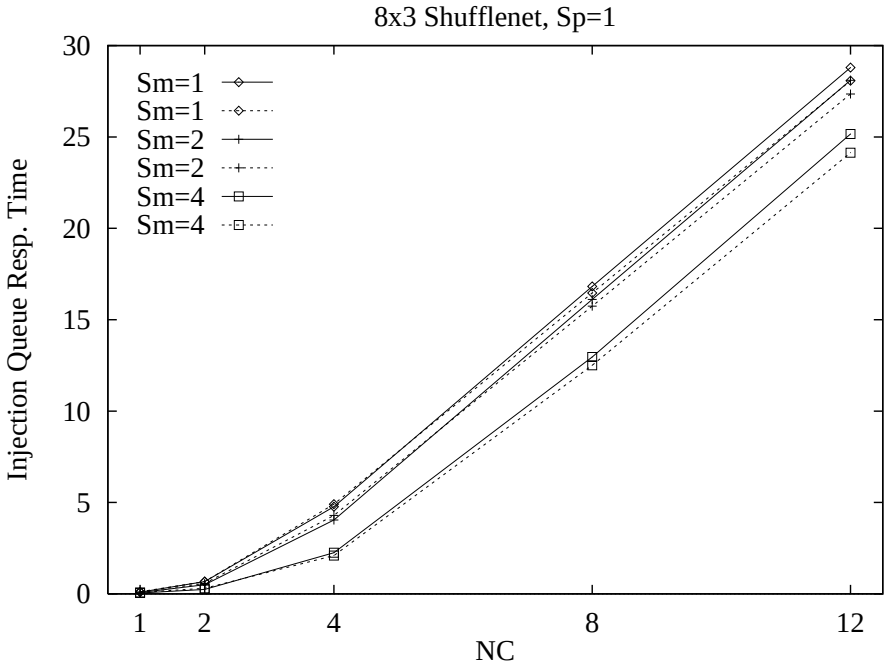

Figure 5: Average injection queue response time, $S_p = 1$.

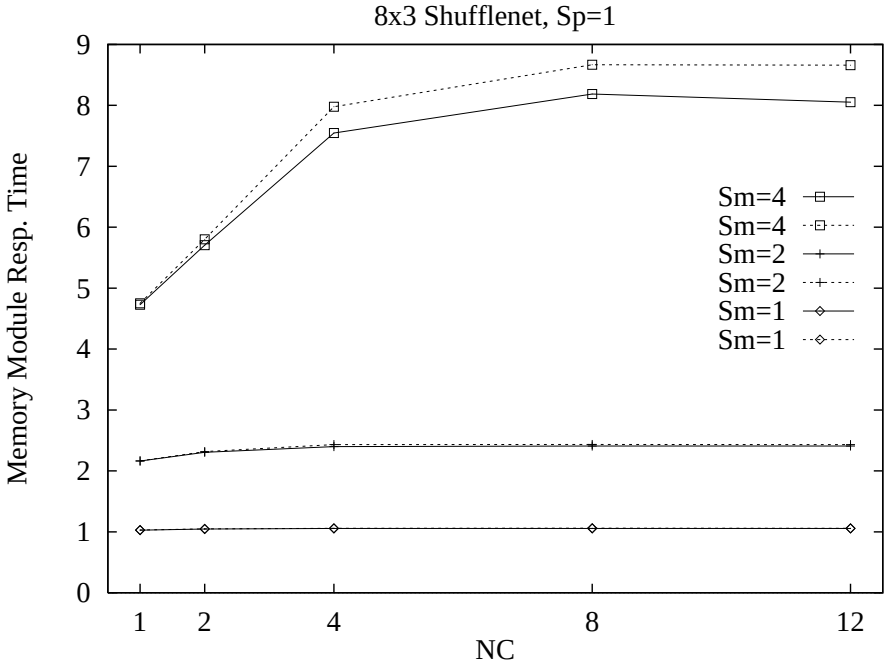

Figure 6: Average memory module response time, $S_p = 1$.

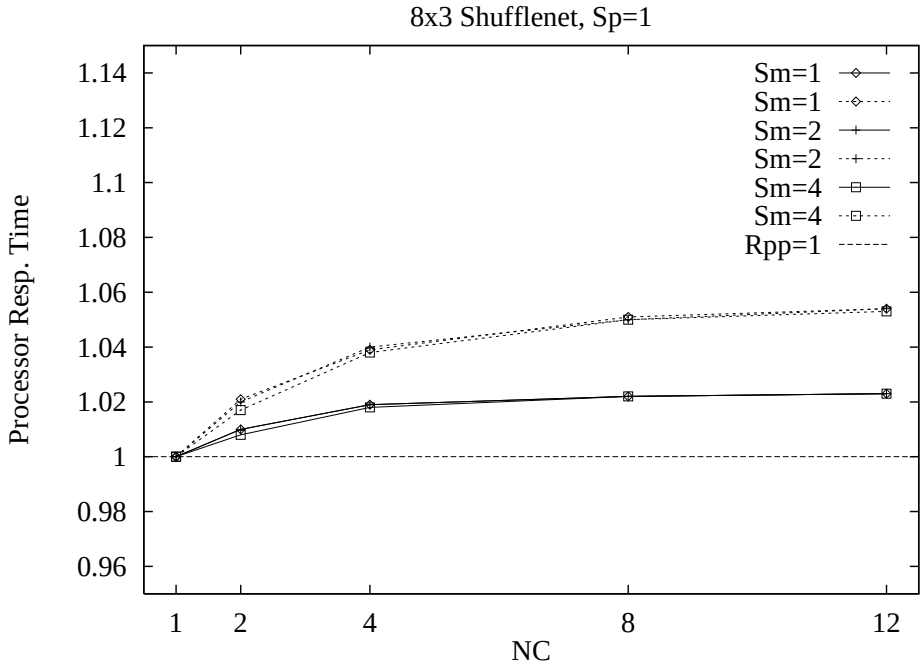

Figure 7: Average processor response time, $S_p = 1$.

Figure 8 shows the average output port utilization. Although this metric is quite challenging to estimate accurately (the model predicts utilizations of exactly 1 at higher populations due to scaling) the agreement is quite good.

Figure 9 shows average total response times for $S_p = 2$ and $S_m = 2, 4, 8$. Note that the curve for $S_m = 8$ has split off from the other two. Examination of Figure 10 immediately shows why: at this point the memory modules become the bottleneck. Figures 11 and 12 corroborate this. Since injection queue times are essentially 0 and output port utilizations are well below 1, the no deflection model works as well as the deflection model in this case.

Figure 13 shows total response times for even larger values of S_p and S_m. As these values increase, the network itself becomes less of a bottleneck, and as a result the no-deflection model comes into better agreement with the deflection model. At $S_p = 10, S_m \geq 10$, which is not shown here, both models are in virtually perfect agreement with the simulation results.

In some rare cases the model does not converge to a unique value, but rather oscillates between two closely-spaced values. This oscillation seems to be centered around the injection queue response times. Figure 14 shows the magnitude of these oscillations for all cases in which they were observed (a total of 3 cases out of 84 parameter settings). What these parameter settings all have in common is that the resulting port utilizations are approaching, but have not reached, their limiting values. This suggests that this phenomenon probably is caused by the throughput scaling technique discussed in Section 2.8.

Another limitation of the model is that it severely underestimates the response time of the injection queue associated with a "hot" memory module. The most probable cause of this discrepancy is that most of the traffic visiting the hot switch is bound for the hot memory module; very

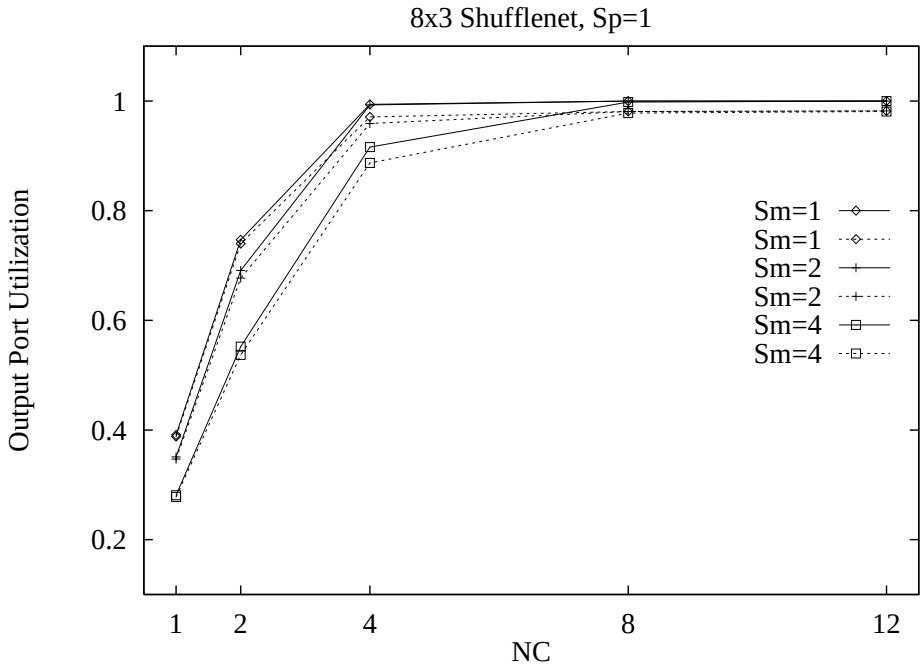

Figure 8: Average output port utilization.

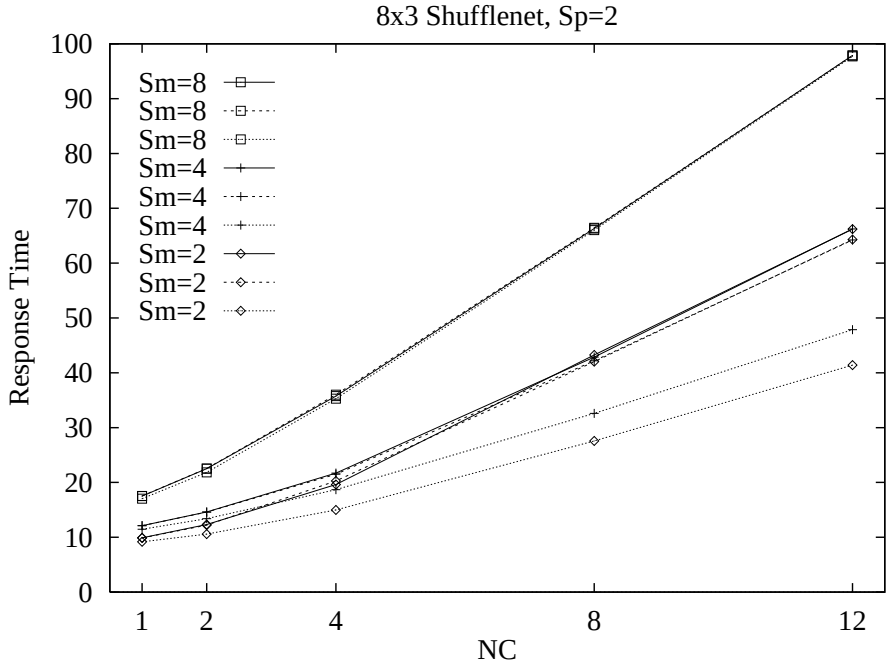

Figure 9: Average per-request total response time, $S_p = 2$.

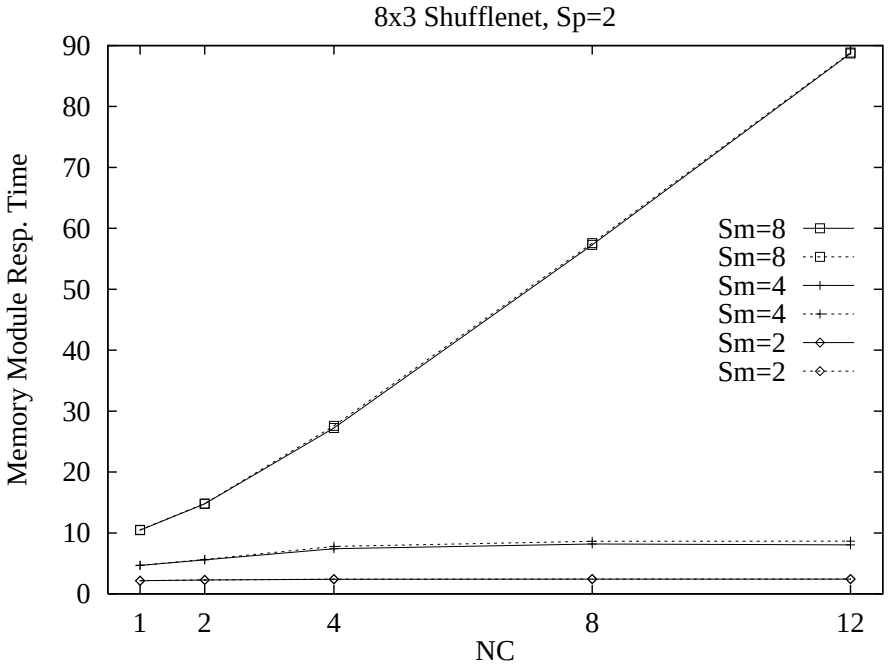

Figure 10: Average memory module response time, $S_p = 2$.

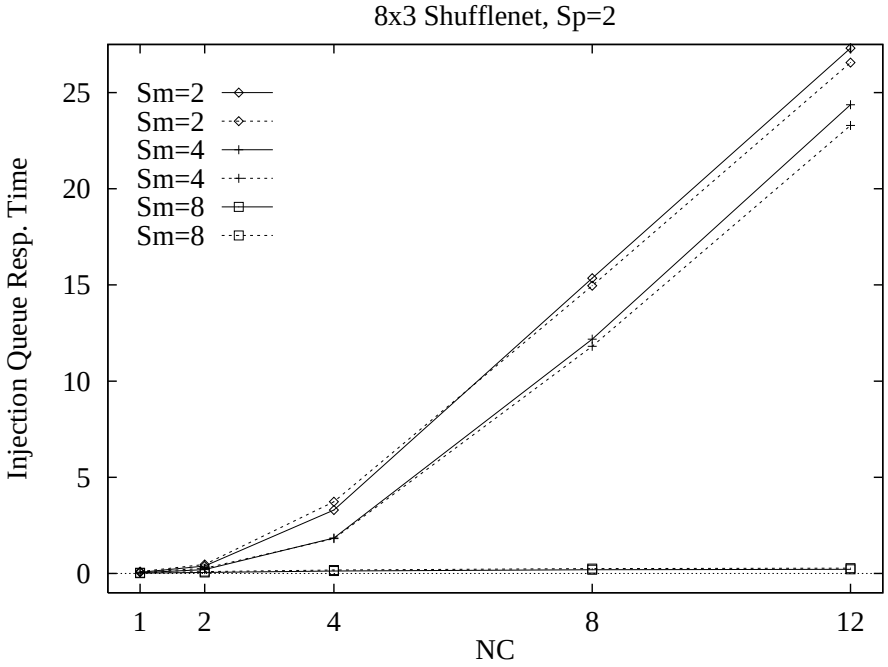

Figure 11: Average injection queue response time, $S_p = 2$.

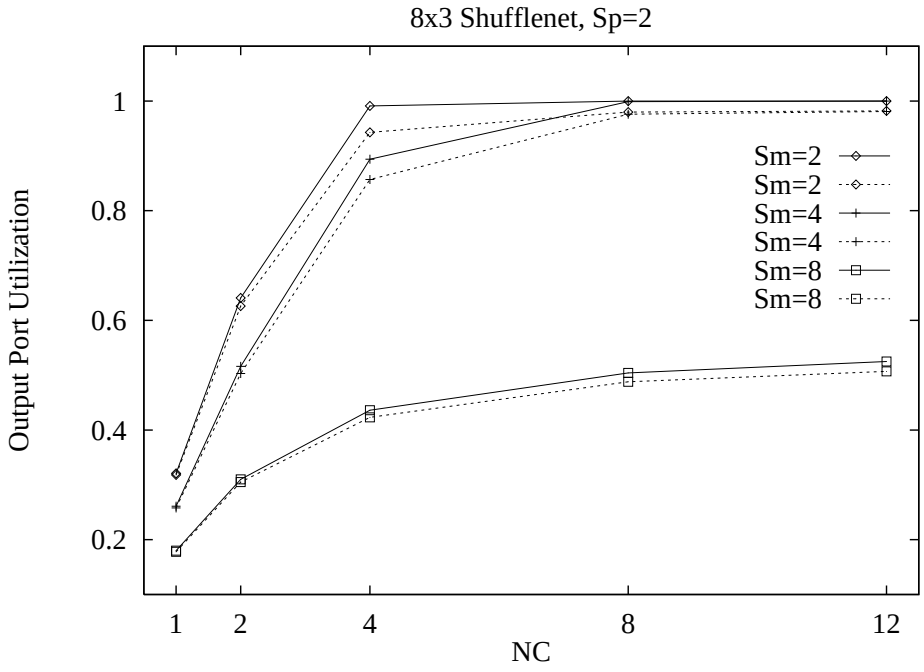

Figure 12: Average output port utilization.

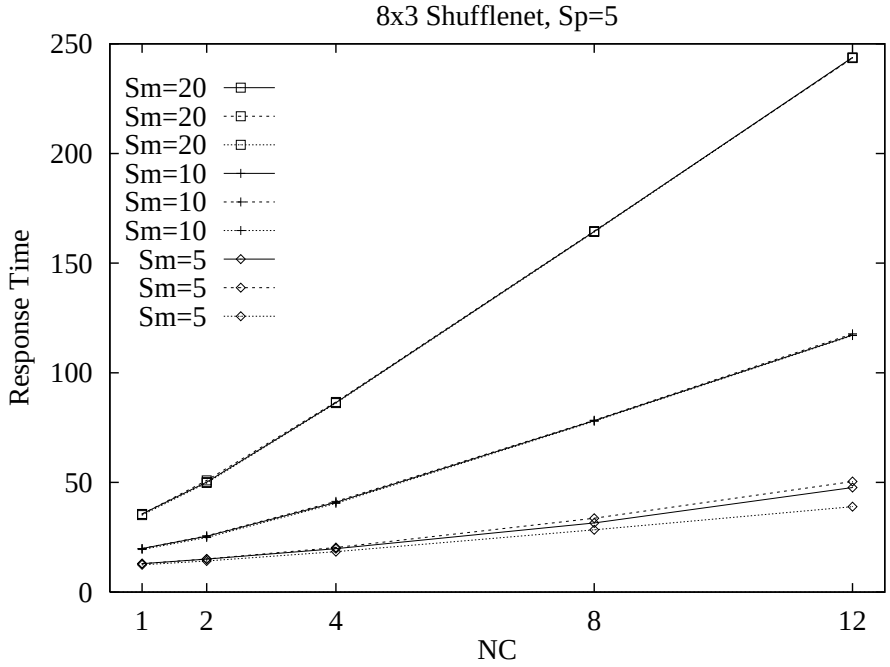

Figure 13: Average per-request total response time, $S_p = 5$.

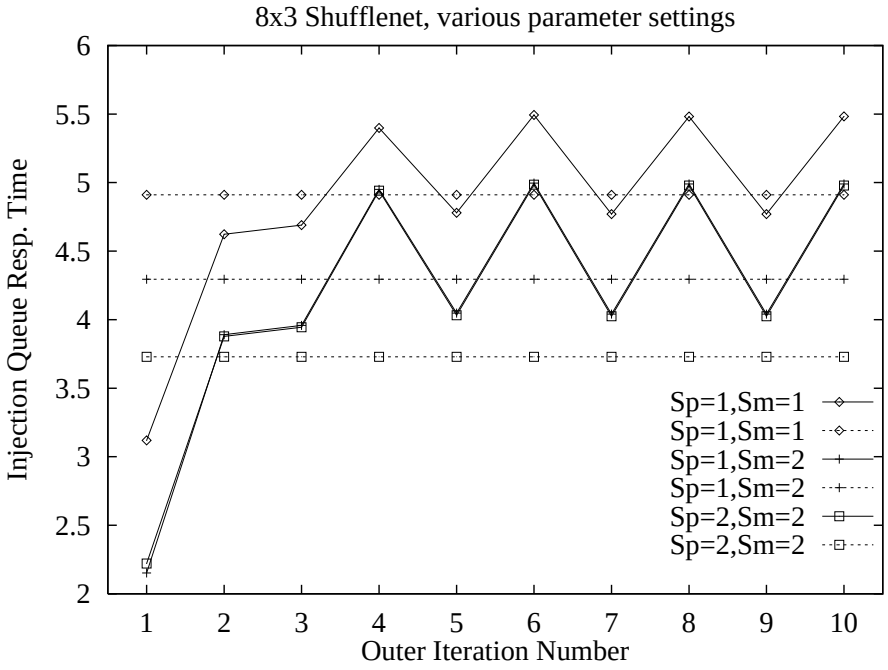

Figure 14: Injection queue response time convergence anomalies.

little traffic passes through this switch without stopping. Hence, $u_{\bar{c}o}$ (the output port utilization due to in-flight messages) is very low. Thus, the model predicts small S_w and small injection queue response times. In reality, however, the high throughput of the injection queue *raises* S_w, since a message about to leave the injection queue for a particular output port o must wait an additional cycle if another message just left the injection queue for o. The model does not take this possibility into account. In fact, the model theoretically allows an arbitrarily large number of messages to leave the injection queue on the same clock cycle, since there is a non-zero probability that $S_w = 0$.

For this reason, we do not feel the model is suitable for non-uniform memory access patterns in its present state of development. Enhancements to overcome this limitation are currently being studied.

4 Conclusions and Future Directions

The primary goal and contribution of this research has been to extend the scope of Approximate Mean Value Analysis to systems in which customer routing is dynamically determined as the model is evaluated. We have applied this technique to the problem of modeling deflection routing in bufferless multistage interconnection networks with uniform memory access patterns. The accuracy of the results is very encouraging.

At the same time, we also are providing the modeling community with further examples of the techniques for analyzing synchronous systems that were introduced in [29].

A secondary goal of this research, which is not yet fully realized, is the accurate modeling of pure blocking centers (the injection queue in our model) whose response time is a function of the state of some *set* of downstream centers (the switch output ports). While the initial efforts are encouraging, much work remains to be done to make this technique more robust. In particular, it appears to be quite challenging to impose an arbitrary, fixed limit on the number of customers that can simultaneously depart from such a server. We are currently experimenting with a number of alternatives in this direction.

An important limitation of the current technique is its computational complexity. For small- to medium-size problems (up to 64 node networks) it is faster than or competitive with careful simulation. However, since the dominant term in its complexity is N^5, it is unsuitable for problems larger than this. There are many promising avenues of exploration that we intend to pursue to improve this situation. For one, it might be possible to reduce the number of linear system solutions in the visit count computations by aggregating visit counts by customer class or location. Although this would compromise accuracy, the degree to which it would do so is unknown. As another possibility, it might be more efficient to use a simple, fast simulation to find the visit counts, alternating with the AMVA calculation. This is an example of *hybrid modeling* [5, 23]. Finally, it might not be necessary to solve the linear systems at all: an alternative approach would be to iterate the set of Equations (19) once during each AMVA iteration. The effect of this technique on accuracy and on the rate of convergence needs to be studied.

We are quite optimistic about the prospects for improving the efficiency of the technique, and we do not see the current inefficiency as a major shortcoming. We view the current technique as a necessary precursor to a more efficient technique, a sort of "existence proof" that AMVA can be applied at all to the architecture described here (at least when cost is no object). After all, the AMVA technique itself is a prime example of a second-generation technique that alleviated the serious computational obstacles of its predecessor, MVA, but this does not diminish the important contribution of MVA itself.

There are several variations of the current architectural model to which we would like to extend the technique. Among these are:

- Pipelining on network links. In an optical network that spans a city, a campus, or even a single building, the length of a fiber link and the speed of message switching are such that several messages can be in transit at the same time on a single fiber, in a pipelined fashion. Thus, there is significant buffering capacity available in the fibers themselves. This is an important real-world consideration that we would like to incorporate into the model.

- Alternative switch routing logic. We can envision many possible alternatives. For example, messages might be injected at switch inputs rather than switch outputs. A priority assignment could be imposed to favor messages coming from the memory modules over messages from the processors. Messages in the injection queue might not enter the network until an optimal output port became available. We can even imagine an ultra-fast network that does completely *random* routing, relying on purely probabilistic guarantees of message delivery.

- Alternative injection queue assumptions. There might be one injection queues per output port, with no opportunity to move between them after the initial choice had been made. There might be a synchronization delay when messages transit from the electronic to the optical domain (and vice-versa).

An obvious omission from the list just given is the handling of multi-packet requests. Unlike purely electronic networks, the bandwith of a fiber link is so high that individual packets can be extremely large; one such technique for accomplishing this is *wavelength division multiplexing*. Furthermore, it is not clear that an optical network could route multi-packet requests any differently than single packet requests even if it wanted to, since the deflection routing discipline completely dashes any hope of keeping related packets traveling together.

Acknowledgements

The author would like to thank the following people: Derek Eager and Darryl Willick for their invaluable technical comments; Dirk Grunwald for suggesting the problem in the first place; Richard Byrd for providing tips on dealing with sparse matrices; and the members of the Center for Opto-Electronic Computing at the University of Colorado, especially Dan Blumenthal, Harry Jordan, and Aruna Ramanan, for helping the author gain a clearer understanding of opto-electronic switch design issues.

The author would also like to acknowledge the National Science Foundation, which funded this research under award #CCR-9010672.

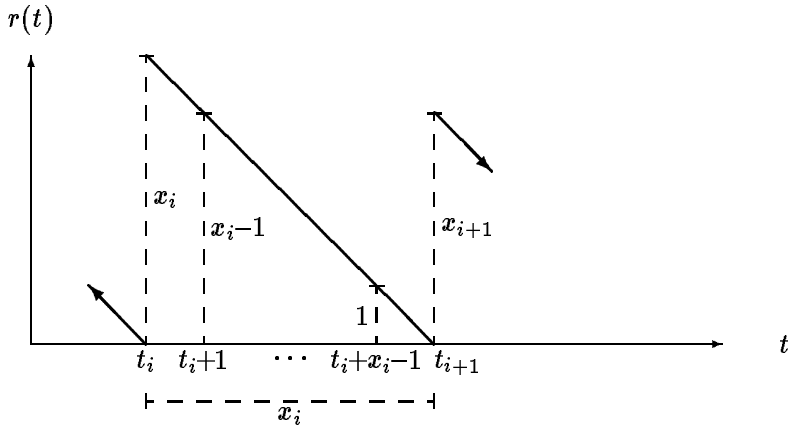

Figure 15: Graphical depiction of residual life.

A Derivation of Mean Residual Life of a Discrete Random Variable

The derivation in this section is a modification of a derivation of Bertsekas and Gallager [2] for the mean residual life of a continuous random variable.

Let $X = \{x_1, x_2, \ldots\}$ be a discrete renewal process with renewal points $\{t_1, t_2, \ldots\}$, and let $r(t)$ be the residual life of X at time t:

$$r(t) = \begin{cases} 0 & \text{if } t \in \{t_1, t_2, \ldots\} \\ x_i - (t - t_i) & \text{if } t_i < t \leq t_{i+1} \end{cases} \tag{23}$$

A graphical depiction of $r(t)$ is shown in Figure 15. Note that we define $r(t)$ to be 0 at all renewal points.[12] Also note that although $r(t)$ is defined for all real t, we assume that it can be observed only at discrete points in time, as indicated on the t-axis.

From this diagram, it is straightforward to calculate the mean residual lifetime of X as the average height of $r(t)$ at all of the discrete points on the t-axis. Let $N(t)$ be the number of renewals up to time t. Then:

$$\begin{aligned} \bar{r} &= \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} r(t) = \lim_{T\to\infty} \frac{1}{T} \sum_{i=1}^{N(T)} \sum_{j=1}^{x_i} r(t_i + j) \\ &= \lim_{T\to\infty} \frac{1}{T} \sum_{i=1}^{N(T)} \sum_{j=1}^{x_i} (x_i - j) = \lim_{T\to\infty} \frac{1}{T} \sum_{i=1}^{N(T)} \frac{x_i(x_i - 1)}{2} \\ &= \lim_{T\to\infty} \left(\frac{N(T)}{T}\right) \left(\frac{1}{N(T)} \sum_{i=1}^{N(T)} \frac{x_i(x_i - 1)}{2}\right) = \frac{1}{E[X]} E\left[\frac{X(X-1)}{2}\right] \\ &= \frac{E[X^2] - E[X]}{2E[X]} \end{aligned} \tag{24}$$

The formulas for $\bar{r}$ in Equations (4,7,13) follow immediately.

[12] This corresponds to our notion that a customer arriving at the end of a clock cycle in which another customer is departing does not "see" the departing customer at all.

References

[1] Adve, V., and Vernon, M. Performance analysis of multiprocessor mesh interconnection networks with wormhole routing. Computer Sciences Dept. 1001a, University of Wisconsin, Madison, WI, June 1992.

[2] Bertsekas, D., and Gallager, R. *Data Networks.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[3] Borodin, A., and Hopcroft, J. Routing, merging, and sorting on parallel models of computation. *J. Computing Systems Science 30* (1985), 130–145.

[4] Brassil, J., and Cruz, R. Nonuniform traffic in the Manhattan street network. In *Proc. ICC '91* (1991), IEEE, pp. 1647–1651.

[5] Browne, J., Chandy, K., Brown, R., Keller, T., Towsley, D., and Dissley, C. Hierarchical techniques for development of realistic models of complex computer systems. *Proc. IEEE 63*, 6 (June 1975).

[6] Choudhury, A., and Li, V. Performance analysis of deflection routing in the Manhattan street network. In *Proc. ICC '91* (1991), IEEE, pp. 1659–1665.

[7] Greenberg, A., and Hajek, B. Deflection routing in hypercube networks. *IEEE Transactions on Communications 40*, 6 (June 1992), 1070–1081.

[8] Hajek, B. Bounds on evacuation time for deflection routing. *Distributed Computing 5* (1991), 1–6.

[9] Hillis, W. *The Connection Machine.* MIT Press, Cambridge, MA, 1985.

[10] Hluchyj, M., and Karol, M. Shufflenet: An application of generalized perfect shuffles to multihop lightwave networks. *IEEE J. Lightwave Tech. 9*, 10 (Oct. 1991), 1386–1396.

[11] Karol, M., Acampora, A., and Hluchyj, M. Terabit lightwave networks: The multihop approach. *AT&T Technical Journal 67*, 1 (1987), 23–34.

[12] Krishna, A., and Hajek, B. Performance of shuffle-like switching networks with deflection. In *Proc. INFOCOM '90* (1990), IEEE, pp. 473–480.

[13] Lavenberg, S., and Reiser, M. Stationary state probabilities of arrival instants for closed queueing networks with multiple types of customers. *J. Applied Probability* (Dec. 1980).

[14] Lawrie, D., and Padua, D. Analysis of message switching with shuffle-exchanges in multiprocessors. In *Interconnection Networks*, C.-L. Wu and T.-Y. Feng, Eds. IEEE Comp. Soc. Press, New York, 1984, ch. 5, pp. 341–348.

[15] Lazowska, E., Zahorjan, J., Graham, G., and Sevcik, K. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Englewood Cliffs, NJ, 1984.

[16] Leutenegger, S., and Vernon, M. A mean-value performance analysis of a new multiprocessor architecture. In *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (May 1988), ACM, pp. 167–176.

[17] Little, J. A proof of the queueing formula $L = \lambda W$. *Operations Research 9* (1961), 383–387.

[18] Maxemchuk, N. Routing in the Manhattan street network. *IEEE Transactions on Communications 35*, 5 (May 1987), 503–512.

[19] Maxemchuk, N. Comparison of deflection and store-and-forward techniques in the Manhattan street and shuffle-exchange networks. In *Proc. INFOCOM '89* (1989), IEEE, pp. 800–809.

[20] Reiser, M., and Lavenberg, S. Mean value analysis of closed multichain queueing networks. *JACM 27*, 2 (Apr. 1980), 313–322.

[21] Robertazzi, T., and Lazar, A. Deflection strategies for the Manhattan street network. In *Proc. ICC '91* (1991), IEEE, pp. 1652–1658.

[22] Schweitzer, P. Approximate analysis of multiclass closed networks of queues. In *Proc. International Conf. on Stochastic Control and Optimization* (1979).

[23] Schwetman, H. Hybrid simulation models of computer systems. *Communications of the ACM 21*, 9 (Sept. 1978), 718–723.

[24] Sevcik, K., and Mitrani, I. The distribution of queueing network states at input and output instants. *JACM 28*, 2 (Apr. 1981), 358–371.

[25] Smith, B. The architecture of HEP. In *Parallel MIMD Computation: HEP Supercomputer and Its Applications*. MIT Press, Cambridge, MA, 1985, pp. 41–55.

[26] Szymanski, T. An analysis of "hot-potato" routing in a fiber optic packet switched network. In *Proc. INFOCOM '90* (1990), IEEE, pp. 918–926.

[27] Vernon, M., Lazowska, E., and Zahorjan, J. An accurate and efficient performance analysis technique for multiprocessor snooping cache-consistency protocols. In *15th Int. Symp. on Computer Architecture* (May 1988), IEEE, pp. 308–315.

[28] Willick, D. Mean value analysis models for multistage interconnection networks. Tech. Rep. 90-11, Department of Computational Science, University of Saskatchewan, Saskatoon, Saskatchewan, Canada, Nov. 1990.

[29] Willick, D., and Eager, D. An analytical model of multistage interconnection networks. In *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (May 1990), ACM, pp. 192–202.

[30] Zhang, Z., and Acampora, A. Analysis of multihop lightwave networks. In *Proc. GLOBECOM '90* (Dec. 1990).