Delay Analysis in Synchronous Circuit-Switched Delta Networks

Amiya Bhattacharya
CSE Department

Ramesh R. Rao
ECE Department

Ting-Ting Y. Lin
ECE Department

University of California, San Diego
La Jolla, CA 92093

Abstract

Multistage interconnection networks (MINs) provide a cost-effective alternative to a full crossbar connection for processor-processor or processor-memory communication in a tightly coupled multiprocessor system. Delta networks, a class of blocking type MIN with unique path property, have been studied extensively for their self-routing capability. A probabilistic analysis of the blocking and its effect on the delay is presented here, for such a network operated in a synchronous circuit-switched mode. Under the assumption of uniformly distributed access requests independently generated at each unblocked source, an upper bound on the expected latency has been established. The bound has been compared with simulation results.

1 Introduction

A wide variety of multistage interconnection networks (MINs) have been proposed for setting up communication between a large number of processors in multiprocessor environments, including the connections between two sets of processors or memory modules. Their increasing popularity is due to the cost savings, as compared to a full crossbar. Low hardware cost is also the reason why self-routing type of MINs are the most popular ones — switches with simple hardware can implement message routing through the network, even in the absence of a centralized controller.

In general, a $N \times N$ (where $N = m^k$) symmetric MIN is configured by arranging $m \times m$ crossbar switches in $k$ stages, with $N/m$ switches in each stage. $2 \times 2$ switches are most common, hence we use $m = 2$ for our bound computation and simulation. Switches of adjacent stages are connected by an interconnection pattern, so that a path can be established between any pair of input and output. However, a given path may inhibit the establishment of another path between a completely disjoint pair, making this type of network of blocking type. As a result, a number of randomly distributed requests may collide. MINs with unique paths between any input-output pair form a class known as delta networks [2, 5], also referred to as generic banyan networks by some authors [1]. These networks have self-routing capability, in the sense that switches at each stage use one bit of the destination address to explore the path for routing. Figure 1 shows an example of such a bit-controlled routing through a

8 x 8 multistage shuffle-exchange or omega network. In this paper, we present a stochastic analysis of the delay due to interaction encountered by uniform traffic through such a synchronous, circuit switched system. The analysis is applicable to all topologies isomorphic to omega, including banyan and baseline networks [3].

Section 2 describes in detail the switching mode and timing of the network, along with the manner in which requests are served. The assumptions and protocols used in the physical operation are clearly

Figure 1: Self-routing through an omega network
specified with justifications. In section 3 we analyze the system based on those assumptions, and compute an upper bound on the average delay encountered by a request. This bound is compared to simulation results in section 4. We conclude by pointing out the benefit of our approach and the possibility of future work.

2 Operational Model

2.1 The Choice of Switching and Timing

In the literature on analysis of multistage interconnection networks, both packet switching [2, 6] and circuit switching [4, 5, 8] models have been considered. In the case of circuit switching, the network can operate in a synchronous or asynchronous mode.

In synchronous circuit-switching, the source modules can submit requests for setting up circuits only at periodically occurring instants. This period is usually referred to as the network cycle time \( t_c \), and will henceforth be called a sweep. Starting from the beginning of a sweep, the apices of partially set up circuits propagate together, at steps of switch delay \( t_a \), through successive stages creating a wavefront. To be consistent with some former literature [8], let us call these steps clock cycles or simply cycles even if the switches are not clocked. After \( k \) cycles, circuits get set up and are subsequently released together after a hold time \( t_h \). All of these constitute the network cycle or sweep length, given by \( t_c = kt_d + t_h \). This is illustrated in figure 2. Sources whose requests are blocked, become

![Timing diagram of two consecutive sweeps](image)

Figure 2: Timing diagram of two consecutive sweeps

aware of their failed attempts, when an indicator signal back-propagates through the partially set-up circuit to reach them within the same sweep, and they resubmit the blocked requests at the beginning of a latter sweep [4, 5].

In the asynchronous mode which is not studied in this paper, sources can submit requests at any time; and as a result, partially set-up circuits face blocking from other fully or partially set-up circuits. Once set up, circuits have a chance to hold for an arbitrary amount of time [8].

Other models for the operation of a MIN can be derived by varying the switching mode, timing, buffer sizes and locations. From an analytical viewpoint, if we only consider the probability of a request reaching the destination, the synchronous and unbuffered packet-switching environment is equivalent to the synchronous circuit-switching environment studied here. Packets are submitted to the network at periodic instants, and they propagate through the stages like the circuit apices. The difference appears when we consider conflicts — blocked packets are simply dropped without any action taken to notify the sources; the responsibility of verifying successful transmission lies with the protocol, and not with the hardware. We explicitly account for the resubmission of blocked requests in our circuit-switched model. We found that this backlog is crucial for tracking the stochastic evolution and very useful for delay analysis. However, arguments for computing blocking probability remains the same for both the cases [6].

2.2 The Distribution of Requests

The most critical factor, other than the network topology or switch hardware, for modelling and analysis of network performance is the distribution of requests at the point of generation. In a multiprocessor environment, this distribution is governed by either the memory reference pattern or the interprocessor communication pattern imposed by the program in execution. Due to locality of reference in programs, locality of interference in the network is a natural phenomenon [4], which leads to creation of hot spots. However, the uniform distribution has become a standard for analysis because of its simplicity. Two variations, namely permutation requests and random requests are widely used. Data skewing algorithms, often used in tightly coupled multiprocessor environments, generate simultaneous memory references to be only permutations of some or all of the memory modules. This restricts each memory module to be addressed by at most one processor at a time. In the context of permutation requests, uniform distribution means that all permutations are equally likely [4, 7]. Random request generation does not adhere to this restriction; and in this context, uniform distribution means that all processors generate requests with equal probability, and they are destined to all the memory modules with equal probability [4, 5, 6]. We chose the latter model for our analysis.

2.3 The Fate of Blocked Requests

The other point of emphasis is on the way blocked requests are handled in a performance model. A nat-
natural demand which arises from the execution of sequential code within each processor is that, if a memory access request is not satisfied due to blocking, it should be resubmitted until success (data dependency assumption). Some of the earlier studies [5] indicate that the steady state blocking probability under independent and uniformly distributed random request pattern does not change considerably if blocked requests are ignored (regenerative assumption), and this leads to a simple closed form expression to the blocking probability. But the strict data dependencies of data skewing algorithms invalidate the usage of regenerative assumption under permutation request pattern, because resubmitted requests may conflict with the newly generated requests and the resulting request pattern may not even be permutations [7]. Moreover, with regenerative assumption, one loses an important performance measure, namely average delay or latency, simply because it cannot be defined. This is why we consider to incorporate the effect of resubmission within the model, which leads us to the following problem.

2.4 The Evil of Mixing

Let us now focus onto the complications which arise when blocked requests are not dropped. A natural choice is to resubmit the same request during the next cycle [7]. However, this is not the only alternative. Data-dependency assumption demands that the blocked request be resubmitted sometime in the future with probability 1, before a new request can be generated at that source. If unblocked sources generate requests with probability $p$ while blocked ones persist with probability $1$, it creates a non-uniform distribution at the beginning of a sweep. This causes a serious analytical problem in extending the existing regenerative models to handle resubmission, as all of them are based on the uniformity assumption for request generation.

For example, the Markov model based on the number of active requests at stage boundaries [4] depends on the preservation of the initial uniformity. Mixing ruins the uniformity at the inputs of the MIN. This also stops us from using the recurrence formula for success rate in regenerative model [5, 6, 7] directly in the data-dependency model.

Although we can identify a Markov chain even for an arbitrary distribution of requests, the associated state space is far too large to lead to a simple analysis.

So, we are motivated to force uniformity at generation by using the same probability of submission for both blocked and unblocked sources. Clearly, if new requests are not to be withheld, this submission probability must be greater than or equal to the generation probability. Inducing uniformity thus calls for generating possibly spurious requests in unblocked sources to match the submission probability for the sake of analysis. The destinations also need to be re-randomized at the same time. This can be justified from the point of view that the addresses were originally generated at random. With this protocol, blocked sources are allowed to resubmit if a success results in a loss of a coin biased with the submission probability. What we achieve as a result is an upper bound on expected value of the latency.

Following are the complete set of assumptions used for our analysis.

1. The source and destination modules are statistically identical within their own group.
2. The requests generated by each source is randomly distributed over all the destination modules.
3. Requests for circuit set-up can only be submitted to the network at periodic instants which are multiple of the network cycle time $t_c$. The partially set up circuit frontiers propagate to the next switch stage also at instants which are multiples of clock cycle time $t_d$, starting from the beginning of a sweep.
4. All switches are identical and can only be connected in either straight or crossed mode. Conflicts occur at a switch during circuit set up only when requests arrive at both of its inputs and try to access the same output port. Switches are unbiased in conflict resolution, and one of the conflicting requests is blocked randomly.
5. Blocked sources do not generate a new request in the next sweep, but resubmit the unserviced request with probability $q$. However, they re-randomize the choice of destination. Unblocked sources, with probability $p$ of generating new request ($q > p$), also submit a new request independently with probability $q$, possibly by loading the network with spurious requests.

3 Analysis of the Model

In our assumptions, we insisted on inducing an independent and uniformly (hence identically) distributed request generation process. The goal is to
use the same counting argument as with the regenerative assumption, for computing the probability of acceptance within a sweep. The difference is that the probability of request generation \( q \) at the input of the network is now a design parameter, which controls the stochastic nature of requests. As the system moves on sweeps over sweep. Following Kruskal and Snir [6, 7], let us note that the uniqueness of paths in delta or banyan networks implies that

**Lemma [Kruskal]:** Let the requests be generated at the source nodes of a uniform banyan network by independent, identically distributed random process that uniformly distributes the requests over all the sink nodes. Assume that the routing logic at each source is "fair", i.e. conflicts are randomly resolved. Then,

1. The patterns of request arrivals at the inputs of the same switch are independent.

2. Requests arriving at an input of a switch are uniformly distributed over the outputs of that switch.

3. For each stage in the network, the pattern of request arrivals at the inputs of that stage have the same distribution.

Let us now consider one of \( m \times m \) crossbars used in the 0th stage. Because of the uniform distribution, the probability that one of its input ports receives a request for a particular output port is \( q/m \), and so the probability that it does not receive a request for that output port is \( 1 - q/m \). Since request generation at each source is an independent process, the probability that a particular output port does not receive any request is \( (1 - q/m)^m \). An output port carries a request if and only if it is requested by one or more input port, the probability of which is then given by \( 1 - (1 - q/m)^m \). This is true for all of the output ports of any of the 0th stage switch, which in turn is the input to the next stage. Proceeding in this way, the distribution of requests at the inputs or outputs of any stage remains uniform, if they start uniformly distributed.

Let \( \{q_i, 0 \leq i < k\} \) denote the probability that an input port of an \( i \)th stage switch carries a request, while \( q_0 \) is the probability that a request appears at any of the output ports of the 4th stage (i.e. the output of the network). Then we have the recurrence \( q_{i+1} = 1 - (1 - q_i/m)^m \) where \( q_0 = q \). The probability of success is given by \( p_s = q_s/q_0 \).

To analyze resubmission following our protocol, we will introduce two integer random variables. Let \( R \) be the total number of unsuccessful attempts made by a request before it succeeds. Clearly \( R \) assumes nonnegative values, and is affected only by \( p_s \), following the geometric distribution, \( Pr[R = r] = p_s(1 - p_s)^r, r \geq 0 \), with mean \( E[R] = \sum_{r=0}^{\infty} r p_s(1 - p_s)^r = 1/p_s - 1 \). The fact that \( \sum_{r=0}^{\infty} Pr[R = r] = \sum_{r=0}^{\infty} p_s(1 - p_s)^r = 1 \) confirms the persisting behavior of blocked requests until success.

Let us now denote by \( T_i \) the length (measured in number of sweeps) of the period following \( i \)th collision until the completion of the next attempt. \( T_i \)s are independent and identically distributed, and take only positive integer values according to the geometric distribution, \( Pr[T_i = t] = q_o(1 - q_0)^{t-1}, t \geq 1 \), with mean \( E[T] = \sum_{t=1}^{\infty} q_o(1 - q_0)^{t-1} = 1/q_o \). Also \( \sum_{t=1}^{\infty} Pr[T_i = t] = \sum_{t=1}^{\infty} q_o(1 - q_0)^{t-1} = 1 \) signifies that a blocked source is sure to resubmit the request.

By conditioning on zero or more unsuccessful attempts, the mean delay or latency encountered by an individual request can be expressed as

\[
E[Delay] = \sum_{r=0}^{\infty} E[Delay | R = r] \cdot Pr[R = r] = 1 \cdot Pr[R = 0] + \sum_{r=1}^{\infty} \{1 + \sum_{t=1}^{\infty} E[T]\} Pr[R = r] = Pr[R = 0] + \sum_{r=1}^{\infty} Pr[R = r] + \sum_{r=1}^{\infty} E[T] \cdot r \cdot Pr[R = r] = 1 + E[T] \cdot E[R] = 1 + \frac{1}{q_o}(\frac{1}{p_s} - 1) = 1 + \frac{1}{q_o} - \frac{1}{q_0}.
\]

This is an increasing function of design parameter \( q \) (= \( q_o \)). Thus, by choosing the minimum value of \( q = p \) within the range \([p, 1]\), the minimum possible upper bound on mean delay can be established.

### 4 Simulation Results

Having the complete specification of the MIN operation, we developed a hardware emulator for validating the bound derived above. The emulator implements software the topology, hardware operations and timing of an omega network, which is a member of the class of MINs under consideration. To be precise, the operational model is simulated with some flexibility for experimentation. Preserving uniformity, which is crucial to the analysis, requires us to submit spurious requests at unblocked sources to match the resubmission rate of blocked ones. However, this is not required in the actual system. Hence, we may compute the systemwide average delay in the simulation runs with different \( p \) and \( q \) (\( p \leq q \)) and compare with the analytical bound. The random number generator
Figure 3: A Comparison of Theoretical and Simulated Results

<table>
<thead>
<tr>
<th>Request</th>
<th>Average delay in simulation for varying $q$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2.136, 2.530, 3.190, 3.772, 4.374</td>
</tr>
<tr>
<td>0.2</td>
<td>2.530, 3.190, 3.772, 4.374</td>
</tr>
<tr>
<td>0.3</td>
<td>3.190, 3.772, 4.374</td>
</tr>
<tr>
<td>0.4</td>
<td>3.772, 4.374</td>
</tr>
<tr>
<td>0.5</td>
<td>4.374</td>
</tr>
<tr>
<td>0.6</td>
<td>5.062</td>
</tr>
<tr>
<td>0.7</td>
<td>5.730</td>
</tr>
<tr>
<td>0.8</td>
<td>6.400</td>
</tr>
<tr>
<td>0.9</td>
<td>7.070</td>
</tr>
<tr>
<td>1.0</td>
<td>7.740</td>
</tr>
</tbody>
</table>

Figure 4: Comparison of Theoretical and Simulation-driven Bound

`rand()` in `stdlib.h` was used for generating random address for unblocked as well as blocked sources (i.e. re-randomization on resubmission was strictly followed), for unbiased conflict resolution at switches, and for implementing generation or resubmission at sources with probabilities $p$ and $q$ respectively. The average delay has been computed by running the emulator over a large number of cycles. The end error of ignoring the blocked requests in last few cycles reduces almost to zero when a run of 100000 cycles is taken. Figure 3 shows these results along with the analytical bound. A plot of the theoretical mean delay along with the bound-generating case ($p = q$) for two runs with 1000 and 100000 cycles is shown in figure 4.

5 Conclusion

In this paper, we have explored the effect of blocking due to traffic interaction in synchronous circuit switched delta networks, with the more realistic assumption of resubmission. Resubmission destroys the uniformity necessary for the easy to analyze model of random request pattern generation. We have been able to analyze it by considering a worse-case protocol, which forces a uniform distribution on the generation process over time and yields in an upper bound on delay. This can be visualized as a space-time analog of appending a randomization network at the input of the actual destination routing network for internally uneven load [1]. The value of this approach is an estimation of mean delay a user faces. This local view of the system performance is especially critical while considering the effect of uneven load or graceful degradation for MIN's with redundancy. It remains to be seen whether this approach can be extended to handle non-uniform traffic originating from sources, with a distribution pattern that conforms with the locality of reference.

References