# Delay and Buffer Bounds in High Radix Interconnection Networks


## Abstract

*We apply recent results in queueing theory to propose a methodology for bounding the buffer depth and packet delay in high radix interconnection networks. While most work in interconnection networks has been focused on the throughput and average latency in such systems, few studies have been done providing statistical guarantees for buffer depth, packet reordering and packet delays. These parameters are key in the design and performance of a network. We present a methodology for calculating such bounds for a practical high radix network and through extensive simulations show its effectiveness for both bursty and non-bursty injection traffic. Our results suggest that modest speedups and buffer depths enable reliable networks without flow control to be constructed.*


## 1 Introduction

High radix Interconnection networks are widely used in supercomputer networks (Merrimac Streaming Supercomputer [3]) and for I/O interconnect (Infiniband Switch fabric [12]). Most research for such interconnection networks focuses on analyzing the throughput and average packet latency of the system. However, little work has been done towards bounding the occupancy of the buffers in the network or the delay incurred by a packet.

The buffer occupancy and the delay distributions play a key role in the design and performance of the network. Bounding the number of packets in a buffer in the network is valuable for network administration and buffer resource allocation. A statistical bound on the packet delay is essential for guaranteeing Quality of Service for delivery of packets. Moreover, the delay distribution is directly related to packet reordering through the network.

Queueing theory [8] provides a huge body of useful results which apply to *product-form* networks. Unfortunately, these results rely on unrealistic assumptions (the most unrealistic being independent exponentially distributed service times at each node as opposed to deterministic service in a real system), and therefore people are reluctant to make use of them. The analysis of a network of deterministic service queues is a known hard problem. The difficulty in analysis primarily arises from the fact that the traffic processes do not retain their statistical properties as they traverse such a network of queues.

Given a myriad of sophisticated techniques developed for analyzing a single deterministic service queue, there has been some recent work that attempts to decompose the network based on large deviations techniques [14, 15]. Most of these results are applicable in convergence regimes, such as in the case when there are several flows passing through a queue, called the *many sources asymptotic* regime. Using the many-sources-aysmptotic, Wischik [14, 15] shows that the distribution of a traffic flow is preserved by passage through a queue with deterministic service, in the limit where the number of independent input flows to that queue increases and the service rate and buffer size increase in proportion. More recently, Shroff et al [6, 5] use similar convergence results to significantly simplify the analysis of such a network. In particular, they show that, if internal nodes in a network are capable of serving many flows, we can remove these nodes from consideration and the queueing behavior of other network nodes remains largely the same.

In this paper, we use the aforementioned convergence results to bound the queue depth and packet delay in high radix interconnection networks. Our simulations show that such convergence results start to kick in when the radix (degree) of the nodes is as small as $16 - 32$. We also use our bounds to study the reordering of packets through the network and to propose a routing mechanism with almost no flow control overhead.

The remainder of this paper is organized as follows: Section 2 reviews recent convergence results in queueing theory. Section 3 discusses how such results can be applied to high radix fat tree interconnection networks. Simulation results are then presented in Section 4. Section 5 discusses the application of such bounds in studying packet reordering and in simplifying the flow control mechanism. Finally, Section 6 concludes.

## 2 Many Sources Queueing Regime

As discussed in Section 1, a recent result by Shroff et al [6, 5] shows how network analysis can be simplified when

the queues in the network carry several flows (called the many sources regime). The authors prove that when an upstream queue serves a large number of traffic flows, the queue length of a downstream queue converges to the queue length of a simplified single queueing system obtained by just removing the upstream queue.

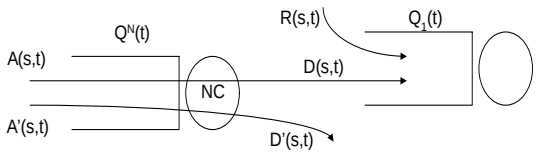


**Figure 1. An upstream queue feeding some flows into a downstream queue**

Consider the set up of two FIFO queues in Figure 1. Let there be $N$ flows going into the upstream queue. The subset of these flows that go on into the downstream queue have a combined arrival process given by $A(s,t)$ which is the total number of packets arriving in the time interval $(s,t)$. The remaining set of flows have arrival process $A'(s,t)$. Let the service capacity of the upstream queue be $NC$, i.e. the service per flow is $C$ packets per time step. The departing flows from the first queue going into the downstream queue have a departure process given by $D(s,t)$. The downstream queue can also receive more cross traffic given by $R(s,t)$. Let the queue depth of the downstream queue at time $t$ be $Q_1(t)$ while that for the upstream queue be $Q^N(t)$. In order to find the Prob$(Q_1 > x)$, we can simplify the above scenario into just one queue.

Figure 2 shows a simplified scenario of Figure 1. In this figure, the effect of the upstream queue on flows $A(s,t)$ is ignored. Let the queue depth of the downstream queue for this scenario be $Q_2(t)$. The authors of [5] prove that as $N \to \infty$, Prob$(Q_1 > x)$ converges to Prob$(Q_2 > x)$ and that the speed of this convergence is exponentially fast. Hence, with a modest number of multiplexed sources, the convergence results start to hold.

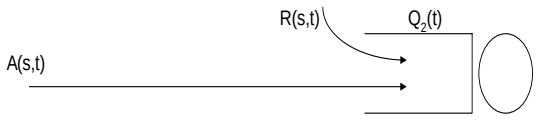


**Figure 2. Simplified scenarion of the set up of two queues**

At first glance, it may seem that the flows traversing the upstream queue should get "rate shaped", making the departure process over a time interval $t$, $D(0,t)$, smoother than the corresponding arrival process $A(0,t)$. However, on closer analysis, this need not be the case. Consider an individual flow, $i$, traversing the upstream queue and going to the downstream queue. Its arrival process is $A_i(0,t)$ and departure is $D_i(0,t)$. Now the arrival and departure processes are related as: $D_i(0,t) = A_i(0,t) + Q_i^N(0) - Q_i^N(t)$, where $Q_i^N(0)$ and $Q_i^N(t)$ are the backlog of flow $i$ in the upstream queue at times $0$ and $t$, respectively. Due to fluctuations in the queue depth and the interaction among the $N$ flows in the upstream queue, $Q_i^N(0)$ can be larger than $Q_i^N(t)$, making the departure process for that flow larger than the arrival process.

## 3 Application to High Radix Fat Trees

In the rest of this paper, we shall apply the many sources regime results to analyzing buffer depth in a popular high radix topology — Clos [2] or fat tree networks [9]. The high radix switch queues are an appropriate application for the many sources asymptotic results. As the radix (and the number of sources) increases, the statistical properties of the flows get preserved as they traverse the network. Our simulations show that the convergence results hold for a radix as small as $16 - 32$.

In our experimental set up, we have simulated a specific kind of fat tree — a $k$-ary $n$ tree network [11]. A $k$-ary $n$ tree network has $n$ levels of internal switches and a total of $k^n$ leaf nodes that can communicate with each other using these switches. There are $nk^{n-1}$ internal switches which have $2k$ incoming ports and $2k$ outgoing ports. The internal switches have buffers inside where packets are stored and serviced to their appropriate output port. Figure 3 shows a diagram for a $2$-ary $4$ tree.

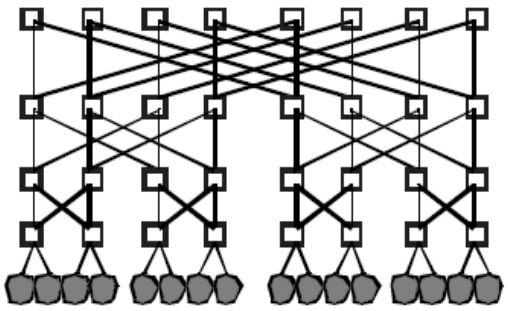

**Figure 3. A 2 ary 4 tree**

In our simulations, we will concentrate on high radix trees where $k$ is of the order of $16$ or $32$. Figure 4 shows a hierarchical schematic for a $k$ ary 3 tree. There are two levels of hierarchy denoted by levels $A$ and $B$ and three levels of switches since $n = 3$. Depending on $k$, each entity of

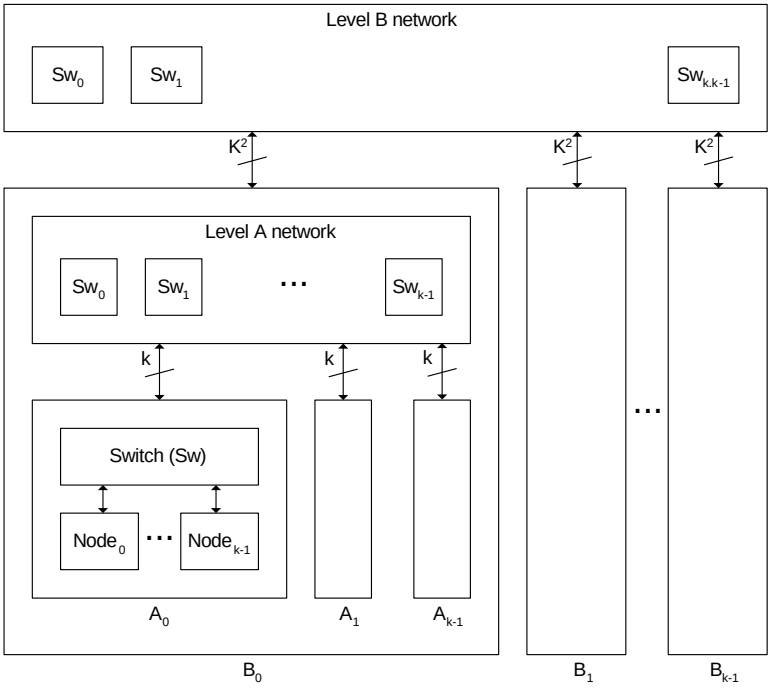


**Figure 4. The k ary 3 tree used in our smulations**

level $A$ may be thought of as a board (or a backplane comprising several boards) while each entity of level $B$ may be thought of as a backplane (or cabinet). The wiring between levels is abstracted out for simplicity.

Load balancing on such a fat tree is easily accomplished using Random Root Routing (RRR) in which each packet is routed to a randomly chosen root switch and then down to the desired destination traversing a total of $2n - 1$ internal switches (hops) for every packet. A more sophisticated approach — called Random Common Ancestor Routing (RCAR) — is to route up to a (randomly chosen) common ancestor of the source and destination leaf node and then down to the destination node. In our analysis, we focus on RRR as it enables us to treat all traffic patterns as two phases of uniformly random traffic, thus making the analysis more tractable. The analysis for RRR is also a conservative analysis for RCAR as the latter has strictly lesser packets using the resources in the upper levels of the network.

# 4 Results

In this section, we first study the impact of the radix on the buffer occupancy distribution in the queues at each hop of the fat tree network. Our approach is to first increase the radix $k$ in a $k$ ary 3 tree network while keeping the per channel bandwidth constant. We then plot the Complementary Cumulative Distribution Functions (CCDFs) of the queue occupancy at each of the $5$ hops of the network. We perform this experiment for non-bursty Bernoulli and bursty injection traffic. For both these injection processes, the many sources convergence results start to manifest themselves at reasonably small values of $k$. Using these values of $k$, we can analytically calculate the exact CCDF from queueing theory for each of the $5$ queues, thus giving us buffer depth bounds. We then use the per hop buffer depth bound to approximate the end-to-end delay of a packet through the network by convolving the per-hop distributions obtained. This approximation gives very accurate results especially at high injection loads.

## 4.1 Increasing the radix $k$

In the very first experiment, we inject packets at each source according to a non-bursty Bernoulli iid process. Each source injects a packet with a probability $p$ at every time step. We increase the radix k of each switch and measure the queue depth at each of the $5$ hops of a $k$-ary 3 tree network. Figure 5 shows that the CCDFs of the queues are quite divergent for a low radix ($k = 2$) but tend to converge to almost identical as $k$ is increased to $16$. This is because, for a high enough radix, the statistical properties of the flows are preserved as they traverse the network. Hence, for the non-bursty injection process, a radix of $16$ is a rea-

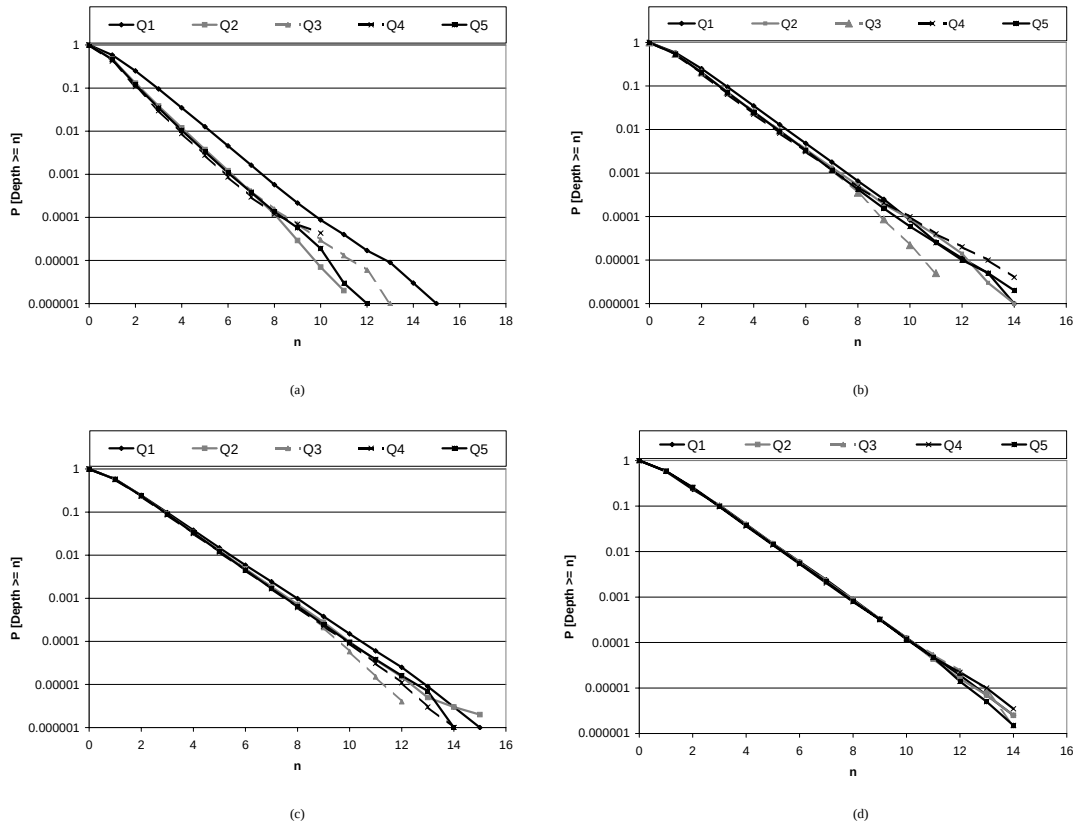


**Figure 5. Queue depth at each hop of a k-ary 3 tree for (a) k=2 (b) k=4 (c) k=8 and (d) k=16**

sonable working parameter to use the convergence results.

### 4.2 Analytically obtaining the CCDF

We now describe our analytical approach for obtaining the CCDFs of the queue depths at each hop. For the 16 ary 3 tree case with non-bursty injection, it suffices to obtain the CCDF for the first hop as the other hops behave almost identically.

Let $A$ be the random variable that represents the total traffic at each time step to the queue at the first hop.

$$A = \sum_{i=1}^{k} X_i \tag{1}$$

where $X_1, X_2, \ldots X_k$ are independent IID Bernoulli random variables corresponding to the $k$ sources such that if $p_i$ is the probability that the source $i$ will send a packet along this queue, then

$$X_i = \begin{cases} 1 & w.p.\ p_i \\ 0 & w.p.\ q_i = 1 - p_i \end{cases} \tag{2}$$

From Equations 1,2 we get

$$E[A] = \sum_{i=0}^{k} p_i$$

Let the service capacity of the queue be $C = 1$ packet per time step. If the queue is stable i.e., $E[A] < C$, we can find the Probability Generating Function (PGF)[1] of the queue size $Q(z)$ using the formula derived in [8].

$$Q(z) = P(Q = 0)\frac{(z-1)A(z)}{(z - A(z))} \tag{3}$$

In our case

$$A(z) = \prod_{i=0}^{k}(q_i + p_i z) \text{ and } P(Q = 0) = 1 - E[A] \tag{4}$$

The derivation for $P(Q = 0)$ is shown in Appendix A. Moreover, $p_i = p$ for all $i$ since all sources inject traffic with the same probability. Using Equation 3 we can find out the queue size distribution in the following way :

$$P(Q = n) = \frac{1}{n!}\left[\left(\frac{d}{dz}\right)^n Q(z)\right]_{z=0} \tag{5}$$

[1]The PGF, G, of a random variable, $X$, is given by $G(z) = E(z^X) = \sum_{i=0}^{\infty} f(i)z^i$, where $f$ is the probability mass function for $X$.

Equation 5 can then be used to derive the CCDF for the queue depth at each hop of the 16 ary 3 tree network.

Figure 6 shows the analytically evaluated queue depth with the error margins derived as in [5] against the measured values obtained in the previous subsection. The queues at each hop are at a utilization of $60\%$ i.e., $E[A]/C = 0.6$. The model matches almost exactly with the simulations. While the observed values were obtained through simulations run for 2 days, the theoretical CCDF could be derived in a matter of minutes using the Maple software [1] for solving Equation 5. The advantage is that we could quickly derive the buffer depth required for which the overflow probability would be of the order of $10^{-15}$, something that was not tractable by simulations alone.

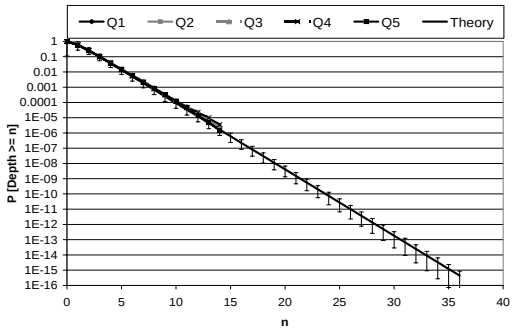


**Figure 6. Analytically derived queue depth against the measured queue depth at each hop for a 16 ary 3 tree at 0.6 load**

### 4.3 Approximating the delay CCDF

Having found the buffer depth distributions at each hop, our next aim is to approximate the end-to-end delay of packets traversing the network. The latency, $T$, incurred by a packet is the sum of two components, $T = H + D$, where $H$ is the hop count and $D$ is the queueing delay. For the 16 ary 3 tree case, $H = 5$. The random variable $D$ is the queueing delay incurred due to buffering at each of the hops in the network. The per-hop delay in each queue is directly related to the occupancy of the queue and its service capacity. In order to find the distribution of $D$, we make the simplifying assumption that the per hop delays at each hop are independent of each other. This assumption has been shown to give accurate results in practice especially as the load is increased [16]. This is because, as the buffer load increases, the output of a queue (which is the input to the downstream queue) gets less correlated with the input process.

Using this independence assumption, we can find the distribution of $D$ by simply convolving the per-hop delay distributions calculated before. Convolving the distributions is equivalent to simply taking the product of their PGFs. For the case when the service is 1 packet per time step, $D$ is given by

$$D(z) = \prod_{i=1}^{H} Q_i(z) \tag{6}$$

Finally, adding a constant $H$ to $D$ gives us the end-to-end delay distribution.

Figure 7 compares the analytical delay CCDF with the measured values for injection loads of $0.2$, $0.6$ and $0.8$. As expected, the analytical and observed plots are a close match for the highest load of $0.8$ and are reasonably good approximations for the lower loads.

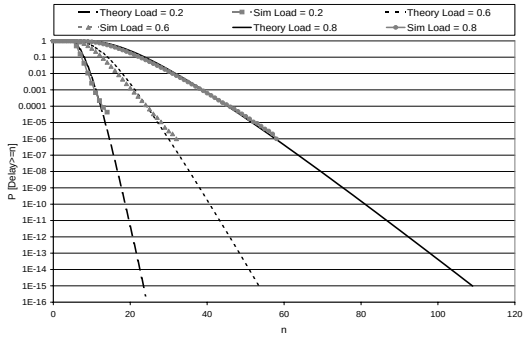


**Figure 7. End-to-end packet delay (theoretical and measured) for a 16 ary 3 tree at different injected loads**

### 4.4 Bursty Flows

The radix at which reasonable convergence holds depends on the size of the network and the nature of the injection sources. In this subsection, we use bursty sources for injection into the network instead of the non-bursty Bernoulli sources that we have used thus far. The injection process is now based on a simple Markov ON/OFF process which produces packet bursts that are geometrically distributed with average burst length of $5$ packets. We repeat the same set of experiments as in the Bernoulli injection case.

As we increase the radix $k$, keeping the load fixed at $0.6$ and the per channel bandwidth constant, we once again observe a convergence in the CCDFs of queue lengths at different hops. For the bursty case, the CCDFs for $k = 16$ do not converge as well as in the Bernoulli case (Figure 8) and a reasonable convergence is observed at a higher value of $k = 32$ (Figure 9).

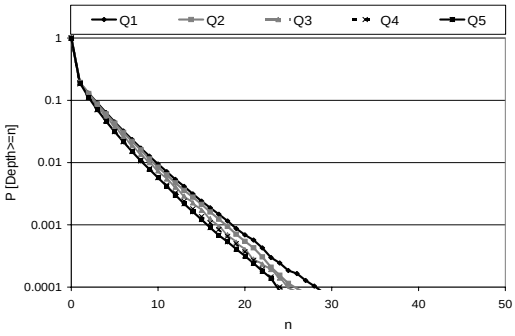


**Figure 8. Queue depth at each hop of a 16 ary 3 tree for bursty traffic and 0.6 load**

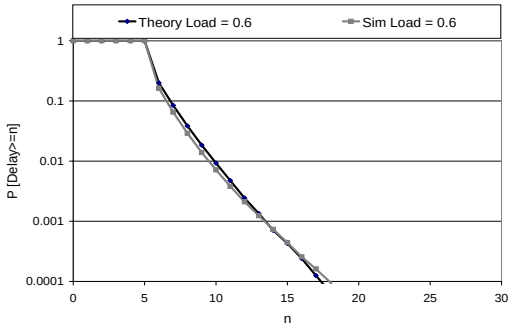


**Figure 10. End-to-end packet delay for a 32 ary 3 tree for bursty traffic at an injected load of 0.6**

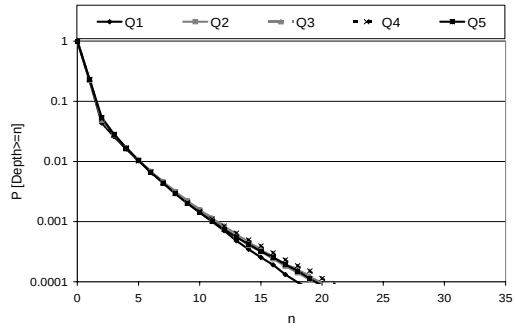


**Figure 9. Queue depth at each hop of a 32 ary 3 tree for bursty traffic and 0.6 load**

The analytical derivation of the queue depth for such a correlated, bursty injection process is mathematically involved and is beyond the scope of this paper. The interested reader is referred to [7] for more details. In order to evaluate the end-to-end delay CCDF for the bursty case, we use the *measured* distribution for the queue delay at the first queue and convolve it five times to get the delay distribution. As seen in Figure 10, the CCDF obtained by the convolution is very close to the measured delay CCDF for $k = 32$.

# 5 Discussion

## 5.1 Using buffer bounds to disable flow control

An interesting fallout from deriving buffer depth bounds is that we can use them to make the flow control of the packets trivial. In practice, the packets traversing the network need to be allocated resources before they can actually use them. This process of allocation of network resources is called flow control. The most commonly used flow control mechanism for allocating buffers to packets is the credit-based flow control [4] (Chapter 13). In order to not drop packets, a packet in the upstream node does not leave for the downstream node until it gets a *credit* signifying there is enough buffer space available there. If we were to send a packet to the downstream node ignoring this flow control information, the packet would be dropped if there was no space in the downstream node. However, we can set our buffer depths to a reasonable size and make sure that the probability of a drop due to buffer overflow is substantially lower than the probability of a drop due to a hard error. In order to do this, we must either police the injection process or provide a modest internal speedup to the network.

Take, for instance, the non-bursty injection process on a $16$ ary $3$ tree. We have successfully computed the buffer depth requirement for a particular injection load of $0.6$ that has a very low probability of exceeding (of the order $10^{-15}$). Repeating our calculations for different loads, we can study how the buffer depth requirements grow as the load is increased keeping the probability of overflow constant at $10^{-15}$.

Figure 11 shows that as the injected load reaches the saturation value of $1$, the buffers start to grow without bound. However, at slightly less than this saturation value, say $0.9$, a buffer size of $160$ packets is required to ensure that the drop probability without flow control is less than $10^{-15}$. Hence, either by policing the injection to make sure that the injection rate stays below $0.9$ or by providing a small internal speedup of $1/0.9 = 1.11$, we can eliminate the flow control overhead from the routing process. A similar analysis can be carried out for other injection processes.

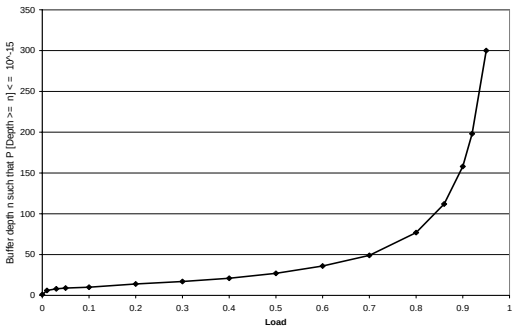


**Figure 11. Buffer depth requirement at various injection loads**

## 5.2 Bounding the reordering of packets

While load balancing algorithms such as RRR and RCAR increase the throughput of the network, they also reorder the packets traveling from a particular source, $s$, to a particular destination, $d$. This happens because packets are sent over different paths with potentially different delays from $s$ to $d$. For instance, Figure 12 shows packets sent over 3 different paths. The black packet is the next packet expected by $d$ but it is delayed in the congested path $P_1$. To deliver packets in sequence to $d$, the other packets (which were injected later than the black packet but reached earlier) must be stored and reordered in a Reorder Buffer (ROB) using the well known sliding window protocol [13].

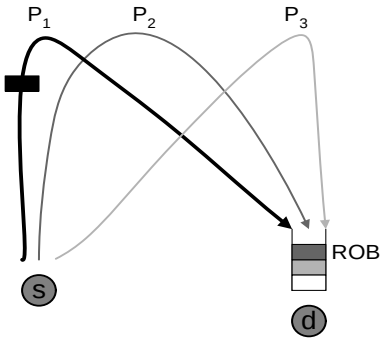


**Figure 12. Reordering of packets sent over 3 different paths**

At each destination, there must be a ROB corresponding to each source. Choosing the size of each ROB is critical to the throughput of the network. If the ROB becomes full, it will apply backpressure to the rest of the network fabric to prevent dropping of packets leading to a reduction in network throughput. Hence, we must evaluate a ROB size such that the probability of it getting full is less than that of a hard error.

Unlike the case of the switch buffers discussed thus far, the ROB occupancy varies with traffic patterns. Consider a permutation traffic pattern like Bit Complement (BC) and one that is not a permutation, Uniform Random (UR). In BC, $s$ sends packets to a fixed destination, $N-1-s$, where $N$ is the total number of destinations while in UR, $s$ sends to a randomly chosen destination. Since reordering occurs for source-destination pairs, permutation patterns use only one ROB at each destination sending more traffic to each ROB than a non-permutation traffic like UR. Consequently, the ROB occupancy for any permutation is bigger than that for a non-permutation traffic for the same injection load. For this reason, we focus on the BC traffic pattern for our ROB size calculations.

Let us denote the CCDF for the delay that we evaluated in Section 4.3 by $D_i$, i.e., $D_i = P[Delay \geq i]$. Let the load on the network be a fraction $l$ of its capacity. Let $Q_r$ be the occupancy of each reorder buffer. Then, for our canonical example of non-bursty Bernoulli arrivals, we can theoretically bound $Q_r$ as follows:

**Claim 1.** *For non-bursty Bernoulli arrivals, if* $\rho_d = P[Delay = d] = D_d - D_{d+1}$, *then*

$$P[Q_r \geq k] \leq \sum_{d=0}^{\infty} \rho_d \, e^{(e^{\eta_d}-1)S_d - \eta_d k} \tag{7}$$

$$\textit{where } \eta_d = ln(k/S_d) \textit{ and } S_d = l\sum_{i=0}^{d-1}(1-D_i)$$

*Proof.* The proof is presented in Appendix B. □

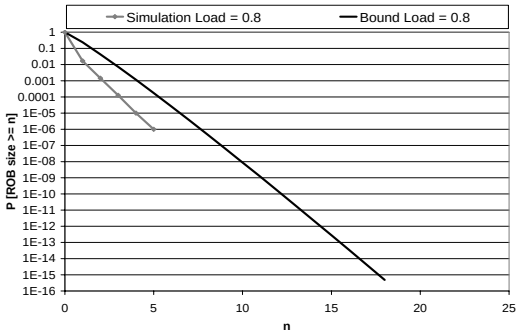


**Figure 13. Bounding the Reorder Buffer Occupancy**

Using the delay distribution derived previously, we can evaluate Equation 7 to get a bound on the ROB occupancy. Figure 13 compares the theoretical bound with the simulated ROB occupancy for an injection load of $0.8$. The probability of overflow for a ROB of size $18$ packets is less than $10^{-15}$. Repeating our calculations for different loads enables us to study how the ROB size requirement grows with load for the same overflow probability.

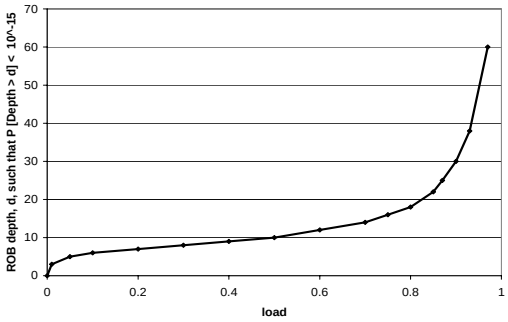


**Figure 14. ROB size requirements at various injection loads**

Figure 14 shows the required ROB size with increasing injection load. As in the case of the switch buffers, the ROB size also grows without bound as the load reaches the saturation value. At a load of $0.9$ or less, we require at most $30$ packet sized ROBs for packets to be reliably reordered and delivered to their destination. A similar calculation can be carried out for bursty injection.

## 6 Conclusion and Future work

In this paper, we have used recent convergence results in queueing theory to propose a methodology for bounding the buffer depth and packet delay in high radix interconnection networks. We have presented extensive simulations for both non-bursty and bursty injection traffic and show that the convergence results start to kick in for radix values as small as $16 - 32$. Using the delay distributions, we study packet reordering in the network and estimate bounds for the reorder buffer size in the network. Finally, we use the bounds to propose a routing mechanism with negligible flow-control overhead by either policing the network at the source or introducing a small internal speedup in the network.

The advantages of disabling flow control are manifold. Significant bandwidth is saved which is otherwise used up by flow control credits. Moreover, practical flow control methods like credit based flow control can require substantial buffer space to maintain full throughput on a single virtual channel. We are currently doing a study to quantify the overhead cost of flow control in interconnection networks.

The methodology described in this paper is applicable only to oblivious routing algorithms. For adaptive algorithms, which make routing decisions taking the network state into account, there is no known technique for evaluating statistical guarantees. However, in practice, adaptive algorithms perform better in the average case compared to oblivious routing algorithms. Developing a technique for bounding delay and buffer depth for adaptive routing remains an open question.

## References


[1] B. Char, K. Geddes, B. G.H. Gonnet, M. Monagan, and S. Watt. *Maple V Language Reference Manual*. Springer-Verlag New York, Inc., 1991.

[2] C. Clos. A study of non-blocking switching networks. *The Bell System technical Journal*, 32(2):406–424, March 1953.

[3] W. J. Dally, P. Hanrahan, M. Erez, T. J. Knight, F. Labonte, J.-H. Ahn, N. Jayasena, U. J. Kapasi, A. Das, J. Gummaraju, and I. Buck. Merrimac: Supercomputing with Streams. In *Proceedings of SuperComputing, SC'03*, Phoenix, Arizona, November 2003.

[4] W. J. Dally and B. Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann, San Francisco, CA, 2004.

[5] D. Y. Eun and N. B. Shroff. Simplification of network analysis in large-bandwidth systems. In *Proceedings of IEEE INFOCOM*, San Francisco, California, April 2003.

[6] D. Y. Eun and N. B. Shroff. Network decomposition in the many-sources regime. *Advances in Applied Probability*, 36(3):893–918, September 2004.

[7] H. S. Kim and N. B. Shroff. Loss probability calculations and asymptotic analysis for finite buffer multiplexers. *IEEE/ACM Transactions on Networking*, 9(6):755–768, 2001.

[8] L. Kleinrock. *Queueing Systems – Volume 1: Theory.*, pages 191–194, Eqn 5.85. Wiley, New York, 1975.

[9] C. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computer*, C-34(10):892–901, October 1985.

[10] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge Univ. Press, 1995.

[11] F. Petrini and M. Vanneschi. $k$-ary $n$-trees: High Performance Networks for Massively Parallel Architectures. In *Proceedings of the 11th International Parallel Processing Symposium, IPPS'97*, pages 87–93, Geneva, Switzerland, April 1997.

[12] G. Pfister. *An Introduction to the InfiniBand Arechitecture (http://www.infinibadta.org)*. IEEE Press, 2001.

[13] A. S. Tanenbaum. *Computer Networks, 3rd ed.* Prentice Hall, 1996. Pages 202-219.

[14] D. J. Wischik. The output of a switch, or, effective bandwidths for networks. *Queueing Syst. Theory Appl.*, 32(4):383–396, 1999.

[15] D. J. Wischik. Sample path large deviations for queues with many inputs. *Annals of Applied Probability*, 11(2):379–404, May 2001.

[16] D. Yates, J. Kurose, D. Towsley, and M. Hluchy. On per-session end-to-end delay distributions and the call admission problem for real-time applications with qos requirement. *Journal on High Speed Networks*, 3(4):429–458, 1994.

## A Finding $P(Q = 0)$

In order to find $P(Q = 0)$ for the set up described in Section 4.2, let us construct the time series equation for the queue depth, $Q$. If $Q_n$ is the depth of the buffer at the end of time slot $n$, then the occupancy in the next step will increase by the number of arrivals at slot $n + 1$ and decrease by 1 (0) if the queue is non-empty (empty) at time $n$. Mathematically, this means

$$Q_{n+1} = Q_n - \Delta_{Q_n} + A_{n+1} \tag{8}$$

where $\Delta_k$ is a shifted discrete step function

$$\Delta_k = \begin{cases} 1 & k = 1, 2, \ldots \\ 0 & k \leq 0 \end{cases} \tag{9}$$

Let us take expectations on both sides of Equation 8. Since, at steady state, we can drop the subscripts we have

$$E(Q) = E(Q) - E(\Delta_Q) + E(A) \tag{10}$$

Now, from Equation 9 it is obvious that $E(\Delta_Q) = P[Q > 0]$. Substituting in (10), we get

$$P[Q = 0] = 1 - E(A)$$

## B Proof of Claim 1

In order to find $P[Q_r \geq k]$, we must find the probability of the event that while the ROB is waiting for the next in-sequence packet (call it $C$), $k$ or more packets injected after $C$ arrive at the ROB. Without loss of generality, let $C$ be injected by the source at time 0. Once $C$ arrives at the ROB after some delay, $d$, the buffer starts to drain. We need to count the number of packets that are generated and reach the ROB through paths different from that of $C$ in the interval $(0, d]$. The probability that this number exceeds $k$ for all $d$ is equivalent to $P[Q_r \geq k]$.

Figure 15 shows the time-line for packet $C$. For each discrete time step, $i$, in the interval $(0, d]$, define an indicator variable, $T_i$, such that $T_i = 1$ if a packet is generated at time $i$ and also reaches the ROB with a delay less than $d - i$ [2].

[2]For simplicity, we ignore the probability that the packet will follow the same path as $C$ as the number of paths to the destination is large.

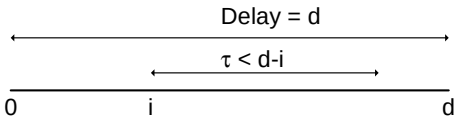


**Figure 15. Time-line for deriving ROB occupancy**

Since the load is $l$ and the delay CCDF is $D_i$, we have

$$T_i = \begin{cases} 1 & w.p.\ l(1 - D_{d-i}) \\ 0 & else \end{cases}$$

Then, if $C's$ delay is given by a random variable $D_C$,

$$P(Q_r \geq k) = \sum_{d=0}^{\infty} P(D_C = d) P(Q_r \geq k | D_C = d) \tag{11}$$

Now, $P(D_C = d) = \rho_d = D_d - D_{d+1}$ and for a fixed $D_C = d$, $Q_r = \sum_{i=1}^{d} T_i$. Thus, to find a bound on $P(Q_r \geq k)$, we need to bound the probability that the sum of the Bernoulli variables $T_i$ exceeds $k$. We use a Chernoff bounding technique similar to the one used in [10].

For a fixed $D_C = d$, and any $\eta_d > 0$,

$$P(Q_r \geq k) = P(e^{\eta_d \sum_{i=1}^{d} T_i} \geq e^{\eta_d k}) \leq \frac{\prod_{i=1}^{d} E[e^{\eta_d T_i}]}{e^{\eta_d k}}$$

The RHS of the above inequality can be simplified to $\frac{\prod_{i=1}^{d}(1+(e^{\eta_d}-1)l(1-D_{j-i}))}{e^{\eta_d k}}$. Finally, using the fact that $1 + x < e^x$, we can simplify the inequality to

$$P(Q_r \geq k | D_C = d) \leq e^{(e^{\eta_d}-1)S_d - \eta_d k} \tag{12}$$

where $S_d = l \sum_{i=0}^{d-1}(1 - D_i)$.

To get the tightest bound, we must minimize the exponent in the RHS of Equation 12. Solving, we get $\eta_d = ln(k/S_d)$. Substituting in Equation 11, we get the desired result.