Age-Based Packet Arbitration in Large-Radix k-ary n-cubes

Dennis Abts
dabts@cray.com

Cray Inc.
1050 Lowater Road
Chippewa Falls, Wisconsin 54729

Deborah Weisser[†]
dweisser@google.com

Google Inc.
1600 Amphitheatre Parkway
Mountain View, California 94043

Abstract

As applications scale to increasingly large processor counts, the interconnection network is frequently the limiting factor in application performance. In order to achieve application scalability, the interconnect must maintain high bandwidth while minimizing variation in packet latency. As the offered load in the network increases with growing problem sizes and processor counts, so does the expected maximum packet latency in the network, directly impacting performance of applications with any synchronized communication. Age-based packet arbitration reduces the variance in packet latency as well as average latency. This paper describes the Cray XT router packet aging algorithm which allows globally fair arbitration by incorporating "age" in the packet output arbitration. We describe the parameters of the aging algorithm and how to arrive at appropriate settings. We show that an efficient aging algorithm reduces both the average packet latency and the variance in packet latency on communication-intensive benchmarks.

General terms: design; architecture; performance

Keywords: multiprocessor; MPP; torus; interconnection network; arbitration; routing; packet-switching

SC07 November 10-16, 2007, Reno, Nevada, USA

[†]This work was performed while Deborah Weisser was a research staff member at Pittsburgh Supercomputing Center, Pittsburgh, Pennsylvania.

1 Introduction

The interconnection network plays a central role in the performance of a parallel computer, often acting as a limiting factor in application performance and scalability. The point-to-point and global bandwidth of the network are critical to large-scale application performance. The latency of the network determines the access time for remote memory references. In addition to the average packet latency, the variance and maximum latency strongly affect performance of applications with synchronized communication, where the maximum latency is the limiting factor. Because of the increase in overall communication volume, the expected maximum length of time for one packet to be delivered increases. For this reason, reducing the variance in communication time is pivotal to performance.

We describe the Cray XT [1] interconnection network in terms of its topology, flow control, virtual channels, and router switch allocation. This paper gives an overview of the system-on-chip (SoC) called "Seastar" that includes the network interface controller (NIC) functionality and an embedded 3-D torus router with six full-duplex network ports and one processor interface port. We provide an overview of the router microarchitecture and a detailed description of the packet age-based arbitration policy for the Cray XT network[1]. We discuss optimal settings of packet routing parameters and demonstrate the effects of these settings on various SeaStar performance register statistics and several benchmarks.

1.1 Topology, routing, and flow control

The Cray XT network is a k-ary 3-cube that scales up to 32K nodes. The flexible routing mechanism of the XT allows a mesh or torus in any of the three dimensions. In practice, packaging constraints make it difficult for the *radix* (k) of the network to be the same for all three dimensions. So, we use the notation k_x, k_y, and k_z to refer to the radix of the x, y, and z dimension in a mixed-radix network. For example, the physical layout of a 2112 node XT system can be organized as an $11\times12\times16$ torus (k_x=11, k_y=12, and k_z=16, or more concisely an 11,12,16-ary 3-cube). To

[1]The Cray XT network supports both k-ary n-mesh (*mesh*) and k-ary n-cube (*torus*) [2] configurations. Each dimension can be independently configured as a mesh or torus.

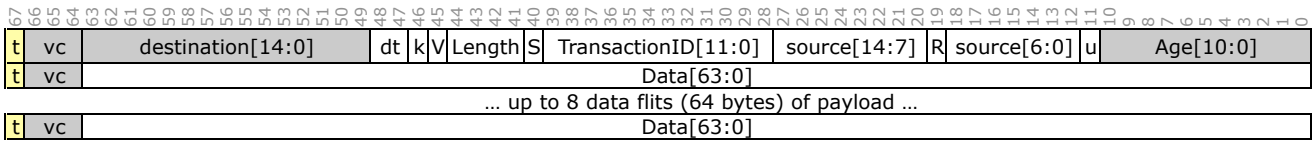


Figure 1. Seastar packet format.

keep the wrap around cable lengths short, the physical layout of the torus is *folded*.

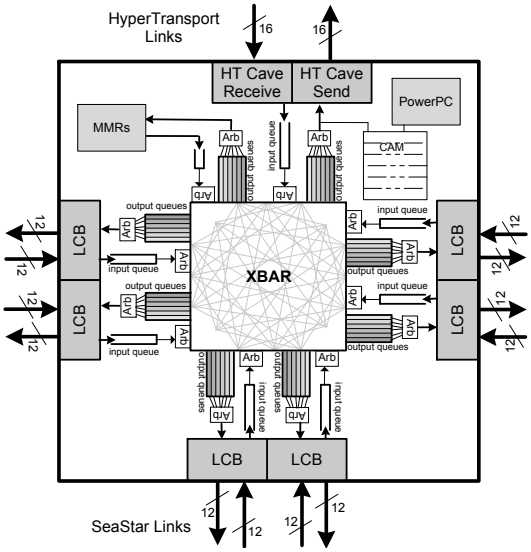
Packet routing is accomplished using a distributed lookup mechanism where each input of the Seastar router has a dedicated routing unit capable of routing a new packet every clock cycle. Routing is performed using *dimension-order* routing (DOR) for deterministic in-order packet delivery. Although DOR is deadlock-free there are several *turn* rules [4] necessary to avoid turn cycles when routing around faulty links. The network supports four virtual channels (VCs) that are segregated into *request* and *response* classes[2]. The VCs are denoted VC0/VC1 and VC2/VC3. Virtual channel dependencies around the torus links are broken using VC *datelines* [3] by routing traffic on VC0→VC1, or VC2→VC3 as it crosses a dateline node. Flow control across the network link uses *virtual cut-through* (VCT) rules [7].

1.2 Router microarchitecture

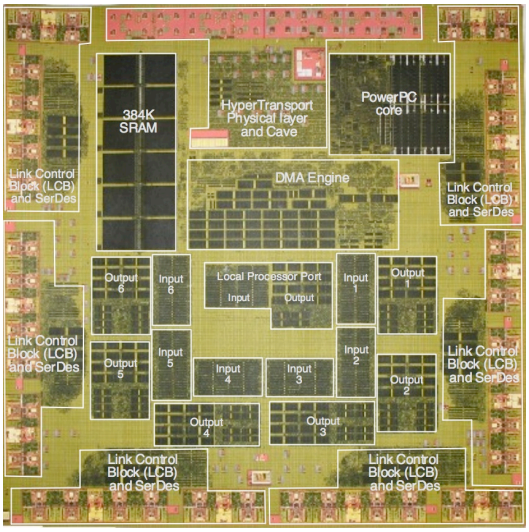
Network packets are comprised of one or more 68-bit *flits* (flow control units). The first flit of the packet (Figure 1) is the *header* flit and contains all the necessary routing fields (destination[14:0], age[10:0], vc[2:0]) as well as a tail (t) bit to mark the end of a packet. The link control block (LCB) implements a sliding window go-back-N link-layer protocol that provides reliable chip-to-chip communication over the network links. The link control block (LCB) implements a sliding window go-back-N link-layer protocol that provides reliable chip-to-chip communication over the network links. Each packet is divided into several *micropackets* which are serialized and transmitted across the network link. Each micropacket contains two flits (134 bits) and 34-bits of *sideband* which carries the sequence number of the last successfully transmitted packet and a 12-bit cyclic redundancy check (CRC) used by the receiver to ensure error-free receipt of the micropacket.

Since most Cray XT networks are on the order of several thousand nodes, the lookup table at each input port is not sized to cover the maximum 32K node network. The Seastar router uses a hierarchical routing scheme where the node name space is divided into *global* and *local* partitions. The upper three bits of the destination field (given by the destination[14:12] in the packet header) of the incoming packet are compared to the global partition of the current Seastar router. If the global partition does not match, then the packet is routed to the output port specified in the global lookup table (GLUT). The GLUT is indexed by destination[14:12] to choose one of eight global partitions. Once the packet arrives at the correct global partition, it will precisely route within a local partition of 4096 nodes given by the destination[11:0] field in the packet header.

The router has six full-duplex network ports and one processor port that interfaces with the Tx/Rx DMA engine (Figure 2). The network channels operate at 3.2 Gb/s ×12 lanes over electri-



(a) Seastar block diagram.



(b) Seastar die photo.

Figure 2. Block diagram of the Seastar system chip.

[2]Since the XT is a distributed memory machine it does not require strict request/reply segregation like a typical shared memory multiprocessor.

cal wires providing a peak of 4.8 GB/s per direction of network bandwidth. The router switch is both input-queued and output-queued. Each port has four 96-entry input buffers, one for each virtual channel. The input buffer is sized to cover the round-trip latency across the network link at 3.2 Gb/s signal rates. There are 24 staging buffers in front of each output port, one for each input source (five network ports, and one processor port), each with four VCs. The staging buffers are only 16 entries deep and are sized to cover the crossbar arbitration round-trip latency[3].

1.3 Contributions and paper organization

Although this paper describes a packet age-based arbitration mechanism used by the Cray XT network, it can be applied to all k-ary n-cubes. Our analysis applies to both *mesh* and *torus* networks, both regular and mixed-radix k-ary n-cubes, making the following contributions:

- We describe hardware-software interface of the aging algorithm and the relevant performance counters for evaluation.
- We present pseudocode for the packet aging algorithm, describe its operation on incoming and outgoing packets and manipulation of the age timestamp.
- We describe the features of the aging algorithm which support mixed-radix networks, where the radix of each dimension may differ, or networks in which each dimension is configured either as a mesh or a torus.
- We describe how to derive initial settings for the aging algorithm parameters and how to evaluate the resulting performance.
- We present the impact of age-based packet arbitration on performance of several communication-intensive benchmarks from the HPC Challenge benchmark suite [5] and representative micro-benchmarks.

The remainder of this paper is organized as follows. Section 2 describes the packet aging algorithm used for output arbitration. Then, Section 3 describes how the key parameters — *age clock period* and *age bias* are derived. Section 4 discusses the effects of age-based arbitration on performance for several communication-intensive benchmarks. Finally, we summarize our contributions and results in Section 5.

2 Age-based Packet Arbitration

We divide the packet latency into two components: *queueing* and *router* latency. The total delay (T) of a packet through the network with H hops is the sum of the queueing and router delay.

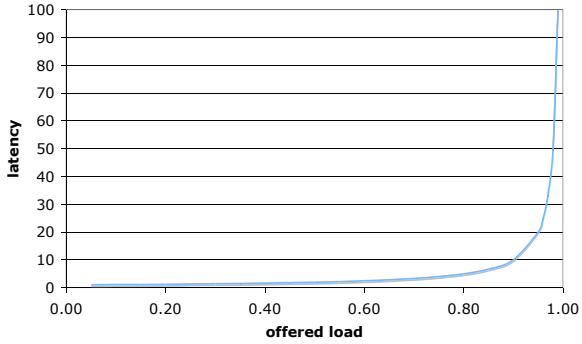
$$T = HQ(\lambda) + Ht_r \tag{1}$$

where t_r is the per-hop router delay[4]. The queueing delay, $Q(\lambda)$, is a function of the offered load (λ) and described by the latency-bandwidth characteristics of the network. An approximation of

[3]We use virtual cut-through flow control into the staging buffers, which must be at least 9 entries deep to cover the maximum packet size.

[4]For the Cray XT Seastar router $t_r \approx 50$ ns.

Figure 3. Offered load versus latency for an ideal M/D/1 queue model.



$Q(\lambda)$ is given by an M/D/1 queue model (Figure 3).

$$Q(\lambda) = \frac{1}{1 - \lambda} \tag{2}$$

When there is very low offered load on the network, the $Q(\lambda)$ delay is negligible. However, when the network is saturated the queueing delay will dominate the total packet latency.

As traffic flows through the network it merges with newly injected packets and traffic from other directions in the network (Figure 4). This merging of traffic from different sources causes packets that have further to travel (more hops) to receive geometrically less bandwidth. For example, consider the 8-ary 1-mesh in Figure 4(a) where processors P0 thru P6 are sending to P7. The switch allocates the output port by granting packets fairly among the input ports. With a round-robin packet arbitration policy, the processor closest to the destination (P6 is only one hop away) will get the most bandwidth — 1/2 of the available bandwidth. The processor two hops away, P5, will get half of the bandwidth into router node 6, for a total of 1/2×1/2 = 1/4 of the available bandwidth. That is, every two cycles router node 7 will deliver a packet from source P6, and every four cycles it will deliver a packet from source P5. A packet will merge with traffic from at most $2n$ other ports since each router has $2n$ network ports with $2n-1$ from *other* directions and one from the processor port. In the worst case, a packet traveling H hops and merging with traffic from $2n$ other input ports will have a worst-case latency of:

$$T_{worst} = \frac{L}{(2n)^H} \tag{3}$$

at the last hop, where L is the length of the message (number of packets), and n is the number of dimensions. In the example shown in Figure 4(a), P0 and P1 each receive 1/64 of the available bandwidth into node 7, a factor of 32 times less than that of P6. Reducing the variation in bandwidth is critical for application performance, particularly as applications are scaled to increasingly higher processor counts. As the network diameter increases, so does the impact of merging traffic and therefore the *variance* in packet latency. A torus is less affected than a mesh of the same radix (Figure 4a and 4b) since it has a lower diameter. Izu [6] shows this effect on throughput and average packet latency in a

k-ary n-cube, but did not provide a solution for the global unfairness.

With dimension-order routing (DOR), once a packet starts flowing on a given dimension it stays on that dimension until it reaches the ordinate of its destination. We route in x, then y, and finally z and prohibit any turns that violate this ordering.

Assuming minimal routing, the average number of hops H is expressed in terms of n and k as

$$H_{torus} = \frac{nk}{4} \textbf{ or} \quad (4)$$

$$= \sum_{i=1}^{n} \frac{k_i}{4} \text{ for mixed–radix network} \quad (5)$$

$$H_{mesh} = \frac{n(k+1)}{3} \textbf{ or} \quad (6)$$

$$= \sum_{i=1}^{n} \frac{(k_i+1)}{3} \text{ for mixed–radix network} \quad (7)$$

where k_i is the radix of dimension i (corresponding to k_x, k_y, and k_z).

To determine the appropriate *age bias* setting we must consider the *channel load*, γ_c. The channel load is the ratio of demand bandwidth to delivered bandwidth. Intuitively, it is a measure of traffic for a particular traffic pattern that traverses channel c when each input injects one packet according to the traffic pattern. As the *radix* (k) of the network grows, under a uniform traffic pattern the average channel load, γ_c, is:

$$\gamma_c = \frac{k}{8} \text{ for a torus} \quad (8)$$

$$\gamma_c = \frac{k}{4} \text{ for a mesh} \quad (9)$$

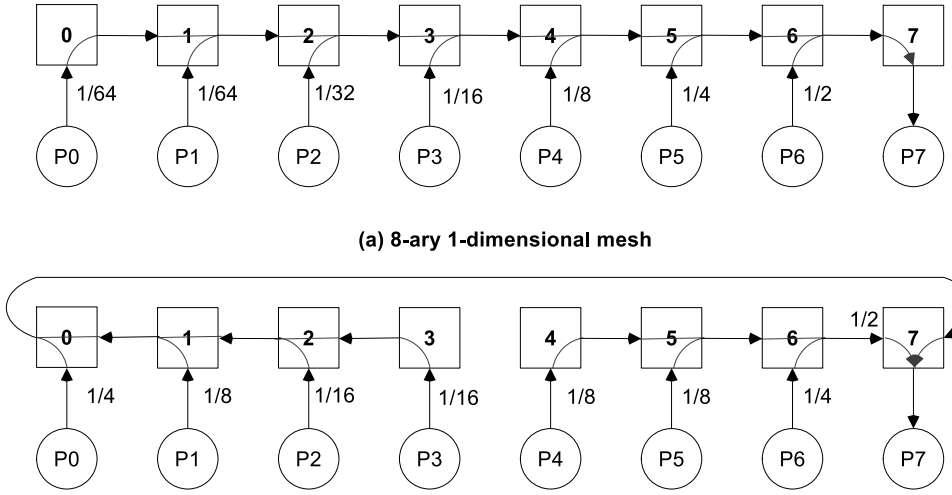
We will use the channel load as a guide to set the packet aging algorithm parameters.

2.1 Key parameters of age-based arbitration

The Seastar router provides a flexible age-based output arbitration to mitigate the effect of traffic merging, thus reducing the variation in packet delivery time. There are three key parameters for controlling the aging algorithm.

- **AGE_CLOCK_PERIOD** – a chip-wide 32-bit countdown timer that controls the *rate* at which packets age. If the age rate is too slow, it will appear as though packets are not accruing any queueing delay, their ages will not change, and all packets will appear to have the same age. On the other hand, if the age rate is too fast, packets ages will saturate very quickly — perhaps after only a few hops — at the maximum age of 255, and packets will not generally be distinguishable by age. The resolution of AGE_CLOCK_PERIOD allows anywhere from 2 nanoseconds to more than 8 seconds of queueing delay to be accrued before the age value is incremented.
- **REQ_AGE_BIAS** and **RSP_AGE_BIAS** – each hop that a packet takes increments the packet age by the REQ_AGE_BIAS if the packet arrived on VC0/VC1 or by RSP_AGE_BIAS if the packet arrived on VC2/VC3. The age *bias* fields are configurable on a per-port basis, with the default bias of 1.
- **AGE_RR_SELECT** – a 64-bit array specifying the output arbitration policy. A value of all 0s will select *round-robin* arbitration, and a value of all 1s will select *age-based* arbitration. A combination of 0s and 1s will control the ratio of round-robin to age-based. For example, a value of $0101\cdots0101$ will use half round-robin and half age-based.

When a packet arrives at the head of the input queue, it undergoes routing by indexing into the LUT with destination[11:0] to choose the target port and virtual channel. Since each input port and VC has a dedicated buffer at the output staging buffer, there is no arbitration necessary to allocate the staging buffer — only flow control. At the output port, arbitration is performed on a per-packet basis (not per flit, as wormhole routing would). Each output port



(a) 8-ary 1-dimensional mesh

(b) 8-ary 1-dimensional torus

Figure 4. All nodes are sending to P7 and merging traffic at each hop.

Table 1. Definition of bits used by the 11-bit packet age.

Field	Bits	Description
age	[7:0]	Input packet age, bits [7:0] of the header flit
carry	8	Carry out bit produced by the sum (age - timestamp)
in_epoch	9	The epoch in which this packet was received at the input
unused	10	Not used in the age calculation

is allocated by performing a 4-to-1 VC arbitration along with a 7-to-1 arbitration to select among the input ports. Each output port maintains two *independent* arbitration pointers — one for round-robin and one for age-based. We use a 6-bit counter that is incremented on each *grant* cycle and indexes into the AGE_RR_SELECT bit array to choose the per-packet arbitration policy.

2.2 Ensuring forward progress

The packet age field of the header is an 11-bit field that is constructed as shown in Table 1. Although *age* occupies an 11-bit field in the packet header (Figure 1), the age is restricted to values $0 \ldots 255$. The additional bits are required for bookkeeping in the aging algorithm. We use the notion of an *epoch* to divide the passage of time into two distinct regions corresponding to epoch values, 0 and 1. When a packet arrives it is assigned to the epoch that was in effect at the time the packet arrived. A set of chip-wide counters, *packet_count*[0] and *packet_count*[1], are maintained to keep track of the number of outstanding packets in each epoch. We use the epoch numbers and counters to determine if the packet has accumulated a substantial amount of time in the router and if we incurred a timestamp rollover. With each roll of the 8-bit timestamp, we switch epochs if and only if the next epoch has no outstanding packets (i.e. *packet_count*[next_epoch] == 0). By following this simple rule, we ensure that all packets that arrived in the previous epoch are drained before we accept packets in the new epoch. To accomplish this, we *inhibit* age-based arbitration until we have drained all the older packets using round-robin arbitration to fairly select the remaining old packets until all the packets in the previous epoch have been sent. This is described in detail by Procedure 3.

2.3 Pseudocode

The variables used to describe the aging algorithm are summarized in Table 2. The *timestamp* variable is the free-running counter that marks the passing of time, and is therefore the centerpiece of the algorithm. The valid range of the packet age is $0 \ldots 255$, with newly injected packets starting with an age of zero[5]. The general properties of the aging algorithm are:

- Promotes global fairness in the network by allowing "older" packets to get a higher priority. In some sense, age-based arbitration is a practical tradeoff between *local unfairness* and *global fairness*. This policy reduces the maximum packet latency, which is important to performance of applications with synchronized communication.
- Differentiates between *per-hop router* latency and *queueing* latency using the *age bias* and *age clock period* parameters, respectively, to control the age rate.
- Supports mixed-radix networks by allowing the age bias to be set on a per *port* basis. For instance, a mixed-radix network such as an 16,12,8 3-cube, the x dimension will have twice the channel load of the z dimension. So, it may be desirable to set the age bias on the x links to be 2, and set the z age bias to 1.
- Supports mesh-tori networks by allowing the mesh links to have a higher priority over the torus links, since the mesh links will have twice the channel load as the torus links for the same size radix.
- Uses the notion of *epochs* to ensure forward progress by avoiding starvation of "younger" packets.

[5]There is no way to inject an *urgent* packet by assigning a starting age >0 for the newly injected packets. We can, however, make the processor age bias 0 instead of the default of 1.

Table 2. Definitions of variables used in describing the packet aging mechanism.

Variable Name	Valid Values	Description
age	0..255	Packet age that corresponds to bits flit[7:0]
timestamp	0..255	Free running counter that serves to mark the passing of time
epoch	{0,1}	The current epoch number for which the timestamp refers
new_age	0..255	Intermediate age calculation
saturation	{0,1}	Flag which indicates if packet age saturates at max of 255
rollover	{0,1}	Flag which indicates the timestamp counter has rolled over
packet_count[epoch]	0..2047	A counter to represent the number of packets in a given epoch
age_clock_period_reg	$0..2^{32}-1$	The internal copy of the AGE_CLK_PERIOD register

```
 1: age ← head_flit[7:0] ; {extract the current age from the input packet}
 2: if vc=0 OR vc=1 then {if the new packet is a request packet}
 3:     age ← age + REQ_AGE_BIAS {add the request age bias from the PORT_CONFIG[63:61] MMR}
 4: else
 5:     age ← age + RSP_AGE_BIAS {add the response age bias from the PORT_CONFIG[60:58] MMR}
 6: end if
 7: if age > 255 then {does the newly computed age overflow?}
 8:     age ← 255 {saturate at 255, otherwise the packet will overflow its age range}
 9: end if
10: increment(packet_count[epoch]) {increment the packet counter for this epoch}
11: head_flit[9] ← epoch {save the epoch number in the header flit for bookkeeping}
12: head_flit[8:0] ← age - timestamp[7:0] {subtract the timestamp when packet arrived}
```

Procedure 1: Upon receipt of a new packet.

- Allows different classes of traffic (VC0/VC1 versus VC2/VC3) to age at different rates. This may be helpful if one set of VCs are used to carry control-oriented packets that are very latency sensitive. This is accomplished by setting the REQ_AGE_BIAS $\neq$ RSP_AGE_BIAS.

To best understand the aging algorithm we divide it's functionality into three procedures and give pseudocode for each.

Procedure 1 Operations at the input port

Procedure 2 Calculate age and select output packet

Procedure 3 Age clock management to ensure fairness

The remainder of Section 2 provides a detailed walk-through of the aging algorithm.

2.3.1 Aging algorithm at input ports

Procedure 1 describes the portion of the aging algorithm at the input ports. When a new packet arrives at an input port, the router must extract the *age* field from the network packet (line 1), which is located at bits *head_flit*[10:0]. Then lines 2 through 6 check the type of packet, either request or response, and add the age bias to the current packet age. The age must saturate at 255, so line 7 checks to see if the age value plus the age bias has overflowed the age range. The counter which tracks the number of outstanding packets in each epoch, *packet_count*, is incremented in line 10. The epoch in which the packet arrived is saved in bit *head_flit*[9] (line 11). Finally, in line 12, the *timestamp* value is subtracted from the current age, and the 9-bit result is saved in the *head_flit*[8:0]. Since the result of the subtraction may produce a carry bit, it must be preserved (in bit *head_flit*[8]) and accounted for when the new age is computed at the output port. We will add in the timestamp when the packet arrives at the output port and is ready for arbitration.

2.3.2 Age calculation and output arbitration

The pseudocode in Procedure 2 describes the steps for processing a packet when it arrives at the head of the output staging buffers and is a candidate for output arbitration. The output arbitration logic considers only non-blocked virtual channels (those with send_credits $\geq$ MAX_PACKET_SIZE). The arbiter inspects the packet at the head of each output staging buffer and computes its new age. The arbiter then does a comparison of the age values to determine the winner, i.e. the packet with the largest computed age value. Ties are broken using a round-robin priority scheme among the candidates.

Lines 1 and 2 initialize the rollover and saturation flags to zero, with the assumption that the age timestamp did not rollover and the new age calculation does not overflow the $0 \ldots 255$ *age* range. The epoch in which the packet arrived is extracted (line 3) from bit *head_flit*[9] — where it was saved by processing done at the

```
 1: rollover ← FALSE {begin with the initial assumption that we do NOT rollover the timestamp}
 2: saturation ← FALSE {begin with the initial assumption that we do NOT saturate the age value}
 3: in_epoch ← head_flit[9] {extract the epoch number when this packet arrived}
 4: if in_epoch ≠ epoch then {are we in a different epoch?}
 5:     rollover ← TRUE {yes, then we have overflowed the timestamp and must account for this}
 6: end if
 7: new_age[9:0] ← head_flit[8:0] + rollover, timestamp[7:0] {compute the new age}
 8: if (new_age[8] = 1) OR (new_age[9] = 1 AND head_flit[8] = 1) then {check if the new age value overflows}
 9:     saturation ← TRUE {the age value overflowed the range 0..255, so we must account for it}
10:     new_age[7:0] ← 255 {saturate at age = 255}
11: end if
12: arbitrate(age-based, new_age) {select the packet to send using age-based priority}
13: head_flit[7:0] ← new_age {stuff the new age into the outgoing packet}
14: decrement(packet_count[in_epoch]) {decrement the number of packets in the arrival epoch}
```

Procedure 2: When a packet arrives at the head of the output staging buffer and is eligible for output arbitration.

```
 1: if inhibit = TRUE then
 2:     inhibit ← FALSE
 3:     if (epoch = 1 AND packet_count[0] > 0) OR (epoch = 0 AND packet_count[1] > 0) then
 4:         inhibit ← TRUE
 5:     end if
 6: else
 7:     decrement(age_clock_period_reg) {decrement the age countdown timer}
 8:     if age_clock_period_reg = 0 then {countdown timer expired}
 9:         age_clock_period_reg ← AGE_CLK_PERIOD {reload the counter}
10:         if timestamp = 255 then {we are about to rollover the timestamp value to zero}
11:             inhibit ← FALSE {assume we are safe to switch epochs}
12:             if (epoch = 1 AND packet_count[0] > 0) OR (epoch = 0 AND packet_count[1] > 0)  then
13:                 inhibit ← TRUE {dont allow the epoch to change!}
14:             else
15:                 increment(timestamp) {adjust the timestamp}
16:                 epoch ← invert(epoch) {switch to the other epoch}
17:             end if
18:         else
19:             increment(timestamp) {timestamp < 255, okay to increment timestamp}
20:         end if
21:     end if
22: end if
```

Procedure 3: On every tick of the system clock, the router will adjust the age countdown timer as follows.

input port when the packet arrived. In line 4 a check is made to determine if the current epoch is the same as the epoch in which the packet arrived. If the current epoch does not match the epoch in which the packet arrived, then the *timestamp* must have rolled over from 255 to 0. Then, line 7 concatenates the rollover bit to the timestamp and performs a 9-bit addition to the age field, *head_flit*[8:0], in the header flit. The 10-bit sum is stored in *new_age* where it is checked against the maximum packet age, lines 8-11. Finally, the arbiter chooses the packet with the oldest age (line 12), and the adjusted age is stuffed into the packet header *head_flit*[7:0] (line 13) before transmitting the packet. As the packet is handed off to the link control block for transmission, the *packet_count*[in_epoch] is decremented, in line 14.

2.3.3 Age clock management

A certain amount of bookkeeping is necessary to manage the aging algorithm. As we described in earlier sections, the rate at which a packet will age is controlled by the AGE_CLK_PERIOD register. A write to AGE_CLK_PERIOD will cause the *internal* (not software-visible) register *age_clk_period_reg* to be updated with the contents of the software-visible AGE_CLK_PERIOD register. Once every clock tick, the router will decrement the value of the internal *age_clk_period_reg*. When it reaches zero, the router will increment the *timestamp* value, and reload the *age_clk_period_reg* counter from the value of the software-visible AGE_CLK_PERIOD MMR. Procedure 3 describes the steps required on every tick of the system clock to adjust the countdown timer and epoch.

3 Finding the appropriate aging parameters

Now that we described the aging algorithm and its properties, this section describes how to derive a set of parameters that will yield good performance. We begin our analysis with the observation that we would like to avoid ties in ages that are presented to the arbiter, since they must be broken using round-robin arbitration and will reduce the benefit of age-based arbitration. Toward this end, we would like the distribution of packet ages to be centered around the middle of the age range, 128. Ideally, these ages would be *uniformly* distributed (not normal or bi-modal) in such a way as to give the most "diversity" in the packet ages that are presented to the arbiter.

3.1 Age bias

Assuming a uniform traffic pattern, P_x, the probability that a packet is ejected from the network from the x dimension is the probability that, upon entering the network, it is not at its ordinate

Table 3. Input ports competing for each output port given dimension-order routing.

Output port	Possible input ports	# inputs
$+x$	$proc$, $-x$	2
$-x$	$proc$, $+x$	2
$+y$	$proc$, $+x$, $-x$, $-y$	4
$-y$	$proc$, $+x$, $-x$, $+y$	4
$+z$	$proc$, $+x$, $-x$, $+y$, $-y$, $-z$	6
$-z$	$proc$, $+x$, $-x$, $+y$, $-y$, $-z$	6
$proc$	$+x$, $-x$, $+y$, $-y$, $+z$, $-z$	6

in the x dimension and is at its ordinate in the y and z dimensions:

$$P_x = \left(\frac{k_x - 1}{k_x}\right)\left(\frac{1}{k_y}\right)\left(\frac{1}{k_z}\right) \quad (10)$$

The probablility that a packet is ejected from the network from the y dimension, P_y, is the probability that the packet does not originate at its ordinate in the y dimension and does originate at its ordinate in the z dimension:

$$P_y = \left(\frac{k_y - 1}{k_y}\right)\left(\frac{1}{k_z}\right) \quad (11)$$

The probablility that a packet is ejected from the network from the z dimension, P_z, is the probability that it is not ejected from the x or y dimension:

$$P_z = 1 - P_x - P_y = 1 - \left(\frac{k_x - 1}{k_x k_y k_z}\right) - \left(\frac{k_y - 1}{k_y k_z}\right) \quad (12)$$

Since the processor can accept packets from 6 ports, one per direction per dimension, the probability that a packet enters at a positive or negative output port, for each dimension i the probability that a packet exits the network via the positive port to the processor, P_{i+} is the same as the probability that it exits via the negative port, P_{i-}:

$$\forall \text{dimensions } i,\ P_{i+} = P_{i-} = \frac{P_i}{2} \quad (13)$$

We must first choose how we should *bias* the age at each hop. For a dimension-order routed (DOR) torus (with routes asserted in x, then y, and finally z), the z dimension will have the best utilization since it is the least constrained for packet egress. To simplify the analysis we assume uniform traffic, and for large-radix k-ary n-cubes, it is likely that a packet will have to traverse all dimensions[6] so this analysis represent a *bound*, rather than the average case. The ejection rate of packets from the z dimension is limited by either the bandwidth of the z dimension or the processor *ejection bandwidth*[7], E_p.

$$U_z = \min\left[B_c\left(\frac{8}{k_z}\right),\ E_p\right] \quad (14)$$

The rate at which packets are ejected from the y dimension is constrained by either the rate at which the z dimension can accept the packets, or the channel bandwidth of the y dimension.

$$U_y = \min\left[B_c\left(\frac{8}{k_y}\right),\ U_z\right] \quad (15)$$

Finally, the rate at which packets are ejected from the x dimension is limited by either the rate at which the y dimension can accept the packets, or the channel bandwidth of the x dimension.

$$U_x = \min\left[B_c\left(\frac{8}{k_x}\right),\ U_y\right] \quad (16)$$

[6]The exact probability that a packet traverses all dimensions of a 3-cube is $(1 - \frac{1}{k_x}) \times (1 - \frac{1}{k_y}) \times (1 - \frac{1}{k_z})$

[7]For Seastar the channel bandwidth, B_c = 4.8 GB/s and ejection bandwidth, $E_p \approx$ 2 GB/s.

Equations 14 - 16 assume uniform traffic[8]. With a mesh topology, the average channel load will be $\frac{k}{4}$ for the bisection links, and the average load gets smaller toward the outside of the mesh. For a mesh, substitute $4/k$ for the $8/k$ terms in Equations 14 - 16.

Intuitively, we want to minimize the variance in total traversal time per packet. Toward that end, we want packets that have more routing hops to get preference in the output port arbitration. For a dimension ordered torus with packets routing in x, then y and then z we would choose our age bias so that $\text{bias}_x > \text{bias}_y > \text{bias}_z$, since it is likely that a packet in the x dimension has more hops remaining than a packet in the y dimension, which in turn has more hops remaining than a packet in the z dimension. For the 11,12,16-ary 3-D torus from our previous example. We choose the age bias so that is satisfies the relation $\text{bias}_x > \text{bias}_y > \text{bias}_z$. Thus we chose bias_z = 1, and bias_y = 2 and bias_x = 3 as a starting point for our experiments.

3.2 Age clock period

Age clock period must be selected carefully for optimal stratification of packet ages. If it is too large, there will not be a steady supply of "old" packets. If it is too small on the other hand, too many packets will fall into the oldest bin, and the variation in their ages will be lost.

To choose the age clock period we will determine the average number of hops in the network and set the age clock so that we have packets with a diverse age value, since ties are broken fairly using round-robin which defeats the age-based arbitration when there are too many ties. To accomplish this we choose the parameters that result in a diverse set of packet ages across the $0 \ldots 255$ range of age values. The age *bias* value will be added at each hop of the network. For our 11,12,16-ary 3-cube, the average number of hops that a packet would take is 3+3+4=10 hops, from Equation 4.

$$\begin{array}{rcccccc} T_{bias} & = & H_x \times \text{bias}_x & + & H_y \times \text{bias}_y & + & H_z \times \text{bias}_z \\ & = & 3 \times 3 & + & 3 \times 2 & + & 4 \times 1 \\ & = & 19 & & & & \end{array}$$

So, on average, the age bias will contribute T_{bias}=19 to the age value. We want the distribution of age values so that they are centered about 128 - 19 = 109. Thus, the age clock must be configured so that, on average, over the 10 hops we accumulate 109 age ticks — or, 109/10 $\approx$ 11 age ticks per hop. If we assume that the network is fully utilized, there will be a total of 10 maximum-sized (9 flit) packets in the input queue, and 1 in the output staging buffer, for a total of 11 packets queued. With two virtual channels, we will move a packet from each input queue every 2×9=18 cycles. Thus, we would have (11 packets)×(18 cycles/packet) = 198 cycles of queueing delay per hop. If we want 198 cycles to represent 11 age ticks, then the *age_clock_period* must be set to 198/11 = 18.

4 Results

The Seastar performance counters (Table 4) are used to measure the effect of age-based arbitration for different work-

[8]For worst-case traffic patterns all the traffic crosses the bisection and the channel load is doubled — $\frac{k}{4}$ for torus, and $\frac{k}{2}$ for a mesh.

Table 4. Performance counter registers

Register	Bits	Description
RTR_PERF_VCv_BLOCKED	63:0	Count of the number of cycles spent *blocked* waiting for buffer resource in virtual channel v.
RTR_PERF_STALLED	63:0	Count of the number of cycles *stalled* where the router does not have sufficient virtual channel send credits to transmit the packet.
RTR_AGE_GROUP0	63:0	Histogram of packet ages. Bits [31:0] count the number of packets with $0 \leq$ age ≤ 63, and bits[63:32] count packets with $64 \leq$ age ≤ 127.
RTR_AGE_GROUP1	63:0	Histogram of packet ages. Bits [31:0] count the number of packet with $128 \leq$ age ≤ 191, and bits[63:32] count the packets with $192 \leq$ age ≤ 255.
RTR_PERF_VCv_PKTS	63:0	Count of the number of packets received on this port for virtual channel v
RTR_PERF_VCv_FLITS	63:0	Count of the number of flits received on this port for virtual channel v

loads. It is likely that some fine-tuning will be required to the *age_clock_period* value because, in practice, our assumption of uniform random traffic is not necessarily representative of real workloads. In fact, applications will likely use spatial decomposition and exploit nearest neighbor communication when possible which will offset the *age_bias* for that dimension. In general, the aging parameters are dependent on the radix of the network, as well as the topology. We experimented with several values of *age_clock_period* and measured the average packet *occupancy* as a metric for evaluation. These experiments were performed using both the default age bias of 1 for all ports as well as age biases of 3, 2, and 1 for x, y, and z. The experiments were performed on an XT3 configured as 11,12,16-ary torus. We wrote scripts to extract the performance counters from the Seastar router, and compute the following metric:

$$occupancy = \frac{T_{stall}}{P_{VC0} + P_{VC1}} \text{ cycles stalled per packet} \tag{17}$$

where T_{stall} is the number of cycles spent waiting at the head of the crossbar because we could not profitably move a packet through the crossbar, and P_{vc0} and P_{vc1} are the number of packets flowing on virtual channels VC0 and VC1, respectively. As the packet occupancy rises, the bandwidth falls and latency increases – thus lower occupancy is better. Tables 5 and 6 illustrate the benefit of age-based arbitration.

Table 5 shows the experimental results for different values of the *age_clock_period* and the *age_bias*. Our goal is to maximize throughput and minimize packet latency, which occurred when bias$_x$=3, bias$_y$=2, bias$_z$=1, and *age_clock_period* = 8. Our initial predication of an *age_clock_period*=18 was too high, but quite close to optimal. An *age_clock_period*=16 performed very close to the value of 8, having on average only 100 ns more latency.

With the default configuration (*age_clock_rate*=0x1000) we experienced very little benefit from age-based arbitration. However, as we increased age rate (by lowering the value of *age_clock_period*) from the default value to *age_clock_rate*=8 was experience a significant improvement in bandwidth and reduction in latency (Table 6) From Table 5 we see that the z dimension is able to eject a packet every ≈11.5 clocks. Thus the channel utilization of the z dimension is 9 flits / (9 + 11.5) = 0.44. We observe that the 0.44×4.8 GB/s = 2.1 GB/s which is approximately the processor ejection bandwidth, E_p, as Equation 14 predicted.

Perhaps more importantly, from Table 5 we see the variance of the packet occupancy coming down as well. This will improve the average packet latency in the network, as well as tighten the packet latency variance. This has important ramifications for performance in applications containing synchronized communication. If all processors are in a barrier, for example, performance is optimized by minimizing the maximum time for a processor to reach the barrier. Indeed, we measure an average reduction in latency of 2.3μs, approximately 31% iimprovement. This result was obtained on a machine running a mix of production jobs over a period of several days with the default age clock and age bias settings, and then again with age_clock_period set to 8 and bias_x to 3, bias_y set to 2, and bias_z set to 1.

In addition to examining packet statistics, we quantify the impact of age-based versus round-robin packet arbitration policies on several benchmarks: MPI-FFTE from the HPC Challenge

Table 5. Age-based arbitration results for different values age_clock_period.

Age clock period	Age bias			Average number of stalled cycles						Queueing delay			Average latency (ns)
	x	y	z	$+y$	$+x$	$+z$	$-z$	$-x$	$-y$	Q(x)	Q(y)	Q(z)	
round robin				18.6	43.0	13.2	12.7	43.1	19.2	737	227	169	7301
4	1	1	1	20.0	36.9	11.7	11.2	37.0	19.9	607	246	153	6585
4	3	2	1	17.4	31.4	11.7	11.5	31.6	17.8	421	247	189	5844
8	1	1	1	21.4	44.1	13.2	13.0	44.1	20.5	611	255	167	6771
8	**3**	**2**	**1**	15.8	24.6	11.6	11.4	24.7	15.7	357	209	156	**5007**
16	1	1	1	24.7	41.2	14.5	14.3	41.5	24.6	554	322	240	7443
16	3	2	1	17.2	30.5	10.9	10.8	30.6	17.1	378	217	148	5101

Table 6. Benchmarks with age-based and round-robin packet arbitration on 2048 processors

Arbitration policy	MPI-FFTE Gflops	MPI_Alltoall MB/s	MPI_Allreduce μs
Age-based	507	194	532
Round-robin	451	142	607
Improvement of age-based over round-robin	12.4%	36.6%	12.4%

benchmark, MPI_Alltoall, and MPI_Allreduce. These benchmarks were selected to be representative of the communication patterns found in communication-intensive production codes. The results, shown in Table 6, are compelling: Performance of MPI-FFTE and MPI_Allreduce improve by 12.4%, and performance of MPI_Alltoall improves by 36.3%.

The *age histogram* registers (Table 4) are critical tools for evaluating if age-based arbitration is performing as desired. Figure 5 shows the distribution of packets for three different settings of the *age_clock_period*. Even with values of 8 and 4, there were disproportionately more packets in the 0-63 age bucket. The distribution of packets differed by only one order of magnitude in Figure 5(b) and (c), whereas it differed by a factor of five orders of magnitude in Figure 5(a) (i.e. the 0-63 age bucket had 1×10^5 more than the 64-127 age bucket).

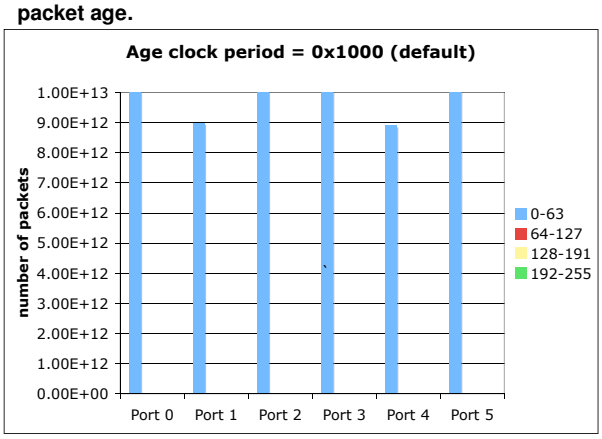
5 Conclusion

Age-based packet arbitration can be highly effective in mitigating the effects of merging traffic in large-radix networks and therefore reducing the variance in packet transit time. In this paper, we describe the age-based packet arbitration scheme used by the Cray XT and present results, both in terms of Seastar performance counters and representative benchmarks, demonstrating the positive effects of age-based packet arbitration.

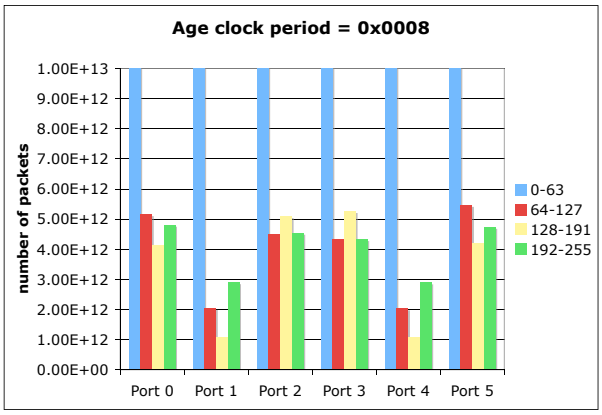
In doing so, we describe how to derive the key parameters of the aging algorithm *bias* and *age_clock_period*. We present a metric for evaluating the effectiveness of aging (occupancy, as described in Equation 17). We demonstrate how to examine packet age distribution using the age *histogram* performance counters.

As applications scale to increasingly high processor counts, minimizing the variance in packet delivery time becomes ever more important to performance. The methods presented in this paper are shown to be highly effective in mitigating the effects of merging traffic in large-radix networks, the effects of which are clearly represented in relevant, realistic benchmarks.

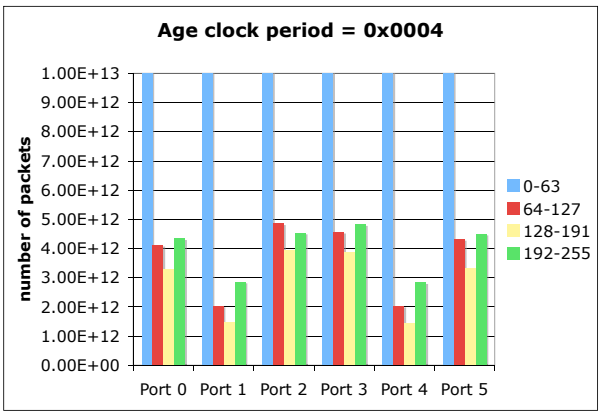
Figure 5. Age histograms showing the distribution of packet age.



(a) default values for age-based arbitration



(b) more uniform distribution with *age_clock_period*=8



(c) about the same distribution as with (b) but with *age_clock_period*=4

References

[1] Cray XT3. http://www.cray.com/products/xt3/.

[2] W. J. Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.

[3] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5):547–553, 1987.

[4] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *ISCA '92: Proceedings of the 19th annual international symposium on Computer architecture*, pages 278–287, 1992.

[5] HPCC Challenge Benchmarks. http://icl.cs.utk.edu/hpcc/.

[6] C. Izu. Throughput fairness in k-ary n-cube networks. In *ACSC '06: Proceedings of the 29th Australasian Computer Science Conference*, pages 137–145, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[7] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.