Incorporating DMA into QoS Policies for Maximum Performance in Shared Memory Systems

Scott Marshall and Stephen Twigg

University of California, Berkeley

Abstract

Modern multiprocessor systems are liable to generate large amounts of communication traffic, potentially bringing the processor-memory interconnection network to near-saturation. As network utilization is pushed closer to these limits, quality of service (QoS) in the network fabric is requisite to ensure individual messages are transferred in a timely manner and nodes are unable to indirectly starve other nodes of network resources. Previous work with network interconnect QoS has failed to provide a desired ability to provide differentiated service to nodes while maximizing network utilization. Also, previous methods fail to consider the impact of given QoS policies in a system allowing for direct memory access (DMA) operations and systems presenting shared memory via hardware-enforced cache coherence protocols. We propose a new QoS policy that extends the age-based routing method by adding admission control objects. These gate objects throttle specific requests from nodes into the network to prevent over-saturation and ensure bandwidth allotments are not violated. We tested the policy using real benchmarks in the gem5 simulator and by using synthetic, high traffic, benchmarks in a network simulator. We found that our policy is capable of providing appropriate fair bandwidth while still exposing enough parameters to fine tune guarantees and priorities among different nodes. Finally, we found that DMA operations require special consideration to compensate for the generation of significant congestion around DMA controllers.

1 Introduction

Computer technology is moving into an era in which clock speeds no longer rise, and instead, the number of CPU sockets and number of CPU cores is ever increasing. These changes bring rise to a new challenge: effectively and fairly utilizing resources in these large-scale shared-memory systems. The best way to overcome this challenge is through quality-of-service (QoS) policies applied to the processor-memory interconnect network, as this layer has complete control over resource distribution in these shared-memory systems.

Previous work [9, 8] has attempted to address the need for a QoS policy for a shared-memory interconnect. However, these solutions have suffered from several limitations and inefficiencies and we show our work solves these problems.

Further complicating shared-memory systems is the increasing need for high-performance I/O. With the growth of fast storage through solid-state-disks (SSD) and fast networking via 10 Gigabit Ethernet, systems need fast and efficient ways of moving data between shared memory and these external sources. Direct Memory Access (DMA), wherein a bulk I/O operation can occur without ongoing handling by the CPU has been, and will continue to be, an effective way to move this type of data quickly and efficiently.

However, little work has addressed the need for a QoS policy which is DMA-aware. Thus, we propose a new QoS design that leverages QoS Gate objects to control and manage network interconnect congestion by using an admission-control strategy that prevents requests from entering the network. We have tested our QoS Gate design with several different QoS policies and have found it is extremely effective. Further, we believe the clean design leaves room for flexibility for future needs and improvements.

In § 2, we introduce related work, while in § 3 we motivate our work by describing the problem we are solving in greater detail. We present our solution in § 4, describe our memory simulator in § 5, and evaluate our design with a collection of benchmarks in § 6. Finally, we discuss future work in § 7 and conclude in § 8.

2 Related Work

Quality of service methods, as previously researched, can be broadly categorized as static-based, age-based, or deadline-based. Static schemes include round-robin scheduling wherein the system temporarily grants priority to specific nodes for small blocks of time. Differentiated service is achieved by adjusting the relative lengths of time slots allotted to each node. Grot notes these systems can easily guarantee bandwidth to nodes but cannot readily maximize network utilization by recovering unused bandwidth [8]. Thus, the system is highly contingent on applications following the network traffic patterns assumed when calculating the static priority schedule. Age-based routing schemes stamp each packet with their time of issue. Routers then prioritize handling packets with the lowest timestamp. This scheme, while readily able to maximize available network bandwidth, does not immediately provide mechanisms for granting differing bandwidth guarantees to individual nodes. Deadline-based routing extends the age-based method, instead posting each packet with a desired deadline and routing according to closest deadline. Differentiated service is achieved by regulating the rate at which individual nodes are required to increase posted deadlines.

Previous work has considered DMA as an explicit traffic type [6]. In that work, the system segregated all traffic type into four broad classes: signaling, real-time, RD/WR, and block-transfer (including DMA traffic). However, the method simply considered systems that prioritized all block-transfer traffic below the other three classes.

2.1 Globally Synchronized Frames

Globally-Synchronized Frames is a loose extension of deadline-based routing [9]. Rather than post deadlines based on the system clock to packets, nodes instead assign packets to coarse routing frames. Each frame is granted a subset of available buffers on all network routers to ensure each frame is guaranteed to be drainable. Like other deadline-based schemes, the earliest frame holds routing priority over all other frames. Each node is limited as to the number of packets it may post with a specific frame, allowing for differentiated service by varying the limits by node. Nodes that exhaust their allocation for a frame are able to post new packets, but must do so to future frames. The variable, but controlled length of time a frame can last is what allows this system to maximize network bandwidth. Frames with few packets will finish and retire quickly, allowing for more frames to be serviced. Grot notes a critical issue with GSF is the requirement for global retirement of frames [8]. Severe congestion at one section of the network (e.g. due to a hotspot) could tie up retirement of multiple frames while another section sits idle waiting for frames to retire and grant buffer space.

2.2 Preemptive Virtual Clock

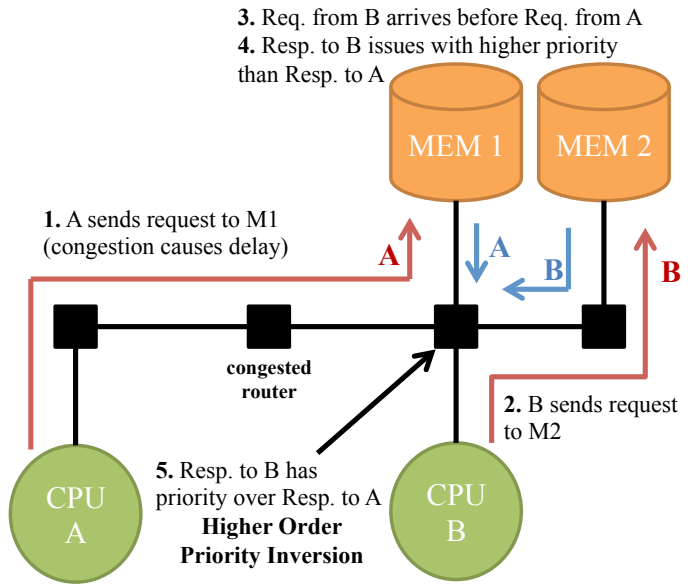
Preemptive Virtual Clock also extends deadline-based routing [8]. Each node holds a virtual clock which determines the deadline to be assigned to issued packets. The clock increments naturally in time or according to the size of packets the node issues onto the network. Occasionally, to prevent nodes consuming large amounts of bandwidth from falling far behind in priority, the clocks are occasionally reset. This setup alone does not prevent priority inversion where low priority packets clog up buffers preventing high priority packets from routing. To fix this, the routers NACK those low priority packets to make way for higher priority packets. While this system allows for differentiated service to nodes and maximizes network utilization, the cost of allowing NACKing is significant. Packet sources are required to save sent packets in a buffer until acknowledgment packet transmission succeeded is received. Also, all bandwidth spent partially routing the packet is effectively wasted as well as extra bandwidth used to handle the ACK and NACK packets.

3 Motivation

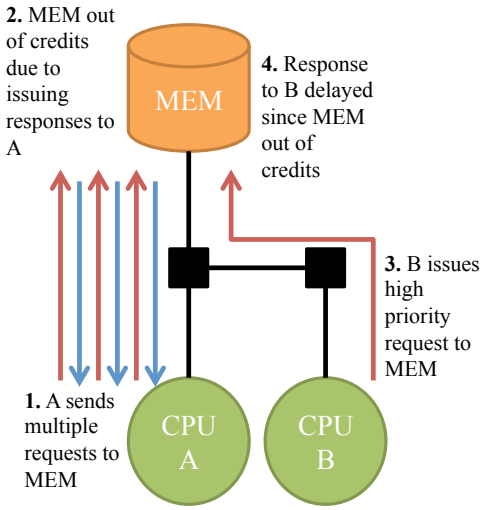
The desire to guarantee processors a portion of the system memory bandwidth is well-understood. For parallel programs, appropriately defining and maintaining these guarantees can help keep nodes coarsely synchronized, even between explicit synchronization. Assuming each node must complete some set of memory accesses during the block of time between synchronizations, the system works best if each node progresses at approximately the same rate. Any nodes receiving undue priority for memory at a high rate will still be blocked at a synchronization barrier until the starved nodes are capable of completing their accesses and consequent calculation.

3.1 Problem Overview

As emphasized by Lee, memory bandwidth guarantees are worthless without corresponding network access guarantees [9]. The node cannot utilize its reserved bandwidth if messages from the node to memory are perpetually stalled on the network. Work on operating systems for Tessellation OS has exposed the desire to provide differing guarantees to nodes in the system dependent on their current application and position in the system [7]. QoS is critical to ensure varying applications



(a) Higher Order Priority Inversion: congestion between CPU A and MEM 1 due to other traffic (not shown) causes higher order priority inversion where Resp. B gains priority even though Req. A was issued first.



(b) Inappropriate Credit Assignment: requiring nodes to pay for response packets leads to priority issues. Even though CPU B has guaranteed bandwidth with MEM, MEM cannot afford to send a response to B.

Figure 1: Illustration of current problems with QoS policies when applied to processor-memory interconnects.

and services in the systems are able to reliably intercommunicate while maintaining enough isolation such that misbehavior by one program does not impede the performance of others.

Previous research into network QoS has considered schemes which apply QoS to singular packets, likely in a message-driven environment. However, shared memory schemes employing cache coherence protocols are not properly served by those methods. Current schemes can only guarantee the request message from the node to a memory controller is serviced with the appropriate priority. However, what the node truly cares about is the time until it receives its response (latency). However, under current schemes, the response message from the controller to the memory is prioritized according to the QoS agreement to the controller. This can lead to two related issues: higher order priority inversion and inappropriate credit assessment.

3.2 Higher Order Priority Inversion

Figure 1(a) demonstrates the potential for higher order priority inversion caused when the QoS system considers only singular packets in isolation. Assuming A and B are of equal priority and issuing requests within their allotted guarantees, the QoS system would ideally work to ensure total request-to-response latency is balanced (with necessary adjustments due to differing path lengths). However, in the detailed scenario, an earlier request from node A to memory 1 is delayed such that it is received after memory 2 receives its request from node B. Thus, the response to node B will most likely have higher priority (under extensions of age/deadline-based schemes) than the response to node A. This will further delay the response to A and thus constitutes a higher order priority inversion since it arises only when the messages are considered as components of the overall cache coherence protocol. This example only demonstrates the case where a response can be immediately generated; however, a full system may require cache invalidates sent to other nodes thus possibly amplifying the problem.

3.3 Inappropriate Credit Assessment

Another issue when implementing a cache coherence protocol on a network with QoS is determining who should pay for response packets. Figure 1(b) demonstrates this issue. Node A first issues many requests to memory, draining its network allotment. Node B then sends its only request to the memory. However, the response is delayed since the memory because it is out of network credits. In general, it is inappropriate to charge system agents for issuing responses. However, as demonstrated by the higher order priority inversion issue, the response packets must be handled with varying priority and therefore cannot escape consideration by QoS

enforcers.

A logical correction to both of these issues is to avoid charging system agents for issuing response packets. The response packet would be granted the same priority (e.g. age, deadline) as the corresponding request packet. However, certain schemes such as QoS are ill-equipped to handle this due to the potential delay in generating the response from the request. For example, in QoS, frame retirement would possibly be held up as DRAM is accessed, caches are invalidated, which could easily take longer than the entire routing time. This motivates the development and analysis of a QoS system that assumes the network is being used to implement some multi-step cache coherence protocol.

3.4 Incorporating DMA

System I/O commonly requires DMA to interface the I/O controllers with the rest of the system. Disk and controllers specifically use DMA to handle the transferring of data between memory and disk when paging [10]. Similarly, high-throughput Ethernet controllers, used when connecting multiprocessor systems to a broader computer network such as the Internet, use DMA to interface with the processor [2]. These transactions are generally distinct from normal memory transactions since they are bursty, accessing multiple contiguous lines of memory over a relatively short period of time. Thus, high bandwidth to memory is required to service a DMA request to avoid needing to buffer up portions of the request. Similarly, the memory responses are liable to cause localized congestions as the large packets route through the system. This scenario motivates consideration of DMA operations when evaluating the QoS policy.

4 Proposed Solution

Given the identified limitations of existing QoS policies, we propose a new processor-memory interconnect network QoS design that not only remedies deficiencies of existing designs but also addresses the need for a DMA-aware strategy. In subsequent sections, we address the need for and strengths of age-based routing, highlight the core component of our design: the QoS Gate, and discuss attributes of several potential QoS policies that the QoS Gate can enforce.

4.1 Age-Based Routing

Previous work [9, 8] demonstrated methods based on age- and deadline-based routing provided fair routing of packets around the network. Our solution draws upon these results and employs age-based routing as a simple method to equalize network priority. In order to prevent issues at the protocol level, response packets are always stamped with the same age as corresponding request packets. More advanced deadline-based schemes such as GSF or PVC cannot safely handle this forced setting of deadlines on new packets and thus are not viable candidates for this solution.

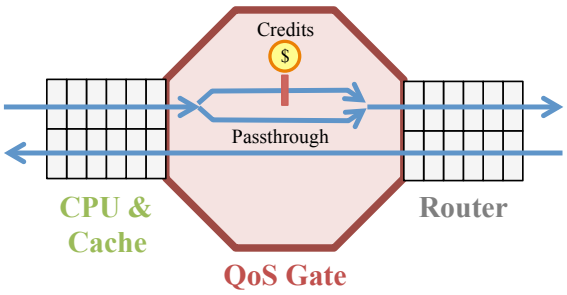


Figure 2: High-level diagram of QoS Gate design. All traffic destined for the CPU along with select traffic from the CPU passes through without credit assessment. Other traffic destined for the network must satisfy the credit check before being permitted to enter the network.

4.2 QoS Gate

The key component of our solution is the QoS Gate, a component in the processor-memory interconnect network that is located inline between each processor's cache and associated switch. Since the QoS Gate is connected inline, it has complete control over all network communication: traffic on all virtual networks flowing both directions. The component is called a *gate* because it works by restricting which types of traffic are allowed to pass through it. A diagram of the Qos Gate is presented as Figure 2. A discussion of these restrictions and how they are applied follows.

The QoS Gate uses the notion of *credits* to manage traffic. Upon initialization, each QoS agent starts with an allowance of credits. As packets flow through the QoS Gate, it deducts credits from its allowance in accordance with the cost of the packets. Finally, each cycle, the credit allowance at each QoS Agent is replenished. If a network packet passing through a QoS Gate costs more than credits the QoS Gate has available, then the QoS Gate rejects the packet - it is queued until the sufficient credits are available. Policies which dictate the initial credit allowance, the cost of each packet type, and the credit replenish rate are independent of the operation of the QoS Gate as described. Discussion of various QoS policy aspects follows.

As we previously indicated in Section 3.3, existing

QoS schemes improperly assess credits, in that endpoints are charged for network traffic they generate in response to a request. Thus, in our design, credit assessment is only performed on traffic entering the network on the request virtual network; credit assessment is not done on response packets entering the network nor the arrival of any packet at its destination.

We previously motivated the need for a DMA-aware QoS policy (Section 3.4). As such, the QoS Gate has two allowances of credits (i.e., one for normal memory requests and one for DMA requests) and assesses credits for requests from a processor's cache to a DMA controller separately from requests from a processor's cache to a memory controller.

Our current design charges a fixed cost for all request types, independent of the type (i.e., while normal memory requests and DMA requests use separate allowances, both packet types are charged the same cost), size (e.g., total data size requested via DMA), and distance (e.g., number of network hops between the source and destination node). A more sophisticated credit assessment algorithm is saved for future work (Section 7).

4.3 QoS Policy Tradeoffs

With the advent of many-core systems, the computer architecture and operating systems research communities have expressed increasing interest in sharing these large-resource systems in a variety of ways. While our primary focus in this work was to establish a QoS policy for resource fairness (in the form of equality of resource utilization), recent literature, such as Tessellation [7], has shown that resource partitioning also holds merit.

Thus, we acknowledge that there will be tradeoffs in defining any QoS policy. For example, in this work, where we emphasize resource fairness and equality, we acknowledge we may sacrifice utilization. Similarly, in a policy that emphasizes differentiated service and minimum guarantees, we acknowledge we may sacrifice fairness.

Therefore, from one perspective, the goal of our work is to show that shared memory systems need a QoS policy that is DMA-aware rather than establishing a claim regarding which policy is the "best". Afterall, different use cases and environments will dictate different needs, and thus, will dictate which QoS policy is most effective. In subsequent sections, we show how our QoS Gate solution can satisfy the needs of several different QoS policies.

4.4 QoS Policy: Fairness

We focused our attention on designing a QoS policy that enforces fairness, e.g., one in which all nodes are able to issue equal numbers of requests. This policy dictates several key aspects of the QoS Gate configuration:

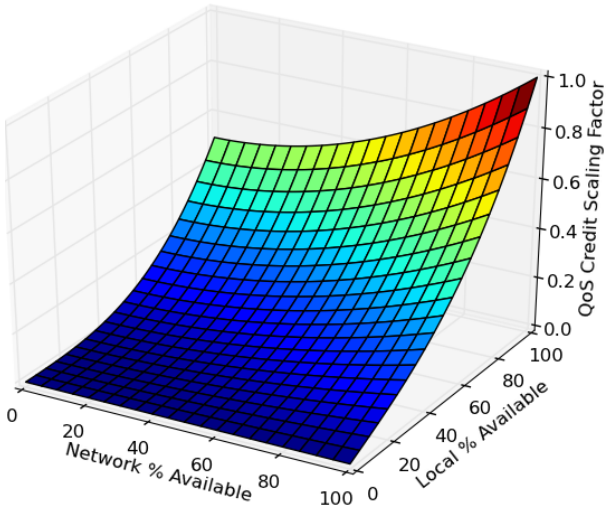


Figure 3: QoS Credit Scaling Factor (QCSF): traffic is never permitted if the local switch is fully saturated, independent of overall network congestion.

- All QoS Gates are initialized with the same numbers of credits.
- The same number of initial credits is given to both the allowance for normal memory requests and that of DMA requests.
- Normal memory requests and DMA requests are of equal cost.
- Credit allowances for normal memory requests and DMA requests are replenished at equal rates, but not necessarily the same across all QoS Gates.

We now address the one aspect of the policy not covered by the criteria above: credit replenishment. To mange credit replenishment, we define a maximum number of credits that can be replenished each cycle, and scale that maximum value by a *QoS Credit Scaling Factor (QCSF)*. In designing a credit replenishment policy, we set out to design an algorithm that would dynamically adjust based on current network congestion levels, and that would have some sense of network locality. To accomplish these goals, our replenishment policy combines knowledge of utilization[1] of the local switch and network utilization[2]

In considering the many ways we can combine the local and network-wide utilization metrics, we treat the

[1]The ratio of utilized bandwidth over total throughput.
[2]Arithmetic mean of all switch utilizations.

two quantities equally, with the exception of never permitting traffic into the network if the local switch is completely congested. Further, we use a conservative policy, wherein the QoS Gate quickly restricts traffic flow should any congestion (anywhere) occur. As a result, the mathematical expression for the QCSF is not linear: $QCSF = local\_avail * (local\_avail^2 + net\_avail^2)$. Figure 3 represents this expression visually.

4.5 QoS Policy: Differentiated Service & Minimum Guarantees

As we previously noted, there are potential scenarios in which different qualities of service are preferred over the fairness model we just introduced. While analyzing a wide collection of QoS Policies was not a goal of this work, we do show that the QoS Gate design does support a differentiated service QoS policy model, wherein selected network nodes are, in effect, given minimum bandwidth guarantees.

The differentiated service QoS policy is achieved through a modification to the QoS credit replenishment algorithm. We augment the QCSF expression to include the sum of a fixed constant which we call the *base*. The QCSF expression for differentiated service is as follows: $QCSF = base + local\_avail * (local\_avail^2 + net\_avail^2)$

4.6 QoS Policy: Partitioning and Grouping

A third potential QoS policy, motivated by the Tessellation [7] work, is the need to group and partition sets of processors in a many-core system. We believe the QoS Gate design is supportive of these policies, though exploration of this area is outside the scope of our current work. Investigation of QoS policies that leverage partitioning and grouping is saved for future work (Section 7).

5 *PyMemSim* Memory Simulator

In order to properly test the developed QoS scheme when the network is placed under high stress, a memory and network simulator, *PyMemSim* was created in Python. The simulator implements a realistic and simple cache coherence protocol. Regular nodes have caches and randomly generate multiple read and write requests to memory controllers. Regular nodes also generate read and write requests to DMA controllers targeted at random memory locations (thus roughly emulating disk I/O).

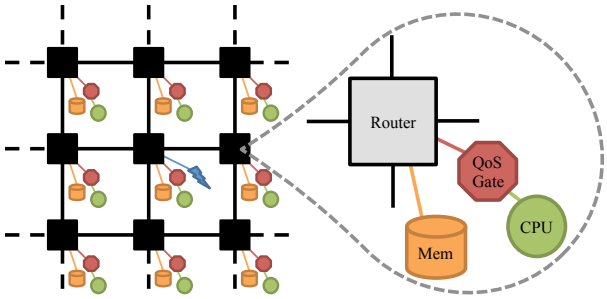


Figure 4: Diagram of sample PyMemSim network: grey/black squares are combination switches and routers, orange cylinders are memory, green circles are combination CPU and caches, red octagons are QoS Gates, and blue lightning bolts are DMA controllers.

5.1 Network Interconnect

The simulated network is arranged as a mesh with each point on the mesh consisting of a router, node, memory controller, and (rarely) a DMA controller. Figure 4 illustrates a portion of this network. Packets are routed on the network via wormhole routing using deterministic dimension-order routing. To prevent deadlock, switches contain a separate input buffers for each possible input as well as a set of input buffers for each virtual network. Packets are routed first by virtual network and then by stated packet age, assuming buffer space at the target and link bandwidth is available. To simplify implementation, it was assumed links between nodes are bidirectional and always reserve half their bandwidth for each direction. This network configuration is provably deadlock free assuming nodes can consume data.

5.2 Coherence Protocol

Figure 4 summarizes the cache side of the MSI cache coherence protocol implemented in the memory simulator. The protocol was tailored to be very simple, minimizing types of messages needed, thus the exclusion of the `E` state and direct `M->S` transition. Each `REQ_READ` and `REQ_EXCL` request is serviced from the cache controller by `REP_READ` or `REP_EXCL` responses, respectively. The memory controller employs an (unlimited) directory with transaction buffers (allowing only one pending transaction per line). The directory is kept accurate by requiring nodes to send `REQ_DONE` to memory in order to close out transactions. The memory controller sends `REQ_INVL` to clear out the directory as needed (to handle). Caches respond with `ACK_INVL` and are also permitted to issue `ACK_INVL` other times at will (e.g. when it must clear out cache space).

Nodes issue requests to a DMA controller using `DMA_READ` and `DMA_WRITE` messages, which specify both

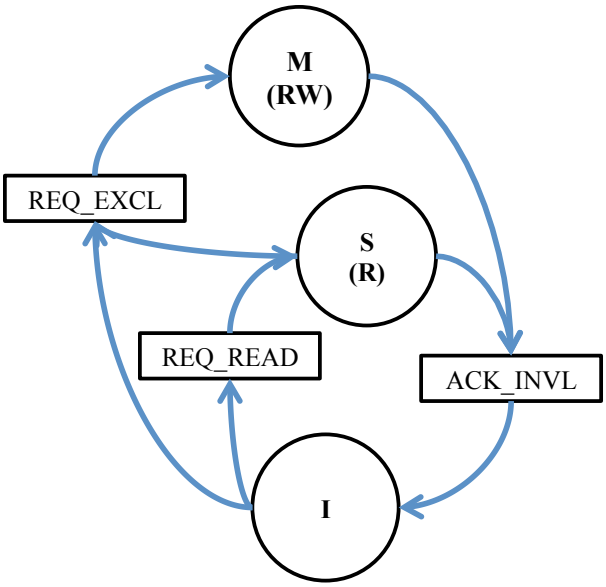


Figure 5: The simulator employs a relatively-simple MSI cache coherence protocol, which allows simulating both cached reads and writes in a shared memory environment.

Packet Types			
Priority	**Name**	**Src**	**Dst**
0	*DMA_READ*	Node	DMACtrl
	DMA_WRITE	Node	DMACtrl
1	REQ_UREAD	DMACtrl	Mem
	REQ_UWRITE	DMACtrl	Mem
	REQ_READ	Node	Mem
	REQ_EXCL	Node	Mem
2	REQ_INVL	Mem	Node
3	REP_READ	Mem	Node
	REP_EXCL	Mem	Node
	REP_UREAD	Mem	Node
	REP_UWRITE	Mem	Node
	REQ_DONE	Node	Mem
	ACK_INVL	Node	Mem
	DMA_DONE	DMACtrl	Node

Table 1: Packet types and associated priority levels. Italicized entries have QoS policies applied.

starting line and number of lines to touch for the request. The DMA controller issues REQ_UREAD and REQ_UWRITE messages in order to read and write lines for the request. Responses to these requests (REP_UREAD, REP_UWRITE) are considered Uncached and thus the DMA controller is not added to the directory. However, as expected, the uncached write operation still requires the directory be used to invalidate all caches. The DMA controller issues a DMA_DONE message to signal the request is done. To prevent deadlock, four priority levels are needed in the system. Assignment of packets to priority level is shown in the Table 1.

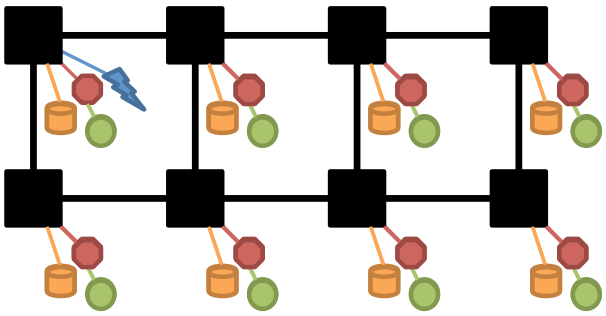


Figure 6: Diagram of interconnection network topology used during gem5 benchmarks. See caption of Figure 4 for identification of network components.

6 Evaluation

We evaluated our proposed QoS Gate design and QoS policies in two environments: the gem5 [4] full-system simulator and our PyMemSim (Section 5) memory simulator.

6.1 gem5

We initially planned on solely using gem5 for implementation and evaluation of our QoS Gate design. Unfortunately, due to stability and performance problems that we repeatedly encountered while using gem5, we elected to shift a majority of our evaluation focus away from gem5 towards PyMemSim.

6.1.1 Simulation Environment

For our gem5 benchmarks, we configured gem5 to use the associated ruby memory environment with a packet-switched network. Ideally, we would have used "garnet", the wormhole-routed version of ruby, but we found it was extremely sluggish and susceptible to frequent deadlock.

We made use of gem5 full-system simulation mode, with a total of 8 x86_64 CPUs, 8 directories, and 1 DMA controller in a 2-row mesh network configuration, as illustrated in Figure 6. The simulation environment was a complete Linux 2.6.22.9 installation provided by the gem5 development team. To conduct DMA benchmarks, we used version 9.39 of the `hdparm` [1] utility, which provides measurements of cached and non-cached disk reads, with or without DMA. To conduct system-wide performance benchmarks, we used the `blackscholes` component of the PARSEC [5, 3] family of benchmarks.

QoS Enabled	Measured Throughput (MB/s)
No	235.52
Yes	233.99

Table 2: `hdparm` DMA cached disk throughput.

Test Case	Measured Runtime (s)
No QoS, No `hdparm`	0.584
Yes QoS, No `hdparm`	0.596
Yes QoS, Yes `hdparm`	0.592

Table 3: `blackscholes` run times.

Parameter	Value
Packet Size	2-10 flits
Bisection Bandwidth	160 flits/cycle
Router Input Queue Depth	8 packets
Max Pending Memory Reqs.	8 requests
Max Pending DMA Reqs.	1 requests
Initial Memory Credits	10 credits
Initial DMA Credits	10 credits
Max Memory Credit Replenish	0.4 credits/cycle
Max DMA Credit Replenish	0.4 credits/cycle
Simulation Runtime	10,000 cycles

Table 4: PyMemSim benchmark parameters.

6.1.2 Fairness

To evaluate the fairness under our proposed QoS policy, we implemented our QoS Gate design into the gem5 memory system. For performance reasons, switch and network utilization metrics are averages over 1,000,000-cycle periods, rather than being updated every cycle.

To validate our fairness claim, we measured the performance of `hdparm` and `blackscholes` individually and together, with and without our QoS Gate in place.

Unfortunately, our gem5 results are inconclusive. In fact, of the measurements we were able to obtain, all configurations (i.e., with and without QoS, `hdparm` alone, `blacksholes` alone, and `hdparm` combined with `blackscholes`) had near-identical results, with variations within the noise. We present our `hdparm` results in in Table 2 and our `blackscholes` results in Table 3.

6.2 PyMemSim

Given the unexpected difficulties we encountered when working with gem5, we performed a majority of our evaluation using our memory simulator, PyMemSim (Section 5). While we acknowledge that this approach strictly uses synthetic benchmarks, rather than real-world application benchmarks, it is still a valid approach for validating our work.

6.2.1 Simulation Configuration

Unless otherwise noted, all benchmarks performed with PyMemSim used a 8x8 mesh network with a single DMA controller attached to the switch at coordinate (3,3) similar to the illustration in Figure 4. Table 4 lists relevant parameters used during the benchmark process.

6.2.2 Fairness

To confirm fairness, we record the number of normal (non-DMA) memory operations that complete during a single run of the memory simulator benchmark. However, note that while we show normal memory operation metrics here, DMA operations are occuring in the simulator simultaneously, with the sole DMA controller located at (3, 3). Our intention is that the combination of age-based routing and our fairness-focused QoS policy should yield an interconnection network in which all nodes are able to issue roughly the same number of requests.

Results of these measurements are show in Figure 7. In comparing Figure 7(a) with Figure 7(b), we see that the baseline is extremely unfair, while age-based routing is quite fair. Comparing Figure 7(b) with Figure 7(c) reveals that in attempting to achieve fairness, the QoS policy reduces traffic surrounding the congested network near the DMA controller at (3, 3). This is consistent with the expected behavior of the policy.

We perform similar analysis with DMA requests. Figure 8 compares the DMA request distribution for the baseline and the fairness QoS policy cases. We see the policy is rather effective, as even though the maximum number of completed requests for most nodes is lower, the distribution is much fairer.

6.2.3 Differentiated Service / Minimum Guarantees

Similar to our analysis for fairness, we wish to evaluate our proposed policy for support of differentiated services. For this benchmark, we indicated that requests from nodes (2, 5) and (7, 7) should be given priority (i.e., guaranteed bandwidth) over requests from other nodes. This benchmark result is plotted in Figure 7(d). Comparing this result to that of the QoS fairness policy in Figure 7(c), it is easy to see that nodes (2, 5) and (7, 7) do in fact complete significantly more requests than other nodes in the mesh.

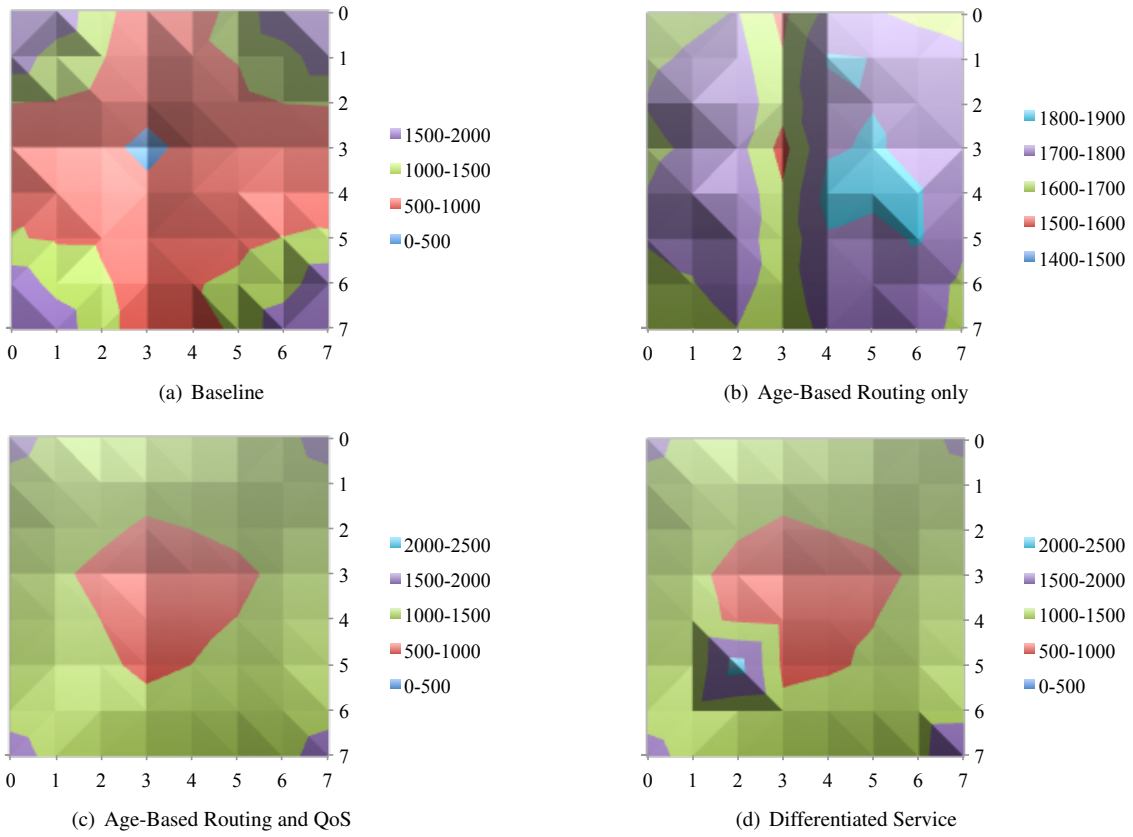


Figure 7: Contour plots showing the distribution of completed memory requests across the 8x8 mesh network.

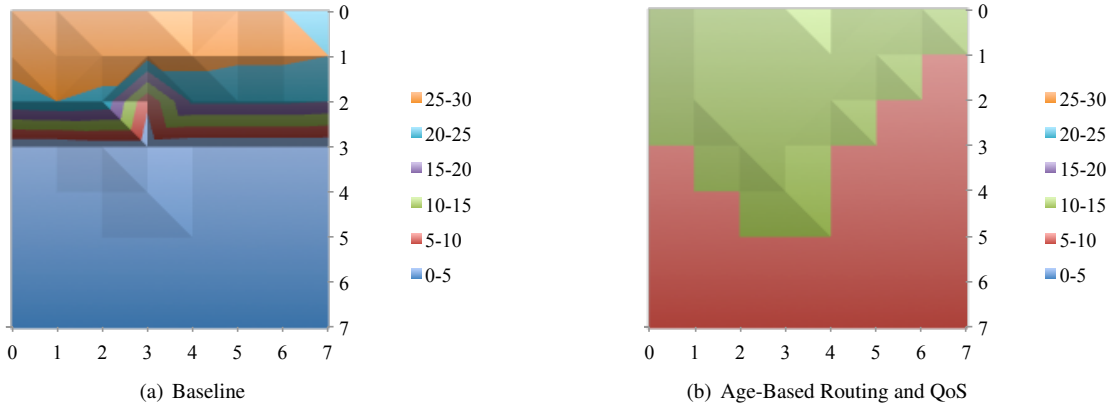


Figure 8: Contour plots showing the distribution of completed DMA requests across the 8x8 mesh network.

6.3 Discussion

Synthetic benchmark results from the previous section confirm that on every level, our proposed design functions as intended. It is clear that age-based routing provides significant benefits for fairness, even in cases where an additional QoS policy is not present. Further, while our QoS policy for fairness does reduce the peak number of requests issued by some nodes and does punish nodes near the single DMA controller, it also brings about greater request fairness across the entire mesh. Further, we see that the fairness policy is equally effective for both regular memory requests and DMA operations. Finally, our results confirm that the QoS Gate approach is capable of supporting several different QoS policies, including differentiated services.

6.3.1 Limitations

Throughout the design and benchmark process, we have discovered two pitfalls of our current approach. First, by incorporating local switch utilization into our QoS fairness policy, we effectively penalize neighbors of nodes with DMA controllers, as DMA controllers will generate lots of network traffic. Second, through our own benchmarks, we have discovered that the effectiveness of our QoS policies are heavily dependent on the credit replenishment policy. The reason this becomes an issue is because the replenishment policy can be tuned with several parameters, and each additional parameter makes it harder and harder to tune the algorithm for general-purpose use.

7 Future Work

Even in light of the success of our current work, we do see several areas for future work. As we mentioned earlier, our current message cost policy is fixed (i.e., all messages are charged the same cost). However, we imagine that dynamically assessing message cost (via message types, message sizes, and/or total distance to travel) might open the opportunity to a wider range of QoS policies. Given the current limitation of our fairness QoS policy, wherein a busy DMA controller reduces the performance of its neighboring nodes due to network congestion, we are considering the value in adding a QoS gate between the DMA controller and the network. Finally, in this work, we only addressed two QoS policies: fairness and differentiated services. However, as we previously acknowledged, there is interest in policies which are capable of grouping several nodes together and/or partitioning the entire network into isolated cells. We believe that the QoS Gate design can support these types of policies, but need additional time to complete this investigation.

8 Conclusion

Our simulator results show that age-based routing along is sufficient to ensure nodes in a system are treated with reasonably equal fairness when executing both normal memory accesses and DMA operations. The QoS Gate we designed acts as the source of enforcement for admission control in our QoS policy and it allows system designers to adjust the relative priority between different nodes in the system while maximizing network utilization. We observed that traffic responsible for servicing DMA requests tends to cause network congestion surrounding DMA controllers, thereby yielding traffic cutbacks of local node requests. Allowing non-congested chip areas to continue operation in light of DMA-induced congestion better utilizes total system bandwidth at the cost of perceived fairness between nodes in the system.

References

[1] hdparm. `http://sourceforge.net/projects/hdparm/`, 2012.

[2] Intel 82599 10 Gigabit Ethernet Controller Datasheet. `http://www.intel.com/content/dam/doc/datasheet/82599-10-gbe-controller-%datasheet.pdf`, 2012.

[3] Princeton Application Repository for Shared-Memory Computers (PARSEC). `http://parsec.cs.princeton.edu/`, 2012.

[4] The gem5 Simulator System. `http://www.gem5.org`, 2012.

[5] BIENIA, C. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[6] BOLOTIN, E., CIDON, I., GINOSAR, R., AND KOLODNY, A. Qnoc: Qos architecture and design process for network on chip. *J. Syst. Archit. 50*, 2-3 (Feb. 2004).

[7] COLMENARES, J. A., BIRD, S., COOK, H., PEARCE, P., ZHU, D., SHALF, J., HOFMEYR, S., ASANOVIC, K., AND KUBIATOWICZ, J. Resource management in the Tessellation manycore OS. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Parallelism (HotPar'10)* (Berkeley, CA, USA, June 2010).

[8] GROT, B., KECKLER, S. W., AND MUTLU, O. Preemptive virtual clock: a flexible, efficient, and cost-effective qos scheme for networks-on-chip. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (2009), MICRO 42, ACM.

[9] LEE, J. W., NG, M. C., AND ASANOVIC, K. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. *SIGARCH Comput. Archit. News 36*, 3 (June 2008).

[10] RUEMMLER, C., AND WILKES, J. An introduction to disk drive modeling. *Computer 27*, 3 (march 1994), 17 –28.