Appears in *35th IEEE/ACM International Symposium on Computer Architecture (ISCA)*, Beijing, China, June 2008.

Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks

Jae W. Lee, Man Cheuk Ng
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
{leejw, mcn02}@csail.mit.edu

Krste Asanović
Computer Science Division, EECS Department
University of California at Berkeley
Berkeley, CA 94720-1776
krste@eecs.berkeley.edu

Abstract

Future chip multiprocessors (CMPs) may have hundreds to thousands of threads competing to access shared resources, and will require quality-of-service (QoS) support to improve system utilization. Although there has been significant work in QoS support within resources such as caches and memory controllers, there has been less attention paid to QoS support in the multi-hop on-chip networks that will form an important component in future systems. In this paper we introduce Globally-Synchronized Frames (GSF), a framework for providing guaranteed QoS in on-chip networks in terms of minimum bandwidth and a maximum delay bound. The GSF framework can be easily integrated in a conventional virtual channel (VC) router without significantly increasing the hardware complexity. We rely on a fast barrier network, which is feasible in an on-chip environment, to efficiently implement GSF. Performance guarantees are verified by both analysis and simulation. According to our simulations, all concurrent flows receive their guaranteed minimum share of bandwidth in compliance with a given bandwidth allocation. The average throughput degradation of GSF on a 8×8 mesh network is within 10 % compared to the conventional best-effort VC router in most cases.

1 Introduction

Advances in fabrication technology allow the integration of many processors on a chip to form a chip multiprocessor (CMP), possibly in the form of a complex system-on-a-chip (SoC) with custom application accelerators (Figure 1). These platforms will be required to support a variety of complex application workloads, with possibly hundreds to thousands of concurrent activities competing for shared platform resources. Without effective quality-of-service (QoS) support, the gap between best-case

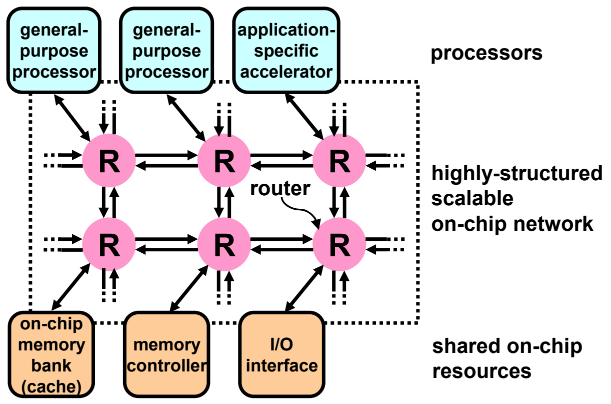

general-purpose processor
general-purpose processor
application-specific accelerator
processors
R
router
highly-structured scalable on-chip network
on-chip memory bank (cache)
memory controller
I/O interface
shared on-chip resources

Figure 1: A complex system-on-a-chip containing a chip-scale multiprocessor and application accelerators communicating with memory and I/O resources over a multi-hop on-chip network.

and worst-case throughput will continue to grow, requiring overprovisioning and hence poor utilization of platform resources [11, 12, 13, 18].

We believe future integrated platforms must implement robust QoS support for both *performance isolation* and *differentiated services*. Performance isolation is the property that a minimum level of performance is guaranteed regardless of other concurrent activities (e.g., preventing denial-of-service attacks to DRAM channels [21]). Differentiated services is the ability to allocate each resource flexibly among competing tasks.

Robust QoS support is only possible if all shared resources are managed together, as an application's guaranteed service level is determined by the weakest guarantee for any of its shared resources. For example, allocating a portion of off-chip memory bandwidth at a memory controller is ineffective if the on-chip network does not guarantee adequate bandwidth to transport memory requests and responses. Even in a case where the on-chip network is not a bandwidth bottleneck, tree saturation [27] can produce a tree of waiting packets that fan out from a hotspot resource, thereby penalizing remote nodes in delivering requests to the arbitration point for the hotspot resource. Figure 2 provides a motivational example, showing how poorly multi-hop on-chip networks perform with no QoS support. Although there has been significant prior work in QoS support for other on-chip resources such as memory controllers (often combined with on-chip bus) [3, 15, 18, 22] and shared cache banks [13, 14, 23, 26], there has been less work on QoS support for multi-hop on-chip networks in programmable platforms.

In this paper, we present a new scheme, *Globally-Synchronized Frames (GSF)*, to implement QoS for multi-hop on-chip networks. GSF provides guaranteed and differentiated bandwidth as well as bounded network delay without significantly increasing the complexity of the on-chip routers. In a GSF system, time is coarsely quantized into "frames" and the system only tracks a few frames into the future to reduce time management costs. Each QoS packet from a source is tagged with a frame number indicating the desired time of future delivery to the destination. At any point in time, packets in the earliest extant frame are routed with highest priority but sources are prevented from inserting new packets into this frame. GSF exploits fast on-chip communication by using a global barrier network to determine when all packets in the earliest frame have been delivered, and then advances all sources and routers to the next frame. The next oldest frame now attains highest priority and does not admit any new packets, while resources from the previously oldest frame are recycled to form the new futuremost frame.

Provided that the pattern of injected packets in each frame does not oversubscribe the capacity of any hardware link, the system can switch frames at a rate that sustains any desired set of differentiated bandwidth flows with a bounded maximum latency. Note that bandwidth and latency are decoupled in this system, as multiple frames can be pipelined through the system giving a maximum latency of several frame switching times. The scheme does not maintain any per flow information in the routers, which reduces router complexity and also avoids penalizing short-lived flows with a long route configuration step. The scheme supports bursty traffic, and allows best-effort traffic to be simultaneously supported with little loss of network utilization compared to a pure best-effort scheme.

2 Related Work

We begin with a survey of related work which we divide into three parts. We first examine schemes to manage resources with centralized arbitration, such as a memory controller or a shared bus, where providing QoS is relatively easy because there is a single gateway through which all requests pass. We next examine earlier work in distributed QoS systems, where QoS is more difficult to provide as each request passes through multiple stages of arbitration. Finally, we examine other proposals for QoS in on-chip networks.

2.1 QoS Support for Resources with Centralized Arbitration

Off-chip memory bandwidth is often a performance bottleneck and is a natural target for QoS support. For example, to target real-time applications, Philips' TM-1 processor supports bandwidth and latency guarantees among one VLIW processor and four DMA engines [3]. More recently, Nesbit et al. have proposed a QoS memory system for CMPs based on a Fair Queueing (FQ) algorithm [22]. The FQ algorithm requires the memory controller to have per-flow queues and maintain per-flow statistics such as virtual start and finish times, which causes a scalability concern for future manycore processors. The resource allocation management unit of the Cell Broadband Engine supports QoS for system memory and I/O interfaces [15]. To reduce the overhead of per-flow data structures, each requester is assigned to a resource allocation group (RAG) and the system allocates a percentage of the managed resources to each RAG.

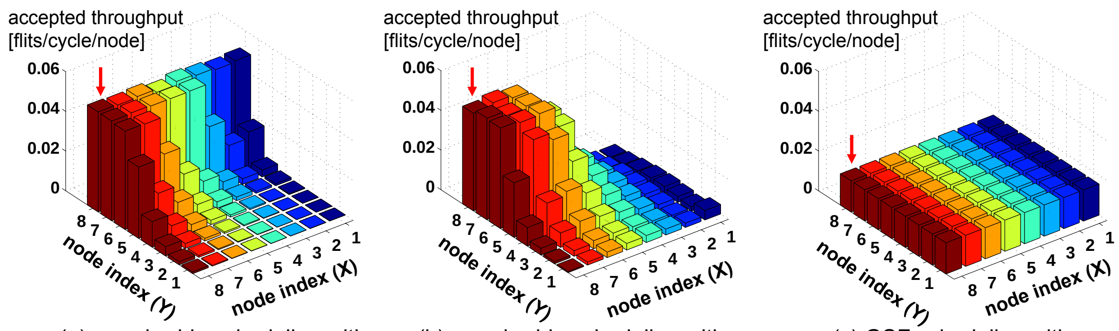
There are also proposals for QoS provision for shared on-chip caches. Suh et al. propose a non-uniform cache partitioning scheme to minimize the overall miss rate, but without consideration of guaranteed services to an individual thread [26]. Iyer's cache controller in the CQoS framework enforces the priority of each thread to allocate cache lines appropriately [14]. Both schemes manage cache space but not access bandwidth. Virtual Private Caches (VPC) manage both shared cache bandwidth and storage [23].

The QoS-capable shared resources discussed in this section are important building blocks for guaranteed QoS systems, but QoS support from the on-chip network is also required to make system-level QoS guarantees possible.

2.2 QoS Support for Resources with Distributed Arbitration

Distributed shared resources, most notably multi-hop on-chip and off-chip networks, require multiple stages of arbitration, which make it more challenging to provide guaranteed service to a flow. Several approaches have been developed to address this issue, either for the IP networks, or for multichip multiprocessor networks.

(Weighted) Fair Queueing [7] and Virtual Clock [32] were developed for QoS in long-haul IP networks where large buffers are available. These achieve fairness and high network utilization, but each router is required to maintain per-flow state and queues which would be impractical in an on-chip network.



(a) round-robin scheduling with dimension-ordered routing

(b) round-robin scheduling with minimal-adaptive routing

(c) GSF scheduling with dimension-ordered routing

Figure 2: Motivation for QoS in on-chip networks. All nodes generate traffic toward a hotspot shared resource located at (8,8) (indicated by arrow) with injection rate of 0.05 (flits/cycle/node), and the bar graph shows accepted service rate per node by the hotspot resource. In (a), locally-fair round-robin (RR) scheduling leads to globally-unfair bandwidth usage, penalizing remote nodes. The minimal-adaptive routing in (b) eliminates the imbalance of traffic between x and y directions in (a), and possibly helps mitigate the problem in a lightly-congested network, but does not fundamentally resolve the problem. Our proposed GSF scheme in (c) guarantees fair bandwidth allocation among all sharers. All three networks achieve comparable average throughput. See Section 6.1 for detailed simulation setup.

In multi-rate channel switching [29], the source rate of a flow is fixed at an integer multiple of a basic rate before the source starts transmission and remains unchanged for the duration of the flow. Because of the fixed rate, it cannot claim unused bandwidth efficiently, which leads to low network utilization.

Source throttling dynamically adjusts the traffic injection rate at a source node primarily for congestion control [27]. This keeps the network from suffering overall throughput degradation beyond the saturation point, but does not provide QoS to individual flows.

Age-based arbitration is known to provide strong global fairness. Each packet (or flit) carries information to indicate its age, either a counter updated at every hop [25], or a timestamp issued when the packet first enters the network from which age can be calculated by subtracting from the current time [5]. The oldest packet wins in any arbitration step. This approach lacks flexibility in bandwidth allocation because it does not allow for asymmetric resource allocation, and requires sophisticated logic to handle aging, arbitration and counter rollover.

Rotating Combined Queueing (RCQ) is designed for a multiprocessor and provides predictable delay bounds and bandwidth guarantees without per-flow queues at intermediate nodes (though it still maintains per-flow statistics) [16].

The idea of rotating priorities in a set of queues is similar to GSF, but GSF further simplifies the router using global information, which is only feasible in an on-chip environment. With RCQ, each packet is assigned a local frame number using per-flow statistics upon arrival at every node on the path. In contrast, the frame number in the GSF is global, which eliminates expensive book-keeping logic and storage at each node.

2.3 QoS-Capable On-Chip Networks

The on-chip environment has different opportunities and challenges compared to the off-chip environment, leading to different design tradeoffs. For example, on-chip networks are buffer (power) limited while multiprocessor networks are often pin-bandwidth limited. Network latencies differ by multiple orders of magnitude, which affects the cost of synchronization. The following briefly introduces several proposals for QoS support in on-chip networks, representing the current state-of-the-art.

Æthereal uses pipelined time-division-multiplexed (TDM) circuit switching to implement guaranteed performance services [10]. Each QoS flow is required to explicitly set up a channel on the routing path before transmitting the first payload packet, and a flow cannot use more than its guaranteed bandwidth share even if the network is underutilized. To mitigate this problem, Æthereal adds a best-effort network using separate queues, but this introduces ordering issues between the QoS and best-effort flows.

SonicsMX supports guaranteed bandwidth QoS without explicit channel setup [30]. However, each node has to maintain per-thread queues, which make it only suitable for a small number of threads (or having multiple sources share a single queue). The Nostrum [20] Network-on-Chip (NoC) employs a variant of TDM using virtual circuits for allocating bandwidth. The virtual circuits are set up semi-statically across routes fixed at design time and only the bandwidth is variable at runtime, which is only suitable for application-specific SoCs. The MANGO clockless NoC [2] partitions virtual channels (VCs) into two classes: Guaranteed Service (GS) and Best-Effort (BE). A flow reserves a sequence of GS VCs along its path for its lifetime. Therefore, the number of concurrent GS flows sharing a physical channel is limited by the number of GS VCs (e.g., 8 in [2]). Felicijan et al. propose a clockless NoC which provides differentiated services by prioritizing VCs. Though this approach delivers improved latency for certain flows, no hard guarantees are provided [8].

3 Globally-Synchronized Frames (GSF)

In this section, we present the design of GSF starting from an idealized deadline-based arbitration scheme for bandwidth guarantees. We then transform this scheme step-by-step into an implementable GSF queueing and scheduling algorithm.

3.1 Global Deadline-Based Arbitration for Bandwidth Guarantees

GSF was originally inspired by deadline-based arbitration, which is a generalization of age-based arbitration [1, 5, 6, 25]. In age-based arbitration, each packet carries a global timestamp, issued when the packet enters the network, and each arbiter (router) forwards the packet with the earliest timestamp first. Instead of using the timestamp, we allow each source to assign a deadline other than the current time according to a deadline assignment policy. Our premise is that we can achieve a desired flow property, including guaranteed minimum bandwidth, by controlling deadline assignment policy, at least in an idealized setup.

Figure 3 shows a network from such an idealized setup, where each queue is a perfect priority queue with infinite capacity, capable of instantly dequeuing the packet with the earliest deadline. Dotted rectangles are a network component which Cruz [4] introduced and named “MUX”. Since we assume zero-cycle delay for arbitration and queue bypassing, the entire network conceptually reduces to a single priority queue having four input ports and one output port, with a total ordering among all packets according to their deadlines.

To provide bandwidth guarantees, we assign the deadline for the n-th packet of Flow i (d_i^n) as follows:

$$\boxed{d_i^n(\rho_i) = MAX[current\_time, d_i^{n-1}] + L_i^n/(\rho_i C)}$$

where ρ_i is the guaranteed minimum bandwidth of Flow i represented as a fraction of channel bandwidth C ($0 \leq \rho_i \leq 1$) and L_i^n is the length of the n-th packet of Flow i.

This formula directly follows from what is known as the *virtual finish time* in the Fair Queueing algorithm [7]. The deadline specifies the time when a packet’s last bit arrives at the destination if the channel were infinitely divisible and shared by multiple packets simultaneously transmitting according to their guaranteed shares (ρ’s), provided we ignore the network traversal delay, which is dependent upon

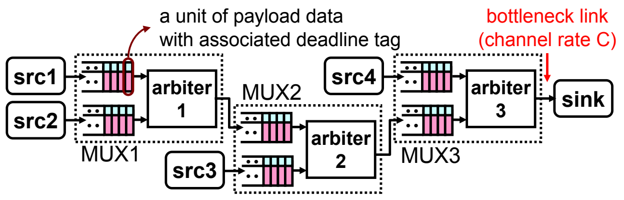


Figure 3: A three-node queueing network with perfect priority queues with infinite capacity. Dotted rectangles are a network component called “MUX”, which merges two incoming flows into a single outgoing one. Each packet carries an associated deadline and the priority queue can dequeue the packet having the earliest deadline. The arbiter and the priority queue are assumed to have zero-cycle delay, which implies a packet just generated at any source can be immediately forwarded to the sink at channel rate C through the combinational path if the packet wins all arbitrations on the path.

each flow’s distance from the destination. Figure 4 compares three arbitration schemes: round-robin, age-based and deadline-based with the deadline assignment policy presented above. Note that the deadline-based scheme provides bandwidth distribution to all flows proportional to the ratio of ρ’s at all congested links. This result holds even if we have non-zero delay for arbitration and/or queue bypassing as long as the priority queue has an infinite capacity. In such a case, two flows sharing a congested link eventually enter into the steady state of proportional bandwidth sharing after a finite winning (losing) streak by the remote (local) node starting when the two flows first meet at the congested link. The length of the winning (losing) streak is determined by the relative difference of the distance to the congested link.

Although deadline-based arbitration provides minimum bandwidth guarantees to flows using the proposed policy in the idealized setup, there are several issues that make this scheme infeasible to implement. First, the scheme is based on perfect priority (sorting) queues and infinite-sized buffers. Second, there is a large overhead for sending and storing the deadline along with the payload data. Therefore, we propose a practical implementation approximating the behavior of the ideal deadline-based arbitration, called *baseline GSF*.

3.2 Baseline GSF

To make deadline-based arbitration practical, we adopt a frame-based approach [31]. The original deadline-based arbitration is a priority-based approach, where competing packets’ priorities are compared to determine which packet is allocated buffer space and switching bandwidth first. In contrast, a frame-based approach groups a fixed number of time slots into a frame and controls the bandwidth allocation by allocating a certain number of flit injection slots per frame to each flow.

Figure 5 shows a step-by-step transformation towards

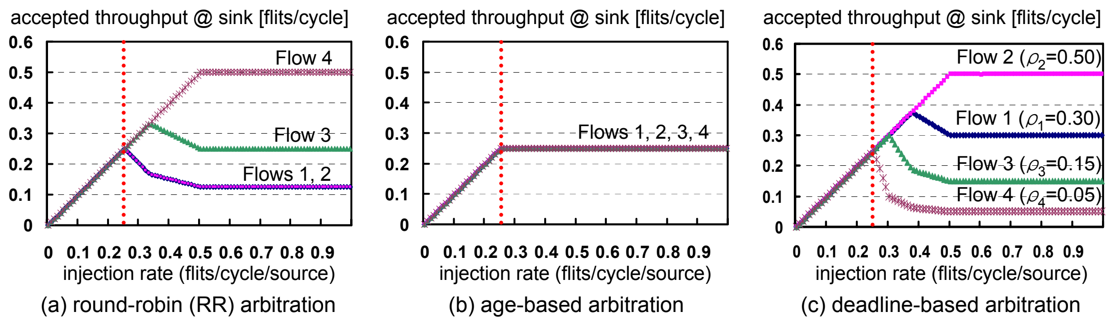


Figure 4: Per-source accepted throughput at the sink with various arbitration schemes in the network shown in Figure 3. Dotted vertical lines indicate minimum injection rate causing congestion. In locally-fair round-robin arbitration in (a), which does not use the deadline field, the throughput of a flow decreases exponentially as the number of hops increases. Age-based arbitration in (b), where deadline is assigned as network injection time, gives fair bandwidth allocation among all flows. With deadline-based arbitration with the policy for bandwidth guarantees in (c), we achieve bandwidth distribution proportional to the ratio of ρ's ($\rho_1 : \rho_2 : \rho_3 : \rho_4 = 0.30 : 0.50 : 0.15 : 0.05$) in face of congestion.

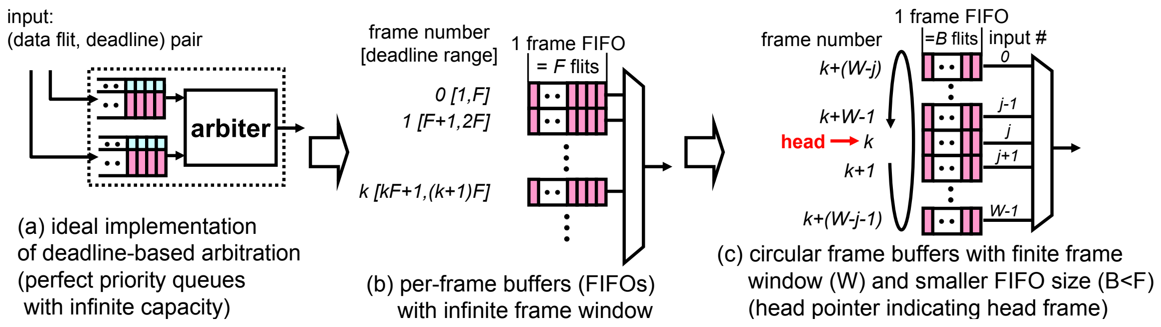

input:
(data flit, deadline) pair
arbiter
(a) ideal implementation of deadline-based arbitration (perfect priority queues with infinite capacity)
frame number [deadline range]
1 frame FIFO = F flits
0 [1,F]
1 [F+1,2F]
k [kF+1,(k+1)F]
(b) per-frame buffers (FIFOs) with infinite frame window
1 frame FIFO
frame number
=B flits
input #
k+(W-j)
k+W-1
head
k
k+1
k+(W-j-1)
0
j-1
j
j+1
W-1
(c) circular frame buffers with finite frame window (W) and smaller FIFO size (B<F) (head pointer indicating head frame)

Figure 5: Step-by-step transformation towards a frame-based, approximate implementation of deadline-based arbitration. (a) shows the ideal MUX introduced in Section 3.1, which is associated with each physical link. In (b), we group all data entries having a range of deadlines (whose interval is F) to form a *frame*. The frame number is used as a coarse-grain deadline, and we now drop the deadline (or frame number) field because each frame has an associated frame buffer. Finally, we recycle frame buffers, with W active frames, as in (c) to give a finite total buffer size.

a frame-based, approximate implementation of deadline-based arbitration. Frame k is associated with packets whose deadline is in the range of $[kF+1,(k+1)F]$, where F is the number of flit times per frame. The frame number k is used as a coarse-grain deadline. By introducing frames, we enforce an ordering *across* frames but not *within* a frame because the service order within a frame is simply FIFO. The baseline GSF arbitration is shown in Figure 5(c), where we have a finite active frame window having W frames (i.e., Frame k through $(k+W-1)$) and each active frame has a dedicated frame buffer (FIFO) whose depth is B flits. The head pointer indicates which frame buffer is currently bound to the earliest frame (frame buffer j in the figure), which we call the *head frame*. Note that the baseline GSF in Figure 5(c) is an asymptotic implementation of the ideal deadline-based arbitration in Figure 5(a) because the former reduces to the latter as $W \rightarrow \infty$ and $F \rightarrow 1$.

Here is a brief sketch of how the GSF network operates. For each active frame, every flow is allowed to inject a certain number of flits, denoted by R_i for flow i. Although our scheme is similar to conventional frame-based bandwidth allocation schemes, we allow W frames to overlap at any given time to accommodate bursty traffic while preventing an aggressive flow from injecting too much traffic into the network.

Once a packet is injected into the network, it traverses the network using only the frame buffers for its given frame number. Therefore, there is no possibility of priority inversion, where a lower-priority packet blocks a higher-priority packet. Combined with earliest-frame-first scheduling for bandwidth allocation, the head frame is guaranteed to drain in a finite amount of time because only a finite sum of packets can be injected into a single frame by all flows. The drained head frame buffers across the entire network are reclaimed and allocated to a newer frame synchronously, which is called an *(active) frame window shift*.

We define an *epoch* as the period of time between adjacent frame window shifts, and the interval of the k-th epoch

Variables	Range	Description
Global parameters and variables*		
W	2 .. ∞	active frame window size
HF	0 .. $(W-1)$	current head frame
F	1 .. ∞	frame size in flits
B	1 .. ∞	frame buffer depth in flits
C	(0, 1]	channel bandwidth in flits/cycle
L^{MAX}	1 .. ∞	maximum packet length in flits
$epoch$**	0 .. ∞	current epoch number
e_k	1 .. e^{MAX}	interval of k-th epoch
e^{MAX}	1 .. ∞	maximum epoch interval (= $\max_{\forall k} e_k$)
T^{epoch}	0 .. e^{MAX}	epoch timer
Per-flow variables		
ρ_i	0 .. 1	a fraction of bandwidth allocated to Flow i (normalized to C)
R_i	0 .. F	flit slots reserved for Flow i in a single frame
IF_i	0 .. $(W-1)$	current injection frame of Flow i
C_i	-L^{MAX} .. R_i	available credit tokens for Flow i to inject flits to IF_i

* Global variable with a subscript i denotes a local copy of the variable maintained by Flow i.
** not implemented in hardware

Table 1: Variables and parameters used in GSF.

(i.e., the period of time when frame k is the header frame) is denoted by e_k. We also define $e^{MAX} \equiv \max_{\forall k} e_k$. Table 1 summarizes variables and parameters used in the GSF algorithm. More detailed description of each network component's operation follows.

Packet injection process: Algorithm 1 describes a packet injection algorithm used by the baseline GSF network. Flow i can inject packets into the active frame pointed to by IF_i as long as it has a positive credit balance ($C_i > 0$) for the frame (Lines 2-4). The flow can go overdrawn, which allows it to send a packet whose size is larger than R_i. To preserve bandwidth guarantees, a flow with negative credit balance cannot inject a new packet until its balance becomes positive again.

If the flow has used up all reserved slots in Frame IF_i, it can use reserved slots further in the future by incrementing IF_i by one (mod W) (Lines 5-13) until it hits the tail of the active frame window. Once the flow uses up all reserved slots in the active frame window, it must stall waiting for a frame window shift to open a new future frame. Note that this injection process is effectively implementing a token-bucket filter [4] with (ρ, σ) = $(R_i/e^{MAX}$, $R_i * W)$ assuming the active frame window shifts at every e^{MAX} cycles.

Switching bandwidth and buffer allocation: Frame buffer allocation is simple because every packet is assigned a frame at the source, which determines a sequence of frame buffers to be used by the packet along the path. There can be contention between packets within a frame but not across frames. In allocating switching bandwidth, we give the highest priority to the earliest frame in the window.

Frame window shifting algorithm: Algorithm 2 shows an algorithm used to shift the active frame window. Source injection control combined with earliest-frame first scheduling yields a finite drain time for the head frame, bounded

Algorithm 1 GSF packet injection algorithm into source queue for Flow i ($\oplus_W$: modulo W addition)

```
Initialize: $epoch = 0, HF_i = HF = 0$
Initialize: $R_i = C_i = \lfloor \rho_i F \rfloor$
Initialize: $IF_i = 1$
 1: AT EVERY PACKET GENERATION EVENT:
 2: if $C_i > 0$ then
 3:     $SourceQueue_i.enq(packet, IF_i)$
 4:     $C_i = C_i - packet.size()$
 5: else {used up all reserved slots in Frame $IF_i$}
 6:     while $(IF_i \oplus_W 1) \neq HF_i$ and $C_i < 0$ do
 7:         $C_i = C_i + R_i$
 8:         $IF_i = IF_i \oplus_W 1$
 9:     end while
10:     if $C_i > 0$ then
11:         $SourceQueue_i.enq(packet, IF_i)$
12:         $C_i = C_i - packet.size()$
13:     end if
14: end if
```

by e^{MAX}. Therefore, we shift the active frame window at every e^{MAX} cycles by default. Every flow maintains a local copy (T_i^{epoch}) of the global epoch timer (T^{epoch}) and decrements it at every clock tick (Lines 9-10). Once the timer reaches zero, all the flows synchronously increment the head frame pointer HF_i (mod W) to reclaim the frame buffer associated with the earliest frame.

The frame window shifting algorithm does not allow a flow to inject a new packet into the head frame (Lines 4-7). Otherwise, we would have a very loose bound on the worst-case drain time of the head frame (e^{MAX}), which would degrade network throughput.

Algorithm 2 GSF frame window shifting algorithm ($\oplus_W$: modulo W addition)

```
Initialize: $T_i^{epoch} = e^{MAX}$
Initialize: $HF_i = HF = 0$
 1: FOR ALL FLOWS, AT EVERY CLOCK TICK:
 2: if $T_i^{epoch} == 0$ then
 3:     $HF_i = HF_i \oplus_W 1$
 4:     if $HF_i == IF_i$ then
 5:         $IF_i = IF_i \oplus_W 1$
 6:         $C_i = MIN(R_i, C_i + R_i)$
 7:     end if
 8:     $T_i^{epoch} = e^{MAX}$
 9: else
10:     $T_i^{epoch} = T_i^{epoch} - 1$
11: end if
```

In effect, the GSF network implements W logical networks sharing physical channels, and each logical network is associated with one frame at any point in time. The W logical networks receive switching bandwidth according to priorities which rotate on every frame window shift. A logical network starts as the lowest-priority logical network when it is just assigned to a new frame, and is promoted throughout the lifetime of the frame to eventually become the highest-priority network, after which it finally gets reclaimed for the new futuremost frame.

The baseline GSF network provides the following guaranteed bandwidth to flow i if (1) none of the physical channels along the path are overbooked and (2) the source queue ($SourceQueue_i$) has sufficient offered traffic to sustain the reserved bandwidth:

$$\boxed{\text{Guaranteed bandwidth}_i = R_i/e^{MAX}}$$

The proof sketch is simple. Flow i can inject R_i flits into each frame, and the network opens a new frame every e^{MAX} cycles. Because the network does not drop any packets and has a finite buffer size, the guaranteed bandwidth holds. In addition, the worst-case network delay is bounded by We^{MAX} because a packet injected in k-th epoch must be ejected from the network by the beginning of ($k+W$)-th epoch.

Although the baseline GSF scheme provides guaranteed services in terms of bandwidth and bounded network delay, there are several drawbacks to the scheme. First, frame buffers are underutilized, which degrades overall throughput for a given network cost. Second, it is difficult to bound e^{MAX} tightly, which directly impacts the guaranteed bandwidth. Even with a tight bound, it is too conservative to wait for e^{MAX} cycles every epoch because the head frame usually drains much faster.

To address these two issues without breaking QoS guarantees, we propose two optimization techniques: *carpool lane sharing* and *early reclamation of empty head frames*.

3.3 Carpool Lane Sharing of Frame Buffers: Improving Buffer Utilization

One observation in the baseline GSF scheme is that guaranteed throughput does not really depend on the active frame window size, W. The multiple overlapping frames only help claim unused bandwidth to improve network utilization by supporting more bursty traffic. As long as we provide a dedicated frame buffer for the head frame at each router to ensure a reasonable value of e^{MAX}, we do not compromise the bandwidth guarantees.

We propose carpool lane sharing to relax the overly restrictive mapping between frames and frame buffers. Now we reserve only one frame buffer to service the head frame (like a carpool lane), called the head frame buffer, but allow all active frames, including the head frame, to use all the other frame buffers. Each packet carries a frame number (mod W) in its head flit, whose length is $\lceil \log_2 W \rceil$ bits, and the router services the earliest frame first in bandwidth and buffer allocation. The frame window shifting mechanism does not change. Note that head-of-line blocking of the head frame never happens because we map frame buffers to virtual channels (allocated on a per-packet basis) and the head frame buffer at each router serves as an escape channel for packets that belong to the head frame.

According to our evaluation, the carpool lane sharing scheme increases the overall throughput significantly because more frame buffers are available to a packet at each node. That is, any packet can occupy any frame buffer, except that the head frame buffers are reserved only for packets in the head frame. To support best-effort traffic, we can simply assign a special frame number (W, for example) which represents the lowest priority all the time.

3.4 Early Reclamation of Empty Frames: Increasing Frame Reclamation Rate

One important factor affecting the overall throughput in the GSF network is the frame window shift rate. According to our analysis, only a small fraction of e_k's ever come close to e^{MAX}, which implies that the head frame buffer is often lying idle waiting for the timer to expire in each epoch.

Therefore, we propose to use a global barrier network to reclaim the empty head frame as quickly as possible. Instead of waiting for e^{MAX} cycles every epoch, we check whether there is any packet in the source or network buffers that belongs to the head frame. If not, we retire the head frame immediately and allocate its associated buffers to the new futuremost frame. Note early reclamation does not break the original bandwidth guarantees, because we always see a net increase, or at worst no change, in available flit injection slots.

Figure 6 shows that early reclamation provides over 30% improvement in network throughput in exchange for a small increase in area and power for the barrier network. The barrier network is only a small fraction of the cost of the primary data network, as it uses only a single wire communication tree and minimal logic.

4 Implementation

The GSF frame structure fits well into the architecture of a conventional virtual channel (VC) router, requiring only relatively minor modifications. This section discusses the GSF router architecture and the fast on-chip barrier network.

4.1 GSF Router Architecture

Figure 7 shows a proposed GSF router architecture. This router implements both carpool lane buffer sharing and early frame reclamation on top of the baseline GSF. Starting from a baseline VC router, we describe various aspects and design issues in the GSF router.

Baseline VC router We assume a three-stage pipelined VC router with lookahead routing [9] as our baseline. The

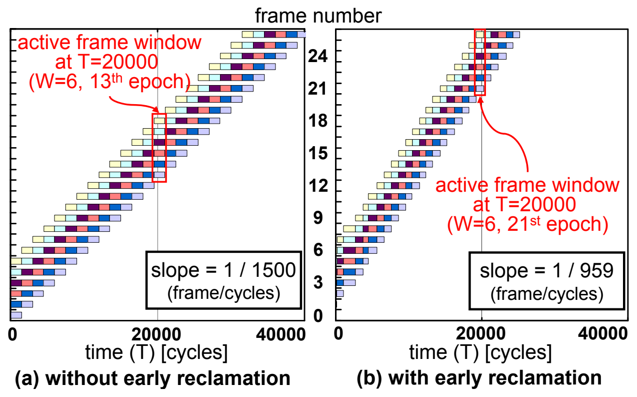


Figure 6: Frame life time analysis for comparison of frame reclamation rates with and without early reclamation. Although the estimated $e^{MAX} = 1500$ is within 10% of the observed worst-case e^{MAX}, the early reclamation increases the frame reclamation rate by $>$30%, which leads to corresponding improvement in the average throughput. See Section 6.1 for simulation setup. We use a hotspot traffic pattern with injection rate of 0.05 flits/cycle/node.

three stages are next-hop routing computation (NRC) in parallel with virtual channel allocation (VA), switch allocation (SA) and switch traversal (ST).

Added Blocks Each router node keeps a local copy of the global head frame (HF) variable. This variable increments (mod W) at every frame window shift triggered by the global barrier network. Each VC has a storage to maintain the frame number (mod W) of the packet it is servicing. The frame number at each VC is compared against HF to detect any packet belonging to the head frame. Then the global barrier network gather information to determine when to shift the frame window appropriately.

Next-Hop Routing Computation (NRC) In order to reduce the burden of the VA stage, which is likely to be the critical path of the router pipeline, we precalculate the packet priority at this stage. The packet priority can be obtained by $(frame\_num - HF)$ (mod W). The lowest number has the highest priority in VC and SW allocation. When calculating the routing request matrix, NRC logic is responsible for masking requests to VC0 from non-head frames, because VC0 is reserved for the head frame only.

VC and SW allocation VC and SW allocators perform priority-based arbitration, which selects a request with the highest priority (the lowest number) precalculated in the previous NRC stage. The GSF router uses standard credit-based flow control.

4.2 Global Synchronization Network

The latency of the global synchronization network affects the overall network throughput because higher latencies leave VC0 (the carpool channel) idle for longer. Although our proposed router is generally tolerant to the latency of the barrier network if the frame size (F) is reasonably large, the impact is more visible for a traffic pattern having a high turnaround rate of frame buffers.

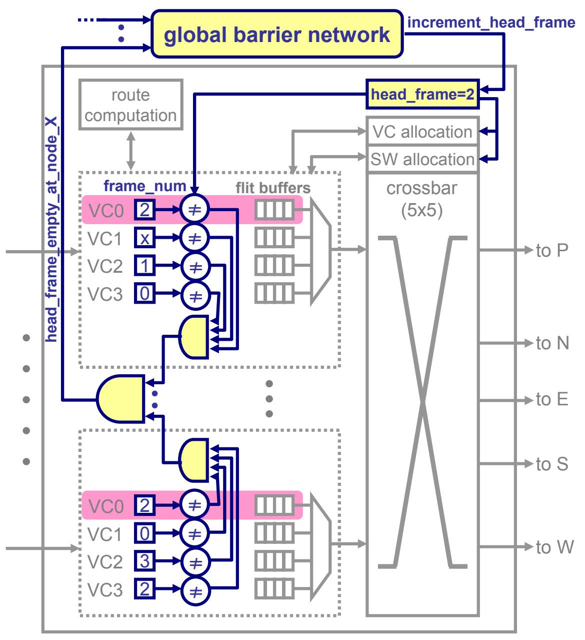


Figure 7: A GSF router architecture for 2D mesh network. Newly added blocks are highlighted while existing blocks are shown in gray.

One way to achieve barrier synchronization is to use a fully-pipelined dimension-wise aggregation network [27]. In this network, assuming a 2D mesh, the center node of each column first collects the status of its peers in the same column. Then, it forwards the information to the center node of its row where the global decision is made. A broadcast network, which operates in reverse of the aggregation network, informs all nodes to rotate their head frame pointers (HF). For k-ary n-cube (or mesh) network, the latency of the synchronization will be $2n\lceil \frac{k-1}{2} \rceil$ cycles assuming one-cycle per-hop latency.

Alternatively, we can implement a barrier network using combinational logic which might take multiple fast clock cycles to settle. This requires significantly less area and power, and provides lower latency as cross-chip signal propagation is not delayed by intermediate pipeline registers. If propagation takes N fast clock cycles, we could sample each router's state and read the barrier signal every N cycles to determine when to perform a frame window shift. For evaluation, we use a variable number of cycles up to $2n\lceil \frac{k-1}{2} \rceil$ cycles.

5 System-Level Design Issues

A QoS-capable on-chip network sits between processors running the software stack and shared platform resources. This section addresses issues in interacting with these two ends of the system to provide robust QoS support.

Admission control: Admission control is a software process that should guarantee that no channel in the network is oversubscribed. That is, suppose $S_c = \{i_1, i_2, \cdots, i_n\}$ is a set of flows that pass through Channel c. Then $\forall$ Channel c in the network, $\sum_{i \in S_c} R_i \leq F$ should hold. To keep network utilization high, each flow can be granted more than the minimum number of slots required where possible, as the maximum number of flits in flight from flow i at any given time is upper bounded by WR_i. If a new flow enters into a previously reserved channel, the software stack redistributes the excess injection slots according to its excess bandwidth sharing policy. Note that our GSF scheme does not require any explicit channel setup, and so only the R_i control register at each source must be changed. If there are multiple clock domains, possibly with dynamic voltage-frequency scaling (DVFS), any channel c should provide at least the sum of guaranteed bandwidths on the channel to preserve QoS guarantees.

Specifying bandwidth requirements: To specify requested bandwidth, one can use either a relative measure (e.g., 10 % of available bandwidth) as in [24] or an absolute measure (e.g., 100 MB/s). If a relative measure ρ_i is given, R_i can be set to be $\rho_i F$. If an absolute measure BW (in flits/cycle) is used, R_i can be set to be $(BW * e^{MAX})$. e^{MAX} is a function of traffic pattern, bandwidth reservation, frame size, packet size, global synchronization latency, and so on, and it is generally difficult to obtain a tight bound. (Currently, we rely on simulation to get a tight bound.)

GSF-based integrated QoS framework: We can extend the domain of GSF arbitration into other QoS-capable shared resources, such as memory bandwidth at a shared memory controller. A single global deadline (frame number) assigned at the source could also be used to manage end-point bandwidth usage to create a GSF-based integrated QoS system. We leave development of this integrated QoS environment for future work.

6 Evaluation

In this section, we evaluate the performance of our proposed GSF implementation in terms of QoS guarantees and average latency and throughput. We also discuss tradeoffs in the choice of network parameters.

6.1 Simulation Setup

Table 2 summarizes default parameters for evaluation. We implemented a cycle-accurate network simulator based on the *booksim* simulator [28]. For each run, we simulate 0.5 million cycles unless the simulation output saturates early, with 50 thousand cycles spent in warming up.

We use an 8×8 2D mesh with four traffic patterns where the destination of each source at Node (i, j) is determined as follows: hotspot ($d_{(i,j)} = (8, 8)$), transpose ($d_{(i,j)} = (j, i)$), nearest neighbor ($d_{(i,j)} = (i + 1, j + 1) \pmod 8$) and uniform random ($d_{(i,j)} = (\texttt{random()}, \texttt{random()})$). Hotspot and uniform random represent two extremes of network usage in terms of load balance for a given amount of traffic. The other two model communication patterns found in real applications, e.g. FFT for transpose, and fluid dynamics simulation for nearest neighbor.

Simulation parameters	Specifications
Common parameters	
Topology	8x8 2D mesh
Routing	dimension-ordered
Router pipeline (per-hop latency)	VA/NRC - SA - ST (3 cycles)
Credit pipeline delay (including credit traversal)	2 cycles
Number of VCs per channel (V)	6
Buffer depth (B)	5 flits / VC
Channel capacity (C)	1 flit / cycle
VC/SW allocation scheme	iSlip [19] (baseline) or GSF
Packet size	1 or 9 flits (50-50 chance)
Traffic pattern	*variable*
GSF parameters	
active_frame_window_size (W)	same as number of VCs
frame_size (F)	1000 flits
global_barrier_latency (S)	16 cycles

Table 2: Default simulation parameters

6.2 Fair and Differentiated Services

We first evaluate the quality of guaranteed services in terms of bandwidth distribution. Figure 8 shows examples of fair and differentiated bandwidth allocation in accessing hotspot nodes. Figure 8 (a) and (b) illustrate QoS guarantees on a 8×8 mesh network and (c) on a 16×16 torus network. In both cases, the GSF network provides guaranteed QoS to each flow. We are able to achieve this without significantly increasing the complexity of the router partly because the complex task of prioritizing packets to provide guaranteed QoS is offloaded by the source injection process, which is globally orchestrated by a fast barrier network. We make the case for using a simple secondary network (barrier network) to control a primary high-bandwidth network to improve the efficiency of the primary network (i.e., to provide more sophisticated services in our case).

For all the simulation runs we performed, we confirmed that bandwidth is shared among all flows in compliance with the given allocation. Therefore, for the rest of this section, we focus on non-QoS aspects such as average throughput and tradeoffs in parameter choice.

6.3 Cost of Guaranteed QoS and Tradeoffs in Parameter Choice

The cost of guaranteed QoS with GSF is additional hardware including an on-chip barrier network and potential degradation of average latency and/or throughput. Un-

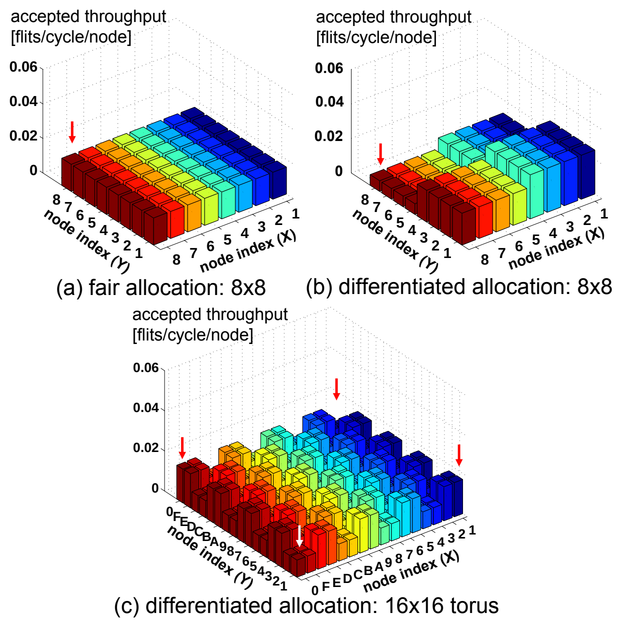


Figure 8: Fair and differentiated bandwidth allocation for hotspot traffic. (a) shows fair allocation among all flows sharing a hotspot resource located at (8,8). In (b), we partition a 8×8 CMP in mesh topology into 4 independent groups (e.g., running 4 copies of virtual machine) and provide differentiated services to them independent of their distance from the hotspot. In (c), we partition a 16×16 CMP in torus topology into 2×2 processor groups and allocate bandwidth to four hotspots in a checkerboard pattern.

less a router has *a priori* knowledge of future packet arrivals, it must reserve a certain number of buffers for future high-priority packets although there are waiting packets with lower priorities. This resource reservation is essential for guaranteed QoS but causes resource underutilization, degrading average-case performance. Therefore, it is our primary design goal to provide robust average-case performance over a wide range of network configurations and workloads. Note that robust performance by the GSF framework entails setting QoS parameters appropriately for a given workload.

Figure 9 shows the average latency versus offered load over three different traffic patterns. For uniform random traffic, we allocate $\lfloor F/64 \rfloor = 15$ flit injection slots per frame to each source (not to each source-destination pair), and these slots are shared by all packets from the same source. For the other traffic patterns, each source-destination pair is regarded as a distinct flow and allocated flit slots considering the link sharing pattern.

We first observe that the GSF network does not increase the average latency in the uncongested region. The network saturation throughput, which is defined as the point at which packet latency is three times the zero-load latency (as in [17]), is degraded by about 12% in the worst case. The performance impact of barrier synchronization overhead is the most visible in uniform random traffic because it has the lowest average epoch interval ($e^{AVG} \equiv (\sum_{k=0}^{N-1} e_k)/N$). Assuming 16-cycle barrier synchronization latency ($S = 16$), S/e^{AVG} ratios are 0.32, 0.15 and 0.09 for uniform random, transpose and nearest neighbor, respectively.

There are two main reasons for degradation of network saturation throughput: underutilization of the head frame VC (VC0) and finite frame window. Because VC0 at each node is reserved to drain packets in the head frame, only (V-1) VCs are available for packets in the other active frames. The finite frame window prevents a flow from injecting more traffic than its reserved flit slots in the active frame window even when there are unclaimed network resources.

Figure 10 explains the impact of these two factors on average accepted throughput. With a small number of virtual channels, e.g. V=2, the throughput gap between GSF and baseline is dominated by underutilization of VC0 and increasing the window size from V to $2V$ does not improve throughput much. As the number of VCs increases, the gap narrows, and the performance gain from a wider window becomes more significant. To have enough overlap of frames to achieve over 90% of the throughput of the baseline VC router, the number of VCs should be at least four in this network configuration. With 8 VCs, the GSF achieves a comparable network throughput at the cost of increased average latency. We choose the 6×5 (virtual channels $\times$ buffers) configuration by default.

In choosing the window size (W), a larger window is desirable to overlap more frames, thereby increasing overall network throughput. However, the performance gain only comes with a cost for more complex priority calculation and arbitration logic. According to our simulation, increasing W to be larger than $2V$ gives only marginal throughput gain. Therefore, we choose the frame window size (W) to be V by default as a reasonable design tradeoff.

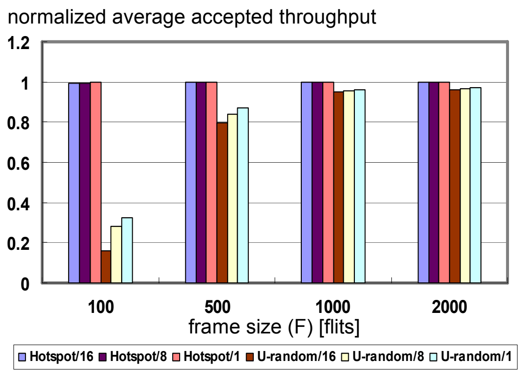


Figure 11: Throughput of GSF network normalized to that of the baseline VC router with variable F (frame window size). Two traffic patterns (hotspot and uniform random) and three synchronization costs (1, 8 and 16 cycles) are considered.

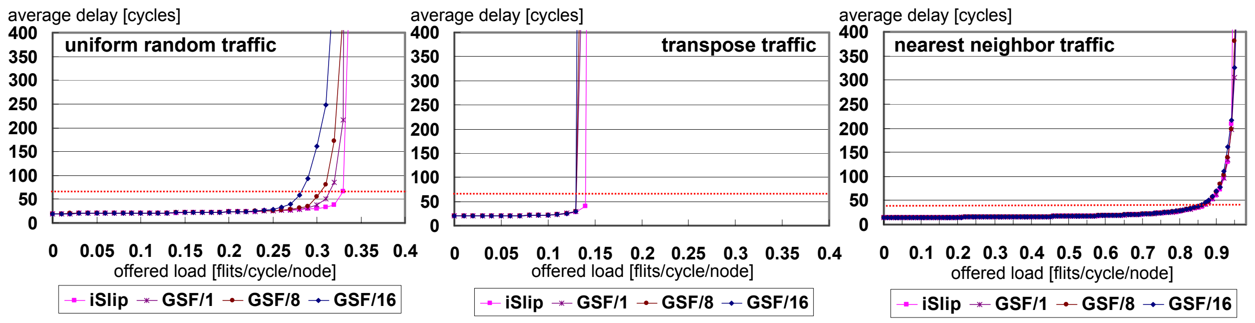


Figure 9: Average packet latency versus offered load with three traffic patterns. For each traffic pattern, we consider three different synchronization costs: 1 (GSF/1), 8 (GSF/8) and 16 cycles (GSF/16). Network saturation throughput is the cross point with dotted line ($3T_{zero-load}$ line) as in [17]. With GSF/16, network saturation throughput is degraded by 12.1 % (0.33 vs. 0.29) for uniform random, 6.7 % (0.15 vs. 0.14) for transpose and 1.1 % (0.88 vs. 0.87) for nearest neighbor, compared to baseline VC router with iSlip VC/SW allocation.

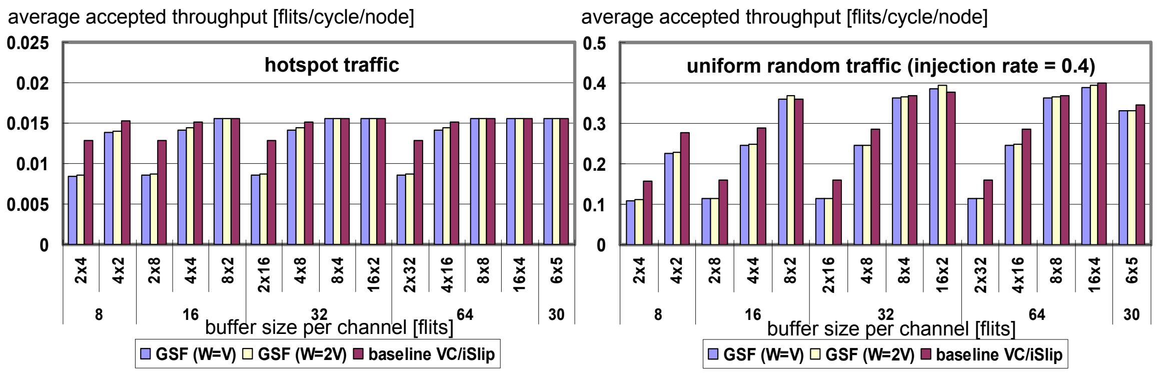


Figure 10: Tradeoff in buffer organization with hotspot and uniform random traffic patterns. VC buffer configuration is given by $V \times B$. The frame window size (W) is assumed to be V or 2V. For both traffic patterns, having $V \geq 4$ achieves 90% or higher throughput compared to the baseline VC router. Generally, increasing VCs improves the throughput at the cost of increased average latency. 6×5, our default, is a sweet spot for the specific router architecture we use.

In Figure 11, we explore the choice of frame size (F). A long frame (whose size is ≥ 1000 in this configuration) amortizes the overhead of barrier synchronization and effectively increases the size of injection window to support more bursty traffic, which is likely to improve the network throughput. The downside is larger source buffers and potential discrimination of remote nodes within a frame. The choice depends on workloads, synchronization overhead and system size.

7 Conclusion

In this paper, we introduced Globally-Synchronized Frames (GSF) to provide guaranteed QoS from on-chip networks in terms of minimum bandwidth and maximum delay bound. We show that the GSF algorithm can be easily implemented in a conventional VC router without significantly increasing its complexity. This is possible because the complex task of prioritizing packets to provide guaranteed QoS is pushed out to the source injection process at the end-points. The end-points and network are globally orchestrated by a fast barrier network made possible by the on-chip implementation. Our preliminary evaluation of the GSF network shows promising results for robust QoS support in on-chip networks.

8 Acknowledgements

We thank Christopher Batten and Heidi Pan at MIT, Rose Liu at the University of California, Berkeley, and Kyle Nesbit at the University of Wisconsin, Madison, as well as anonymous reviewers for their helpful comments. This work was partly funded by Nokia Inc.

References

[1] D. Abts and D. Weisser. Age-based packet arbitration in large-radix k-ary n-cubes. In *SC*, 2007.

[2] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. In *DATE*, 2005.

[3] B. Case. First Trimedia chip boards PCI bus. *Microprocessor Report*, Nov 1995.

[4] R. L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991.

[5] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[6] W. J. Dally. Virtual-channel flow control. *IEEE Trans. Parallel Distrib. Syst.*, 3(2):194–205, 1992.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM*, 1989.

[8] T. Felicijan and S. B. Furber. An asynchronous on-chip network router with quality-of-service (QoS) support. In *Proceedings IEEE International SOC Conference*, 2004.

[9] M. Galles. Spider: A high-speed network interconnect. *IEEE Micro*, 17(1):34–39, 1997.

[10] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Des. Test*, 22(5):414–421, 2005.

[11] F. Guo, H. Kannan, L. Zhao, R. Illikkal, R. Iyer, D. Newell, Y. Solihin, and C. Kozyrakis. From chaos to QoS: case studies in CMP resource management. *SIGARCH Comput. Archit. News*, 35(1):21–30, 2007.

[12] F. Guo, Y. Solihin, L. Zhao, and R. Iyer. A framework for providing quality of service in chip multi-processors. In *MICRO*, 2007.

[13] L. R. Hsu, S. K. Reinhardt, R. Iyer, and S. Makineni. Communist, utilitarian, and capitalist cache policies on CMPs: caches as a shared resource. In *PACT*, 2006.

[14] R. Iyer. CQoS: a framework for enabling QoS in shared caches of CMP platforms. In *ICS*, 2004.

[15] C. R. Johns and D. A. Brokenshire. Introduction to the Cell Broadband Engine architecture. *IBM J. Res. & Dev.*, 51(5), 2007.

[16] J. H. Kim and A. A. Chien. Rotating Combined Queueing (RCQ): Bandwidth and latency guarantees in low-cost, high-performance networks. In *ISCA*, 1996.

[17] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *ISCA*, 2007.

[18] J. W. Lee and K. Asanović. METERG: Measurement-based end-to-end performance estimation technique in QoS-capable multiprocessors. In *RTAS*, 2006.

[19] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw.*, 7(2):188–201, 1999.

[20] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *DATE*, 2004.

[21] T. Moscibroda and O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX Security*, 2007.

[22] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair queuing memory systems. In *MICRO*, 2006.

[23] K. J. Nesbit, J. Laudon, and J. E. Smith. Virtual private caches. In *ISCA*, 2007.

[24] K. J. Nesbit, J. Laudon, and J. E. Smith. Virtual Private Machines: A resource abstraction for multicore computer systems. In *University of Wisconsin - Madison, ECE TR 07-08*, 2007.

[25] R. S. Passint, G. M. Thorson, and T. Stremcha. United States Patent 6674720: Age-based network arbitration system and method, January 2004.

[26] G. E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *HPCA*, 2002.

[27] M. Thottethodi, A. R. Lebeck, and S. S. Mukherjee. Self-tuned congestion control for multiprocessor networks. In *HPCA*, 2001.

[28] B. Towles and W. J. Dally. Booksim 1.0. `http://cva.stanford.edu/books/ppin/`.

[29] J. S. Turner. New directions in communications (or which way to the information age). *IEEE Communications*, 24(10):8–15, Oct. 1986.

[30] W.-D. Weber, J. Chou, I. Swarbrick, and D. Wingard. A quality-of-service mechanism for interconnection networks in system-on-chips. In *DATE*, 2005.

[31] H. Zhang and S. Keshav. Comparison of rate-based service disciplines. In *SIGCOMM*, 1991.

[32] L. Zhang. Virtual Clock: a new traffic control algorithm for packet switching networks. In *SIGCOMM*, 1990.