SurfNoC: A Low Latency and Provably Non-Interfering Approach to Secure Networks-On-Chip

Hassan M. G. Wassel[†], Ying Gao[†], Jason K. Oberg[*],
Ted Huffmire[‡], Ryan Kastner[*], Frederic T. Chong[†], Timothy Sherwood[†]

[†]UC Santa Barbara [*]UC San Diego [‡]Naval Postgraduate School

hwassel@cs.ucsb.edu, yinggao@ece.ucsb.edu, jkoberg@cs.ucsd.edu,
tdhuffmi@nps.edu, kastner@cs.ucsd.edu, {chong, sherwood}@cs.ucsb.edu

ABSTRACT

As multicore processors find increasing adoption in domains such as aerospace and medical devices where failures have the potential to be catastrophic, strong performance isolation and security become first-class design constraints. When cores are used to run separate pieces of the system, strong time and space partitioning can help provide such guarantees. However, as the number of partitions or the asymmetry in partition bandwidth allocations grows, the additional latency incurred by time multiplexing the network can significantly impact performance.

In this paper, we introduce SurfNoC, an on-chip network that significantly reduces the latency incurred by temporal partitioning. By carefully scheduling the network into waves that flow across the interconnect, data from different domains carried by these waves are strictly non-interfering while avoiding the significant overheads associated with cycle-by-cycle time multiplexing. We describe the scheduling policy and router microarchitecture changes required, and evaluate the information-flow security of a synthesizable implementation through gate-level information flow analysis. When comparing our approach for varying numbers of domains and network sizes, we find that in many cases SurfNoC can reduce the latency overhead of implementing cycle-level non-interference by up to 85%.

Categories and Subject Descriptors

C.2.1 [**PROCESSOR ARCHITECTURES**]: Multiple Data Stream Architectures (Multiprocessors) Interconnection architectures ; C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time and embedded systems

General Terms: Design, Reliability, Security, Verification.

Keywords: High Assurance Systems, Networks-on-chip, Non-interference.

This work was funded in part by Grant No. CNS-1239567, CNS-1162187 and CCF-1117165. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the sponsoring agencies.

1. INTRODUCTION

Programmers are increasingly asked to manage a complex collection of computing elements including a variety of cores, accelerators, and special purpose functions. While these many-core architectures can be a boon for common case performance and power-efficiency, when an application demands a high degree of reliability or security the advantages become a little less clear. On one hand, the ability to spatially separate computations means that critical operations can be physically isolated from malicious or untrustworthy components. There are many advantages to providing physical separation which have been well explored in the literature [28, 40]. On the other hand, real systems are likely to use different subsets of cores and accelerators based on the needs of the application and thus will require a shared communication network. When a general-purpose interconnect is used, analyzing all the ways in which an attacker might influence the system becomes far more complicated. The problem is hard enough if we restrict ourselves to considering only average-case performance or packet ordering, but the difficulty of the problem increases even further if we attempt to *prevent even cycle-level variations.*

In high-assurance systems it is a common practice to break the system into a set of domains, which are to be kept separate. These domains should have *no effect* on one another. For example, the Mars Curiosity rover software runs on a RAD750 processor, a single-core radiation-hardened version of the Power architecture with a special-purpose separation kernel [1]. The kernel partitions the tasks, such as guidance, navigation and the various science packages from one another to help prevent cascading failures. Future space missions are looking to use multicore systems [23, 33], which adds another layer of communication, but there are serious concerns about the introduction of opportunities for interference between system components [25, 26].

The problem is that typical networks-on-chip have many internal resources that are shared between packets from different domains, which we would otherwise wish to keep separate. These resources include the buffers holding the packets, the crossbar switches, and the individual ports and channels. Such resource contention introduces "interference" between these different domains, which can create a performance impact on some flows, pose a security threat by creating an opportunity for timing channels [37], and generally complicates the final verification and certification process of the system because *all* of the ways in which that interaction might occur must be accounted for. Non-interference means that injection of packets from one domain cannot affect the timing of delivery of packets from other domains.

These concerns are similar to, but distinct from, the problem of providing quality-of-service guarantees. While QoS can minimize the performance impact of sharing between domains by providing a minimum guaranteed level of service for each domain (or class) [12, 13, 19, 14], as shown by Wang and Suh, quality of service techniques will still allow timing variations and thus do not truly support non-interference [37]. The only way to be certain that the domains are non-interfering is to statically schedule the domains on the network over time. However, a straightforward application of time multiplexing leads to significant increases in latencies as each link in the network is now time multiplexed between many different domains.

The core idea behind our approach is that, if a strictly time multiplexed link is seen as an oscillating behavior, we can stagger the *phases* of these oscillations across the network such that a set of "waves" is created. As these waves traverse the network they provide an opportunity for packets of the corresponding domain to travel unimpeded along with these waves (thus avoiding excessive latency), while still requiring no dynamic scheduling between domains (thus preventing timing corruption or information leakage). Channels in the same dimension and direction appear to "propagate" different domains such that after passing through the pipeline of the router, the channel is ready to forward a packet coming from the same dimension and domain without any additional wait (unless there is contention from packets of the same domain). In this way packets "surf" the waves in each dimension. We identify the many potential challenges of achieving non-interference in a modern network-on-chip router microarchitecture using gate-level analysis, and we discuss the details and ramifications of our surf scheduling methodology, and we argue that our approach truly does not allow even cycle-level cross-domain interference. Specifically in this paper:

1. We present a channel scheduling scheme and network router design which simultaneously supports both low-latency packet-switched operation and non-interference between domains.

2. We show that as the network grows in size, as the number of domains increases, and as the asymmetry between domains becomes larger, the benefit for a surf-scheduled network over TDMA continues to increase.

3. We evaluate the latency, throughput, area, and power consumption of these approaches through a detailed network simulation.

4. Finally, we argue that the technique is truly sound through an analysis of the router micro-architecture and with the help of formal verification via gate-level information flow analysis.

The rest of the paper is organized as follows. We begin with a discussion of related work and how our proposed solution fits in the design space in Section 2. Next, in Section 3, we describe the core idea behind the SurfNoC schedule followed by a detailed router micro-architecture discussion in Section 4. Section 5 presents the evaluation of the system and explores the relationship between domains, partition asymmetry, and scheduling. Then, we provide a gate-level information-flow analysis in Section 6. Finally, Section 7 concludes the paper with our final thoughts and a discussion of future directions.

2. RELATED WORK

Our proposed solution to non-interference in NoCs touches on many problems that have been proposed in the literature, such as timing channels in micro architecture, QoS in networks-on-chip, fault-containment and composability in system-on-chips, and security in NoCs. In this section, we will try to review some of this related work and show how our work fits in the design space.

Timing Channels and Non-interference in Micro-architecture: There has been a recent renewed interest in the analysis of timing channel attacks and mitigations through micro-architecture state such as cache interference [2, 38, 39] and branch predictors [3, 4]. One approach to these problems is a technique that can verify non-interference of hardware/software systems (including high-performance features such as pipelining and caching) using gate-level information flow tracking [36, 34, 35]. More recently, a NoC timing channel protection scheme for a system with security lattices was been proposed [37]. This paper proposes a priority-based arbitration scheme to allow packets with LOW labels to always win arbitration (except when they reach a pre-specified quota during each system epoch to prevent denial-of-service attacks from the LOW domain). This ensures that information cannot flow from the domain with a HIGH label to the domain with a LOW label, but allows for information flow in the other direction. It can be extended to multiple security labels as long as they form a lattice. In this work, we propose a technique that assures multi-way non-interference in NoCs with low latency overhead to allow for verification of high-assurance systems such as those in aerospace and automotive systems.

QoS in Network-on-chips: Techniques for achieving NoC quality-of-service guarantees have been proposed based on solutions to analogous problems in macro-scale networks. These approaches for the most part attempt to limit the rates of each flow [12, 13, 19, 14]. However, quality-of-service guarantees are known to be not sufficient for timing channel protection [37]. Optimizations that allow flows to go over their designated rate when uncontended and the lack of fault containment are problematic for high-assurance systems [28] because of the high cost of any unaccounted variation in such systems. The time division approach proposed here provides for both fault containment and timing channel elimination.

Security in NoCs: Security in NoCs has been studied from several aspects that focus on specific attack mitigations such as defending against denial-of-service (DoS), battery-draining attacks [9] and maintaining access control of specific memory regions in shared- memory systems [9, 27], and buffer overflow attacks [20, 21]. Gebotys and Zhang have focused on confidentiality by providing encryption techniques for data transmitted over the NoC in a SoC setting [10].

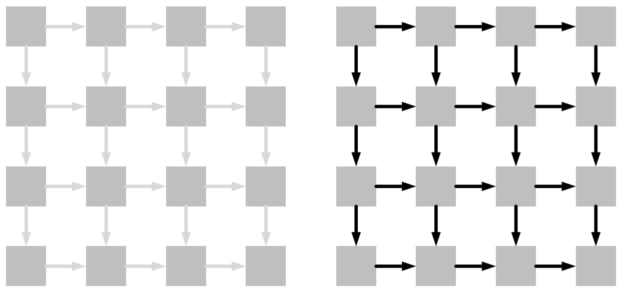

(a) Odd cycles. (b) Even cycles

Figure 1: Time-division multiplexing scheduling in a 16-node 2D mesh (only one direction of channels is shown for illustration purposes).

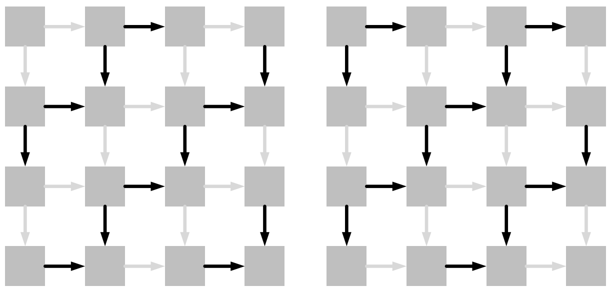

(a) Odd cycles (b) Even cycles

Figure 2: Surf scheduling in a 16-node 2D mesh (only one direction of channels is shown for illustration purposes).

Availability is handled in the Tile64 iMesh networks by separating (and in fact physically separating) the network accessible by user applications from the network used by the OS and IO device traffic [40]. Our scheme can protect against DoS and bandwidth depletion attacks between domains because of the static time allocation to different domains.

Non-interference in NoCs: Non-interference in NoCs has been studied in the system-on-chip domain to provide composability and fault containment as well as predictability of latency for real-time performance guarantees [15, 24]. Composability means that the system can be analyzed as a set of independent components, which allows for easier verification of the overall system without having to verify all possible interleavings of events in the system. This has been especially critical in high-assurance systems that require a very high level of verification because of the safety ramifications of the system. Æthearal proposed a time-division multiplexed (TDM) virtual circuit switching network to provide guaranteed services (GS) for performance-critical applications with real-time deadlines and a packet-switched best-effort (BE) network for applications with fewer requirements [11]. A lighter version that only provides GS was proposed to further simplify routers [31, 16]. More recently, Stefan and Goossens proposed a modification to Æthearal that enables multi-path routing, both static and dynamic (based on a true random number generator), in order to enhance security by using a non-deterministic path instead of the source-routing used in Æthearal [30]. In addition, the need for real-time worst-case execution time (WCET) analysis inspired a set of work, such as, the T-CREST project, which tries to build a time-predictable multi-core for real-time applications. They proposed an integer programming technique to minimize the length of static schedule of all-to-all circuit switching connections in a TDM way [29]. Bui et al. proposed an on-time network-on-chip using real-time packet scheduling, admission control, and run-time path configuration [6]. We believe that such an admission control technique is orthogonal to SurfNoC. SurfNoC can be augmented by an admission control mechanism to provide time-predictable packet delivery.

Regarding packet-switching networks, the Avici TSR [8] uses separate virtual channels for each destination in the network, but packets destined to different locations share physical channels. Under saturation, physical channels are allocated fairly, but destinations can go over their fair share when the network is not saturated, which can leak information by detecting the variation of bandwidth a certain node receives.

To the best of our knowledge, our scheme is the first to provide a packet-switched network that can guarantee two-way (or multi-way) non-interference and timing channel protection in a way that is both a) provable down to the gate-level implementation and b) provides low-latency overhead.

3. SURFNOC ARCHITECTURE

3.1 A Motivating Example

Consider the 16-node half-mesh network (channels are drawn in one direction left-to-right and top-down for illustration purposes) in Figure 1, assuming that even nodes belong to domain 0 and odd nodes are part of domain 1. A straightforward way to support non-interference is by partitioning the virtual channels and time-multiplexing the physical channels and crossbars between different domains such that channels are only allowed to propagate packets from domain 0 (black) on even cycles and packets from domain 1 (grey) on odd cycles (assuming a single-cycle routers), as shown in Figures 1a and 1b. This time-multiplexing scheme ensures that the latency and throughput of each domain is completely independent of the other domain's load. However, this baseline scheme means that packets will have to wait an extra cycle at each hop. Even worse, as we scale the number of domains from 1 to D, assuming a single-cycle router, each packet will have to wait $D - 1$ cycles per hop. This is a high price to pay, and one that continues to get worse the further away you attempt to communicate. If we want to maintain to non-interference, we will still need these strict time-varying partitions, but by changing the phase of their oscillations we can dramatically reduce the latencies involved.

A better schedule for time-multiplexing will make sure that domains wash over the network as a wave, such that each dimension appears to be "propagating" one domain in a pipelined fashion. Figure 2a shows a simplified view of this point. Every link still rotates evenly through domain 0 and domain 1, but if we consider the top row in Figure 2a, we can see alternating channels (grey, black, grey). In the next cycle (shown in Figure 2b, the channels used to propagate packets from domain 0 (black) will carry packets from domain 1 (grey), and vice versa.

Before entering the network, the packet waits in the injection port until its domain's turn. The schedule ensures that when the packet is ready to egress the router, there will be no delay waiting for its domain's turn at the downstream router. The only exceptions to this rule are when a packet needs to change dimensions (such as when the packet turns from traveling along the X dimension to the Y dimension) and when there is contention from packets in the *same* domain.

As an optimization, we constrain our schedule such that two directions of the router propagate packets from the same domain at the same time. For example, the top-left router in Figure 2b propagates packets from domain 0 (black) both to the right and down. In this case, any packet which is sent in a downward and/or rightward direction will only have to wait to enter the network and will have no additional waits during turns between dimensions (again, unless there is intra-domain contention). Of course, this example is very simple as it has only two domains, even divisions, and does not consider the latency of the network routers. In the next section, we will show how to devise a detailed strategy for k-ary n-cube meshes and tori networks and discuss how non-interference can be shown at the level of an implementation.

3.2 SurfNoC Scheduling

The most basic routing algorithm in meshes and tori is dimension-ordered routing. That is, a packet walks through a dimension until it cannot move further without going farther from the destination and then transfers to an other dimension. Thus, routing is linear in each dimension, which provides an opportunity to reduce wait time between hops. This way, packets will only have to wait when they enter (exit) the network from (to) the injection (ejection) channel and when they change dimensions. We will describe this idea in detail in the rest of this section.

The straightforward way to support time-division multiplexing is to operate the whole network in time slices that are divided between application domains. That is a packet waits at each hop until the network is forwarding packets from its domain. This approach leads to a zero-load latency L_0 that is proportional to the number of application domains D, pipeline depth P, and the number of hops H, as shown in Equation 1. This solution might work efficiently for a small number of domains such as two to four domains, but in high-assurance applications as many as tens or hundreds of domains can be found [28].

$$T_0 = HP + H(D-1) \tag{1}$$

Building on the technique we developed in the motivating example, we propose SurfNoC scheduling, in which different routers (and in fact different ports of the same router) can forward packets from different domains in the same cycle. In this schedule, a packet waits until it can be forwarded in one dimension (i.e., its output channel is forwarding packets from its domain in this cycle) and then does not experience any wait at any downstream router in this dimension (assuming there is no contention from packets from the same domain) in a way similar to the schedule developed in the half-mesh example. After finishing the first dimension, the packet may experience another wait until it can be forwarded in the next dimension. We call this schedule Surf scheduling because a packet is like a surfer who waits to "ride" a wave until some location and then waits to "ride" another wave. In this analogy, waves are dimension pipelines. Equation 2 shows the maximum zero-load latency and clearly shows that the overhead is additive not multiplicative as in the straightforward way. The term $(n-1+2)$ comes from $n-1$ transitions between dimensions and two waits during injection and ejection. It is worth noting that this is the maximum wait, not the typical one, as the schedule may require less wait.

$$T_{0max} = HP + ((n-1)+2)(D-1) \tag{2}$$

The way to implement these different "waves" is by scheduling different directions in a router independently; an idea inspired by dimension-slicing used in dimension-ordered routing in meshes and tori. We used what we call direction-slicing of the pipelines, such that each direction has its own pipeline. This pipeline is a *virtual* one going through different routers (not in the same router). We will describe this idea in the case of a 2D mesh or torus.

In a 2D mesh or torus, each dimension has two directions (E and W for the x-dimension; N and S for the y-dimension). The pipelines of directions of the same dimension (i.e. N, S and E, W) are running in opposite directions as shown in Figure 3. In this technique, each port of a router is scheduled independently of all other ports in a pipelined way such that the downstream router in the same direction will forward packets from the same domain after P cycles where P is the pipeline depth of the router. These schedules are imposed on output channels of each router to avoid timing channels based on contention in the allocator (as detailed in the next section).

Figure 3 illustrates an example of a 16-node 2D mesh schedule of three domains (colored white, grey, black). There are two waves south-east (SE) (as the one shown in Figure 2) and north-west (NW) running in the mesh. Each channel propagates packets according to the following schedule (white, white, gray, and black) and repeats. It is worth noting that using such a schedule results in half of the bandwidth being allocated to the white domain, whereas the black and grey domains guarantee only a quarter of the bandwidth for each of them. This illustrates the benefit of our schedule in statically allocating non-uniform bandwidth to domains.

4. ROUTER MICROARCHITECTURE

The micro-architecture of the SurfNoC router has two main goals:

1. Ensuring a timing-channel-free contention between packets, i.e., contention can occur between packets from the same domain but not between packets from different domains;
2. Scheduling the output channels of each router in a way that maintains the surf schedule across the whole network;

In order to achieve these two goals, we used a dynamic number of virtual channels that are partitioned between domains independent of load (§4.1). We analyzed the VC allocator and switch allocators to make sure they are timing-channel free (§4.2). The scheduling of output channels is done through masking requests to the switch allocator from packets until its turn to use the output channel arrives in the wave pipeline (§4.3).

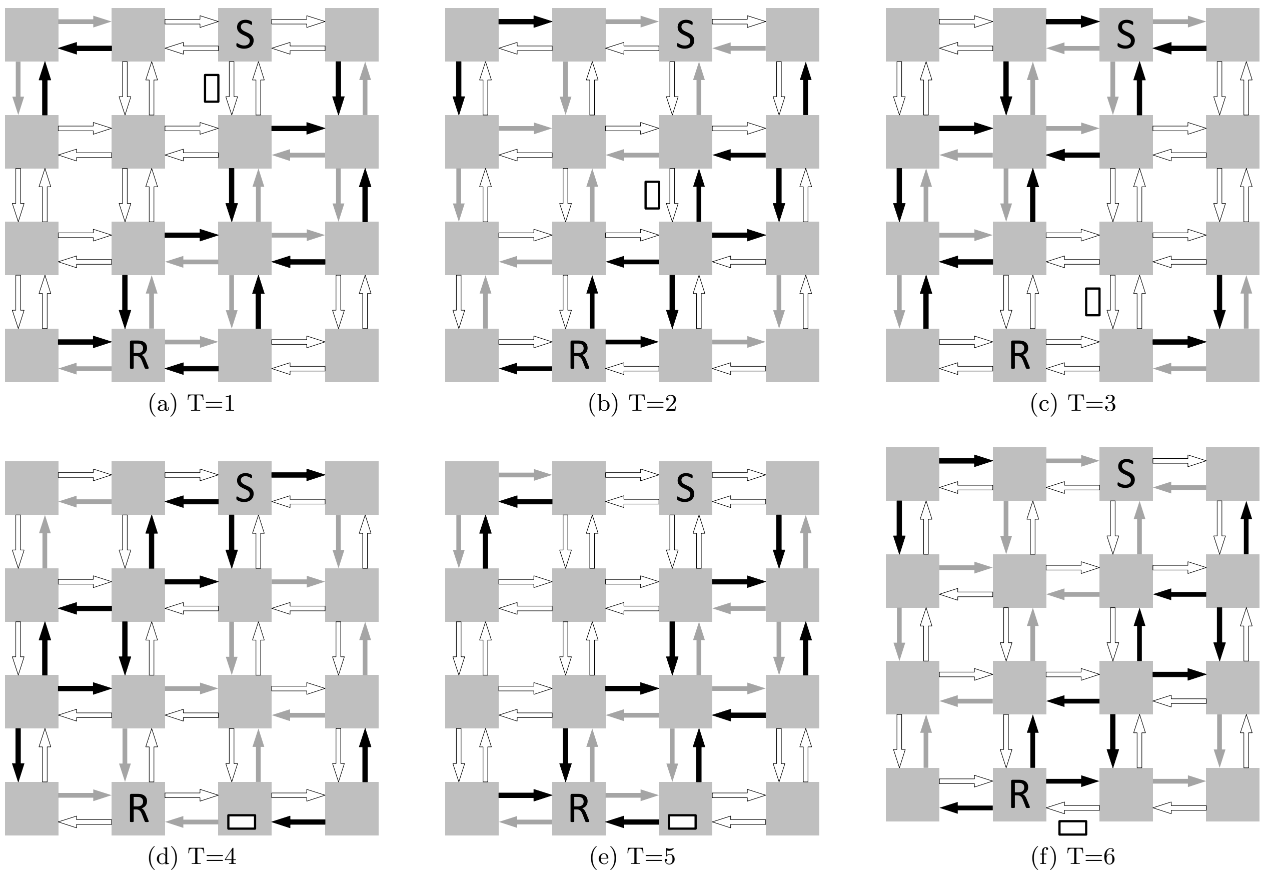


Figure 3: Surf scheduling in a 16-node 2D mesh with three application domains (denoted by white, grey, and black) assuming single-cycle routers for the purpose of illustration. The schedule runs as white, white, grey, and black and repeats, giving the white domain half the bandwidth. A packet (the white box under the node S) belonging to the white domain is sent from the node marked by S to the node marked by R. The figure contains six consecutive cycles. At $T = 1$, the packet is forwarded on the S port in the y-dimension (which is scheduled to forward white packets). It keeps moving in the y-dimension until $T = 3$ when it needs to move in the x-dimension on the W port. The packet waits 2 cycles (T=4 and T= 5) until it is the white domain's turn on the W port, and finally it is forwarded to its destination on $T = 6$. Another wait may happen again in the destination router (R) to forward the packet on the ejection port waiting for the white domain's turn.

4.1 Partitioning Virtual Channels

Spatial partitioning of the queues is not a new idea [37, 8]. Static partitioning of virtual channels is done through restricting the routing algorithm so that it generates output virtual channels in the range allowed for the domain of the packet. This partitioning ensures non-interference between packets from different domains while they wait in the buffers before being forwarded, i.e., eliminating the head-of-line (HOL) problem between domains.

4.2 Allocators

The SurfNoC router has two allocators, the VC allocator and the SW allocator. We used a separable allocator as the baseline allocator. These allocators use round-robin arbiters. This may lead to timing channels if requests are allowed from different domains to the same resource. We will detail how we prevent that from happening for both allocators.

Virtual Channel Allocators: The requesters of the VC allocator are packets requesting the upstream router virtual channels. The resources are virtual channels of the upstream routers. By restricting the routing circuit to only issue requests for virtual channels that belong to the corresponding domain, contention is guaranteed to be between packets from the same domain. Actually, we can use this property to reconstruct the VC allocator to be D VC allocators of size $v \times v$, where D is the number of domains, and v is the number of virtual channels per domain (across ports, not per port) instead of one large VC allocator of size $V \times V$, where $V = D.v$. This design can help save power by power-gating some of these allocators if the number of required domains is less than D for a certain application. Figure 4 depicts an example of a 3×3 VC allocator and illustrates the rationale behind the non-interference support in the VA stage as well as the optimization of separate D allocators. This also shows that we can use any arbiter or allocator design for VC allocation because it is intrinsically interference-free.

Switch Allocator: The SW allocator assigns output ports to virtual channels. Since any virtual channel can request any port, we cannot apply the same technique we used for

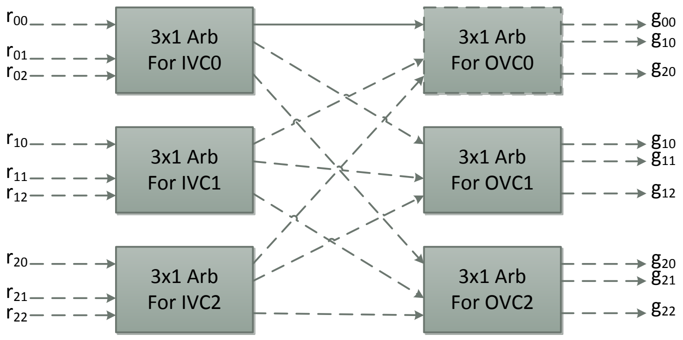
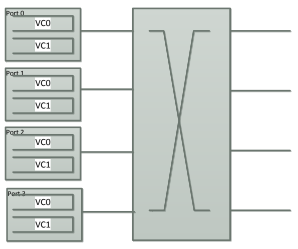
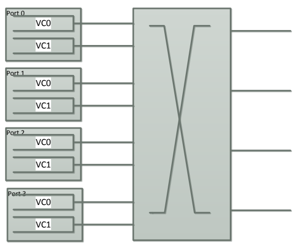


Figure 4: Virtual channel allocator: A 3x3 separable input-first VC allocator. In this example, we assume that VC0 and VC2 are assigned to domain 0 and VC1 is assigned to domain 1. Dashed lines show signals that can never be 1 due to route computation restrictions. This example shows that we can reconstruct the allocator into smaller ones.



(a) No speedup



(b) Crossbar with input speedup

Figure 5: Crossbar with input speedup to eliminate contention on switch input port between VCs from different domains.

the VC allocator of dividing the allocator into separate smaller allocators. Another problem arises from the fact that switch ports are shared among virtual channels from different domains (as shown in Figure 5a), which means that requests to the switch can be denied if two VCs (belonging to two different domains) on the same input port and requesting two legitimate (according to the surf schedule) output ports will contend on the crossbar input port leading to one of them being delayed, and thus a timing channel exists. We can solve this problem by using the input speedup parameter of the crossbar with value D, and hence no contention between domains on switch input ports exists. Figure 5b shows an example of such configuration.

By solving the input port request of the allocator, we can now design the switch allocator as a separable one of size $Dp \times p$ where p is the number of ports of the router. It is worth noting that it does not matter if the allocator is input-first or output-first because of two reasons. First, an input arbiter is responsible for one input to the crossbar that is shared between VCs from the same domain. Second, by using dimensional order routing and the surf scheduling, a VC can request only one output port. Requests to an output port are masked using the scheduler state so that only requests from the domain which owns the current time slot reaches the allocator (i.e., no contention between different domains can happen in the output arbiter).

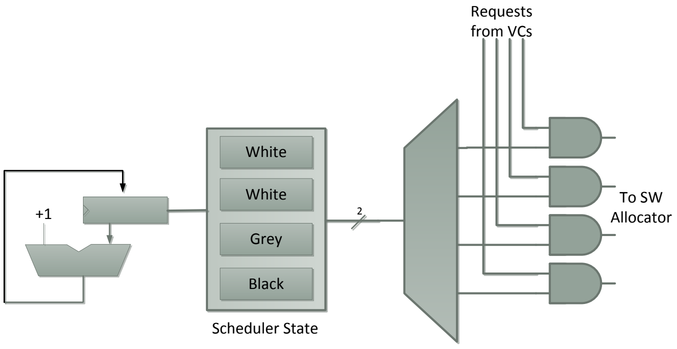


Figure 6: Scheduler: The scheduler output is used to mask requests to the switch output ports according to the surf schedule.

4.3 Scheduler

The scheduler is a set of p tables each indexed by a counter, one for each router output port. The initial state of the counter is pre-determined at design time in order to enforce the surf schedule. The number of slots in the tables is determined by the number of domains. The selected element from the array is used as input to a decoder. The decoder output is used to mask requests to the switch allocator as shown in Figure 6. If the number of domains D is greater than the router pipeline depth (including channel traversal) P, the schedule table is initialized according to Equation 3, where $S_i[d]$ is the schedule of port i at index d, l is the location of the node in the dimension of port i, and l' is the location of the node in the other dimension.

$$S_i[d] = \begin{cases} ((D-P)(l+l')+d) \bmod D & \text{if } i \in \{0,2\} \\ (-(D-P)(l+l')+d) \bmod D & \text{if } i \in \{1,3\} \end{cases} \tag{3}$$

4.4 Pipelining and separation discussion

We have so far discussed separation with respect to each pipeline stage separately, but the question remains whether pipelining and pipeline stalls can cause interference or not. We will discuss each pipeline stage, and the basic idea is to ensure that stalls do not induce interference between separate domains.

Buffer write and route computation (BW/RC): This stage is the first stage of the pipeline, and because we are assuming a credit-based flow control, flits do not enter the router unless there is a guaranteed space in the buffer for them. Spatial separation is ensured because VC allocation is done in the upstream router. Route computation can be done in parallel for all flits at the front of all virtual channels (waiting for RC). No interference can be caused in this stage.

Virtual channel allocation (VA): At this stage, all flits send requests to the VC allocator. Using our design, interference can happen between virtual channels from the same domain but not between those from distinct domains. Stalled flits because of lack of free virtual channels (in the downstream router) prevent only flits from the same virtual channel from making progress. This can be ensured by recording state in the pipeline for each virtual channel, i.e., stalls due to virtual channel allocation have to be per virtual channel (not per input port).

Parameter	Baseline-small	Baseline-fast	Surf and TDMA
VCs	12	32	See Table 2
Buffers per VC	4	4	See Table 2
Input Speedup	1	32	See Table 2
Flits per packet	1		
Router delay	4 cycles		
SW and VC Allocators	Seperable (input-first)		
Routing	DoR		

Domains	1	2	4	8	16	32
Number of VCs per port	16	16	16	32	32	32
Number of flits per VCs	8	8	8	4	4	4
Input speedup	1	2	4	8	16	32

Table 2: Different configurations

Switch allocation (SA): Switch allocation can fail, due to contending flits for switch ports (limited to virtual channels from the same domain), which causes stalls in the pipeline. We avoid stalling the whole port (which leads to interference between domains) by having a separate state in the pipeline stage for each virtual channel. Switch allocation can also be stalled because of lack of buffering in the downstream router, i.e., waiting for a credit. The effect of this stall is limited to a single virtual channel, and can be handled by the same way we addressed a stall resulted from a failed SW allocation.

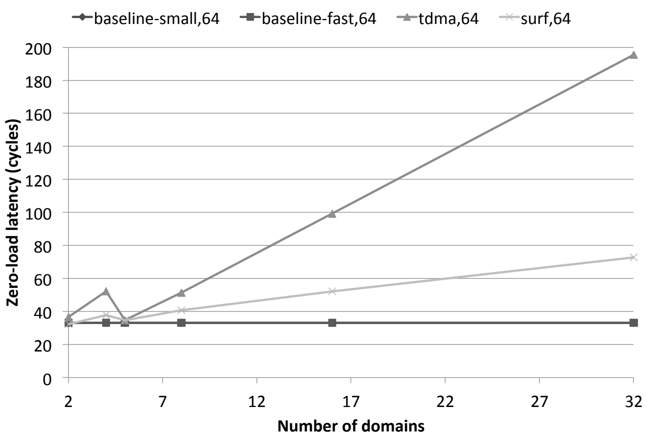
The key idea here is that stalls can affect flits in the stalled stage and all previous stages only from the same virtual channel. Thus, we can guarantee separation because we statically assign virtual channels to domains.

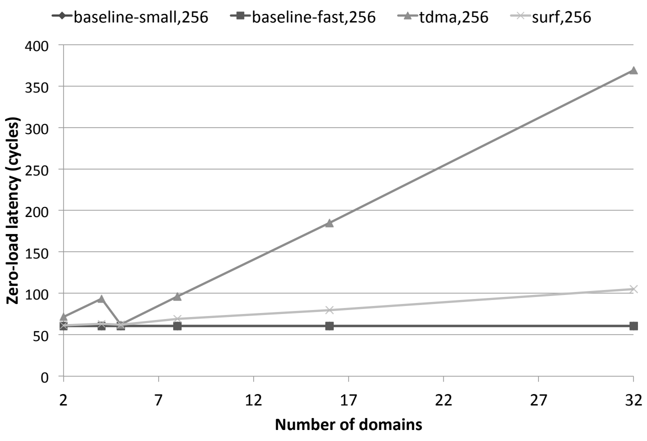
5. EVALUATION

In this section, we evaluate the performance and separation features of our SurfNoC scheme. We also evaluate the area and power overhead compared to a mesh network without non-interference support.

5.1 Experimental setup

We implemented a model of the SurfNoC router in BookSim 2.0 [7], a cycle-level interconnection network simulator. The simulator is warmed up until steady state is reached and statistics are reset, then a sample of the packets is measured from the time it enters the source queue until it is received. For latency measurements, the simulation runs until all packets under measurement leave the network. Table 1 provides the simulation parameters used for different schemes. We evaluated four schemes, two which do not provide separation guarantees, while the other two support strong separation. The non-separation baselines are an input-queued router with minimal resources, which achieves almost 40% saturation throughput (*Baseline-small*), and a similar router but with much more resources (buffers and input-speedup in the crossbar switch), which we call *Baseline-fast*. We chose to use two baselines because the separation





(b) 256 Nodes

Figure 7: Zero-load latency for different network size and different number of security domains (the two baselines are overlapped because zero-load latency does not depend on buffers and crossbar input speedup).

supporting router includes more resources and would achieve more throughput than a baseline with minimal area, which will hide the lost throughput due to the static scheduling. The non-interference supporting schemes are a straightforward (*TDMA*) (the whole network forwards packets from the same domain) and an input-queued router which enforces the surf schedule (*Surf*). Table 2 shows the different configurations used for different numbers of domains for Surf and TDMA.

5.2 Impact on latency

We first examine the impact of our non-interference support on latency with a different number of domains, and a different number of nodes under the uniform random traffic pattern. In order to understand the effect of time-division multiplexing of channels, we measure zero-load latency (latency at offered load of 0.1% of capacity for only one domain) and plot it for different numbers of domains in Figure 7. In this figure, we plot latency in cycles (y-axis) vs. number of domains on the x-axis for two network sizes of 64-nodes (Figure 7a) and 256-nodes (Figure 7b). We compare four configurations: baseline-small, baseline-fast, TDMA and Surf. It is clear that the latency overhead of Surf scales much bet-

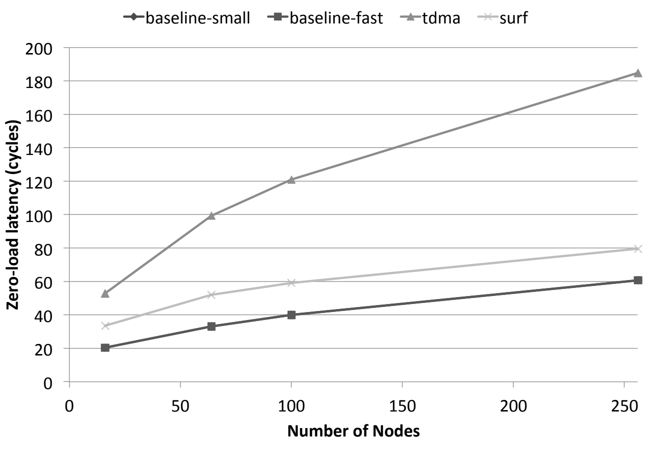
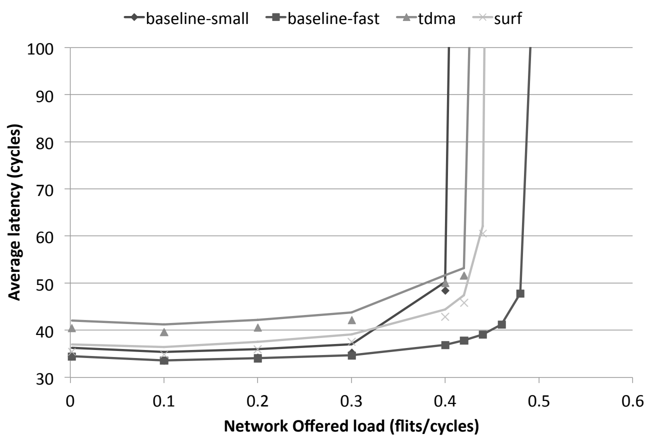
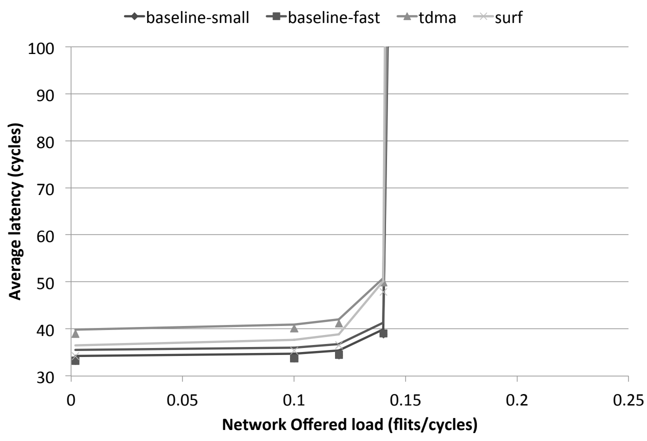


Figure 8: Zero-load latency versus different network size with 16 domains (the two baselines are overlapped because zero-load latency does not depend on buffers and crossbar input speedup).





(b) Transpose (2 Domains)

Figure 9: Average latency as a function of aggregate offered load of all domains for 2D mesh network of 64 Nodes: We can see that latency is stable below the network saturation point.

ter than TDMA for the same network size (for example, the overhead is reduced the overhead from 66 (19.1) to 19 (4.6) cycles by 71.3% (75.8%) for network sizes of 64 nodes with 16 (4) domains. The savings is even greater (up to 84.7%) for a 256-node network. We can see that there is one exception to this reduction in latency which happens for five domains. It is a subtle case that happens only for five domains, because the packet leaves the router after one cycle of switch traversal (ST), spends one cycle for link traversal (LT) and after two cycles of buffer write (BW) and virtual channel allocation (VA) in the upstream router (total of four cycles during which the upstream router propagates packets form other domains), it becomes ready for switch allocation (SA) without any wait using TDMA leading to the same latency overhead of surf scheduling. One would also notice that the benefits are higher for larger networks because of the increased average number of hops. We can conclude that, in general, the savings of surf scheduling is more scalable with larger networks as well as a higher number of domains.

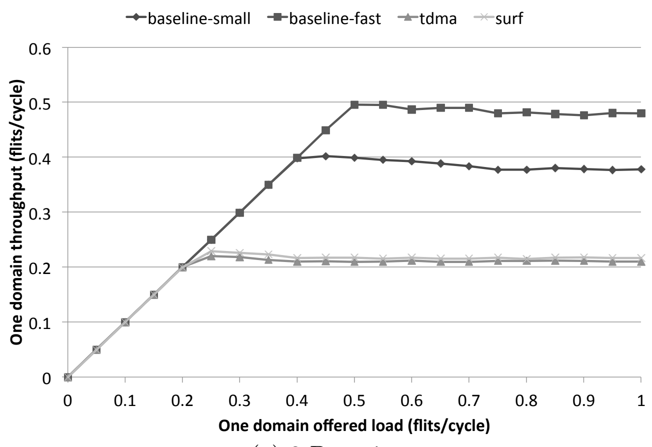
In order to clearly understand how the overhead scales with network sizes or average number of hops, we re-plotted zero-load latency of 2D mesh networks of sizes varying from 16 to 256 nodes with 16-domains under the uniform random traffic pattern. It is clear that the latency of both baselines increases with network size due to a higher average number of hops. We can see that the overhead of surf scheduling is almost independent of network size (average number of hops), leading to a line parallel to the baseline with constant overhead of 19 cycles (except for 16-nodes) because the packet wait-time depends only on the number of dimensions and the number of domains. On the other hand, the larger the network, the higher the overhead for TDMA scheduling because a packet has to wait for its turn at each hop in the path to its destination. This clearly shows that our scheme is scalable with network size and proves our intuition of latency overhead independence of number of hops.

Zero-load latency is just one latency metric; thus, we now study latency as a function of network offered load. Figure 9 shows average latency measured after convergence as a function of offered load for a 2D mesh network of 64-nodes under uniform random and transpose traffic patterns. We vary aggregate offered load on the x-axis, i.e., if we have D domains, the value of the x-axis is the sum of offered load of all D domains. We used two domains in this experiment. We can see that surf scheduling maintains its latency savings at all offered load values lower that the saturation point of the network. We can also see that loss saturation bandwidth of the separation supporting networks is small compared to that of the baseline-fast configuration. We will examine individual domain throughput of the network in the next section.

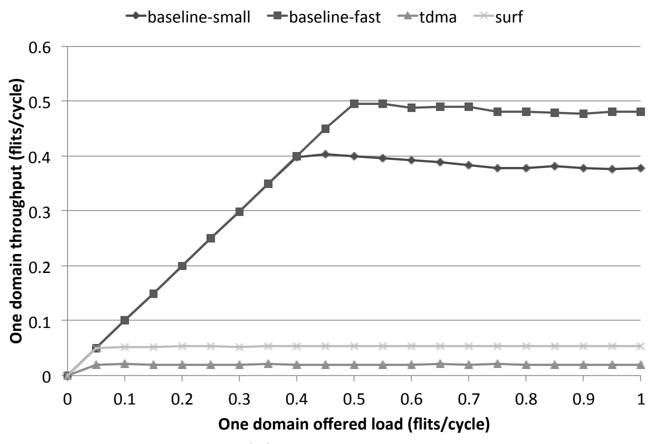
5.3 Throughput

We want to understand the effect of non-interference on throughput from three perspectives: single-domain throughput, aggregate network throughput and single-domain throughput independence of other domains load. We checked these properties for a 2D mesh network with 64-nodes with two and sixteen domains.

Figure 10 shows the effect of supporting non-interference on single-domain throughput for the two schemes: TDMA and Surf. We observe that before the saturation point, the throughput of a single domain (only one domain is allowed

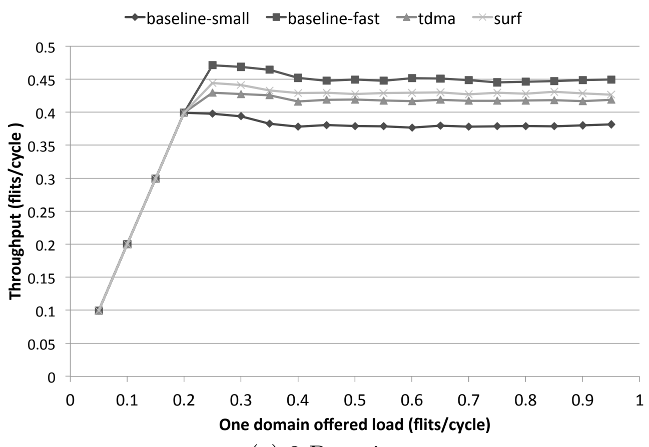


(a) 2 Domains

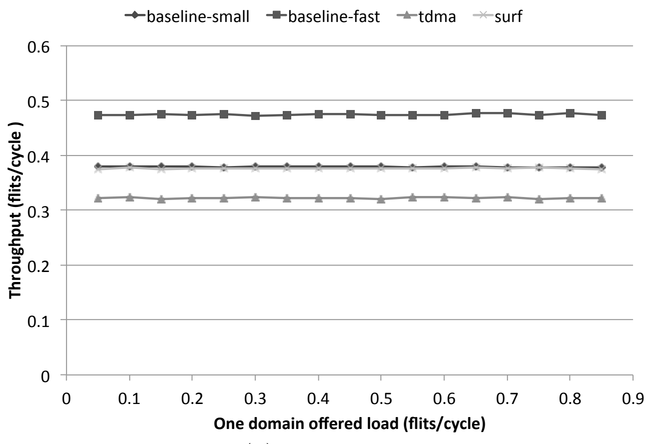


(b) 16 Domains

Figure 10: Throughput as a function of offered load of one domain (only one domain is injecting) for 2D 64-nodes mesh using different number of domains.



(a) 2 Domains



(b) 16 Domains

Figure 11: Aggregate network throughput as a function of offered load of one domain (all domains are injecting packets) for 2D 64-nodes mesh using different number of domains.

to inject packets in the network regardless of the number of domains) is exactly the same as if the network is not partitioned. However, we can also see that one domain saturation throughput is inversely proportional to the number of domains. In fact, it is almost half (one-sixteenth) of the saturation throughput of the baseline configuration using the same resources (buffers and input speedup of the switch), as can be seen in Figure 10a (10b) for two (sixteen) domains. This even distribution of bandwidth is expected because of uniformly dividing the virtual channels among domains and time-division multiplexing of channels.

In order to understand the effect of separation on aggregate throughput of the whole network, we run an experiment varying offered load of all domains from zero to one and measuring the aggregate network throughput (average number of packets received during a certain time slot) of all domains for all configurations. The results are plotted in Figure 11 for 2 and 16 domains. In this experiment "baseline-fast" uses the same buffer and input speedup values of the separation-supporting configurations(TDMA and Surf) in order to measure the performance loss due to non-interference support using the same set of resources. Although we can see that saturation throughput is reduced by around 11.7%, aggregate throughput loss is only limited to 4.9% and 20.5% for 2 and 16 domains, respectively. Figure 11a clearly shows that the network can operate when offered load is below saturation throughput without any performance loss. Non-interference configurations have higher saturation throughput than the small baseline because they use more resources, and lower than the fast baseline that includes the same resources because of unused time slots due to schedule enforcement. Moreover, we can see in Figure 10b that if all domains are trying to inject packets at just 10% of the network capacity, the network reaches saturation, leading to increased latencies. This can be tackled by non-uniformly allocating the bandwidth according to application-specific requirements.

Non-uniform bandwidth allocation: In order to verify the benefits of assigning bandwidth non-uniformly, we performed an experiment on a 2D mesh network with 64 nodes and three domains. Bandwidth (VCs and time slots in the schedule) is assigned as follows: a quarter of the bandwidth is assigned to domain-0 and domain-1, each; and half of the bandwidth is assigned to domain-2. This non-uniform allocation is done by devising a schedule with four slots

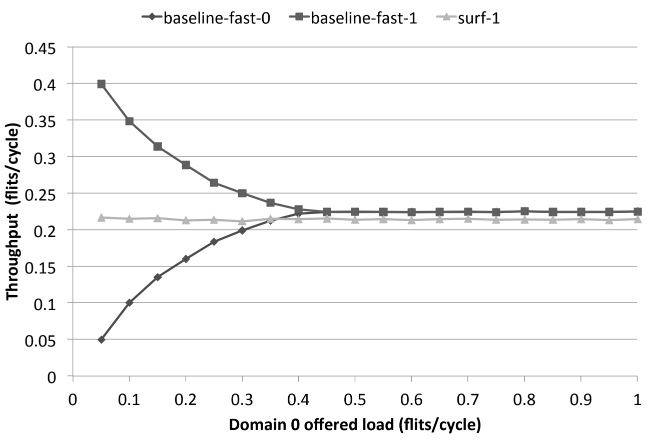


Figure 12: Separation of uniformly distributed bandwidth. Throughput as a function of domain 0 offered load. We can see that, by using surf scheduling, domain 1 throughput is independent of domain 0 load (the same trend was measured for domain 0 throughput while varying domain 1 load).

and assigning domain-3 time slots to domain-2. Saturation throughput, as expected, is 0.09 for both domain 0 and 1, while it is 0.21 for domain 2. Latency at a 5% injection rate is 36 (53) cycles for domain-2 and 39 (53) cycles for domains 0 and 1 using surf scheduling (straightforward tdma). This shows that our scheme can have latency benefits as well as throughput benefit by designing a non-uniform surf schedule.

We examine the non-interference between domains by varying one domain's offered load while keeping the other domain's offered load constant at the maximum in a 2D 64-node mesh with two domains. We plot both domain's throughput for baseline-fast and domain 1 throughput for surf as a function of domain 0 offered load in Figure 12. We can see that domain 1 throughput is independent of domain 0 traffic if we use the surf scheduling but not for the baseline case.

5.4 Area and power overhead

Area: The SurfNoC router requires modifications to the crossbar, more buffering, and a bigger switch allocator (due to a bigger crossbar). For a D domain network, we added D input speedup in the crossbar. Crossbar area scales linearly with the input speedup D because we increase only one dimension of the crossbar. We verified this trend using Synopsis Design Compiler (version E-2010.12-SP5-2 using TSMC 45nm CMOS library) to synthesize a parameterized RTL crossbar implementation [5], and it scales linearly. For example, while a 5×5 crossbar occupies 620.93 library units, a 20×5 crossbar consumes area of 2540.16 library units, which is a factor of 4.16 for input speedup four.

Baseline-small uses 48 entries per input port, assuming 32-bit flits, DSENT [32] (with 45 nm bulk LVT running at 1 GHz with 0.3 injection rate) estimates the buffer area of 0.0125 mm^2. On the other hand, surf and baseline-fast uses 128 entries occupying 0.0327 mm^2, a factor of 2.62 overhead against the baseline-small.

We also added the scheduler, which is mainly p copies of a a counter (where p is the number of output ports), a table of D entries, a $D\times2^D$ decoder, and $D.p$ AND gates (assuming that each domain requests one port regardless of the number of VCs per domain). We estimate the scheduler to be of negligible area compared to the router. For example, the storage requirement for a 16-domain 5-port router is just 324 bits.

Power: Having seen area overhead, we now discuss power consumption overhead. Using DSENT's estimates, the power consumption of buffers increases from 11.9 mW for the baseline-small to 29.3 mW for the baseline-fast and surf schemes, an overhead of 146%. Crossbar with input speedup D power consumption scales linearly with D because dynamic power consumption is directly proportional to capacitance, which is directly proportional to wire length, which increases only linearly with input speedup without output speedup. Using the same synthesis results, Design Compiler estimates a 5×5 crossbar to consume 74.26 μW of power, while a 20×5 crossbar consumes 309.26 μW, a factor of 4.1 for an input speedup of 4.

Having an input-speedup of D might be prohibitive in cases of large D. However, there is a trade-off between wait time at the switch allocator (SA) and input-speedup of the crossbar switch, i.e., performance/resources trade-off. Keeping our surf schedule in place while arbitrating the crossbar input port between VCs from different domains in a static deterministic round-robin manner (regardless of requests), is the most straight-forward way. For example, in the case of 32 domains, we can use an input speedup of 4 instead of 32, and a flit will wait up to 7 cycles until it enters the crossbar. In general, if input speedup is S and D is the number of domains (where $1\leq S\leq D$), flits can wait up to an extra $D/S-1$ cycles to enter the crossbar and would wait longer than $D-1$ in turns. This would be one way to avoid crossbars of excessive size (and the slower clock rates they incur) as well. Maximum zero-load latency of such a scheme will be given using Equation 4. This essentially creates a continuum of design between a strict TDMA (in fact, slightly worse for $S=1$) and a full surf schedule ($S=D$).

$$T_{0max}=HP+((n-1)+2)(DD/S-1)+H(D/S-1) \quad (4)$$

6. NON-INTERFERENCE VERIFICATION

In order to prove non-interference between domains of our arbitration scheme, we used Gate-level information-flow tracking (GLIFT) logic [36, 35]. GLIFT logic captures all digital information, including implicit and timing-channel flows, because all information flows represent themselves in decision-making circuit constructs such as multiplexers. For example, an arbitration operation leaks information if the control bits of the multiplexers depend on one of the two domains, but it will not leak information (or cause interference) if arbitration is based on a static schedule. GLIFT tracking logic can accurately capture this fact because it is precise (i.e., not conservative in the primitive shadow gates but is conservative in the compositional shadow circuit). For example, a shadow-AND gate propagates a label of HIGH only if the output of the AND gate depends on the HIGH input (i.e., if one input of a two-input AND gate is LOW zero, the output is guaranteed to be zero and thus does not depend on the HIGH input). GLIFT automatically generates conservative *shadow logic* that can be used to prove non-interference between domains for a given circuit. Shadow logic is a tracking logic used as a verification technique (and is not intended to be part of the final system, thus does not cost any area or power). Using gate-level analysis, we discovered interference in the switch allocator during initial designs of the system. Moreover, contention between domains on the crossbar switch input ports was discovered using the same analysis technique (hence, our input-speedup idea). In essence, we used GLIFT analysis to design the architecture in addition to verifying the final design.

We integrated the scheduler (§4.3) enforcing the surf schedule into a Verilog implementation of a switch allocator [18]. We used a two-domain allocator that allocates requests of different virtual channels to output ports. We modified the allocator to have a request per VC rather than per input port (as in the original design [18]). We synthesized the allocator using Synopsis Design compiler, then generated its shadow logic and verified the separation property using simulation of the resulting circuit. We assigned a LOW label for VC 0 requests and a HIGH label for VC 1. We tested inputs for VCs sharing the same input port requesting different and same output ports. In all cases, grant signals had the same label of their respective virtual channel, which proves that grants are independent of requests from the other domain. We also reversed labels of VC0 (HIGH) and VC1 (LOW) to verify that separation holds for the other direction of information flow (Domain 0 to Domain 1). This proves that the crossbar arbitration, and consequently sharing of physical channel, are timing-channel free, which (in addition to static VC allocation) ensures network non-interference. Freedom of two-way information flow, or complete non-interference, was verified.

7. CONCLUSIONS

Networks-on-chip play an important role in integrating many components, whether they are accelerators, cores, or memories. Not only are they increasingly prevalent in consumer general-purpose silicon, but they are also seeing introduction in high-assurance domains, where security and verification accuracy are crucial to saving time, money, and potentially even lives. Separation is an important property that allows designers to reason about systems efficiently by defining sub-components that can be verified independently while limiting the design space.

While we believe this paper is an important step regarding gate-level separation in NoCs, there are many questions that merit further investigation. First, we make no use of application-level knowledge that might shed light on the expected communication patterns. Co-scheduling communicating tasks (with global traffic knowledge) [22] to be near to one another might introduce the opportunities for non-homogeneous yet non-interfering schedules across the network. Application-level knowledge of lattice-based information flow policies might be combined with this work to allow more flexibility in scheduling between comparable domains [37]. Second, our approach uses a dimension-ordered routing that is oblivious to our time-division multiplexing scheme (surf-scheduling). As such, a packet will only change dimensions after it finishes traversing one dimension. A non-interference-aware routing technique might minimize wait time by introducing more turns opportunistically. In general, the relationship between interference and more aggressive optimizations would be interesting to explore. Third, there are other topologies to consider, e.g., high-radix routers such as flattened butterflies [17]. Although flattened butterflies can use dimension-order routing and thus surf scheduling might directly be applied, non-minimal routing is usually required to improve throughput. Enforcing a surf-like schedule with adaptive routing might increase latency. However, all of these open questions require a foundation from which to build.

The foundation we propose here is SurfNoC, a low-latency time-division-multiplexed packet-switched k-ary n-cube network. SurfNoC exploits the dimension-ordered routing algorithms in mesh networks by scheduling channels in each dimension in a pipelined fashion so that packets propagate in the dimension as if there are no domain restrictions on channels. Packets have to wait for their domain's turn only when they enter, exit, and potentially when changing dimensions. We discuss our wave-based domain scheduled network and describe the implementation at the level of the router micro-architecture with respect to non-interference support. Importantly, while several works have discussed interference at a high level, we believe this is the first time that true cycle-level non-interference has been proven to hold at the gate-level. In addition to the formal gate-level analysis needed to demonstrate that, we show that our schedule latency overhead scales efficiently with the number of separation domains compared to a straightforward synchronous TDMA scheme (saving up to 75% of latency overhead in the case of a 64-node with 32 domains). Although each domain's throughput suffers as the network is partitioned (as would be expected), the aggregate network performance remains very close to the no-separation baseline. More importantly, the latency overhead remains constant with respect to network size.

8. ACKNOWLEDGMENTS

The authors would like to thank Tim Levin, Cynthia Irvine, Xun Li and the anonymous reviewers for insightful comments on this paper.

9. REFERENCES

[1] NASA'S Mars rover Curiosity powered by Wind River. `http://www.windriver.com/announces/curiosity/Wind-River_NASA_0812.pdf`.

[2] O. Aciiçmez. Yet another microArchitectural attack: exploiting I-Cache. In *Proceedings of the 2007 ACM workshop on Computer security architecture*, CSAW '07, pages 11–18, New York, NY, USA, 2007. ACM.

[3] O. Aciiçmez, c. K. Koç, and J.-P. Seifert. Predicting secret keys via branch prediction. In *Proceedings of the 7th Cryptographers' track at the RSA conference on Topics in Cryptology*, CT-RSA'07, pages 225–242, Berlin, Heidelberg, 2006. Springer-Verlag.

[4] O. Aciiçmez, c. K. Koç, and J.-P. Seifert. On the power of simple branch prediction analysis. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, ASIACCS '07, pages 312–320, New York, NY, USA, 2007. ACM.

[5] D. U. Becker. *Efficient Microarchitecture for Network-on-Chip Routers.* PhD thesis, Stanford University, August 2012.

[6] D. Bui, A. Pinto, and E. A. Lee. On-time network on-chip: Analysis and architecture. Technical Report UCB/EECS-2009-59, EECS Department, University of California, Berkeley, May 2009.

[7] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[8] W. J. Dally, P. P. Carvey, and L. R. Dennison. The Avici terabit switch/router. In *IEEE Hot Interconnects*, 1998.

[9] L. Fiorin, G. Palermo, and C. Silvano. A security monitoring service for NoCs. In *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, CODES+ISSS '08, pages 197–202, New York, NY, USA, 2008. ACM.

[10] C. H. Gebotys and Y. Zhang. Security wrappers and power analysis for SoC technologies. In *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '03, pages 162–167, New York, NY, USA, 2003. ACM.

[11] K. Goossens, J. Dielissen, and A. Radulescu. AEthereal network on chip: concepts, architectures, and implementations. *Design Test of Computers, IEEE*, 22(5):414 – 421, Sept.-Oct. 2005.

[12] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu. Kilo-NOC: a heterogeneous network-on-chip architecture for scalability and service guarantees. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 401–412, New York, NY, USA, 2011. ACM.

[13] B. Grot, S. W. Keckler, and O. Mutlu. 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*, MICRO 42, pages 268–279, New York, NY, USA, 2009. ACM.

[14] B. Grot, S. W. Keckler, and O. Mutlu. Topology-aware quality-of-service support in highly integrated chip multiprocessors. In *Proceedings of the 2010 international conference on Computer Architecture*, ISCA'10, pages 357–375, Berlin, Heidelberg, 2012. Springer-Verlag.

[15] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken. CoMPSoC: A template for composable and predictable multi-processor system on chips. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):2:1–2:24, Jan. 2009.

[16] A. Hansson, M. Subburaman, and K. Goossens. Aelite: A flit-synchronous network on chip with composable and predictable services. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 250 –255, april 2009.

[17] J. Kim, J. Balfour, and W. Dally. Flattened butterfly topology for on-chip networks. In *MICRO 40: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 172–182, Washington, DC, USA, 2007. IEEE Computer Society.

[18] M. Kinsy and M. Pellauer. Heracles: fully synthesizable parameterized MIPS-based multicore system. Technical Report MIT-CSAIL-TR-2010-058, MIT Computer Science and Artificial Intelligence Laboratory, December 2010.

[19] J. W. Lee, M. C. Ng, and K. Asanovic. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 89–100, Washington, DC, USA, 2008. IEEE Computer Society.

[20] S. Lukovic and N. Christianos. Enhancing network-on-chip components to support security of processing elements. In *Proceedings of the 5th Workshop on Embedded Systems Security*, WESS '10, pages 12:1–12:9, New York, NY, USA, 2010. ACM.

[21] S. Lukovic and N. Christianos. Hierarchical multi-agent protection system for NoC based MPSoCs. In *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems*, S&D4RCES '10, pages 6:1–6:7, New York, NY, USA, 2010. ACM.

[22] S. Ma, N. Enright Jerger, and Z. Wang. DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 413–424, New York, NY, USA, 2011. ACM.

[23] M. Malone. Talk on OPERA RHBD Multi-core. `https://nepp.nasa.gov/mapld_2009/talks/083109_Monday/03_Malone_Michael_mapld09_pres_1.pdf`.

[24] R. Obermaisser and O. Hoftberger. Fault containment in a reconfigurable multi-processor System-on-a-Chip. In *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*, pages 1561 –1568, june 2011.

[25] E. Ong, O. Brown, and M. J. Losinski. System F6: Progress to Date. `http://digitalcommons.usu.edu/cgi/viewcontent.cgi?filename=0&article=1016&context=smallsat&type=additional`. in Small Satellite Constellations: Strength in Numbers, Logan, UT, 2012, p. 7.

[26] W. R. Otte, A. Dubey, S. Pradhan, P. Patil, A. Gokhale, G. Karsai, and J. Willemsen. F6com: A component model for resource-constrained and dynamic space-based computing environments. In *16th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing*, 2013.

[27] J. Porquet, A. Greiner, and C. Schwarz. NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –4, march 2011.

[28] J. Rushby. Partitioning for Avionics Architectures: Requirements, Mechanisms, and Assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999. Also to be issued by the FAA.

[29] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, NOCS '12, pages 152–160, Washington, DC, USA, 2012. IEEE Computer Society.

[30] R. Stefan and K. Goossens. Enhancing the security of time-division-multiplexing networks-on-chip through the use of multipath routing. In *Proceedings of the 4th International Workshop on Network on Chip Architectures*, NoCArc '11, pages 57–62, New York, NY, USA, 2011. ACM.

[31] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens. A TDM NoC supporting QoS, multicast, and fast connection set-up. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1283 –1288, march 2012.

[32] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic. DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, NOCS '12, pages 201–210, Washington, DC, USA, 2012. IEEE Computer Society.

[33] J.-L. Terraillon. Multicore processors - the next generation computer for ESA space missions. `http://www.cister.isep.ipp.pt/ae2012/presentations_pdf/thursday/k/terraillon.pdf`. "Keynote address.".

[34] M. Tiwari, X. Li, H. M. G. Wassel, F. T. Chong, and T. Sherwood. Execution leases: a hardware-supported mechanism for enforcing strong non-interference. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 493–504, New York, NY, USA, 2009. ACM.

[35] M. Tiwari, J. K. Oberg, X. Li, J. Valamehr, T. Levin, B. Hardekopf, R. Kastner, F. T. Chong, and T. Sherwood. Crafting a usable microkernel, processor, and I/O system with strict and provable information flow security. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 189–200, New York, NY, USA, 2011. ACM.

[36] M. Tiwari, H. M. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood. Complete information flow tracking from the gates up. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS '09, pages 109–120, New York, NY, USA, 2009. ACM.

[37] Y. Wang and G. Suh. Efficient timing channel protection for on-chip networks. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 142 –151, may 2012.

[38] Z. Wang and R. B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 494–505, New York, NY, USA, 2007. ACM.

[39] Z. Wang and R. B. Lee. A novel cache architecture with enhanced performance and security. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 83–93, Washington, DC, USA, 2008. IEEE Computer Society.

[40] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, Sept. 2007.