Networks on Chip with Provable Security Properties

In systems where a lack of safety or security guarantees can be catastrophic or even fatal, noninterference is used to separate domains handling critical (or confidential) information from those processing normal (or unclassified) data for fault containment and ease of verification. SurfNoC significantly reduces the latency incurred by strict temporal partitioning.

Hassan M.G. Wassel
Google

Ying Gao
University of California, Santa Barbara

Jason K. Oberg
University of California, San Diego

Ted Huffmire
Naval Postgraduate School

Ryan Kastner
University of California, San Diego

Frederic T. Chong
Timothy Sherwood
University of California, Santa Barbara

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.

The difficulty in designing such strong separation functionality into typical networks on chip (NoCs) is that they 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,[1] and generally complicate the final verification and certification process of the system because all of the ways in which that interaction might occur must be accounted for. Noninterference means that the injection of packets from one domain can never have any effect on the delivery of packets from other domains, even in their timing.

These concerns are similar to, but distinct from, the problem of providing quality-of-service guarantees. Although QoS can minimize the performance impact of sharing between domains by providing a minimum guaranteed level of service for each domain (or class),[2-5] as Wang and Suh show, QoS techniques still allow some degree of timing variations and thus do not truly support noninterference.[1] The only way to be certain that the domains are noninterfering is to statically schedule them on the network over time. However, a straightforward application of time multiplexing leads to significant increases in latencies because each link in the network is now time-multiplexed between many domains.

The core idea behind our approach, for meshes and tori, 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"

0272-1732/14/$31.00 © 2014 IEEE Published by the IEEE Computer Society

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 can 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 noninterference in a modern NoC router microarchitecture using gate-level analysis, discuss the details and ramifications of our surf scheduling methodology, and demonstrate that our approach truly does not allow even cycle-level cross-domain interference. (For information on previous research, see the "Related Work in Noninterference" sidebar.)

SurfNoC scheduling

The straightforward way to support time-division multiplexing (TDM) 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 begins 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:

$$T_0 = HP + H(D-1) \quad (1)$$

This solution might work efficiently for two to four domains, but in high-assurance applications, as many as tens or hundreds of domains can be found.[6]

The most basic routing algorithm in meshes and tori (k-ary n-cube networks) 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 another dimension. Thus, routing is linear in each dimension, which provides an opportunity to reduce wait time between hops. In SurfNoC scheduling, different routers (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 (that is, 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). After finishing the first dimension, the packet might 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 to some location and then waits to ride another wave. Equation 2 shows the maximum zero-load latency and clearly shows that the overhead is additive, not multiplicative as in the straightforward approach:

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

The term $([n-1]+2)$ comes from the $n-1$ transition between dimensions and two waits during injection and ejection. Note that this is the maximum wait, not the typical one, because the schedule might require less wait.

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 (east and west for the x-dimension; north and south for the y-dimension). The pipelines of directions of the same dimension (that is, north, south, east, and west) run in opposite directions, as Figure 1 shows. 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 router's pipeline depth. These schedules are imposed on each router's

Related Work in Noninterference

Our proposed solution to noninterference in networks on chip (NoCs) touches on many problems that have been addressed in previous research, such as timing channels in microarchitectures, quality of service (QoS) in networks on chip, and fault containment in systems on chip.

Timing channels and noninterference in microarchitecture

Recently there has been renewed interest in the analysis of timing channel attacks and mitigations through microarchitecture state such as cache interference[1-3] and branch predictors.[4,5] One approach to these problems is a technique that can verify noninterference of hardware/software systems (including high-performance features such as pipelining and caching) using gate-level information flow tracking.[6-8] More recently, researchers proposed a NoC timing-channel protection scheme for a system with security lattices.[9] This priority-based arbitration scheme 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. Our proposed technique enables multiway noninterference in NoCs with low latency (allowing even packets with noncomparable labels to share the network). However, the techniques working in tandem would still provide the greatest possible benefit.

QoS in networks on chip

Techniques for achieving NoC quality-of-service guarantees have been proposed based on solutions to analogous problems in macroscale networks. These approaches attempt to limit the rates of each flow.[10-13] However, quality-of-service guarantees are insufficient for timing-channel protection.[9] Optimizations that allow flows to go over their designated rate when uncontended and the lack of fault containment are problematic for high-assurance systems[14] because of the high cost of any unaccounted variation in such systems. Our time-division approach provides for both fault containment and timing channel elimination.

Noninterference in NoCs

Noninterference 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.[15,16] 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 system's safety ramifications. The Æthearal NoC is a time-division multiplexed (TDM) virtual circuit-switching network that provides guaranteed services for performance-critical applications with real-time deadlines and a packet-switched best-effort network for applications with fewer requirements.[17] A lighter version that only provides guaranteed service was proposed to further simplify routers.[18,19] More recently, Stefan and Goossens proposed a modification to Æthearal that enables multipath routing, both static and dynamic (based on a true random number generator), to enhance security by using a nondeterministic path instead of the source-routing used in Æthearal.[20] In addition, the need for real-time worst-case execution time (WCET) analysis inspired a set of works, such as the T-Crest project (www.t-crest.org), which tries to build a time-predictable multicore for real-time applications. T-Crest researchers proposed an integer programming technique to minimize the static schedule length of all-to-all circuit switching connections in a TDM way.[21] Dai Bui and his colleagues proposed an on-time NoC using real-time packet scheduling, admission control, and runtime path configuration.[22] 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.

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 I/O device traffic.[23] Our scheme can protect against denial-of-service (DoS) and bandwidth depletion attacks between domains because of the static time allocation to different domains.

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

References

1. O. Aciiçmez, "Yet Another MicroArchitectural Attack: Exploiting I-Cache," *Proc. 2007 ACM Workshop Computer Security Architecture* (CSAW 07), 2007, pp. 11-18.

output channels to avoid timing channels based on contention in the allocator (as detailed in the next section).

Figure 1 shows a 16-node 2D mesh schedule of three domains (colored white, gray, black). There are two waves, southeast (SE) and northwest (NW), running in the mesh. Each channel propagates packets according to the schedule white, white, gray, and black and repeats. Using such a schedule results in half of the bandwidth being allocated to the white domain, whereas the black and gray domains guarantee only a quarter of the bandwidth for each of them. This

2. Z. Wang and R.B. Lee, "New Cache Designs for Thwarting Software Cache-Based Side Channel Attacks," *Proc. 34th Ann. Int'l Symp. Computer Architecture* (ISCA 07), 2007, pp. 494-505.
3. Z. Wang and R.B. Lee, "A Novel Cache Architecture with Enhanced Performance and Security," *Proc. 41st Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2008, pp. 83-93.
4. O. Aciiçmez, C.K. Koç, and J.-P. Seifert, "Predicting Secret Keys via Branch Prediction," *Proc. 7th Cryptographers' Track at the RSA Conf. Topics in Cryptology* (CT-RSA 07), 2007, pp. 225-242.
5. O. Aciiçmez, C.K. Koç, and J.-P. Seifert, "On the Power of Simple Branch Prediction Analysis," *Proc. 2nd ACM Symp. Information, Computer, and Comm. Security* (ASIACCS 07), 2007, pp. 312-320.
6. M. Tiwari et al., "Execution Leases: A Hardware-Supported Mechanism for Enforcing Strong Non-interference," *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2009, pp. 493-504.
7. M. Tiwari et al., "Crafting a Usable Microkernel, Processor, and I/O System with Strict and Provable Information Flow Security," *Proc. 38th Ann. Int'l Symp. Computer Architecture* (ISCA 11), 2011, pp. 189-200.
8. M. Tiwari et al., "Complete Information Flow Tracking from the Gates Up," *Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 09), 2009, pp. 109-120.
9. Y. Wang and G. Suh, "Efficient Timing Channel Protection for On-Chip Networks," *Proc. 6th IEEE/ACM Int'l Symp. Networks on Chip* (NoCS 12), 2012, pp. 142-151.
10. B. Grot et al., "Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees," *Proc. 38th Ann. Int'l Symp. Computer Architecture* (ISCA 11), 2011, pp. 401-412.
11. B. Grot, S.W. Keckler, and O. Mutlu, "Preemptive Virtual Clock: A Flexible, Efficient, and Cost-Effective QoS Scheme for Networks-on-Chip," *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2009, pp. 268-279.
12. B. Grot, S.W. Keckler, and O. Mutlu, "Topology-Aware Quality-of-Service Support in Highly Integrated Chip Multiprocessors," *Proc. Int'l Conf. Computer Architecture* (ISCA 10), 2010, pp. 357-375.
13. J.W. Lee, M.C. Ng, and K. Asanovic, "Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks," *Proc. 35th Ann. Int'l Symp. Computer Architecture* (ISCA 08), 2008, pp. 89-100.
14. J. Rushby, *Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance*, NASA Contractor Report CR-1999-209347, NASA Langley Research Center, 1999.
15. A. Hansson et al., "CoMPSoC: A Template for Composable and Predictable Multi-Processor System on Chips," *ACM Trans. Design Automation of Electronic Systems*, vol. 14, no. 1, 2009, pp. 1-24.
16. R. Obermaisser and O. Hoftberger, "Fault Containment in a Reconfigurable Multi-Processor System-on-a-Chip," *Proc. IEEE Int'l Symp. Industrial Electronics* (ISIE 11), 2011, pp. 1561-1568.
17. K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal Network on Chip: Concepts, Architectures, and Implementations," *Design & Test of Computers*, vol. 22, no. 5, 2005, pp. 414-421.
18. A. Hansson, M. Subburaman, and K. Goossens, "Aelite: A Flit-Synchronous Network on Chip with Composable and Predictable Services," *Proc. Design, Automation & Test in Europe Conference & Exhibition* (DATE 09), 2009, pp. 250-255.
19. R. Stefan et al., "A TDM NoC Supporting QoS, Multicast, and Fast Connection Set-Up," *Proc. Design, Automation Test in Europe Conf. & Exhibition* (DATE 12), 2012, pp. 1283-1288.
20. R. Stefan and K. Goossens, "Enhancing the Security of Time-Division-Multiplexing Networks-on-Chip through the Use of Multipath Routing," *Proc. 4th Int'l Workshop Network on Chip Architectures* (NoCArc 11), 2011, pp. 57-62.
21. M. Schoeberl et al., "A Statically Scheduled Time-Division-Multiplexed Network-on-Chip for Real-Time Systems," *Proc. IEEE/ACM 6th Int'l Symp. Networks-on-Chip* (NOCS 12), 2012, pp. 152-160.
22. D. Bui, A. Pinto, and E.A. Lee, *On-Time Network On-Chip: Analysis and Architecture*, tech. report UCB/EECS-2009-59, Electrical Engineering and Computer Science Dept., Univ. of Calif., Berkeley, 2009.
23. D. Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, vol. 27, no. 5, 2007, pp. 15-31.

illustrates the benefit of our schedule in statically allocating nonuniform bandwidth to domains.

Router microarchitecture

The microarchitecture of the SurfNoC router has two main goals:

- Ensuring a timing-channel-free contention between packets—that is, contention can occur between packets from the same domain but not between packets from different domains.
- Scheduling the output channels of each router in a way that maintains

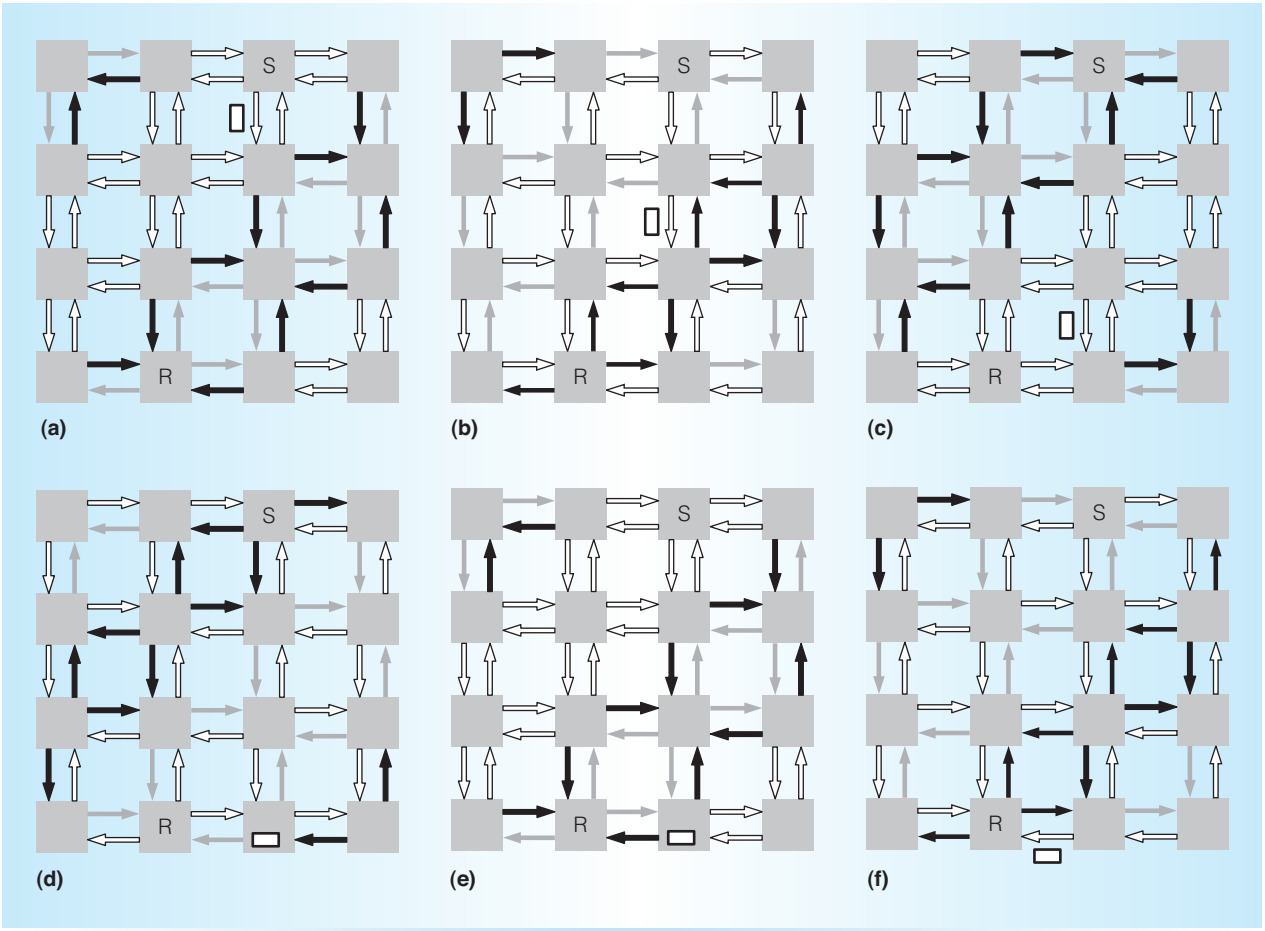

Figure 1. Surf scheduling in a 16-node 2D mesh with three application domains (denoted by white, gray, and black) assuming single-cycle routers for illustration purpose. The schedule runs as white, white, gray, and black and repeats, giving the white domain half the bandwidth. A packet (the white box under the node S) belongs to the white domain and is sent from the node marked S to the node marked 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 two 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.

the surf schedule across the whole network.

To achieve these goals, we used a static partitioning of virtual channels (VCs) and carefully designed the VC and switch allocators to be timing-channel free. In essence, the VC allocator is divided into several allocators that allocate VCs belonging to the same domain because VCs are statically divided between domains. The resources (switch I/O ports) in the switch allocator, on the other hand, can be requested from multiple domains. Moreover, switch inputs are shared between VCs belonging to different domains. To solve this problem, we use input speedup of D to remove this contention. This observation was discovered through gate-level information flow analysis of the allocator microarchitecture. The scheduling of output channels is done through masking requests from packets to the switch allocator until its turn to use the output channel arrives in the wave pipeline. Traversing the switch does not require any router modification because all resources have been arbitrated for.

Pipelining and separation

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. We will discuss each pipeline stage, with the basic idea being to ensure that stalls do not induce interference between separate domains.

Buffer write and route computation. BW/RC 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 VCs (waiting for RC). No interference can be caused in this stage.

Virtual channel allocation. At the VA stage, all flits send requests to the VC allocator. Using our design, interference can happen between VCs from the same domain but not between channels from distinct domains. Stalled flits resulting from a lack of free VCs (in the downstream router) prevent only flits from the same domain from making progress. This can be ensured by recording the state in the pipeline for each VC—that is, stalls due to VC allocation must be per VC (not per input port).

Switch allocation. SA can fail owing to contending flits for switch ports (limited to VCs from the same domain), which cause 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 VC. SA can also be stalled because of lack of buffering in the downstream router—that is, waiting for a credit. The effect of this stall is limited to a single VC, and can be handled the same way we addressed a stall resulting 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 VC. Thus, we can guarantee separation because we statically assign VCs to domains.

Noninterference verification

To prove noninterference between domains of our arbitration scheme, we used gate-level information-flow tracking (GLIFT) logic.[7,8] 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 and arbiters. 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 (that is, not conservative in the primitive shadow gates but conservative in the compositional shadow circuit) and sound (that is, it will definitely capture illegal information flows). For example, a shadow-AND gate propagates a label of "high" only if the output of the AND gate depends on the "high" input (that is, 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 noninterference 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 circuit, enforcing the surf schedule, into a Verilog implementation of a switch allocator.[9] We used a two-domain allocator that allocates requests of different VCs to output ports. We modified the allocator to have a request per VC rather than per input port (as in the original design[9]). We synthesized the allocator using the 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

Table 1. Simulation parameters.

Parameter	Baseline-small	Baseline-fast	Surf and TDMA
Virtual channels (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	Separable (input-first)		
Routing	Dimension-ordered routing		

Table 2. Different configurations of partitioned schemes.

	Number of domains					
Parameter	**1**	**2**	**4**	**8**	**16**	**32**
No. of VCs per port	16	16	16	32	32	32
No. of flits per VC	8	8	8	4	4	4
Input speedup	1	2	4	8	16	32

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 VC, which proves that grants are independent of requests from the other domain. We also reversed labels of VC 0 (high) and VC 1 (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 the sharing of the physical channel, are timing-channel free, which (in addition to static VC allocation) ensures network noninterference. Freedom of two-way information flow, or complete noninterference, was verified.

Evaluation

We evaluate the performance of our SurfNoC scheme and compare the area and power overhead to a mesh network without noninterference support. A more detailed evaluation can be found in the original paper.[10]

Experimental setup

We implemented a model of the SurfNoC router in BookSim 2.0,[11] 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 lists the simulation parameters used for different schemes. We evaluated four schemes, two that do not provide separation guarantees, and two that support strong separation. The nonseparation baselines are an input-queued router with minimal resources, which achieves almost 40 percent saturation throughput (*baseline-small*), and a similar router that has many more resources (buffers and input-speedup in the crossbar switch), which we call *baseline-fast*. We used two baselines because the separation-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 noninterference-supporting schemes are a straightforward time-division multiple access (TDMA), where 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.

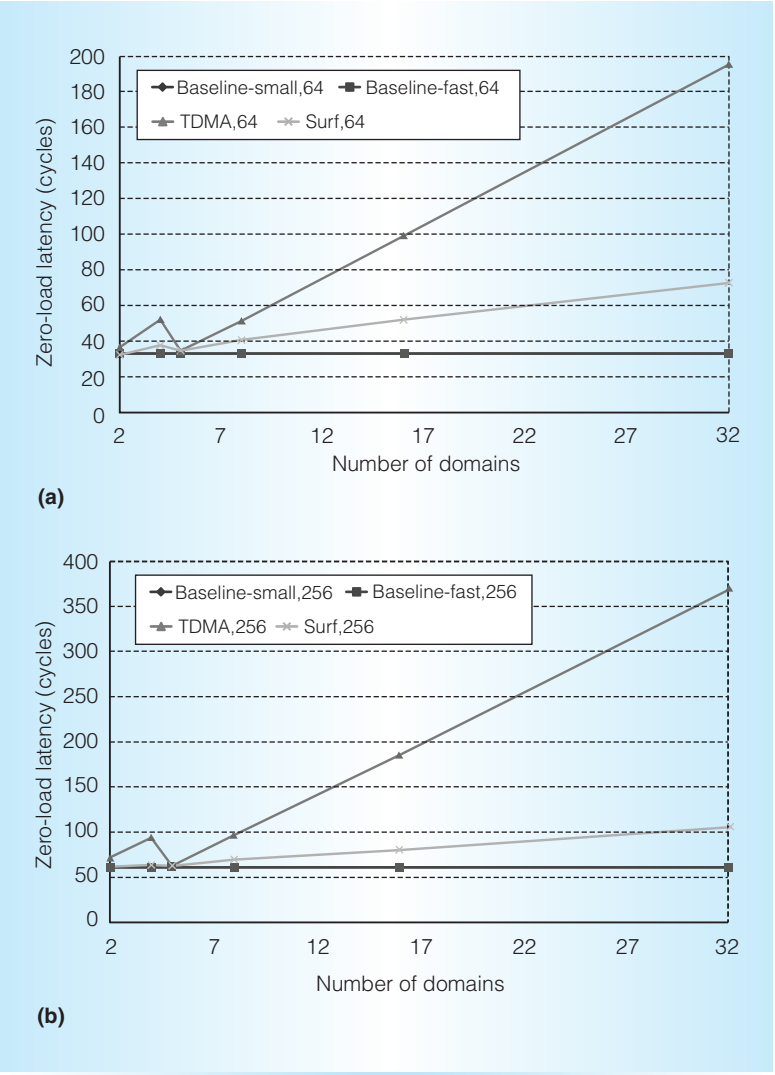

Figure 2. Zero-load latency for different network sizes and a different number of security domains: 64 nodes (a) and 256 nodes (b). The two baselines overlap because zero-load latency does not depend on buffers and crossbar input speedup.

Impact on latency

We first examine the impact of our noninterference support on latency with a different number of domains, and a different number of nodes under the uniform random traffic pattern. To understand the effect of TDM of channels, we measure zero-load latency (latency at offered load of 0.1 percent of capacity for only one domain) and plot it for different numbers of domains in Figure 2. In this figure, we plot latency in cycles (y-axis) versus number of domains on the x-axis for network sizes of 64 nodes (Figure 2a) and 256 nodes (Figure 2b). It is clear that the latency overhead of Surf scales much better than TDMA for the same network size (for example, the overhead is reduced from 66 (19.1) to 19 (4.6) cycles by 71.3 percent (75.8 percent) for network sizes of 64 nodes with 16 (4) domains. The savings is even greater (up to 84.7 percent) for a 256-node network.

We can see that there is one exception to this reduction in latency. It is a subtle case that occurs 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 VA in the upstream router (a total of four cycles during which the upstream router propagates packets from other domains), it becomes ready for 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.

To clearly understand how the overhead scales with network size or average number of hops, we replotted zero-load latency of 2D mesh networks of sizes varying from 16 to 256 nodes with 16 domains under the uniform random traffic pattern in Figure 3. The latency of both baselines increases with network size due to a higher average number of hops. The overhead of surf scheduling is almost independent of network size (average number of hops), leading to a line parallel to the baseline with a constant overhead of 19 cycles (except for 16 nodes) because the packet wait time depends only on the number of dimensions and domains. On the other hand, the larger the network, the higher the overhead for TDMA scheduling because a packet must 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 independent of the number of hops. We can conclude that, in general, the savings of surf scheduling are more scalable with larger networks and a higher number of domains.

Zero-load latency is just one latency metric. Thus, we now study latency as a function of network offered load. Figure 4 shows

average latency measured after convergence as a function of offered load for a 2D mesh network of 64 nodes under uniform random traffic patterns. We vary aggregate offered load on the *x*-axis—that is, if we have *D* domains, the value of the *x*-axis is the sum of the 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.

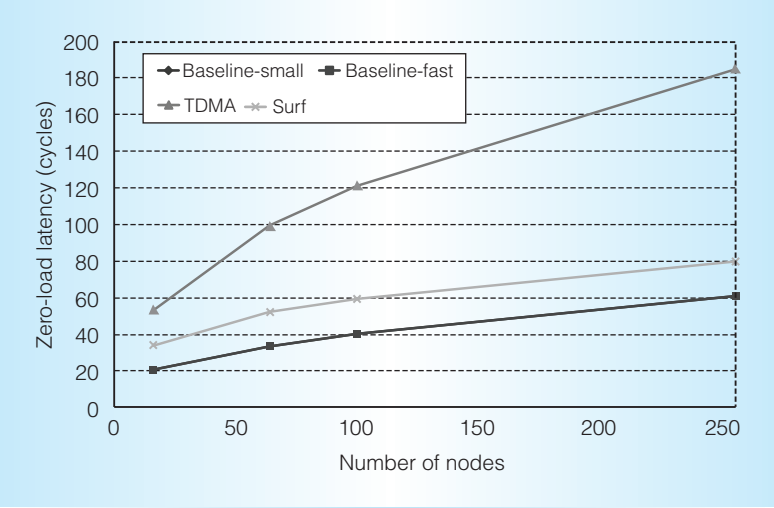

Figure 3. Zero-load latency versus different network size with 16 domains. The two baselines overlap because zero-load latency does not depend on buffers and crossbar input speedup.

Throughput

In Figure 4, although we can see that saturation throughput is reduced by about 11.7 percent, aggregate throughput loss is limited to 4.9 percent for two domains. Noninterference configurations have a 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.

To verify the benefits of assigning bandwidth nonuniformly, 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 nonuniform allocation is done by devising a schedule with four slots and assigning domain 3 time slots to domain 2. Saturation throughput, as expected, is 0.09 for both domain 0 and 1, and 0.21 for domain 2. Latency at a 5 percent 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 both latency and throughput benefits by designing a nonuniform surf schedule.

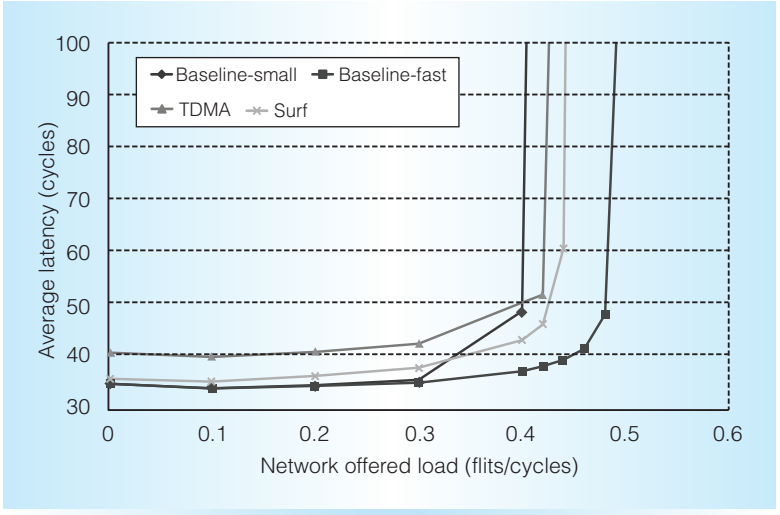

Figure 4. Average latency as a function of aggregate offered load of all domains for a 2D mesh network of 64 nodes. Latency is stable below the network saturation point.

Area and power overhead

The main source of power and area overhead is the increased size of the crossbar with an input speedup of *D*. This increases both the crossbar and switch allocator area and power consumption linearly with *D*. Having an input speedup of *D* might be prohibitive in cases of large *D*. However, there is a performance/resources trade-off between wait time at the switch allocator and input speedup of the crossbar switch. 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 straightforward approach. 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 seven cycles before entering 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 is one way to avoid excessively large crossbars (and the slower clock rates they incur) as well. We use Equation 3 to find the maximum zero-load latency of such a scheme:

$$T_{0\max} = HP + ([n-1]+2)\left(D\frac{D}{S} - 1\right) + H\left(\frac{D}{S} - 1\right) \quad (3)$$

This essentially creates a continuum of design between a strict TDMA (in fact, slightly worse for $S = 1$) and a full surf schedule ($S = D$).

Programmers are increasingly asked to manage a complex collection of computing elements, including a variety of cores, accelerators, and special-purpose functions. Although 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 becomes 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 explored in the literature. On the other hand, real systems are likely to use different subsets of cores and accelerators based on an application's needs 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 it becomes even more difficult if we attempt to prevent even cycle-level variations.

High-assurance systems are often divided into a set of domains, which are 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. 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 that add another layer of communication,[12,13] but there are serious concerns about the introduction of opportunities for interference between system components.[14,15] Such interference can be catastrophic or even fatal.

The SurfNoC scheduling technique for meshes and tori reduces latency overhead while maintaining noninterference. Perhaps more importantly, it shows that security properties of on-chip networks can be reasoned about rigorously and that doing so has important ramifications on the network microarchitecture. Of course our design is not the end-all design point: there are certainly inefficiencies remaining in our design and how this work can extend to more general and modern NoC techniques remains to be seen. However, by coupling our design efforts with gate-level information flow analysis, we show that verifying noninterference properties at the level of gates and wires for a full router microarchitecture is indeed possible.

We do not foresee the trend toward larger and more diverse NoCs on systems reversing anytime soon. Thus, we believe the techniques we've described will continue to grow in importance. SurfNoC might be extended to support more general routing functions, bandwidth sharing through time donation between domains in a security lattice, a reconfigurable router microarchitecture design, and predictable latency through throttling of source nodes. Similar design and analysis techniques could be extended beyond regular packet-switched networks to include virtual circuit-switching and irregular SoC-style interconnects. Although it is not clear where the limits are between security and performance in this space, if we are to avoid the continuing loop of "patch-and-pray" that inevitably falls out of the ad hoc approach we traditionally have taken to computer systems security, we will increasingly have to involve the architecture, hardware design, and verification teams in

providing a formally sound foundation for integration. MICRO

Acknowledgments

This work was funded in part by grants CNS-1239567, CNS-1162187, and CCF-117165. Jason K. Oberg is funded by a US National Science Foundation graduate research fellowship. 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.

References

1. Y. Wang and G. Suh, "Efficient Timing Channel Protection for On-Chip Networks," *Proc. 6th IEEE/ACM Int'l Symp. Networks on Chip* (NoCS 12), 2012, pp. 142-151.
2. B. Grot et al., "Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees," *Proc. 38th Ann. Int'l Symp. Computer Architecture* (ISCA 11), 2011, pp. 401-412.
3. B. Grot, S.W. Keckler, and O. Mutlu, "Preemptive Virtual Clock: A Flexible, Efficient, and Cost-Effective QoS Scheme for Networks-on-Chip," *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2009, pp. 268-279.
4. B. Grot, S.W. Keckler, and O. Mutlu, "Topology-Aware Quality-of-Service Support in Highly Integrated Chip Multiprocessors," *Proc. Int'l Conf. Computer Architecture* (ISCA 10), 2010, pp. 357-375.
5. J.W. Lee, M.C. Ng, and K. Asanovic, "Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks," *Proc. 35th Ann. Int'l Symp. Computer Architecture* (ISCA 08), 2008, pp. 89-100.
6. J. Rushby, *Partitioning for Avionics Architectures: Requirements, Mechanisms, and Assurance*, NASA contractor report CR-1999-209347, NASA Langley Research Center, 1999.
7. M. Tiwari et al., "Crafting a Usable Microkernel, Processor, and I/O System with Strict and Provable Information Flow Security," *Proc. 38th Ann. Int'l Symp. Computer Architecture* (ISCA 11), 2011, pp. 189-200.
8. M. Tiwari et al., "Complete Information Flow Tracking from the Gates Up," *Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 09), 2009, pp. 109-120.
9. M. Kinsy and M. Pellauer, *Heracles: Fully Synthesizable Parameterized MIPS-based Multicore System*, tech. report MIT-CSAIL-TR-2010-058, MIT Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2010.
10. H.M.G. Wassel et al., "SurfNoC: A Low Latency and Provably Non-Interfering Approach to Secure Networks-on-Chip," *Proc. 40th Ann. Int'l Symp. Computer Architecture* (ISCA 13), 2013, pp. 583-594.
11. W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2003.
12. M. Malone, talk on OPERA RHBD Multicore, Military and Aerospace Programmable Logic Devices (MAPLD) Workshop, Greenbelt, MD, Aug. 2009; https://nepp.nasa.gov/mapld_2009/talks/083109_Monday/03_Malone_Michael_mapld09_pres_1.pdf.
13. J.-L. Terraillon, "Multicore Processors—The Next Generation Computer for ESA Space Missions," keynote address, 17th Int'l Conf. Reliable Software Technologies, 14 June 2012; www.cister.isep.ipp.pt/ae2012/keynote#MP.
14. E. Ong, O. Brown, and M.J. Losinski, "System F6: Progress to Date," *Proc. Small Satellite Conf.: Enhancing Global Awareness through Small Satellites*, 2012, p. 7; http://digitalcommons.usu.edu/smallsat/2012/all2012/10/.
15. W.R. Otte et al., "F6com: A Component Model for Resource-Constrained and Dynamic Space-Based Computing Environments," *Proc. 16th IEEE Int'l Symp. Object/Component/Service-Oriented Real-Time Distributed Computing*, 2013; www.isis.vanderbilt.edu/node/4552.

Hassan M.G. Wassel is a software engineer at Google, working in the Platforms group.

His research includes computer architecture and its interaction with software and novel hardware technologies in order to build energy-efficient, high-performance, and reliable systems. Wassel has a PhD in computer science from the University of California, Santa Barbara, where he performed the work for this article. He is a member of IEEE and the ACM.

Ying Gao is a PhD student in computer engineering at the University of California, Santa Barbara. Her research focuses on security computer architectures and improving program security by using intelligent hardware. Gao has a BS in optoelectronics from Tianjin University, China.

Jason K. Oberg is a PhD candidate in the Computer Science and Engineering Department at the University of California, San Diego. His research interests include testing and verification methods for secure hardware design and methodologies for identifying timing-based side channels in hardware. Oberg has an MS in computer engineering from the University of California, San Diego.

Ted Huffmire is an assistant professor of computer science at the Naval Postgraduate School. His research focuses on the intersection of computer architecture and computer security. Huffmire has a PhD in computer science from the University of California, Santa Barbara.

Ryan Kastner is a professor in the Department of Computer Science and Engineering at the University of California, San Diego. His research focuses on embedded system design, particularly the use of reconfigurable computing devices for digital signal processing as well as hardware security. Kastner has a PhD in computer science from the University of California, Los Angeles.

Frederic T. Chong is the director of the Greenscale Center for Energy-Efficient Computing, director of the Computer Engineering Program, and a professor of computer science at the University of California, Santa Barbara. His research interests include emerging technologies, multicore and embedded architectures, computer security, and sustainable computing. Chong has a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology.

Timothy Sherwood is a professor in the Department of Computer Science at the University of California, Santa Barbara. His research focuses on the development of novel computer architectures for security, introspection, and embedded applications. Sherwood has a PhD in computer science from the University of California, San Diego. He is a member of IEEE and the ACM.

Direct questions and comments about this article to Hassan Wassel, 1600 Amphitheater Pkwy, Mountain View, CA 94043; hwassel@gmail.com.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.