Efficient Timing Channel Protection for On-Chip Networks

Yao Wang and G. Edward Suh
Computer Systems Laboratory
Cornell University
Ithaca, NY 14853, USA
{*yao, suh*}*@csl.cornell.edu*

***Abstract*—On-chip network is often dynamically shared among applications that are concurrently running on a chip-multiprocessor (CMP). In general, such shared resources imply that applications can affect each other's timing characteristics through interference in shared resources. For example, in on-chip networks, multiple flows can compete for links and buffers. We show that this interference is an attack vector through which a malicious application may be able to infer data-dependent information about other applications (side channel attacks), or two applications can exchange information covertly when direct communications are prohibited (covert channel attacks). To prevent these timing channel attacks, we propose an efficient scheme which uses priority-based arbitration and a static limit mechanism to provide one-way information-leak protection. The proposed technique requires minimal changes to the router hardware. The simulation results show that the protection scheme effectively eliminates a timing channel from high-security to low-security domains with minimal performance overheads for realistic traffic patterns.**

***Keywords*-on-chip network, security, side channel, covert channel**

I. INTRODUCTION

Future computing systems are expected to integrate a large number of processing and memory elements on a die in order to continue scaling the overall throughput within a limited power budget. Today's processors often contain two to eight cores on a chip. More specialized processors such as the Intel Single-chip Cloud Computer (SCC) [7] or the Tilera TILE64 family [14] already contain several tens of cores. For efficient communications, such many-core systems are likely to rely on an on-chip interconnect network and often called network-on-chip (NOC).

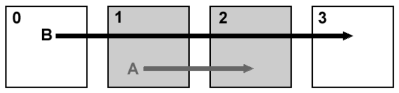
To fully utilize a large number of processing elements, many-core systems need to execute a number of parallel workloads and share resources dynamically. For example, cloud computing infrastructures may support multiple virtual machines with shared physical resources. Therefore, applications on a NOC system can interfere with one another's execution through contention on the network; an application's communication traffic may experience contention and delays that would not have occurred if each application had been run individually. The network contention over shared channels is obviously a concern from the performance, and also introduces fairness and quality-of-service issues, which have been studied recently [11], [5].

In this paper, we study the implications of the shared on-chip network from a security point of view, namely information leak through network interference, and proposes an efficient protection mechanism. In general, the latency and throughput variations from network interferences can be used as timing channels by an attacker either to infer confidential information from a protected high-security program (side channel attacks) or to have a malicious program deliberately leak information covertly when direct communication channels are protected (covert channel attacks). While a potential denial-of-service (DoS) attack on the network has been studied, to the best of our knowledge, this paper represents the first study on timing channels.

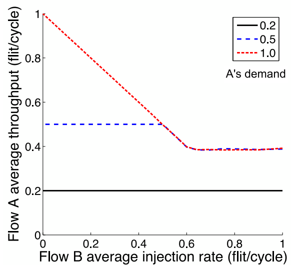
Handling of both side channel and covert channel concerns is critical for systems that require high levels of assurance. For example, consider cloud computing infrastructures that allow virtual machines from multiple customers to share physical hardware. In order for such systems to be viable for business or military customers, a system must provide an assurance that critical trade secret cannot be leaked. In fact, today's cloud computing service contracts may prevent a service provider from sharing physical systems among multiple customers to handle such a concern. Similarly, safety critical systems such as an automobile engine controller cannot make use of multi-core platforms for multiple tasks unless there is a strong guarantee of isolation in order to meet timing deadlines.

Unfortunately, eliminating the network interference can result in a significant performance degradation due to inefficient resource management even when there is no malicious traffic. For example, static partitioning implies that a network can never be fully utilized by one program. In essence, non-interference requires that resource allocation decisions should be independent from application demands.

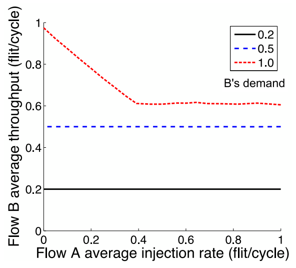
In this paper, we present an efficient solution to remove the timing channel based on the observation that practical systems often need to prevent only *one-way* information flows from high security to low security levels. In this case, we found that the network capacity can be dynamically allocated based on application demands without a security concern by giving a strict priority to the low-security traffic. The priority ensures that the low-security traffic is not affected by the high-security traffic demands. The priority



(a) Network and flow setup.

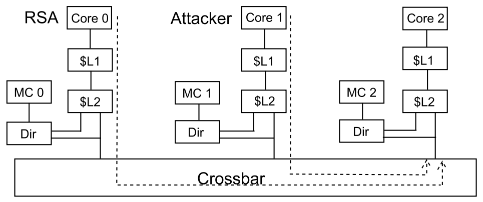


(b) A's throughput, varying B's demand.

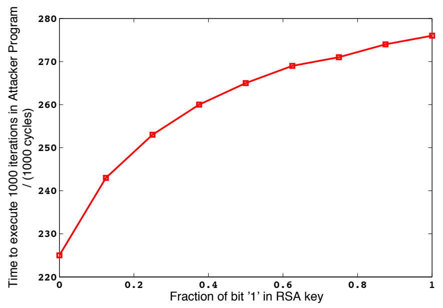


(c) B's throughput, varying A's demand.

Figure 1. A simple network-interference example.



(a) A platform setup.



(b) Attack program's execution time.

Figure 2. A side-channel attack on RSA.

scheme can be easily incorporated into a network router in a manner similar to traditional quality-of-service (QoS) mechanisms. To prevent the low-security traffic to cause a denial-of-service (DoS) attack, we throttle the low-security traffic based on conditions that do not depend on the high-security traffic. For example, the network can enforce a static bandwidth limit on the low-security traffic.

Simulation studies show that the proposed technique indeed eliminates one-way information flows from high-security to low-security levels. Also, the technique only requires minor changes to traditional router designs to add a simple priority-based arbitration and a static virtual channel allocation along with a bandwidth limit. Moreover, the simulation results indicate that the performance overhead of the proposed protection can be minimal, especially when communication demands change over time. The technique allows high-security traffic to fully utilize the network capacity as long as there is no low-security traffic. Similarly, the low-security traffic can utilize the network up to its limit.

The rest of the paper is organized as follows. Section II discusses the timing channel problem in on-chip networks. Section III and Section IV present simple static partitioning approaches as well as our proposal to address this problem. Section V evaluates the effectiveness and performance overheads of the protection techniques. Section VI discusses related works, and Section VII presents our conclusions.

II. Timing Channels in On-Chip Networks

A. *On-chip Network Interference*

Network interference happens when flows from multiple applications contend for shared resources such as links and buffers. As an example, Figure 1(a) illustrates a simple scenario where two flows, Flow A and Flow B, share a common link between Node 1 and 2 on a typical 1-D mesh with 4 virtual channels. Because only one flow can use the link in each cycle, the network performance of one flow, such as throughput and latency, can be affected by the other flow's demand. Figure 1(b) and Figure 1(c) show the throughput of Flow A and B respectively as a function of the other flow's injection rate. In the experiments, we fixed one flow's injection rate (0.2, 0.5, or 1 flit/cycle) and measured its delivered throughput while varying the other flow's injection rate from 0 to 1 flit/cycle. The experiments uses a cycle-level network simulator, named Darsim [12]. As shown in the figure, when a flow has a low injection rate (0.2 flit/cycle), its throughput is not significantly affected by the other flow's injection rate because a round-robin arbitration allocates roughly half of the link capacity to each flow, which is enough to satisfy the demand. However, when the injection rate is high (1 flit/cycle), a flow's throughput is highly dependent on the other flow's injection rate.

The network interference creates a potential timing channel where sensitive information can be leaked intentionally or unintentionally even when legitimate communication channels are disallowed. For example, a program with a high security clearance may covertly leak confidential information to a low-security program by controlling the amount of network traffic to reflect the secret (say, high demand for 1 and low demand for 0). The low-security program can obtain the secret by measuring the throughput or latency of its own flow through the shared network. This covert channel allows the two programs to communicate covertly even when they are not allowed to explicitly send messages to each other or access the same memory locations.

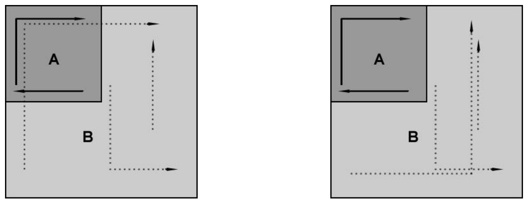
The timing channel can also be exploited to extract secrets from a program, which leaks information unintentionally. This scenario is often called a side channel. As an example, Figure 2(a) shows a case where an RSA algorithm (Core

0) runs on a multi-core platform along with a malicious program (Core 1). RSA [13] is a public key cryptographic algorithm that is widely used to secure electronic communications through encryption or digital signatures. The core of the RSA algorithm performs a modulo multiplication of two large numbers (often 1024 or 2048 bits) depends on each bit in a secret key, and is shown to be prone to timing attacks [8]. Essentially, the algorithm examines each bits in the key and only performs a multiplication if the bit is 1. In our example, we studied a case where the multiplication operation in RSA causes additional network traffic to the memory controller 2 (MC2) due to cache conflicts. Meanwhile, the attack program runs a loop which causes cache misses in every iteration, which also sends memory requests to MC2. Packets from the RSA program and the attack program share the output port of the crossbar and experience network interference. The experiments were performed using McSim [1], which provides timing models for a multi-core platform based on Intel's Pin tool. Figure 2(b) shows the attack program's execution time as the number of 1s in the RSA key varies. As shown in the figure, the execution time has a high correlation with the fraction of 1s in the key. This result suggests that the attacker can roughly estimate the number of 1s from its own execution time, which can greatly reduce the search space to find the correct key.

B. Security Model

The objective of our timing channel protection technique is to prevent sensitive information flows from a high-security domain to a low-security domain through network interference. In particular, our goal is to remove dynamic data-dependent network interference. For example, an application in the low-security domain may still be able to observe that there is another application, but should not be able to observe run-time changes in the high-security domain. In this way, an on-chip network can still be shared among multiple applications. Instead of referring to individual applications, we use the term *domain* to refer to a set of applications that do not need to be isolated from each other.

The security model is based on the traditional multi-level security model (MLS) for confidentiality where information can flow from a lower security level to a higher security level, but not in the other way. As we will discuss later, only preventing high-to-low information flows instead of flows in both directions is essential in enabling efficient techniques that can allow dynamic resource management. Fortunately, the multi-level security model with the one-way protection is applicable to a wide range of applications. In fact, the model is the standard for military and is also widely adopted by industry. For example, the model can be applied to cloud computing by placing security-sensitive corporate virtual machines in a high-security level and common compute workloads in a low-security level, allowing more efficient utilization. While our technique is applicable to multiple security levels, for simplicity, our discussion will focus on the case with two levels - a high-security level and a low-security level.



(a) Baseline. (b) Spatial partitioning.

Figure 3. Non-interference through spatial network partitioning.

In this work, we assume that an attacker controls software on a subset of processing cores along with its placement and scheduling. We also assume that the attacker knows the characteristics of high-security programs including their placement, scheduling, and traffic patterns. In that sense, we assume that attackers can cause its flow to share the network resources with arbitrary flows in the high-security domain. However, the attack program is assumed to be in the low-security domain. Because he only has access to software, the attacker can only observe the network traffic through the latency and the throughput of its own flows.

III. Static Network Partitioning

To remove a timing channel via on-chip networks, the communication latency and throughput of the low-security domain should be independent from the dynamic behavior of applications in the high-security domain. This independence condition implies that all shared network resources, both link bandwidth and buffer space, must be allocated between the security domains in a way that is independent from run-time demands. The most straightforward approach to achieve this goal is to statically partition resources.

Spatial Network Partitioning (SNP) As shown in Figure 3, a network can be partitioned in space to remove interference. The example shows two security domains, A and B, where each domain is allocated with a subset of cores. In the baseline without any timing channel protection, some flows from the two domains can use the same link using the traditional Y-X routing as shown in Figure 3(a). The interference can be removed by requiring each domain to only use the routers that are associated with the nodes that are allocated to itself as shown in Figure 3(b).

Temporal Network Partitioning (TNP) Instead of restricting routers for each security domain, a more flexible partitioning approach can allow sharing of the routers and links while still statically allocate their resources. In this approach, we statically allocate virtual channels (VCs) in each input port to security domains and use temporal scheduling to have security domains take pre-determined turns on a per-cycle basis. For example, each security domain can be allocated with a half of input virtual channels for exclusive use, and be allowed in the switch allocation and link traversal in every other cycle. Effectively, this scheme statically allocates a half of network resources to each domain.

Strengths and Weaknesses Both spatial and temporal partitioning schemes eliminate network interference between the security domains. In fact, the static partitioning remove a timing channel in both directions, not only from high-security to low-security domains. However, both schemes can incur significant performance overheads because resource allocation cannot be dynamically adjusted to match the actual demands from each security domain. For example, even if one security domain has a large bandwidth demand while the other domain has a very small demand, the simple partitioning only allows the domain with the large demand to use at most the half of the output bandwidth. Even if application's traffic patterns are known, the static partitioning cannot efficiently deal with bursts or changes in application phases.

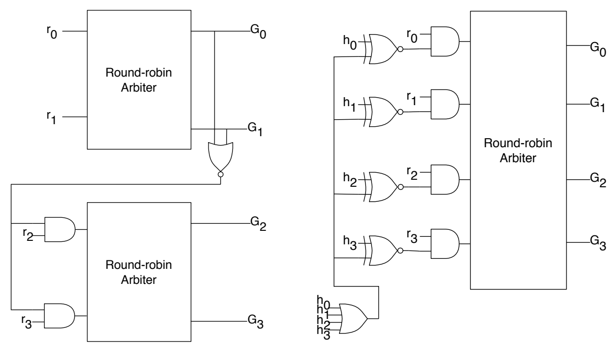
IV. One-Way Information Leak Protection: Reversed Priority with Static Limits (RPSL)

In this section, we propose an efficient mechanism to eliminate a timing channel from high-security to low-security domain. As shown in the previous section, eliminating network interference in both directions suffers from inefficiency because resource managements cannot reflect run-time demands from either domain. On the other hand, protection in one direction allows at least the low-security domain's demand to be used in resource allocation.

A. Overview

In the multi-level security model, the goal is to prevent information flows from the high-security domain to the low-security domain. To achieve this goal without statically allocating resources, our scheme uses a priority-based arbitration for resources such as the router crossbar along with static allocation of virtual channels. The basic idea is to assign a high priority to low-security traffic so that its behavior is not affected by high-security traffic. In other words, when flows from two different security levels compete for the switch traversal, the low-security flow always wins, in both input port and output port arbitrations in a separable allocator. To remove interference in buffers, virtual channels are statically allocated to each security domain. This static allocation removes head-of-line blocking between packets from the two different security domains. In this way, the low security-level flows are not affected by the dynamic demands of the high-security domain. At the same time, the approach allows each security domain to use more network resources dynamically when the other domain has low demands.

While assigning a higher priority to a lower-security domain can prevent the information leak without statically restricting network resources, the approach introduces an obvious concern for fairness. In fact, the strictly higher priority allows a malicious program in the low-security



(a) Input arbiter. (b) Output arbiter.

Figure 4. A priority-based switch allocator design.

domain to easily perform a Denial-of-Service (Dos) attack on high-security programs by sending packets at a high injection rate and occupying all network resources.

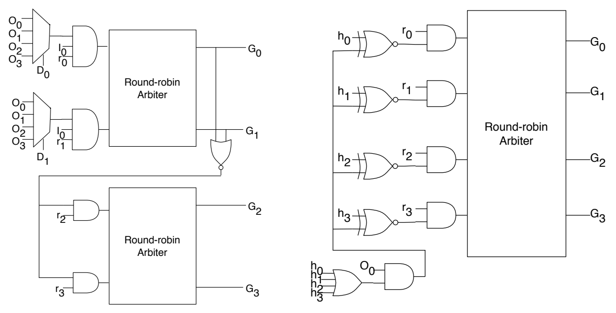
To prevent the DoS attack, we add an additional mechanism that monitors and limits the traffic amount of the low-security domain in a way that is independent from the demand from the high-security domain. This mechanism sets a static limit per port on the number of flits that can be sent by the low-security domain over a certain interval. Once the limit is reached, the port does not send the low-security flits until the next interval, allowing the high-security flits to go through. Note that the static limit does not create a timing channel because it does not depend on the high-security traffic. Also, the limit can change over time as long as it does not reflect sensitive information - network demands from the high-security level.

The rest of the section discusses in detail how this overall approach can be efficiently realized in a modern router architecture, and how the technique differs from traditional quality-of-service schemes.

B. Information Leak Protection

To provide one-way information leak protection, our scheme requires a priority-based allocator that gives a high priority to a lower security traffic. Here, we describe how this mechanism can be implemented based on a typical separable allocator, which consists of input arbiters and output arbiters [3], with statically allocated virtual channels. The input arbiter decides which virtual channel can use each input port to the crossbar, and the output arbiter decides which input port can use each output channel. For simplicity, we illustrate our design for a router with four virtual channel buffers per input port, which are numbered from 0 to 3. To avoid the interference from head-of-line blocking as discussed above, we statically allocate virtual channel 0 and 1 to the low-security domain (domain A), and 2 and 3 to the high-security security domain (domain B).

Figure 4(a) shows our design of an input arbiter. In the diagram, r_0-r_3 are request signals from virtual channels, which compete for the input port to the crossbar. G_0-G_3



(a) Input arbiter. (b) Output arbiter.

Figure 5. A switch allocator design with a static bandwidth limit.

are grant signals indicating which virtual channel wins the input arbitration. The arbiter uses a round-robin arbitration among virtual channels within the same security domain, but gives a priority to the first two virtual channels (0 and 1). A request from a low-priority channel (2 or 3) is given to the round-robin arbiter only if there is no request from the high-priority virtual channel (AND gates in the figure).

Figure 4(b) shows our design of an output arbiter for a 4-by-4 crossbar. Here, r_0-r_3 represent request signals from four input ports, and h_0-h_3 are signals indicating whether the request signals are from the high priority virtual channels ('1') or from the low priority virtual channels ('0'). Again, G_0-G_3 are grant signals indicating which request wins the arbitration. Similar to the input arbiter design, the output arbiter masks a request signal from a low-priority virtual channel if there is any high-priority request. As shown in the figures, adding a priority to the switch arbitration is quite simple and only adds a couple of gate delays to the typical round-robin arbiter.

C. Denial-of-Service Protection

To prevent a potential DoS attack, our approach enforces a static limit on the number of low-security flits that can use each input and output port over a certain interval. For this purpose, a counter is added to each input and output port and records the number of flits from the low-security domain. The counter is compared to a static limit C, and resets every N cycles. C/N represents the maximum fraction of the network capacity that can be used by the low-security domain. For example, if $C = 80$ and $N = 100$, the low-security flows can at most send 80 flits every 100 cycles. The remaining cycles can only be used by the high security flows.

The switch allocator design with this capability is shown in Figure 5. For the input arbiter, we add a signal I_0 to indicate whether the low-security packets has reached its limit for this input port. The signal comes from the input port counter. In addition, we also add control signals from the output ports to prevent low-security flows from exceeding the limit on each output port; O_i indicates whether the output port i reached the limit. Finally, D_0-D_1 indicate the output port of each low-security flow, which is used to select a proper output limit signal. For the input arbiter, a request from a low-security flow is invalided (AND) once the limit is reached for either the input or output ports. Similarly, for the output arbiter, we add O_0 to invalidate the requests from low-security flows once the output limit is reached.

D. Extension to More Security Domains

It is relatively straightforward to extend the RPSL scheme to support more than two security domains. Suppose we have N security domains, and we rank them based on their security levels. The flows from the lowest security domain get the highest priority to use the router crossbar and vice versa. In this way, the traffic from the lower security domain is not interfered by the traffic from the higher security domain, thus preventing information flow from higher security domain to lower security domain. To extend to N security domains, RPSL needs N-1 static limits to avoid DoS attacks, one for each domain except for the highest security domain. The limits may specify the maximum bandwidth usage over a certain period for corresponding security domains or the maximum usage for each security level and below. The arbiter designs can be extended to support more priority levels and static limits with simple additions to the design for two security domains. We need N round-robin arbiters and similar priority control signals for input arbiter. For output arbiter, we need to add compare logic for h_0-h_3 signals to decide which flow has the highest priority.

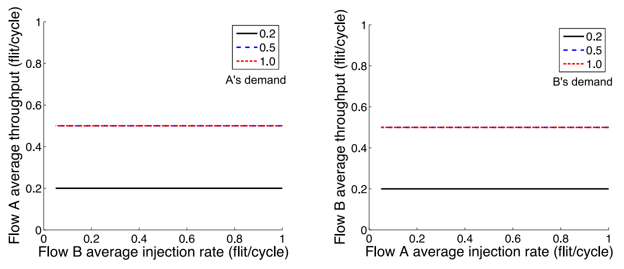
E. Comparison with QoS schemes

The proposed scheme uses the priority-based allocation, which is a common technique in network quality-of-service (QoS) schemes. In fact, at a glance, the timing channel protection may seem similar to guaranteeing a certain level of services in terms of latency and throughput in the traditional QoS context. However, a closer inspection reveals that the QoS properties are insufficient to prevent information leak through network interference. In particular, in a traditional QoS system, a flow can utilize the full capacity of a network beyond its allocation if there is no competing flow. However, such an optimization creates a timing channel in the context of security by reflecting the demand from the high-security domain. For example, the fact that a flow can use more resources than its QoS allocation reveals that other (high-priority) flows have low resource usage at the time.

V. Evaluation

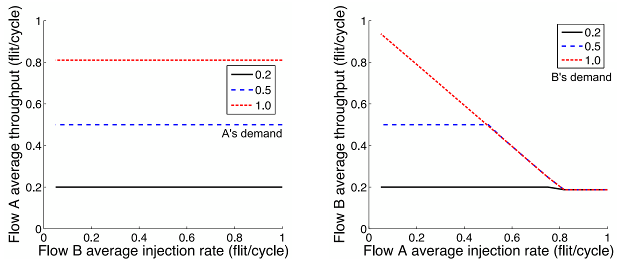
A. Experimental Setup

We evaluate proposed timing channel protection scheme, named RPSL, compared to the baseline without protection and the static temporal network partitioning (TNP, see Section III). The simulations are performed by modifying Darsim [12], which is an open-source cycle-level NOC simulator. The network is configured to be a standard mesh



(a) Flow A (low security). (b) Flow B (high security).

Figure 6. The effectiveness of TNP on a simple example.



(a) Flow A (low security). (b) Flow B (high security).

Figure 7. The effectiveness of RPSL on a simple example.

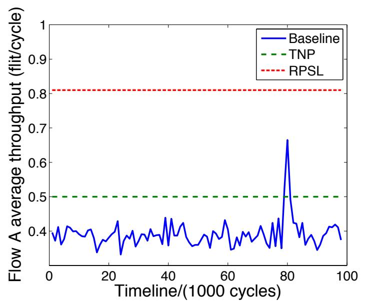
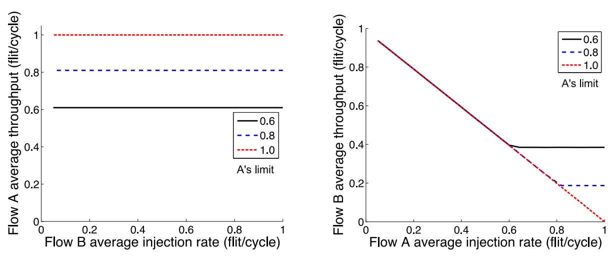


Figure 8. Flow A's throughput over time with varying demands from Flow B. Flow A: low security, Flow B: high security.



(a) Flow A (low security). (b) Flow B (high security).

Figure 9. Flow throughputs with a static limit on the low-security domain.

with four virtual channels per port, and uses iSLIP [3] as the baseline allocator. For TNP, we allocate half of the virtual channels and switch time slots to each security domain. For RPSL, we allocate half of the virtual channels to each domain while prioritizing the low-security flows in arbitration. The static limit for the low-security domain in RPSL is set to be 80 flits per 100 cycles. Each packet consists of eight data flits and one head flit. All our experiments are done with a warm-up period of 20000 cycles, followed by simulation of 100000 cycles.

B. Security

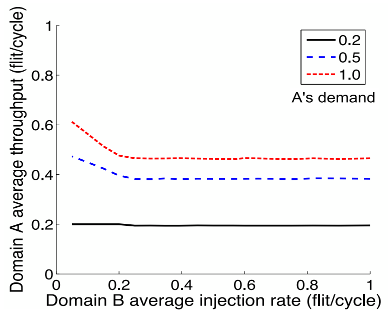
1) Timing Channel Protection: We first study the security of each protection scheme using a simple scenario, which is discussed in Section II (see Figure 1). In this scenario, two flows share a link in the network, causing interference. In the experiments, we assign Flow A to the low-security domain and B to the high-security domain. Then, we observe the throughput of each flow, with a fixed injection rate, while varying the other flow's injection rate.

Figure 6 and Figure 7 show the results for the two protection schemes, TNP and RPSL, respectively. The results shows that both schemes can eliminate the timing channel from the high-security domain to the low-security domain; the throughput of Flow A does not depend on the injection rate of Flow B. In fact, the throughput of both flows are constant under the TNP scheme, indicating that the static partitioning prevents information flows in both directions. On the other hand, Flow B's throughput under RPSL is sensitive to the injection rate of Flow A because the technique only removes a timing channel in one direction.

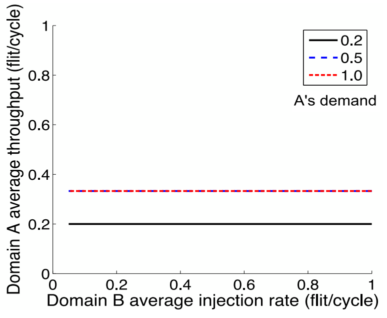
Another interesting observation from Figure 6 is that neither of the two flows' throughput can achieve higher than 0.5 flit/cycle. This is the consequence of statically allocating virtual channels and time slots evenly between two flows, so each of them can at most use half of the network bandwidth. In contrast, the results for RPSL in Figure 7 show that each flow's throughput can be higher than 0.5 flit/cycle because the network resources can be dynamically allocated.

For the experiments above, the injection rates of both flows are kept constant during each simulation run in order to obtain the average throughput. However, in practice, an attack program will measure its throughput over time while keeping its injection rate high. The variations in the observed throughput is used to obtain information about the high-security domain's behaviors. To mimic such an attack process, we also measured the dynamic change in the Flow A's throughput while keeping Flow A's injection rate at 1 flit/cycle and varying the injection rate of Flow B over time. More specifically, we randomly change the injection rate of flow B among 1/3, 1/2, and 1 flit/cycle every 1000 cycles. Figure 8 shows the dynamic throughput changes of Flow A. As the results show, the throughput of Flow A varies significantly for the baseline scheme, depending on the injection rate of Flow B. As expected, the throughput of Flow A stays constant under TNP and RPSL, which further illustrates the two schemes can provide information-leak protection against timing channel attacks.

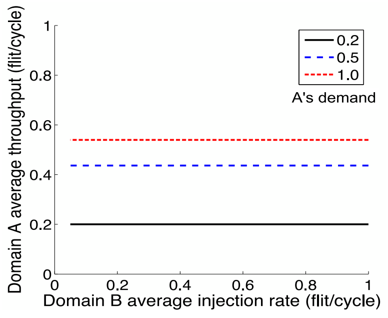
2) DoS Protection for RPSL: To study the effectiveness of having the limit on the low-security domain, Figure 9 shows experimental results with a few different limits. In the first experiment, we fixed Flow A's injection rate at 1



(a) Baseline

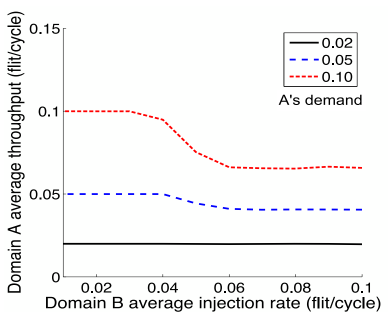


(b) TNP

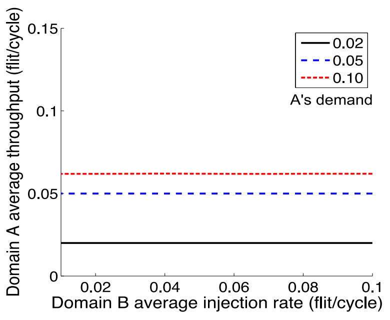


(c) RPSL (limit = 0.8 flit/cycle)

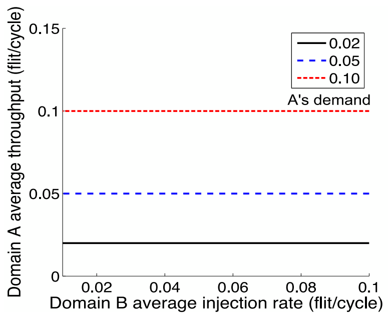
Figure 11. Low-security (domain A) throughput as a function of the high-security (domain B) demand for the transpose traffic pattern.



(a) Baseline



(b) TNP



(c) RPSL (limit = 0.8 flit/cycle)

Figure 12. Low-security (domain A) throughput as a function of the high-security (domain B) demand for the hotspot traffic pattern.

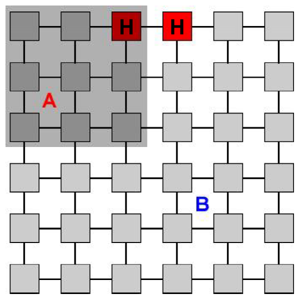


Figure 10. Experimental setup in the 6-by-6 mesh network.

flit/cycle, and measured its throughput while varying Flow B's injection rate under three different limits for Flow A: 0.6, 0.8, and 1.0. Figure 9(a) shows that the throughput of Flow A stays constant and matches the specified limit, which confirms the bandwidth limit for the low-security domain is effective and does not introduce a timing channel.

Figure 9(b) shows that the throughput of Flow B while varying Flow A's injection rate. As shown in the figure, Flow B's throughput decreases with the increasing injection rate of Flow A. However, the bandwidth limit on Flow A allows Flow B's to utilize the network even when Flow A's injection rate is at 1 flit/cycle. Also, note that when the limit is 1.0 flit/cycle, which effectively means no limit, Flow A utilizes the full bandwidth of the network and Flow B's throughput drops to zero, which illustrates a DoS attack. The experimental results show that the bandwidth limit can effectively avoid DoS attacks. Also, the results suggest that the limit can be chosen to be an arbitrary value without introducing a timing channel. Therefore, the limit can be optimized to match the general performance demand of each domain. For example, if the high security domain also has a high bandwidth requirement, we can set the limit to a low value such as 0.2 flit/cycle. In this way, the high secuirty domain can get at least 80% of the bandwdith while still maintaining the security guarantee.

3) Complex Traffic Patterns: To evaluate the protection schemes under more complex interference patterns, we simulated more complicated traffic patterns on a 6-by-6 mesh network with two security domains. As shown in Figure 10, Domain A, which is the low-security domain, was given one-fourth of the cores. We place the cores in contiguous locations because this minimizes the communication cost within a security domain. We simulated two traffic patterns, namely transpose and hotspot. In transpose, a node communicates with the node that is symmetric with respect to the diagonal. In hotspot, all nodes communicate with a single hotspot node, which are marked red in Figure 10. We use Y-X routing for all configurations. As a result, flows from different domains share some links in the network, potentially causing interference. As before, we plot the average throughput of one domain while varying the average injection rate of the other domain. Due to the space limit, we only show domain A's throughput results, which show if there exists a timing channel from high-security to low-

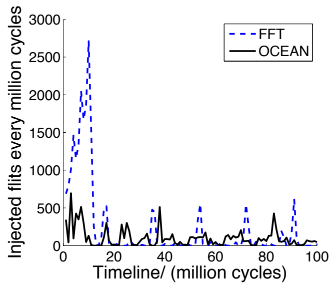


Figure 13. On-chip network traffic for SPLASH-2 benchmarks.

security domains.

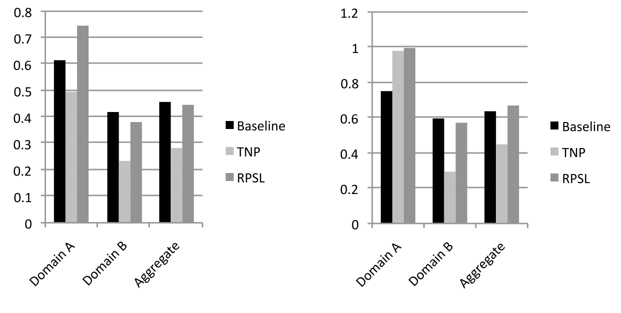
Figure 11 shows the results for the transpose traffic. As expected, in the baseline scheme, the throughput of domain A (low security) changes with the injection rate of domain B (high security). In contrast, domain A's throughput stays constant under both TNP and RPSL regardless of the injection rate of domain B. Therefore, both schemes provides an effective protection against timing channels. However, a distinct difference between the results of TNP and RPSL is that domain A achieves higher average throughput in RPSL than in TNP when domain A's injection rate is high (0.5 or 1 flit/cycle), which again shows the performance advantage of RPSL over TNP scheme. The results of the hotspot traffic pattern in Figure 12 show similar trends.

C. Performance

To provide the capability of information-leak protection, we put restrictions on virtual channel usage and crossbar arbitration, which can potentially decrease network performance. To explore how much performance overhead is incurred, we simulated the three schemes (baseline, TNP, and RPSL with the low-security limit of 0.8) with transpose and hotspot traffic patterns mentioned above.

To mimic the real application behavior, we studied the network traffic for program in the SPLASH-2 benchmark suite using McSim [1]. The experiments collected the traffic statistics for each network flow. From the study, we found that in most programs, the injection rates of network flows vary significantly over time. Figure 13 shows the injection rate of one network flow for FFT and OCEAN benchmarks over time. As shown in the figure, the injection rate changes with time and shows a large dynamic range. Based on this observation, in our experiments, we randomly changed the network flow's individual injection rate every 1000 cycles, for both transpose and hotspot traffic patterns.

For the performance metric, we use actual throughput over throughput demand, which equals to the number of flits received divided by the number of flits injected. Our results for transpose and hotspot traffic are shown in Figure 14(a) and Figure 14(b), respectively. For domain A, RPSL scheme achieves higher performance than that of the baseline scheme. This is because RPSL gives domain A a higher prority to use the crossbar. For domain B, the



(a) Transpose. (b) Hotspot.

Figure 14. Impacts of timing channel protection mechanisms on the network throughput.

performance of RPSL is a slightly lower than that of the baseline. However, the aggregated performance of RPSL is comparable to the performance of the baseline scheme. For the hotspot pattern, we even see an performance improvement over the baseline scheme. Overall, the results show that RPSL has a small performance overhead over the baseline because RPSL can usually utilize available bandwidth at runtime. The only case that the available bandwidth is wasted in RPSL is when the low-security domain has reached its limit and the other domain is sending very little traffic which is insufficient to utilize the remaining bandwidth. Furthermore, the cap can be reconfigured to fit the demand if this situation does happen.

By contrast, the TNP scheme incurs a significant performance overhead. In fact, the performance suffers in TNP scheme whenever one security domain's demand on a single link exceeds a half of the capacity and the other domain's demand is less than half. Therefore, TNP cannot allow each flow to fully utilize the network even for a relatively short period for a burst. Although TNP can provide two-way information-leak protection, the performance overhead is likely to be too significant to be practical in most applications.

VI. Related Work

A. Side-channel Protection

Previous research has studied the possibility of side channel attacks caused by contention over other shared resources such as caches. Wang and Lee showed how SMT, control speculation, and shared caches can create such interference between threads and thereby promote side and covert channel attacks [15]. They then suggested architectural partitioning and randomization schemes to eliminate or decorrelate cache interference from data dependencies [16]. Kong also suggested software solutions to the cache interference problem [9]. However, the cache side-channel protection schemes cannot be directly applied to NOC interference. Also, the side-channel protection does not handle covert channels where a program intentionally leaks information through timing channels.

B. On-chip Network Security

Regarding NOC security, previous work [4] has focused on preventing unauthorized memory operations and denial-of-service. This paper studies a new security problem of timing channels, and show how to efficiently address the problem using a priority with a static limit. To the best of our knowledge, we are not aware of previous work on network timing channels.

C. Quality of Service

Many QoS techniques have been proposed to provide performance isolation and differentiated services to different network flows. Virtual Clock [17] provides performance isolation by assigning virtual clock values to packets based on the reserved rates, and the virtual clock value is then used for crossbar arbitration, thus guaranteeing the reserved rates for each flow. The idle resources are dynamically reallocated among flows. Preemptive Virtual Clock [5] keeps track of each flow's bandwidth consumption and prioritizes packets based on the consumed bandwidth and established rate of service, thus providing guaranteed performance. Another QoS technique, Globally-Synchronized Frames [11], achieves performance isolation by reserving slots in a global frame for each flow. A key difference between these QoS techniques and the RPSL scheme is that the QoS techniques traget at using network bandwidth efficiently and do not keep the available bandwidth unused if there is demand. In RPSL, to provide information-leak protection, the lower security-level flow cannot use the available bandwidth once it exceeds the limit.

D. Composability

Due to interference between applications, the integration and verification complexity of multi-processor SoC grows exponentially with the number of applications. Composability is proposed to remove all interference between applications so that each application can be verified individually without slow system-level simulation of all use cases. Unlike our RPSL security scheme which utilizes the one-way information leak protection, a composable system requires bi-directional non-interference between applications, thus incurring a larger performance or hardware overhead. An approach of achieving composability is not sharing any resources, which is expensive and mostly used by federated architectures in aerospace industries. Kopetz [10] and Hansson [6] proposed the Time Division Multiple Access (TDMA) approach, which schedules the network communication based on a static time table, similar to the TNP approach in our paper. As shown in our experiments, the TDMA approach suffers from large performance overhead compared to our RPSL scheme. Akesson [2] used an approach based on latency-rate servers. Although it overcomes some shortcomings of the TDMA approach, it adds significant hardware to achieve the goal of composability. In summary, due to the fact that composability needs a bi-directional non-interference, the composability approaches have larger performance overhead than our RPSL scheme.

Also, because the goal of composability is to reduce the complexity of system integration rather than removing timing channels, some composability approaches allow dynamically reconfiguring the resources to fulfill the changing application behavior. As a result, these approaches are not secure from the information flow perspective.

VII. Conclusion

In this paper, we proposed a security technique RPSL to prevent timing channel attacks caused by interference in on-chip networks. We demonstrated that the network interference among applications executing concurrently on the same CMP can be used as timing channels to obtain confidential information. The RPSL mechanism eliminates a one-way timing channel from high-security to low-security domains by assigning a high priority to the low-security domain. For complete elimination of timing channels, we also discussed two static network partitioning schemes in either space or time. These protection schemes are compared through cycle-accurate network simulations. The results suggest that both RPSL and static partitioning provide effective protection against timing channels, and the one-way protection through RPSL can be realized with minimal performance overheads.

VIII. Acknowledgments

This work was partially supported by the National Science Foundation grants CNS-0746913, CNS-0708788, and CNS-0905208, and an equipment donation from Intel Corporation. We thank Skyler Schneider and Kun Ki for their contributions to the experimental framework, and anonymous reviewers for constructive comments.

References

[1] http://cal.snu.ac.kr/mediawiki/index.php/McSim.

[2] B. Akesson, A. Hansson, and K. Goossens. Composable Resource Sharing Based on Latency-Rate Servers. In *Proceedings of the EUROMICRO Conference on Digital System Design*, pages 547–555, Los Alamitos, August 2009. IEEE Computer Society Press.

[3] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[4] 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*, pages 197–202. IEEE, 2008.

[5] B. Grot, S. 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*, pages 268–279. ACM, 2009.

[6] A. Hansson, K. Goossens, and M. Bekooij. CoMPSoC: A Template for Composable and Predictable Multi-processor System on Chips. In *Transactions on Design Automation of Electronic Systems*, pages 1–24, 2009.

[7] Intel. Intel Single-chip Cloud Computer. product overview. http://techresearch.intel.com/spaw2/uploads/files/SCC-Overview.pdf.

[8] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology CRYPTO '96*, pages 104–113. Springer-Verlag, 1996.

[9] J. Kong, O. Aciicmez, J.-P. Seifert, and H. Zhou. Hardware-software Integrated Approaches to Defend Against Software Cache-based Side Channel Attacks. In *IEEE 15th International Symposium on High Performance Computer Architecture.*, pages 393–404, February 2009.

[10] H. Kopetz, C. El Salloum, B. Huber, R. Obermaisser, and C. Paukovits. Composability in the Time-triggered System-on-Chip Architecture. In *SOC Conference, 2008 IEEE International*, pages 87–90, sept. 2008.

[11] 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*, pages 89–100. IEEE, 2008.

[12] M. Lis, K. S. Shim, M. H. Cho, P. Ren, O. Khan, and S. Devadas. DARSIM: A Parallel Cycle-level NoC Simulator. In L. Eeckhout and T. Wenisch, editors, *MoBS 2010 - Sixth Annual Workshop on Modeling, Benchmarking and Simulation*, Saint Malo, France, 2010.

[13] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[14] Tilera. Tilera TILE64 Processor. product overview. http://tilera.com/sites/default/files/productbriefs/PB010_TILE64_Processor_A_v4.pdf.

[15] Z. Wang and R. B. Lee. Covert and Side Channels Due to Processor Architecture. In *Proceedings of the 22nd Annual Computer Security Applications Conference*, pages 473–482. IEEE, 2006.

[16] Wang, Z. and Lee, R. B. New Cache Designs for Thwarting Software Cache-based Side Channel Attacks. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 494–505. IEEE, May 2007.

[17] L. Zhang. Virtual Clock: A New Traffic Control Algorithm for Packet-switched Networks. *ACM Transactions on Computer Systems (TOCS)*, 9(2):101–124, 1991.