ConNOC: A practical timing channel attack on network-on-chip hardware in a multicore processor

Usman Ali and Omer Khan
University of Connecticut, Storrs CT USA
{usman.ali, khan}@uconn.edu

***Abstract*—Shared hardware resources in today's microprocessors have emerged as a target for adversaries to leak secret information via timing-based software side channels. This paper characterizes such attacks on the non-persistent network-on-chip (NoC) hardware, and demonstrates its practicality on a real multicore machine. State-of-the-art 4-core baseline setup shows an average of less than 2-cycle latency variation due to contention at the NoC hardware resources. This noisy and unpredictable timing channel achieves ∼18% accuracy when the attacker is assumed to have no replay capability. However, in the presence of a high number of replays, accuracy improves significantly but slows down the speed of the attack. ConNOC proposes a novel attack setup that better exploits interference at NoC hardware to make it less noisy and more predictable. It demonstrates ∼7-cycle latency variation under no replay capability. The evaluation of covert communication and information leakage attacks shows 100% accuracy using five replays to leak information. This translates to 2 *mbps* (mega bits per second) attack throughput on the Tilera *TileGx72* multicore processor executing at 1GHz.**

I. Introduction

In today's highly virtualized systems, adversaries are actively using co-located software on commercial processors to leak secret information through timing-based software side-channel attacks (sSCA). Timing-based sSCA leaks information with the help of contention-induced timing variations in the chip-level shared hardware resources. Google recently published their proof-of-concept code showing the practicality of Spectre exploits within modern web browsers on real processor hardware [1]. The shared hardware side-channels are broadly categorized as persistent and non-persistent. The persistent channels constitute resources that hold information for a larger time interval, e.g., caches, TLBs, or even main memory. On the contrary, non-persistent channels hold information in shared hardware for a shorter time interval, e.g., execution units or network-on-chip routers and wires. These vulnerabilities have motivated the development of a plethora of mitigation schemes for persistent channels [2], [3], [4]. Consequently, adversaries are now more inclined to target non-persistent channels for timing-based sSCA attacks.

The non-persistent channels are not easy to exploit and require a sophisticated setup due to their shorter time for holding data [5], [6]. It has been shown in literature that non-persistent channels have low latency variations of about 2–10 cycles, as compared to 10s or even 100s of cycles for persistent channels [7], [8]. Another limitation of non-persistent channels is their high noise levels. Thus, an attack must rely on replay capabilities to confidently leak information [9]. Due to these limitations, a realistic attack on non-persistent channels requires evaluation under realistic constraints on a real machine. Existing strategies [6], [10] unfortunately do not consider these challenges with the attack setup, and focus on mitigation techniques based on spatial and/or temporal isolation principles. In order to design an efficient mitigation scheme for a non-persistent timing-based sSCA, the adversarial attack capabilities need further investigation.

This paper focuses on timing-based sSCA that exploits network-on-chip non-persistent hardware channels in a real multicore machine setup. An attack scenario [6] (baseline attack) is proposed in the literature that consists of two independent applications, where their code and data are spatially distributed across 4-cores in a multicore. These cores are interconnected using the NoC hardware routers and wires. Simulated evaluation using a cycle-level network simulator shows that these applications observe timing variations due to interference at the NoC hardware. The timing variations depend on the contention level, whereby when the NoC hardware is under contention, it leads to an additional latency delay as compared to the no-

contention scenario. Based on these timing variations, followup research proposes mitigation strategies for NoC hardware [10], [2]. However, it is unclear if the prior literature correctly characterizes this non-persistent hardware side-channel. Can timing variations in this setup correctly capture the adversarial attack on a real machine? How can one manage the high noise challenge in this channel and ensure high accuracy to conduct an attack at high performance? Moreover, no prior work has demonstrated the use of NoC hardware channel to conduct real attacks, i.e., covert communication and information leakage. The covert-communication attack allows two unauthorized malicious applications to communicate based on timing variations in shared hardware channels [11]. Whereas, in an information leakage attack, an adversary infers secret from a secure application (e.g., AES or RSA crypto-system private keys) based on timing variation patterns [11].

We revisit the baseline attack on NoC hardware and characterize its timing variations on a real Tilera *TileGx72* multicore processor. Our evaluation shows that the baseline attack's placement of code and data observes an average of 1.58-cycle latency variation due to contention at the NoC hardware. This leads to an unreliable attack setup with ∼18% accuracy for inferring a secret bit of information. However, accuracy increases when the adversary possesses replay capability, where each bit leakage step is repeated to increase the latency under contention. The baseline setup is shown to require 55 replays to leak a bit with ∼100% accuracy.

ConNOC proposes a novel method for code and data placement of victim and adversarial applications that efficiently exploits the NoC hardware side-channel. The proposed setup maximizes the number and probability of activating the contended NoC hardware resources, and results in an average of ∼7.3-cycle latency variation without replay. The statistical mean of contention and no-contention cases is used as a threshold to differentiate the high-noise timing variations at the NoC hardware with ∼62% accuracy, a 3.4× improvement over the baseline setup. Moreover, a heuristic is developed that uses replay capability to infer each secret bit of information with high accuracy, yet high speed. The efficacy is measured using accuracy (true positive rate) and rate (minimum number of replays) metrics. The proposed 3-core setup is demonstrated for covert communication and information leakage attacks with 100% accuracy using 5 replays, as compared to 55 in the baseline setup. This translates to 2 *mbps* compared to baseline 180 *kbps* attack throughput, which is an 11× speedup for the proposed attacks on the target multicore processor executing at $1GHz$.

II. Related Work

Persistent hardware side-channels are an easy target for an adversarial software to leak secret information in today's microprocessors [1]. For example, Bernstein et al. [12] exploit timing variations in on-chip caches to leak AES private key. Osvic et al. [8] propose prime+probe, whereas Yarom et al. [7] propose flush+reload attack that targets cache variations to leak secret keys of AES and RSA crypto-systems. In cache attacks [7], the adversary observes up to 80-cycle variations between cache-hit and cache-miss scenarios. This allows the attacker to build a highly reliable sSCA. Researchers have also targeted other persistent hardware channels, for example TLBs [13], branch target buffers [14] [15], and main memory DRAM rows [16]. With the emergence of mitigation mechanisms being deployed in commercial processors, non-persistent channels are an emerging target for attackers. Recent work [5] [17] successfully exploits execution engine to bypass security policies and leaks secret information. Moreover, in the context of multicore processors, the on-chip network hardware has been shown as another non-persistent hardware side-channel [6].

Suh et al. [6] proposed an attack setup that consists of two applications, X and Y that are spatially distributed across 4-cores in a multicore setting. The cores are interconnected using the NoC hardware. Figure 1-a shows the two applications executing on cores 0 and 1 respectively. The application X accesses its data from core 3's last-level shared cache slice, while application Y from core 2. In a cache coherent shared memory system, these data accesses are performed at the cache line granularity. However, the network bandwidth is limited to a fixed flit size, whereby a cache line is split into multiple flits and moved between an application's code and data locations using the NoC hardware. It has been shown that while these applications are isolated processes at the software level, they share NoC hardware, thus creating a side-channel for information leakage. One application creates contention at the NoC hardware that results in observable latency variations as compared to no-contention case for the other application. This latency variation is exploited to create timing-based sSCA. However, prior work falls short of characterizing attacks on a real machine

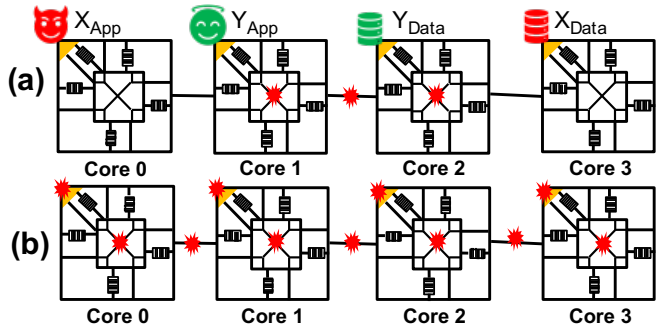

Fig. 1. (a) Layout for a 4-core baseline attack scenario with three points of contention. (b) Sources of contention in NoC hardware: switch, link, and host-interface.

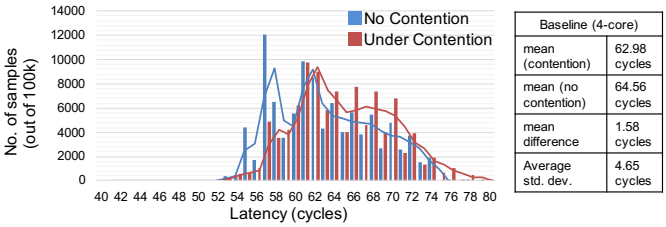

Fig. 2. The latency histogram distribution for the 4-core baseline setup with statistical mean and standard deviation data for no-contention and contention scenarios.

to confirm the practicality of the proposed setup [6]. Instead, research efforts have focused on mitigation strategies based on this potential NoC attack setup [10]. Without a clear understanding of the adversarial capabilities, it is unrealistic to assume the efficacy of the prior mitigation schemes for this non-persistent channel.

III. Motivation

This work characterize the baseline setup proposed by Suh et al. [6] on a real multicore processor, Tilera *TileGx72*. The NoC hardware in the evaluated processor consists of queues, switches and wires. We identify three sources of contention that result in timing variations, i.e., switch, link, and host-interface. The switch establishes a connection between the incoming input bridges to create a channel for communication. Once the connection is established, the switch becomes unavailable for other bridges. If there is any data on non-connected bridges, it is stored in attached buffers. This non-availability of switch causes contention delay. The links are wires that connect NoC hardware with adjacent cores. At any given point, these wires can be occupied by a single data transfer, while the other application's data must await its turn in the core's buffer. Data multiplexing at links is another source of contention delay. A host-interface connects core resources (i.e., execution units and caches) with the switch. Again, the wires interconnecting the incoming and outgoing traffic at the host-interface are a source of contention. As shown in Figure 1-b, the 4-core setup has many highlighted points of contention that are open for an adversary to create a reliable attack. However, the data requests and replies from the two applications must co-locate on one or more of these hardware resources to create latency variations. The objective is to maximize the number of contented resources, as well the probability of co-location at runtime. It is clear from Figure 1-a that the baseline attack setup [6] only activates three points of NoC hardware contention. Since both adversarial and victim application's data accesses must align on one or more of these sources, the probability of success is limited, as discussed next.

A. Low Timing Variations in Baseline Setup

The non-persistent NoC hardware channel holds data for a much shorter time. For example, a data item (flit) traversing over an un-contended NoC reaches its destination in 1-2 cycles per hop. The baseline setup (Figure 1-a) shows data movement paths in the presence or absence of the victim application Y. The victim application Y accesses its data from core 2 periodically. A well synchronized contended NoC causes a delay in data access latency. As shown in Figure 2, when application Y is inactive (no contention), the adversarial application's data access request takes 62.98 cycles on average. However, when application Y is actively making data access requests to core 2 (contention points activated on NoC hardware), the adversary's each data access now takes 64.56 cycles on average. This leads to a ∼1.58-cycle timing variation that the adversarial application X must exploit to construct an sSCA attack. However, this low timing variation is unpredictable as observed by the high standard deviation of access latencies in Figure 2. The low latency and highly variable timing variations exacerbate the detection capabilities to conduct a successful attack.

B. High Noise and Low Accuracy in Baseline Setup

The co-located concurrent applications introduce unwanted interference (or noise) on the NoC hardware channel, further limiting the success of the adversarial

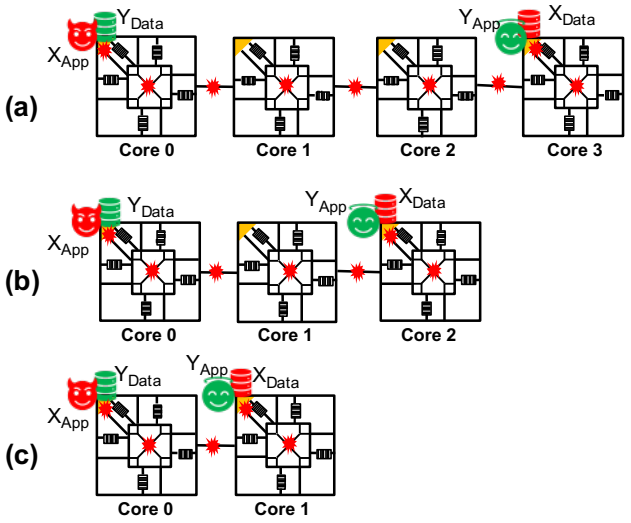

Fig. 3. Proposed ConNOC setups: (a) 4 node placement of data & code, (b) 3 node data & code placement, and (c) 2 node data & code placement.

application. For the baseline setup, Figure 2 shows the latency histogram under contention and no contention scenarios. In the absence of contention, the latency varies from 50 to 80 cycles with a mean of 62.98 cycles. However, when the victim application creates contention, the latency still ranges from 50 to 80 cycles, but the mean shifts to 64.56 cycles. For both scenarios, the standard deviation is observed in excess of 4 cycles, which shows high overlap between the two scenarios. In order to confidently differentiate between the no-contention and contention cases, an adversary must rely on replay capability [9], i.e., repetition of contention or no contention to remove noise. Our real machine evaluation shows that for the baseline attack, true positive rate is ∼18% without the replay capability and ∼100% when the replay capability is utilized 55 times. This makes the baseline setup adequate to create a practical attack, but at slow speed. Therefore, the main objective of this work is to develop an attack setup that utilizes the NoC hardware side-channel to create predictable, and efficient sSCA attacks.

IV. ConNOC

The baseline attack on the NoC hardware side-channel is primarily limited by the placement of code and data for the adversarial and victim applications in a multicore setting. As observed in Figure 1-a, the setup severely limits the number of potential contended hardware resources. Moreover, the success of the attack depends on the probability of aligning contented data accesses on one or more of these sources. ConNOC proposes a novel data placement scheme that maximizes the number of contention points, as well as the probability of success to activate them at runtime. The key idea is to activate multiple core's NoC routers concurrently in such a way that at any given time instance, the flits from each cache line access occupy as many switches, adjacent core links, and host-interface links as possible. Moreover, the cache line accesses from two applications are setup to traverse in opposite directions to maximize the probability of contending the NoC hardware resources. This is done by placing the code and data for each application on the extreme ends of the multiple cores setup in a multicore. An important factor is the number of cores that actively participate in the attack setup. When the number of cores in small, the number of potential contention points are constrained while the probability of activating them is high. On the other hand, when the number of cores is large (potentially each flit of a cache line occupying a core), the setup utilizes a much larger number of contention points. However, the probability of activating contention on multiple NoC resources decreases. This leads to a tradeoff that must be characterized and quantified to configure the setup with the right number of cores.

A. Code and Data Placement Setup

ConNOC is a data and code placement strategy that aims to maximize the number of NoC hardware contention points. The maximum occupancy of a path from an application's data access point of view is the number of flits per access. For example, if the processor implements 64-bytes cache line and 8-bytes (64-bits) flit size, the maximum number of cores occupied by a single path is 8 cores. In this scenario, ConNOC utilizes 8-cores where the code and data placement are done on the extreme points of the interconnected cores. The adversarial and victim application aims to execute two concurrent paths in opposite directions to create contention in the various NoC hardware resources. The objective of this paper is to demonstrate the feasibility of sSCA attacks, thus the evaluation is conducted on 2- to 8-core setups. To convey key tradeoffs in the setups under ConNOC, Figure 3 shows the proposed 4-core, 3-core, and 2-core setups.

Figure 3-a shows the 4-core placement of code and data. An application X (adversary) is pinned on core 0, where data of application X is pinned on core 3's last level shared cache slice. Application Y is pinned

on core 3, whereas its data is pinned on core 0. This placement activates switches of all four cores, three links interconnecting the four cores, and the two host-interface links on cores 0 and 3. The flits of data accesses originating from the two applications can contend on one or more of these contention points at any given time instance, creating a timing side-channel. ConNOC's 4-core setup activates 3× more contention points as compared to the baseline setup from Figure 1-a. Therefore, depending on the number and probability of activated contention points, this setup is expected to improve both statistical mean difference, as well as the standard deviation. Although not shown in Figure 3, five or higher core count setups introduce more contention points. However, the probability of activating one or more of the contention points decreases. This potentially leads to higher variability in access latencies under the contention case, thus leading to a degradation in statistical mean difference and an increase in standard deviation.

To improve the probability of activating contention points, ConNOC's strategy is to reduce the number of cores involved in the setup. Figure 3-b shows the 3-core placement of code and data using the same strategy as described for the 4-core setup. This placement activates switches of three cores, two links interconnecting the cores, and the two host-interface links on cores 0 and 2. Although the number of contention points is reduced compared to the 4-core setup, the benefits arise from increasing the probability of the flits activating one or more of the contention points. Consequently, the 3-core setup is expected to improve both statistical mean difference and standard deviation over the 4-core setup. At the extreme end, only two cores are utilized to create the ConNOC setup, as shown in Figure 3-c. Although this 2-core setup maximizes the probability of activating contention points by the two applications, it's efficacy is limited by the number of exploitable contention points. This setup is expected to improve the standard deviation as compared to the 3- and 4-core setups. However, the statistical mean difference is lower due to its limitations on the number of contention points.

B. Information Leakage Heuristic

One of the challenges of timing-based sSCA on non-persistent channels is high noise. ConNOC proposes a heuristic that efficiently differentiates no-contention and contention scenarios to overcome this challenge. A threshold acts as a boundary value between the two scenarios, resulting in a clear identification of contention and no contention cases. From the adversarial application view point, all latency values that are equal or less than the threshold are considered no-contention, while greater than threshold are considered contention cases. However, in the presence of high noise in this non-persistent channel, a single access is insufficient to infer these cases with high confidence. ConNOC proposes to replay each bit-leakage (contention and no contention) scenario multiple times, where each access is repeated multiple times, and the aggregate latency variations are used as a timing-channel. For example, if the adversarial application replays a data access n times, the latency measurement is aggregated by n times. Consequently, the mean difference increases between the two scenarios, which reduces noise between the no-contention and contentions scenarios. The objective is to reduce n while achieving near 100% accuracy of bit-leakage. Accuracy is defined as the *true positive rate*, which is quantified as the percentage of classifications that result in correct inference of the contention and no contention using the heuristic's threshold.

The selection of the threshold value is an important factor in the heuristic. ConNOC utilizes a learning phase where statistically significant number of samples are collected for both no-contention and contention cases. The mean latency value that differentiates both distributions with associated number of replays is consequently used to set the threshold value.

C. Practical Attacks

Timing-based sSCA are classified into two categories, i.e., covert-communication and information-leakage attacks. In covert-communication attacks, a trusted but malicious application with access to secure data leaks secrets to another application. The leaked data is later exfiltrated to malicious actors. In information-leakage attacks, an adversarial application infers secrets from a non-compromised trusted application. For example a trusted application like AES or RSA with private keys is the target for an adversary. Practical attacks require a shared synchronization structure (i.e., global cycle counter) to achieve maximum accuracy.

1) Threat Model: This work assumes a common threat model for timing-based sSCA [7]. An adversary executes an application with user-level privileges, and possesses the capability to co-locate and place code and data across different cores in a multicore processor setting. Modern processors allow user-level programs to control code placement and data for higher per-

formance in parallel applications. The adversary also has replay capability that allows repeating operations multiple times. The operating system (OS) controls the executions of instructions, creates checkpoints, and replay instructions for consistency purposes. The threat model considers the OS as a malicious entity, and requires a compromised OS for an information-leakage attack. However, the covert-communication attack works without a malicious OS since both application collude to conduct this attack.

2) Covert communication attack: Two independent applications, X and Y exploit the ConNOC setup to communicate covertly. The 3-core setup from Figure 3-b is used to explain the covert-communication attack. Applications X and Y act as receiver and transmitter respectively. Figure 4 shows the pseudo-code of a covert-communication attack. To transmit a secret bit 1, the transmitter waits for synchronization event (i.e., a fixed time quanta), and makes *num of replays* number of memory accesses that result in contention at the NoC hardware. On the other hand, to transmit secret bit 0, the transmitter remains idle and makes no accesses for the given time quanta. Similarly, the receiver waits for the synchronization event, and makes *num of replays* accesses to memory, and measures their aggregate latency. The receiver uses the ConNOC replay-based heuristic to infer the secret bit from this timing information. For example, if the measured latency is equal to or less than the set threshold, it infers a secret bit 0. Otherwise, if the measured latency is greater than the threshold value, the receiver infers a secret bit 1. This process is repeated to transmit a stream of secret information between the two applications.

```
// transmitter pseudo code
wait_sync_tick()
for every bit:
  if (bit == 1):
    for (num of replays):
      function1()
  else if (bit == 0):
    do_nothing()
```

```
// receiver pseudo code
wait_sync_tick()
for every bit:
  timer_start()
    for (num of replays):
      function1()
  stop_timer()
  if (time > threshold):
    bit = 1
  else
    bit = 0
```

Fig. 4. Covert-Communication attack pseudo-code.

3) Information leakage attack: An adversarial application exploits the ConNOC setup to leak secret information from a secure application. For information leakage attack, a scenario is modeled where a secure application operates on a secret key. Figure 5 shows the pseudo-code for the information leakage attack. Again, the 3-core setup from Figure 3-b is used for this attack. A secure application Y consists of a function that makes memory accesses based on a secret bit value. For example, if the secret bit value is 1, function1 is called followed by function2. Whereas, if the secret bit value is 0, only function1 is called. RSA square and multiple algorithm is an example that performs such function sequences based on the secret key [7]. An adversary application X executes and makes timing measurements for periodic memory accesses. For example, if the measured latency is greater than the threshold, it implies function1 is executed. Whereas, if measured latency is less than or equal to the threshold, it implies function2. An execution of function1 followed by function2 is inferred as secret bit 1, where detection of function1 followed by function1 is inferred as secret bit 0. Note that in this attack, the malicious OS is assumed to control synchronization and replays for the secure application.

```
// secure pseudo code
wait_sync_tick ()
function(secret_bit):
  if (secret_bit == 1):
    for (num of replays):
      function1 ()
      function2 ()
  else if (secret_bit == 0):
    function1 ()
```

```
// adversary pseudo code
wait_sync_tick ()
for every secret_bit:
  start_timer ()
    for (num of replays):
      read_data()
  stop_timer ()
  if (time > threshold):
    current_bit = 1
  else
    current_bit = 0
  if (last_bit == 1 && current_bit = 0):
    secret_bit = 1
  else if (last_bit == 1 && current_bit = 1):
    secret_bit = 0
```

Fig. 5. The pseudo-code for information leakage attack using secret data-dependent functions.

V. Methodology

The Tilera® *Tile-Gx72™* multicore processor is used for evaluation [18]. It is a tiled architecture with 72 independent tiles, and Network-on-Chip hardware comprises of 5 independent 2-D mesh networks (called iMesh™ [18]) with X-Y routing. Each tile consists of a 64-bit multi-issue in-order core, $32KB$ private level-1 (L1) data and instruction caches, and a shared level-2 (L2) cache of $256KB$ (LLC capacity of $18MB$). The cache line size is 64 bytes, while the NoC flit size is 8 bytes (64 bits). iMesh comprises five independent

networks, i.e., cache coherence traffic (TDN), memory controller traffic (MDN), static network (STN), I/O dynamic network (IDN), and user dynamic network (UDN). Moreover, it consists of four on-chip 72-bit ECC-protected DDR memory controllers to access off-chip memory. The system is booted with GNU/Linux operating system with kernel version *3.10.55-MDE-4.3.2.182362*. Tile-Gx72™ API library, Tilera Multi-core Components (TMC) is used to manage network traffic and tile resources.

The cache coherent TDN NoC hardware of iMesh is used in this paper. We use the baseline 4-core setup to explain the attack methodology. To measure the timing variations of TDN, the GNU/Linux command *numactl* pins the code for application X to tile 0. The TMC library allocates a memory page ($4KB$ data structure) using *tmc_alloc_set_home()* on tile 3 (i.e., the data structure uses tile 3's L2 cache slice as it's home location). The TMC library call *tmc_alloc_set_caching()* is used to disable local caching for tile 0. Whenever application X on tile 0 fetches data from main memory (i.e., read a variable *int x*), the data is moved to tile 3's shared L2 cache, and after that to tile 0's register file (local caching on tile 0 is disabled). The TMC library function *get_cycle_count()* fetches the cycle counter value. The time is measured in four steps. (1) Initially, application X on tile 0 reads data (*int x*) that is brought on-chip from main memory and placed on tile 3's shared L2 cache. (2) Read the time counter value using *get_cycle_count()* and store into a temporary variable. (3) Application X on tile 0 reads *int x* again. The data (a cache line, 64 bytes) moves from tile 3 shared L2 cache, but only the requested word is stored in the register file of tile 0. (4) Fetch time counter value, and subtract from the earliest time counter value to measure the timing latency. This latency includes time to fetch data from tile 3's L2 cache and its traversal over the TDN NoC hardware of the iMesh network towards tile 0.

In the no-contention scenario, only application X executes in the system. However, in the case of the contention scenario, the code for another application Y is pinned on tile 1, whereas its data structure is allocated to tile 2 using the appropriate TMC library calls. While keeping the measurement data size the same (i.e., a single cache line), application Y makes concurrent access from tile 1 to tile 2, thus creating opportunities for contention in the shared NoC hardware resources. In the case of replay enabled attack, both applications make n repeated accesses under both no-contention and contention scenarios to create aggregated latency measurements. Here n is the number of replays for an sSCA attack.

VI. Evaluation

The baseline and ConNOC setups are evaluated with and without replay capability for their latency variations, as well as the true positive rate metric. The covert-channel and information-leakage attacks are demonstrated using ConNOC 3-core placement on Tilera TileGX-72.

A. Evaluation without replay capability

This section evaluates ConNOC placement using 2-core, 3-core, 4-core, and 5-core setups. The application X on core 0 makes a variable read access (that translates to cache line access at the hardware level), and measures the latency with and without contention on NoC hardware. Under no-contention scenario, only application X executes. However, for contention scenario, application Y makes continuous read accesses to its data placed on core 0. This occupies the NoC hardware resources, and application X observes contention latencies. This process is repeated for 100,000 samples for both contention and no-contention scenarios.

Figure 6-(a) shows the results for the 5-core setup, where a large number of contention sources are possible but the probability of activating one or more of these sources is low. The mean latency difference is 2.76 cycles with standard deviation of 4.45 cycles. Compared to the baseline setup (cf. Figure 2), the latency difference is 75% higher that helps create less overlap in the latency distributions. Consequently, as shown in Figure 7, the true positive rate of the 5-core setup improves to ∼55%, as compared to ∼18% for the baseline setup.

The 4-core setup, shown in Figure 6-(b) reduces the number of sources of contention, but increases the probability of contention. This improves the mean latency difference to 5.18 cycles with a standard deviation of 5.21 cycles. This trend is further improved in 3-core setup (Figure 6-(c)), where mean latency difference is 7.3 cycles with a standard deviation of 3.73 cycles. As a result, the 4-core setup improves the true positive rate to ∼62%, while the 3-core system gives the best result at ∼65%.

For the 2-core setup, although the probability of contention increases, the setup exposes a lower number of sources for contention. Therefore, as shown in Figure

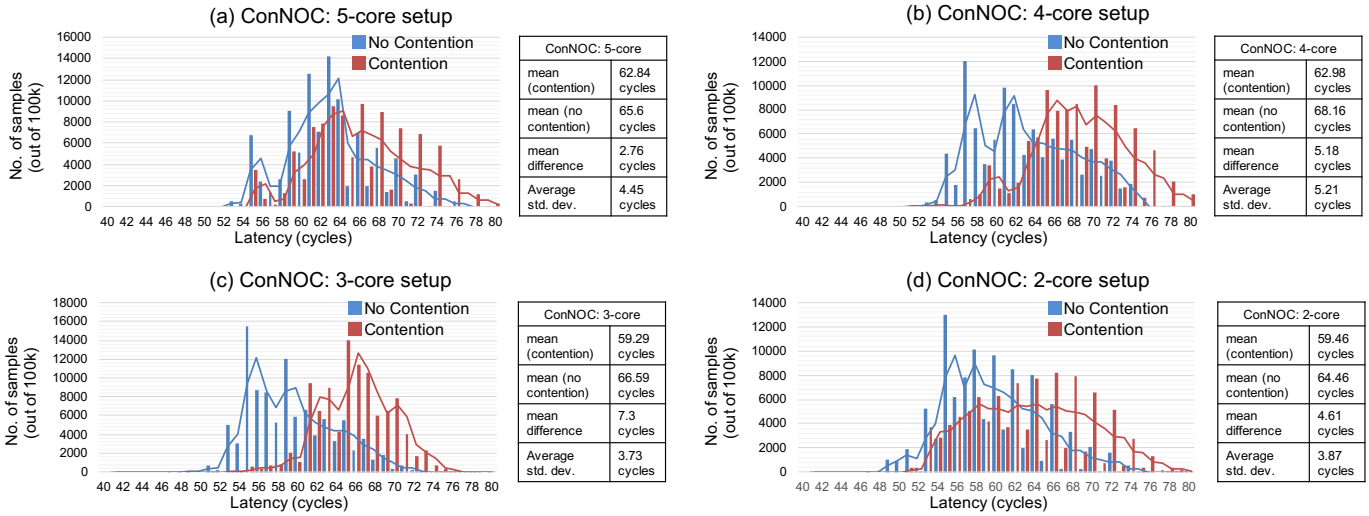

Fig. 6. The latency histogram distributions for 5-core, 4-core, 3-core, and 2-core ConNOC setups under no-contention and contention scenarios.

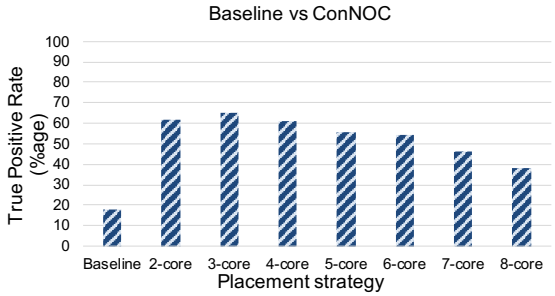

Fig. 7. True positive rate comparisons for 4-core baseline placement versus ConNOC setups without replay.

6-(d), the mean latency difference drops to 4.61 with a standard deviation of 3.87 cycles. Consequently, this setup does not perform as well as the 3-core setup, and its true positive rate drops back to ∼62%. Since the 3-core ConNOC setup performs best under no replay, it is used as the default setup.

B. Evaluation with replay capability

We observe that all setups are inadequate for practical attacks without the replay capability. Replay allows an adversary to repeat each operation of a given application, thus enlarging the aggregate latency difference between no-contention and contention scenarios. The evaluation is performed using n replay rates, where n ranges from 2 to 100. Each replay consists of reading a data variable n times from the core hosting the L2 cache for the data, and measurement of aggregated latency is recorded. For example, if $n = 2$, a word is read twice. This operation is repeated for both applications X and Y with contention, and for application X in the case of no contention.

Figure 8 represents latency histograms for various replay rates for the 3-core ConNOC setup. As expected, Figure 8-No-Replay shows a high overlap between latencies for contention and no-contention scenarios that justifies the ∼65% true positive rate for this setup. However, with replay the distributions start to shift away from each other. At 2 replays, considerable overlap persists, while at 5 and higher number of replays, no overlap is observed. Therefore, ConNOC uses 5 replays as its default since it delivers ∼100% true positive rate.

Figure 9 summarizes the true positive rates as a function of replay capability for the baseline and various ConNOC setups. The baseline shows a true positive rate of ∼100% at 55 replays, which translates to a throughput rate of ∼180 *kbps*. The 3-core ConNOC setup delivers 100% accuracy at just 5 replays that translates to a throughput rate of 2 *mbps*. This is an 11× improvement compared to the baseline setup. The 2-core and 4-core ConNOC setups are able to achieve 100% accuracy at 6 and 7 replays respectively. The number of required replays is observed to increase with 5-core and higher ConNOC setups. However, they all outperform the baseline setup.

C. Covert-communication attack demonstration

This section evaluates covert communication attack using 3-core ConNOC setup with 5 replays, and a

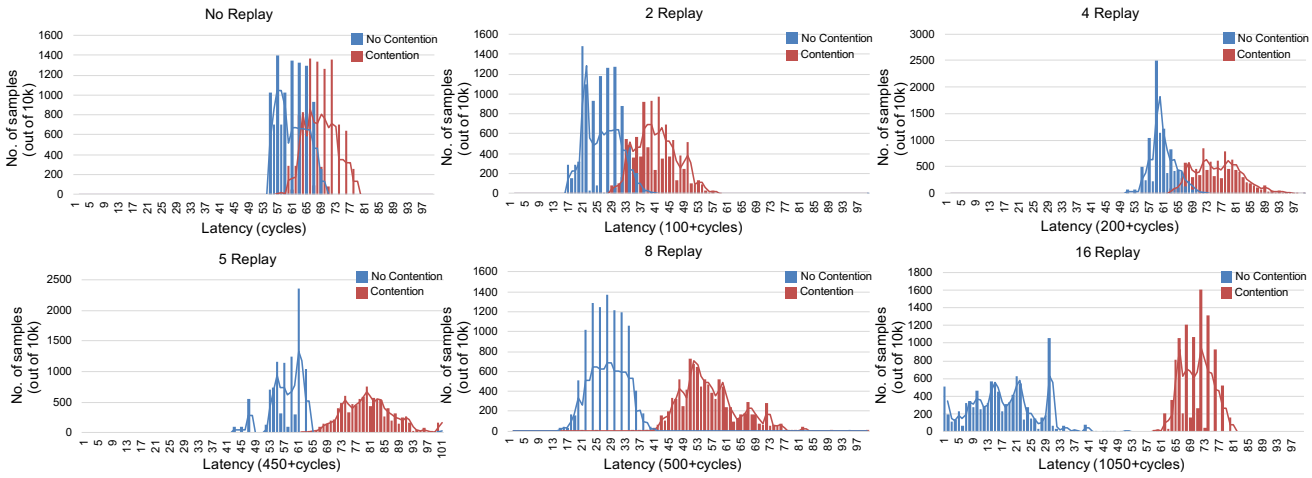

Fig. 8. Aggregate effect of no replay, 2, 4, 5, 8, and 16 replays in 3-core ConNOC setup.

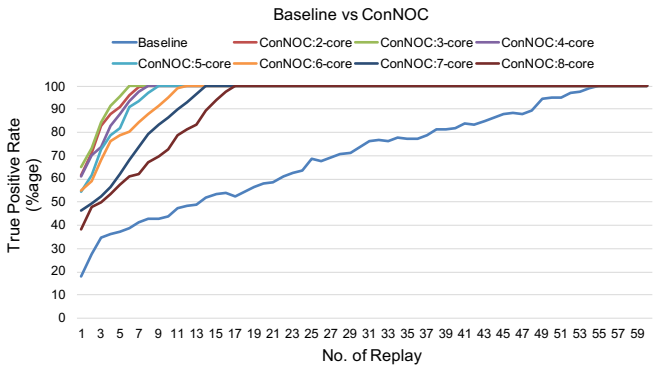

Fig. 9. True positive rate comparisons for 4-core baseline setup versus ConNOC setups.

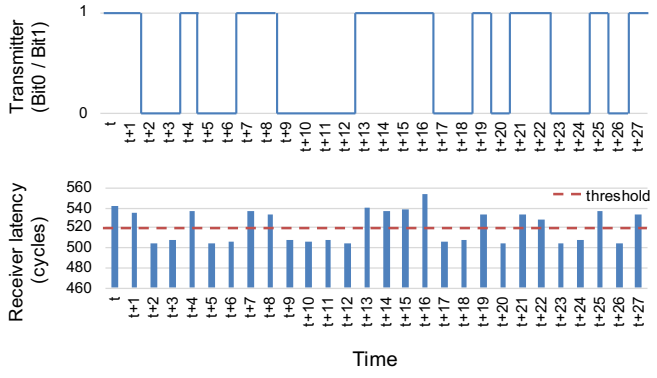

Fig. 10. A covert-communication of 28 bits of data using 3-core ConNOC setup, 5 replays, and the threshold of 522 cycles.

threshold of 522 cycles. In this demonstration, the objective of the transmitter application is to transmit 28 bits to the receiver application covertly. For a span of t cycles of time, the transmitter application creates contention on NoC hardware to transmit bit 1. To transmit bit 0, the adversary stays idle for t cycles. Figure 10 shows that a bit 1 is transmitted at $t = 0$, where receiver observes a latency of 540 cycles. Similarly, at $t+2$, the transmitter application stays idle. Thus, the receiver application measures the latency of 504 cycles. Using the threshold value of 522 cycles, the receiver application infers all secret bits with 100% accuracy. This covert-communication channel attack achieves a peak throughput of 2 *mbps*.

D. *Information leakage attack demonstration*

This section evaluates information leakage attack using 3-core ConNOC setup with 5 replays, and a threshold of 522 cycles. This attack is modeled after the RSA crypto-system square and multiple algorithm to demonstrate the use case of ConNOC. An adversarial application X monitors contention level at shared NoC hardware. The secure RSA-like application accesses two functions, function1 and function2 based on an 18-bit secret key. The function1 causes high contention, while the function2 causes low contention at the NoC hardware. The sequencing of the two functions reveal secret bit 0 or 1. For example, if the secure application transmits secret bit 0, it only calls function1. Whereas, for secret bit 1, it performs function1 followed by function2. In the case of secret bit 1, a high contention followed by low contention is observed by the receiver application.

Figure 11 presents a demonstration of attack on a time scale. An adversarial application infers secret key value based on the pattern of contention observations.

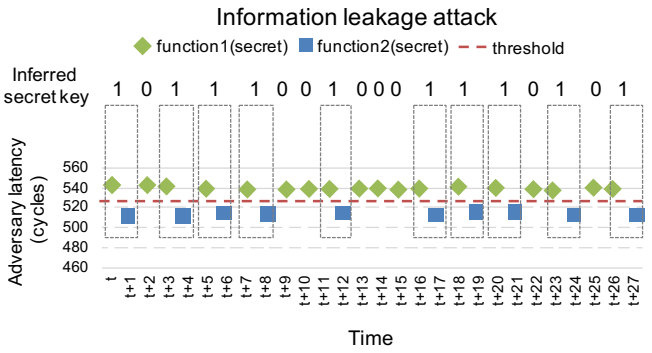

Fig. 11. Information leakage of 18 secret bits using 3-core ConNOC setup, 5 replays, and the threshold of 522 cycles.

A high contention at $t = 0$, followed by low contention at $t + 1$ indicates that function1 is accessed followed by function2. The receiver classifies this as secret bit 1. In contrast, a high contention level at $t+2$ followed by another high contention at $t + 3$ is classified as secret bit 0. This pattern is repeated to retrieve all 18 bits of secret with 100% accuracy at 2 *mbps* speed.

VII. Discussion on mitigation schemes

Prior works almost exclusively rely on isolation-based mitigation schemes based on spatial or temporal partitioning of NoC hardware to remove interference. For example, for temporal partitioning, the NoC operates between applications (or domains) in time slices. Thus, a packet waits at each hop until the NoC allows forwarding packets from its domain. Both spatial and temporal partitioning schemes are effective at mitigating NoC interference. However, they suffer significant performance overheads because hardware resource allocation is not adjusted dynamically to match the demands of each domain.

Suh et al. [6] proposed a one-way leakage protection-based scheme for NoC, called reversed priority with static limits (RPSL). RPSL assigns a HIGH priority to non-secure applications with static limits on resources, and LOW priority to secure applications. This scheme prevents non-secure applications from observing the contention at NoC hardware due to always HIGH priority. The main drawback of this approach is that it depends on assigning priorities to the co-located applications, which may not be possible in practice. Another mitigation strategy that builds on temporal partitioning idea relies on two-way protection using surf scheduling paradigm [10]. This scheme is shown to improve performance over the baseline time-division multiplexing approach. Recent secure processor architectures propose a spatiotemporal approach to protect against timing-based attacks [2]. Although existing mitigation schemes are sufficient to provide security guarantee for ConNOC, their performance impact must be evaluated.

Other obfuscation-based mitigation schemes exist for persistent channels, like [3]. However there is no existing secure obfuscation-based scheme for NoC hardware attacks. We plan to explore this mitigation approach for ConNOC as our future work.

VIII. Conclusion

This paper characterizes timing-based sSCA on a real Tilera *TileGx72* multicore processor that exploits NoC hardware, a non-persistent, low timing variation and high noise side-channel. We implement a state-of-the-art 4-core baseline attack that places adversarial application across distributed cores, and shows that this placement has ∼18% true positive rate which is inadequate for a realistic attack. A novel code and data placement strategy called ConNOC is proposed, which better exploits NoC hardware for timing-based sSCA and shows true positive rate for ∼65%. A replay-based heuristic improves ConNOC accuracy to ∼100% with only five replays, whereas the baseline setup requires 55 replays for high accuracy. This translates to a 2 *mbps* throughput, whereas the baseline setup is limited to 180 *kbps* throughput. ConNOC shows 11× improvement on a real processor when compared to baseline. We also demonstrate covert communication and information leakage attacks with 100% accuracy based on the ConNOC strategy.

IX. Acknowledgments

This research was supported by the National Science Foundation under Grants No. CNS-1929261 and CNS-1916756.

References

[1] S. Röttger and A. Janc, "A spectre proof-of-concept for a spectre-proof web," Mar 2021.

[2] H. Omar and O. Khan, "Ironhide: A secure multicore that efficiently mitigates microarchitecture state attacks for interactive applications," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 111–122, Feb 2020.

[3] M. K. Qureshi, "Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 775–787, 2018.

[4] F. Liu and R. B. Lee, "Random fill cache architecture," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 203–215, 2014.

[5] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, 2019.

[6] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pp. 142–151, 2012.

[7] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*, (San Diego, CA), pp. 719–732, USENIX Association, Aug. 2014.

[8] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of aes," in *Topics in Cryptology – CT-RSA 2006* (D. Pointcheval, ed.), (Berlin, Heidelberg), pp. 1–20, Springer Berlin Heidelberg, 2006.

[9] D. Skarlatos, M. Yan, B. Gopireddy, R. Sprabery, J. Torrellas, and C. W. Fletcher, "Microscope: Enabling microarchitectural replay attacks," *IEEE Micro*, vol. 40, no. 3, pp. 91–98, 2020.

[10] H. Wassel, Y. Gao, J. Oberg, T. Huffmire, R. Kastner, F. Chong, and T. Sherwood, "Surfnoc: a low latency and provably non-interfering approach to secure networks-on-chip," in *ISCA*, 2013.

[11] J. Szefer, "Survey of Microarchitectural Side and Covert Channels, Attacks, and Defenses," *Journal of Hardware and Systems Security*, vol. 3, no. 3, pp. 219–234, 2019.

[12] D. J. Bernstein, "Cache-timing attacks on aes," 2005.

[13] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 955–972, USENIX Association, Aug. 2018.

[14] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Jump over aslr: Attacking branch predictors to bypass aslr," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–13, 2016.

[15] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside sgx enclaves with branch shadowing," SEC'17, (USA), p. 557–574, USENIX Association, 2017.

[16] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for cross-cpu attacks," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 565–581, USENIX Association, Aug. 2016.

[17] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 973–990, USENIX Association, Aug. 2018.

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