Characterization of Timing-based Software Side-channel Attacks and Mitigations on Network-on-Chip Hardware

USMAN ALI, SHEIKH ABDUL RASHEED SAHNI, and OMER KHAN, University of Connecticut, United States

Modern network-on-chip (NoC) hardware is an emerging target for side-channel security attacks. A recent work implemented and characterized timing-based software side-channel attacks that target NoC hardware on a real multicore machine. This article studies the impact of system noise on prior attack setups and shows that high noise is sufficient to defeat the attacker. We propose an information theory-based attack setup that uses repetition codes and differential signaling techniques to de-noise the unwanted noise from the NoC channel to successfully implement a practical covert-communication attack on a real multicore machine. The evaluation demonstrates an attack efficacy of 97%, 88%, and 78% under low, medium, and high external noise, respectively. Our attack characterization reveals that noise-based mitigation schemes are inadequate to prevent practical covert communication, and thus isolation-based mitigation schemes must be considered to ensure strong security. Isolation-based schemes are shown to mitigate timing-based side-channel attacks. However, their impact on the performance of real-world security critical workloads is not well understood in the literature. This article evaluates the performance implications of state-of-the-art spatial and temporal isolation schemes. The performance impact is shown to range from 2–3% for a set of graph and machine learning workloads, thus making isolation-based mitigations practical.

CCS Concepts: • **Security and privacy → Side-channel analysis and countermeasures;**

Additional Key Words and Phrases: Secure network-on-chip, side-channel attack, hardware security

ACM Reference format:
Usman Ali, Sheikh Abdul Rasheed Sahni, and Omer Khan. 2023. Characterization of Timing-based Software Side-channel Attacks and Mitigations on Network-on-Chip Hardware. *ACM J. Emerg. Technol. Comput. Syst.* 19, 3, Article 21 (June 2023), 23 pages.
https://doi.org/10.1145/3585519

1 INTRODUCTION

Homogeneous many-core processors and heterogeneous system-on-chip-based systems are widely deployed in the real world due to efficient utilization of shared hardware resources (i.e., caches, speculation units, interconnect hardware). Although resource sharing brings substantial performance benefits, adversaries exploit such resource sharing for timing-based **side-channel attacks (SCA)**. Google [20] released a proof-of-concept for timing attacks that target execution

This research was supported by the National Science Foundation under Grant No. CNS-1929261. This research was also supported in part by the Semiconductor Research Corporation (SRC).
Authors' address: U. Ali, S. A. R. Sahni, and O. Khan, University of Connecticut, 371 Fairfield Way, Storrs, Connecticut 06269; emails: {usman.ali, abdul.rasheed, omer.khan}@uconn.edu.

1550-4832/2023/06-ART21 $15.00
https://doi.org/10.1145/3585519

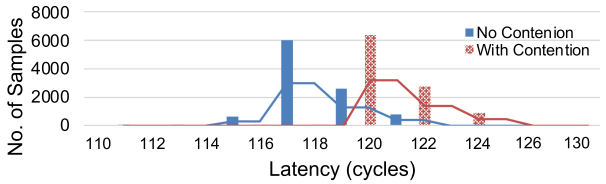

Fig. 1. (a) Timing variations to access shared NoC hardware under contention and no contention situations.

units and leaks data from chrome browser. Timing-based channels on shared hardware resources are classified as persistent and non-persistent channels. In persistent channels (i.e., caches and main memories), the hardware stores code or data until evicted (i.e., for a longer time), whereas in non-persistent channels (i.e., execution units and network-on-chip), the shared hardware does not store code or data for a long time period; rather, it aides with the movement of code or data. The persistent channels have proved to be a reliable source (i.e., high timing variations and less prone to noise) for timing-based SCA [13, 19, 27], and thus they have been the focus of interest for security researchers. Contrary, the non-persistent channels are noisy, less explored, and not easy to attack.

Prior work [24] indicated the potential of timing-based SCA on the non-persistent **network-on-chip (NoC)** hardware and proposed an attack setup (baseline). The setup consists of two independent applications (i.e., a transmitter and receiver) with spatially distributed code and data across four cores in a multicore machine. Although these applications are fully isolated due to process isolation, they share the underlying NoC hardware. The transmitter application occupies shared NoC resources by accessing data and causes latency variations for all other applications receiving data over the NoC hardware, including the receiver application. Figure 1 shows the timing histogram of receiver application data accesses with NoC hardware under contention and no contention. These latency variations are exploited for timing-based side-channel attacks. A recent work, ConNOC [8] implements the baseline attack setup on a real machine and shows that timing variations are inadequate for practical SCA attacks. It proposes a novel code and data placement method to improve the accuracy of the attack using timing variations in the NoC hardware. It quantifies the attack to deliver ∼100% accuracy at a much higher speed as compared to the baseline setup. Moreover, it demonstrates practical covert-communication and information leakage attacks on a real multicore machine. Although ConNOC claims to show the practicality of attack, it does not consider realistic situations such as the impact of system noise on the attack efficacy. For example, will an SCA attack work in the presence of external system noise? Can noise reduction approaches from information theory, such as differential signaling, de-noise unwanted interference in the attack?

This article evaluates the baseline and ConNOC attack setups that target NoC hardware in the presence of random external noise and characterizes the effect of noise on the efficacy of attack. Three applications are utilized to generate low, medium, and high system-level noise in the multicore machine. The evaluation shows that baseline attack efficacy drops to 88%, 78%, and 55% for the low, medium, and high noise scenarios. We propose to utilize information theory concept of differential signaling to de-noise the attack. Differential signaling requires transmission of a reference signal alongside the original signal; thus, if external noise affects both signals, then a difference between the original and reference signal reveals original information. Our evaluation of the information theory attack shows that the efficacy of SCA attack improves to 97%, 88%, and 78% under low, medium, and high noise, respectively. A 78% efficacy of attack is sufficient to leak critical information, specifically for covert-communication. Therefore, we implement and characterize a practical attack to demonstrate covert leakage of a 240 × 180 pixel image over the NoC covert-communication channel in the presence of high system noise in our multicore machine setup.

Literature shows that SCA attacks can be mitigated using obfuscation- [14, 15, 28] and isolation- [18, 24, 25] based approaches. Both schemes have security and performance implications. Obfuscation schemes mitigate timing-based SCA by making constant time for contention and no-contention cases. One approach is to introduce random noise on NoC hardware to obfuscate timing variations. For example, Reference [28] generates dummy traffic to corrupt timing variations (i.e., constant time for contention and no contention cases) to mitigate the timing-based SCA. Although noise injection schemes are claimed to obfuscate timing variations, they are not evaluated in terms of timing-based SCA on NoC hardware. For example, how much noise is required to mitigate attack that targets NoC hardware effectively? Further, are de-noising schemes such as repetition codes and differential signaling from information theory sufficient to overcome obfuscation mitigation schemes? Our information theory attack answers these questions and concludes that random noise can be de-noised, thus making the obfuscation-based mitigation schemes inadequate for practical purposes. To successfully mitigate timing attacks on NoC hardware, isolation-based schemes must be considered.

Isolation schemes [18, 24, 25] distribute shared resources spatially or temporally to provide strong security guarantees by removing interference that causes timing variations. However, these schemes incur performance implications. For example, spatial isolation distributes shared hardware into secure and non-secure domains and limits the efficient usage of hardware resources. However, the temporal isolation schemes allow efficient utilization of resources by allocating shared hardware to each application for small time quanta. The temporal isolation approach allows full utilization of hardware resources for a unique application, but co-located applications need to wait. Prior work [24, 25] characterizes temporal isolation schemes using micro-benchmarks and synthetic traces, but performance implication study on the real-world applications is missing in the literature. Further, a comparison study of performance implications of spatial and temporal mitigation schemes is missing in the literature.

Isolation schemes, such as Ironhide [18], TDM [24], and SurfNOC [25] require hardware support that is not available on commercial off-the-shelf machines. Therefore, to study their performance impact on real workloads, a detailed application-level simulator is needed. In this article, we implement three state-of-the-art isolation-based NoC mitigation schemes on a RISC-V multicore simulator [4, 12, 17]. We extend hardware performance models in the simulator to study the performance implications of the mitigation schemes for a set of graph and machine learning workloads executing real-world inputs. First, we demonstrate that the simulator has enough fidelity for timing-based attacks that exploit the underlying shared NoC hardware. Second, we show a detailed performance evaluation of the mitigation schemes and conclude performance degradation of 2.29%, 12.5%, and 2.97% for Ironhide, TDM, and SurfNOC, respectively. Ironhide spatially partitions the available cores into hardware isolated clusters of cores. The NoC traffic is not allowed to cross cluster boundaries, thus ensuring isolation. The hardware support needed for Ironhide is at the core-level cluster formation. This technique works best when workload resource demands are matched with the spatial partitioning of core-level resources. However, SurfNOC temporally isolates NoC traffic from different applications with properties that minimize stalls in the NoC. This technique is shown to be as effective as Ironhide, but it comes at the cost of hardware support in the NoC router microarchitecture. Both Ironhide and SurfNOC approaches are practical to protect against timing-based software side-channel attacks on NoC hardware.

Overall, this article makes the following contributions:

(1) Shows that system noise significantly reduces the efficacy of state-of-the-art baseline and ConNOC attack setups that target NoC hardware for a timing-based side-channel attack on real hardware.

(2) Propose an information theory-based analysis of the attack that uses repetition codes and differential signaling to de-noise the attack setup and demonstrate a practical covert-communication attack with high accuracy on real hardware.
(3) It shows that de-noising methods make obfuscation-based mitigation schemes inadequate. Therefore, it evaluates isolation-based mitigation schemes using real graph and machine learning workloads and quantifies their performance implications using an application-level RISC-V multicore simulator.

2 BACKGROUND AND MOTIVATION

2.1 Timing-based SCA on NoC

Timing-based SCA exploits timing variations in performing same operation under different conditions. For example, a data read from a remote core takes varying time based on the availability of shared hardware resources. NoC hardware consists of wires, buffers and crossbars that are shared among all available cores. Prior work [24] proposed an attack setup that consists of code and data of two applications spatially distributed over four cores in multicore configuration. These applications are virtually isolated but share underlying non-persistent NoC hardware. ConNOC [8] implemented a novel placement strategy that better exploits the underlying NoC hardware for timing variations. An adversary can carefully control contention on NoC hardware to create a timing-based SCA, which is used for practical covert communication and information leakage attacks.

2.2 Information Theory and Noise

Noisy communication channels (i.e., wireless communication) are common problem in information theory, and various techniques are used to improve signal quality and de-noise signal at bit level, word level, and packet level. This work focus on information theory concepts of repetition codes [21, 23] and differential signaling [16] for bit-level timing-attack.

2.2.1 Repetition Codes. The repetition codes require a re-transmission of the original signal multiple times. This approach introduces redundancy of transmittable information and increases the probability of the signal reaching its destination. For example, a repetition code of three will convert a bit-level information 101 into bit-level information 111 000 111 (i.e., each bit is transmitted three times). This increases the probability of contention on NoC hardware.

2.2.2 Differential Signaling. The differential signaling technique de-noises the original signal from unwanted noise. The differential signaling technique requires a simultaneous transmission of two signals for each bit of information, an original signal and a reference signal (i.e., bit 0 or low signal). Later, a difference between the original signal and the reference signal is used to infer original information. For a bit 1, the differential signaling approach requires transmission of "10," and a positive difference indicates original bit 1. For bit 0, the "00" is transmitted, and no difference indicates the original bit 0. This approach makes the attack resilient to external noise.

2.3 Mitigation Schemes

Timing-based SCA is possible due to controlled interference at underlying shared hardware resulting in timing variations. Such attacks can be mitigated by either hiding timing variations or prohibiting adversaries from making controlled interference on shared hardware. Mitigation schemes are classified into obfuscation and isolation schemes.

2.3.1 Obfuscation Schemes. Obfuscations schemes [14, 15, 28] introduce random noise to hide the timing variations by attempting to make constant timing for all operations. For example, an

obfuscation-based mitigation scheme can create contention on shared NoC hardware with the help of external noise to deceive adversaries into inferring bit 1, where it was bit 0.

2.3.2 Isolation Schemes. Isolation schemes prevent adversaries from creating contentions by isolating shared hardware spatially or temporally. Spatial isolation [18] allocates dedicated hardware to adversary applications and secure applications. However, temporal schemes [24, 25] alternatively allocate hardware to each application for a small interval of time.

2.4 Limitations of Prior Work and Goals

State-of-the-art work on timing-based SCA on NoC hardware implements baseline and ConNOC attack setups, but a study on the impact of noise targeting NoC attacks is missing in the literature. Further, the literature lacks answers to the following questions: (1) Can unwanted noise reduce the efficacy of the SCA attack?, (2) Can we use information theory concepts to improve the attack in the presence of noise?, (3) Can we develop a real attack using the NoC channel? Will noise-based mitigation schemes work for SCA that targets NoC hardware?, and Can we defeat noise-based mitigation schemes? Contrary to attacks and noise-based mitigations, there are practical mitigation schemes based on isolation for timing attacks, but evaluation of these schemes focuses on security aspects and performance of synthetic traffic. However, performance implication studies and comparison of these mitigations on real-world applications are missing in the literature.

This work aims to answer missing literature questions and comprehensively characterize timing-based attacks on NoC hardware under realistic conditions. This work also evaluates performance characterizations of three isolation-based mitigation schemes using six real-world security-critical applications.

2.5 Threat Model

The work adopts threat model from timing-based SCAs on shared hardware resources [8, 24, 27]. The threat model considers that two applications (i.e., transmitter and receiver) are unauthorized to communicate with each other. Both applications have user-level privileges and control placement of their data on different cores. Modern multicore processors enables user-level applications to pin code and data to certain cores for higher performance gains and benefit from parallelism. For example, the numactl command pins application code on certain cores and allocate memory controllers in a real Tilera TileGx-72 multicore processor running the Linux operating system *numactl* command [2] is used pin application code to certain cores and allocate memory controllers. *Numactl* is user-privileged command and ensure the pinning of application to certain cores and prevent operating system to re-allocate the application to random cores. Application data are pinned to certain tiles using the Tilera **Tile Multicore Components (TMC)** Library [10]. Further, the threat model assumes that all applications have access to a cycle accurate timer to measure timings of operations. The threat model focus on software-based SCAs and considers physical SCAs such as thermal, power, sound, and EM outside of the threat model. Consequently, it focuses on software-introduced (i.e., parallel running applications to SCA attack) noise and ignores physical noise such as thermal or coupling noise outside of the threat model. It also considers performance degradation attacks such as denial-of-service beyond the scope of this work.

3 TIMING ATTACKS ON NOC HARDWARE

The timing-based SCA targets shared resources in NoC hardware, including shared buffers, wires, and crossbars. For example, if an application running on core 0 requests data read from a remote core 3, then the data will initially move from the remote core's cache to NoC hardware, where it stays in buffers until the crossbar is available. The data are multiplex over wires to utilize resources efficiently. An adversary application may simultaneously occupy the buffers, wires, and

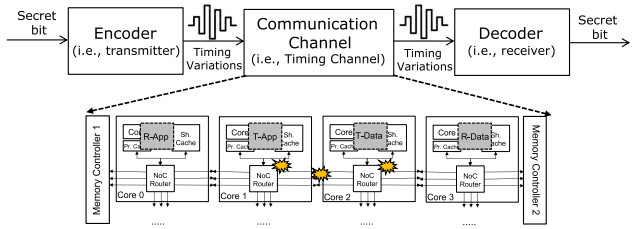

Fig. 2. Three components of timing-based attack on NoC hardware: (1) encoder, (2) timing-based communication channel, and (3) decoder.

crossbars and force other application's data to wait. This leads to timing variations in data access when the NoC hardware resources are contended or not. An SCA attack exploits these timing variation to conduct covert communication or information-leakage attacks. The efficacy of the attack depends upon (1) the maximum time difference that an adversary application can create between contention and no contention, (2) synchronization between creation of contention and detection of contention, and (3) unwanted resource occupation from non-malicious applications sharing NoC hardware. Prior work characterized maximum timing variations and their amplification using repetition codes, but the efficacy of attack NoC hardware in the presence of system-level noise is missing. This section explains components of the attack setup and the proposed information theory attack to de-noise unwanted interference that affects the attack efficacy.

3.1 Timing-based SCA Components

The attack setup consists of two applications, a transmitter application and a receiver application with code and data spatially distributed across cores such that they share underlaying NoC hardware. Figure 2 shows a layout of attack components with (1) a timing-based communication channel, (2) an encoder, and (3) a decoder. A timing-based communication channel [8, 24] is underlaying shared NoC hardware that causes timing variations for data movement. The encoder is implemented in the transmitter application, which creates a contention pattern on the timing channel. Contrary, the decoder is implemented in the receiver application and decodes the timing-variation pattern on the NoC hardware timing channel. The success of the attack also depends on synchronization between encoder and decoder applications. The timing-based SCA are synchronized using cycle-accurate global cycle counters or shared global flags. Both applications initiate a transmission and receive at pre-agreed time ticks in a fully synchronized attack. For example, at every t cycle, the transmitter application invokes an encoder to generate contentions or vice versa, and the receiver application invokes a decoder to monitor contention situation on NoC hardware.

3.1.1 Timing-based Communication Channel. A timing-variations-based side-channel on NoC hardware is a channel for communication between two applications. Figure 2 shows a transmitter application T hosted on core 1 and the transmitter data T_d pinned on core 2. A communication between T and T_d causes contention on NoC hardware between core 1 and core 2. Similarly, the receiver application R is hosted on core 0 and receiver data R_d pinned on core 3. A contention on core 1 and 2 NoC hardware will cause a delay in the latency of communication between R and R_d, and application R will observe timing variations based on contention level. This placement of code and data is used to create a timing-based SCA. For example, to transmit a bit 1, an application creates a high contention on the NoC hardware channel, whereas to transmit a bit 0, no contention is created on NoC hardware. Although this is a low timing variation and a high noise channel, an encoded stream of information with information theory concepts overcome such challenges. The

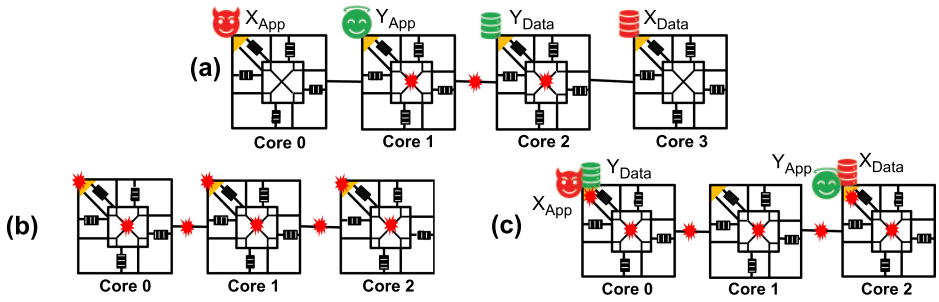

Fig. 3. Layout for timing-based SCA on NoC hardware. (a) Baseline attack setup and contention points. (b) Point of contention in NoC hardware. (2) ConNOC placement of code and data with contention points.

baseline attack proposes a timing-based communication channel, which later ConNOC improves by means of better code and data placement strategies.

3.2 Baseline and ConNOC Attack Setup

The baseline attack setup consists of victim (i.e., transmitter) and adversarial (i.e., receiver) applications where code and data are spatially distributed across four cores. Figure 3(a) shows the baseline attack setup where transmitter application is hosted on core 0, and its data are hosted on core 3, whereas receiver application is pinned on core 1, and its data are hosted on core 2. The baseline attack partially exploits contention points (i.e., wires, buffers, and crossbars) on core 1 and core 2. 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. Figure 3(b) shows a possible point of contention. Moreover, the success of the attack depends on the probability of aligning contented data accesses on one or more of these sources. A ConNOC data placement scheme improves the number of contention points, as well as the probability of success to activate them at runtime, and thus requires fewer repetition codes and results in high-speed attack. 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 is small, the number of potential contention points are constrained while the probability of activating them is high. However, 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. 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 64-bits flit size, then the maximum number of cores occupied by a single path is eight cores. In this scenario, ConNOC utilizes eight cores where the code and data placement are done on the extreme points of the interconnected cores. However, ConNOC placement works efficient in three-core configurations. Figure 3(c) shows a three-core placement of code and data to maximize attack efficacy.

3.3 Impact of System Noise

The efficacy of baseline and ConNOC attack setup depends on the successful creation of contention and no-contention situations on NoC hardware by transmitter and receiver applications. For example, the transmitter application may create a no-contention situation in NoC hardware for t time, whereas the receiver measures its data access latency to detect the no-contention scenario.

Without system noise, this results in a data access latency for the receiver that falls within the prescribed range. Contrary, in the presence of system noise the receiver data access latency is perturbed. The receiver may detect this situation as a contention case, while the transmitter intended a no-contention scenario. This unwanted interference drastically reduces attack efficacy and requires additional methods to make the attack more resilient to system noise.

3.4 Information Theory Attack Setup

The baseline attack setup and ConNOC are studied without presence of noise, where noise drastically reduces the efficacy of attacks and makes them inadequate for practical attacks that target noisy and non-persistent timing-based SCA on NoC hardware. This section improves the baseline and ConNOC attack setups with information theory concepts of repetition codes and differential signaling to overcome the challenges of unwanted external noise. Although information theory improves the efficacy of an attack, it reduces the attack speed. A practical attack requires a full synchronization of two applications. Otherwise, attack efficacy reduces drastically. A practical covert-communication attack is implemented to show the efficacy of the information theory attack in presence of noise.

3.4.1 Repetition Codes. The repetition codes is simple information theory [22] approach to improve the quality of signal over noisy channel. The repetition codes require a re-transmission of the original signal r times. This approach introduces redundancy of transmittable information and increases the probability of the signal reaching its destination. For example, a repetition code of three will convert a bit-level information 101 into bit-level information 111 000 111 (i.e., each bit is transmitted three times). However, the receiver has 3-times-higher probability of receiving the correct information. In our attack, a repetition is implemented by occupying the NoC hardware resources r times. This increases the probability of creation of contention for transmitter application on NoC hardware and consequently detection of contention by receiver application.

3.4.2 Differential Signaling. The differential signaling technique is used in electrical circuits [16] to remove unwanted electrical interference. The differential signaling technique requires a simultaneous transmission of two signals for every bit of information—an original signal (could be bit 0 or bit 1) and a reference signal (i.e., bit 0 or low signal). The receiver will collect both signals. Later, a difference between the original signal and the reference signal is used to infer original information. The differential signaling works on the assumption that noise will effects both signals equally, while a difference will remain constant, i.e., a positive difference for bit 1 and no difference for bit 0. For example, in bit 1, the differential signaling approach requires transmission of "10," and a positive difference indicates original bit 1. For bit 0, the "00" is transmitted, and no difference indicates the original bit 0. This approach makes the attack resilient to external noise.

3.4.3 Information Theory Attack. The encoder in transmitter application consists of repetition codes and differential signaling components. Figure 4 shows layout for information theory attack setup. In the first step, the secret bit is encoded using repetition code and differential signaling. In step 2, a contention and no contention pattern is generated on timing-based communication channel based on encoded information. This attack uses 4-core baseline attack setup as a communication channel. In step 3, the receiver application detects contention and no-contention patterns and decodes differential signaling and repetition codes to retrieve the secret bit. Figure 5 shows a pseudo code of transmitter and receiver applications for information theory attack. The next sections explain encoder and decoder implementation of repetition codes and differential signaling.

3.4.4 Encoder. The encoder includes a bit-level repetition code to overcome low timing variations. Repetition codes introduce redundancy in transmittable information and amplify the effect

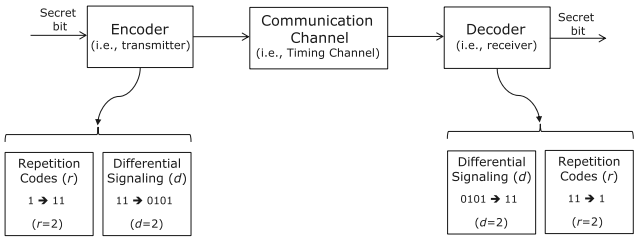

Fig. 4. Baseline attack with repetition codes and differential signaling.

```
// transmitter pseudo code

set_sync_bit();

while(sync == 1) {
   if (bit==1){
      // generate contention at NOC
      for(loop < replay_rate){
         read_data();
      }
      for(loop < replay_rate){
         //do nothing;
      }

   }else if(bit==0){
      for(loop < replay_rate){
         //do nothing;
      }
      for(loop < replay_rate){
         //do nothing;
      }
   }
}
```

```
// receiver pseudo code

while(sync == 1) {
   t_start = timer();
      for(loop < replay_rate){
         read_data()
      }
   t_diff0 = timer() - t_start;

   t_start = timer();
      for(loop < replay_rate){
         read_data()
      }
   t_diff1 = timer() - t_start;

   if (t_diff1 - t_diff0 > threshold){
      secret_bit = 1;
   }else{// infer secret = 0
      secret_bit = 0;
   }
}
```

Fig. 5. Pseudo code for information theory attack.

of timing variations. This amplification causes a larger timing difference between contention and no contention cases. For example, an r times repetition of a bit 1 will generates a stream of r times bit 1, which results in a d times increase in timing variations. Further, this encoder includes a differential signaling component to overcome the challenge of external system noise. Differential signaling is a technique used in information theory to de-noise the signal. The differential signaling encodes a single bit into a pair of bits. For example, a bit 1 is encoded into a low and high pattern (i.e., 01), and bit 0 is encoded into a low and low pattern (i.e., 00). Later a latency difference of two signals is used to infer transmitted bit. The external noise affects both bits of the encoder, but the difference between the two signals remains constant—for example, a positive difference for bit 1 and zero for bit 0.

3.4.5 Decoder. The decoder monitors latency values to infer contention levels at NOC hardware and decodes components for differential signaling and repetition codes. If latency is greater than a set threshold, then this is inferred as a contention case or vice versa. The decoder repeats this process and collects multiple latency samples, and the differential signaling component decodes these t samples into a stream of repetition codes value. These repetitions are later resolved to a secret bit value. For example, the decoder detects a case of 0101, and the differential signaling component will convert this into 11, which later resolves to secret bit 1 using repetition code decoder.

3.5 Covert-Communication Attack

To measure the efficacy of the information theory attack, we have implemented a covert-communication attack. A covert-communication attack allows communication between otherwise two unauthorized applications. The *transmitter* application has access to confidential data and uses

a timing-based channel to exfiltrate information to a *receiver* application. The transmitter application uses the information theory attack setup to leak secret information. For example, the transmitter application occupies shared NoC hardware for a short time quanta, where the receiver application observes contention on receiver data read within the same time quanta. This contention results in an additional latency, which the receiver application compares against a threshold to infer the transmitted information. If latency is greater than a threshold, then the receiver infers it as a contention case, and if it is equal or less than the threshold, then the receiver application infers it as a no contention case. The decoder converts this pattern into secret information. This process is repeated to leak a large amount of data. This covert-communication attack is synchronized using a cycle-accurate global cycle counter. The fully synchronized attack that uses information theory slows the overall speed of baseline attack, and external noise affects the efficacy of the attack. We demonstrate that a slow and low efficacy attack is sufficient to leak useful information.

4 MITIGATIONS SCHEMES

The controlled contention is an essential requirement to enable timing-based SCA on non-persistent NoC channels. The obfuscation schemes depend on external noise to obfuscate timing variations, and information theory attacks can de-noise the external noise, which makes obfuscation schemes inadequate for practical purposes. Thus, this section focuses on the aforementioned isolation-based schemes, i.e., TDM, SurfNoC, and Ironhide [18, 24, 25]. These isolation-based mitigation schemes have been shown in literature to provide strong security guarantees. However, their performance implications on real applications is missing in the literature, which is characterized in this article.

4.1 Temporal Isolation: TDM

In a TDM-based temporal mitigation scheme [24], all NoC routers are allocated temporally to different domains at once. Figure 6(a) shows temporal scheduling of NoC hardware allocation for two domains. A domain is a specific set of applications that requires isolation for security or other reasons. The domain 0 application traffic will use NoC routers, queues, and links in odd cycles, whereas domain 1 traffic will use NoC hardware in even cycles. For example, Figure 6(a) shows an X-Y routing scheme where a domain 1 packet originating from core 7 in an odd cycle will have to wait for an even cycle to move to core 6. For a one-cycle per-hop NoC hardware, the packet requires a total of eight cycles to reach its destination compared to four cycles without the TDM. This temporal isolation scheme eliminates the chances of contention occurrence due to interference from other domains applications, thus guaranteeing against timing variations. Although TDM protects against timing variations, performance overheads are directly proportional to the number of domains and cores. For example, every packet will have to wait for an additional cycle, multiple to the number of cores and domains to traverse to the target NoC router.

4.2 Temporal Isolation: SurfNOC

SurfNOC [25] aims to improve the performance overhead of TDM and temporally schedules isolation of NoC hardware resources. In SurfNOC, data from different domains flow like surfs. This wavelike packet traversal guarantees non-interference while reducing latency overhead compared to the TDM scheme. Figure 6(b) shows surflike scheduling of two domains using SurfNOC. For example, the domain 0 data flow on black waves, whereas domain 1 data flow on gray waves. For an X-Y routing scheme, the Figure 6(b) shows a domain 0 packet originating at core 7 and having a destination at core 1 observes a one cycle delay at start time while waiting for a black wave. The packet reaches its designation on cycle 5. The SurfNOC takes an additional one cycle during traversal compared to four additional cycles of TDM. Each domain observes a

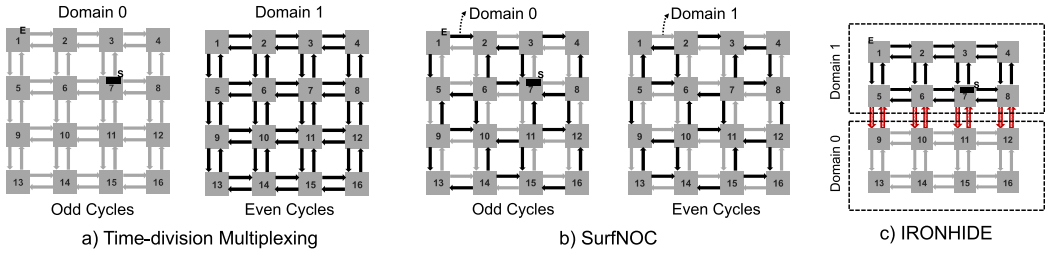

Fig. 6. A sample scheduling of two domains-based mitigations using (a) TDM-based temporal isolation, (b) SurfNOC-based temporal scheduling, and (c) Ironhide, a cluster-based spatial isolation scheme.

delay directly proportional to the number of domains − 1 at start time, and once data ride the wave, there are no delays. The only exception is for a higher number of domains (i.e., greater than 2) where an additional delay could add at the turn while the packet changes the wave (i.e., dimension change). This work focus on two domains. This scheme reduces up to 75% performance implications compared to TDM, where the overhead depends on the number of domains and cores.

4.3 Spatial Isolation: Ironhide

Ironhide [18] dynamically partitions NoC hardware, including routers and links, into two strongly isolated clusters of cores, creating two domains. One cluster is assigned to domain 0 applications, while the other executes domain 1 applications. This spatial isolation prohibits the sharing of NoC hardware and protects against timing leakages. Figure 6(c) shows the architectural structure of Ironhide in sample 4×4 tiles multicore architecture. Based on scheduled applications resource demands, Ironhide performs dynamic allocation of cores to the domain in a multicore setting. The security kernel in Ironhide implements the heuristics to determine optimal resource allocation points by **monitoring misses per-kilo instructions (MPKI)** of scheduled processes. MPKI trend contains a saturation point that indicates resource allocation point beyond which process performance saturates. These saturation points for each process are stored in the security kernel, and during the process scheduling, the cluster size is reconfigured to match these saturation points.

Processes are executed in multiple domains by pinning their threads into cores of respective domains. The shared cache slices are also homed within the same domain, while the off-chip memory is statically distributed among processes of different domains. The memory controllers are also statically distributed among different domains to avoid interference at memory controller queues. The network traffic is routed in such a way that all network packets remain within the cluster boundary. A deterministic X-Y routing protocol is deployed to confine each packet source to the destination path to never violate the cluster boundary. Though Ironhide protects against timing SCA by creating strongly isolated clusters, partitioning resources can affect application performance due to reduced number of available cores. Applications with high core-level parallelism show degraded performance due to partitioning of cores into clusters. However, processes with fewer resource demands do not observe any degradation. Figure 6(c) shows two clusters with each cluster of size 8—a packet traversing from core 7 to core 1 in cluster 1 (i.e., domain 1). The packet will take four cycles to reach its destination, compared to 5 in SurfNOC and 8 in TDM.

5 METHODOLOGY

This section explains methodology for information theory attack and mitigation schemes implementation. The information theory attacks are implemented on a real multicore processors [10]. Contrary, the TDM and SurfNOC mitigation schemes requires hardware modification capabilities that are not available in commercial hardware. Thus, are implemented on a RISC-V simulator.

5.1 Information Theory-based Attacks

The baseline and information theory attacks are characterized on a 72-core Tilera Tile-Gx72 processor [10].

5.1.1 Tile-Gx72. Tile-Gx72 is a tiled architecture with five independent two-dimensional (2D) mesh NOC hardware (iMesh) with X-Y routing. Each tile consists of 32 KB private and 256 KB shared level-2 cache. The cache line size is 64 bytes, whereas the NOC flit size is 8 bytes. This work explicitly targets the cache-coherence network (TDN) of iMesh. The GNU/Linux operating system with kernel version 3.10.55-MDE- 4.3.2.182362 uses the Tilera TMC library to manage network traffic and tile resources.

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

In the no-contention scenario, only receiver executes in the system. However, in the case of the contention scenario, the code for another transmitter application is pinned on core 1, whereas its data are pinned to core 2 using the appropriate TMC library calls. While keeping the measurement data size the same (i.e., a single cache line), the transmitter makes concurrent access from core 1 to core 2, thus creating opportunities for contention in the shared NoC hardware resources.

5.1.3 Noise Generation Applications. Noise is an unwanted perturbation in timing variations of the receiver application due to parallel running applications. Noise can be generated by normal parallel running applications or specialized applications designed for mitigation purposes. This work use *LYNX* [5], *CURL* [1], and *STRESS* [6] for low, medium, and high noise generations. *LYNX* is a command-line-based web browser for Linux that cause low compute and memory activity. Web browsing causes sparse traffic on NOC hardware, which causes low disturbance. *CURL* is command-line for data transfer from internet. A continuous data transfer causes significant noise (medium) on NOC hardware. *STRESS* is a benchmark application to generate noise on various hardware components. We use it to generate a high noise situation involving memory hierarchy and NOC hardware.

5.2 Characterization of Mitigation Schemes

Mitigation schemes are modeled using RISC-V-based large core count simulator.

5.2.1 RISC-V Multicore Simulator. This work use an in-house application class multicore RISC-V simulator [4]. The simulator uses performance models of MIT's Graphite simulator [17] and uses

Architecture Description Language [3] models that describe the functional aspects of RISC-V simulated multicore, e.g., Instructions implementation, register states, MMU, and memory hierarchy models. The simulator is equipped with performance models to capture latency of hop-by-hop NOC hardware [17]. In NoC performance model, each packet movement traces (including NoC latency) are collected at each core level, and overall NoC latency for an application is measured using these traces.

5.2.2 Implementation of Mitigation Schemes. The simulator NoC performance models are modified to include additional latency penalties for TDM and SurfNOC. In TDM implementation, we have doubled the latency of routers for the two domains experiment at every NoC router. Contrary to TDM, the SurfNOC requires additional latency while waiting for surf and changing a surf (i.e., dimension shift). To model SurfNOC performance implications for two domains, we have doubled router latency at first NoC router and later added additional router latency if we detect an X to Y turn. To model Ironhide on the simulator, all available resources are distributed into two domains. Security-critical applications are executed in domain 1 such that they are spatially isolated from other applications of domain 0 running in parallel. The challenge here is to find the optimal resource distribution point between domains. As mentioned in Section 4.3, the security kernel in Ironhide captures the MPKI trend using a representative input.

The security kernel first allocates all the resources to user application and captures the MPKI value. The resource allocations are monotonically decreased to obtain an MPKI trend as a function of core count. The MPKI curve contains the saturation point representing the resource allocation point beyond which the performance scaling saturates, as shown in Figure 15. The domain size for a secure application is configured to match the saturation point to avoid any performance degradation. The secure application threads are pinned to respective cores of domain 1. The L2-homing policy is updated to map local misses to L2 slices within the cluster, whereas L2 misses are routed to memory controllers mapping the respective DRAM regions. The X-Y routing protocol for on-chip network makes sure that network packets of a cluster do not violate cluster boundary.

5.2.3 Timing-variations Fidelity on Multicore Simulator. A Table 1 configuration is used to experiment and collect timing information. Like the attack setup on Tilera, the receiver thread is pinned on core 0 and data on core 3. The transmitter application is pinned on core 1, and transmitter data are pinned on core 2. To pin data on respective cores, we reserved a large array of "char" variables of 1-byte size and shortlisted variables such that they store in respective cores L2 cache memory—first access load data into respective core L2 and application core private caches. The private caches are flushed in the next step, and data are accessed again. Meanwhile, the time to load that data are measured. This process is repeated to take latency measurement under contention and no contention scenarios.

5.3 Evaluation Metrics

This work use **True Positive (TP)** rate [22] and **Discrimination Index (DI)** [16] as timing-variation metrics to evaluate the efficacy and reliability of information theory attacks. The TP rate is based on correct inference of contention situation at underlying NoC hardware-based timing-channel. For example, if the transmitter application creates contention on NoC hardware, and the receiver application successfully infers the contention, then it calculates the TP rate using following equation $TP = \frac{b_1+b_0}{t_1+t_0}$. Here t_1 is total contention samples that transmitter application creates, where t_0 was total samples when the transmitter was idle (i.e., no contention on NoC hardware). The b_1 is the number of samples inferred as contention when the transmitter application creates contention, where b_0 is the number of samples recorded as no contention when there was no contention on NoC hardware.

Table 1. Architectural Parameters for Evaluation

Architectural Parameter	Simulator	TILE-Gx72
Number of Cores	up to 64 @ 1 GHz	72 @ 1 GHz
Compute Pipeline per Core	In-Order, Single-Issue	64-bit VLIW
Memorgy Subsystem		
L1–I Cache per core	32 KB, 2–way Assoc.	32 KB, 2-way Assoc., 2 cycles
L1–D Cache per core	32 KB, 2-way Assoc.	32 KB, 2-way Assoc., 2 cycles
L2 Inclusive Cache per core	256 KB, 4-way Assoc.	256 KB, 4-way Assoc., 10 cycles 2 cycle tag, 4 cycle data
Directory Protocol	Invalidation-based MESI	Invalidation-based
Num. of Mem Cntrl	4	4
DRAM Bandwidth per Cntrl	10 GBps	12 GBps
Electrical 2D Mesh with XY Routing		
Hop Latency	2 cycles (1–router, 1–link)	
Contention Model	Only link contention (Infinite input buffers)	
Flit Width	64 bits	64 bits

Contrary to the TP rate, which gives attack efficacy, the DI is used to measure the reliability of the timing channel. The DI is a statistical tool that quantifies the timing variations distributions under contention and no contention cases. The DI calculations include the statistical mean of timing variations distribution and the variance under contention and no contention scenarios. We have calculated the DI using following equation: $DI = \frac{\mu_1+\mu_0}{\sqrt{\sigma_1^2+\sigma_0^2}}$. Here, the μ_1 is the mean of timing-variation distribution under the contention situation, and μ_0 is the statistical mean of timing-variation distribution under a no-contention case. The variance of timing-variation distributions is represented by σ_1^2 for contention and σ_0^2 for no contention scenarios, respectively.

The implementation of information theory concepts directly affects the speed of attack, and this work uses the traditional communication metric of speed, i.e., kilobits per second (kbps), to measure the maximum speed of attack. The mitigation schemes are evaluated using the completion time metric for security-centric applications.

5.4 Performance Benchmarks

The performance implications of these mitigation schemes have been evaluated using six applications from three different security-critical computing domains. Three graph benchmarks, **Single Source Shortest Path (SSSP)**, PageRank, and Triangle Counting, are executed using a real-world California Road Network graph [7] A mission planning algorithm, **Artificial Bee Colony (ABC)** [26], is also analyzed. Two machine learning workloads AlexNet and Squeeze-Net, are also considered. ImageNet [11] is provided as an input to these benchmarks.

6 EVALUATION

This section evaluates implemented attacks with information theory concepts and uses TP rate, DI, and speed metrics to quantify attack efficacy.

6.1 Attack with Repetition Codes

This section evaluates the baseline and ConNOC attack setup alongside encoder and decoder with repetition codes only and without any external noise. Figure 7 shows a detailed dot plot of baseline

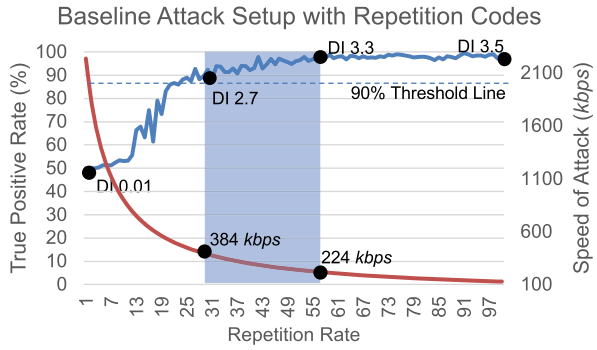

Fig. 7. Baseline attack TP rate and speed comparison with DI values.

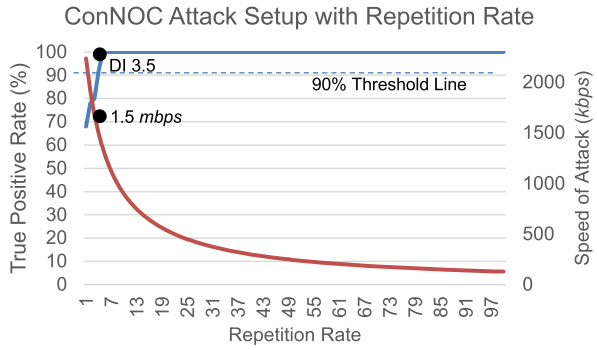

Fig. 8. ConNOC attack TP rate and speed comparison with DI values.

attack for TP rate against repetition codes value from a range of 1 to 100. The experimental results show that without repetition codes, the TP rate is 50%, which is equivalent to randomly guessing the contention and no contention cases. We have set a threshold of 90% TP rate to consider an attack a practical threat. A 30 times repetition of contention and no contention cases is required to achieve a threshold of 90%, with a DI value of 2.7. We observed that the proposed setup with repetition codes only shows 100% TP rate at 55 repetitions with DI of 3.3. Further increase in repetition codes does not improve the TP rate where DI continuously increases at a slow rate. Although TP rate is an important metric, the attack speed largely affects practical attacks. Figure 7 shows that with an increase in repetition rates, the speed of attack decreased attack. The speed is maximum with no replay and reduces to 384 kbps at 30 repetition rate and 224 kbps at 55 repetition rate.

Figure 8 shows the dot plot for ConNOC attack setup with TP rate against repetition codes values. The experiment results shows a 68% TP rate without any repetition codes compared to 50% of baseline attack setup. The repetition code of 4 is required to achieve threshold of 90% where repetition codes of 5 is required to achieve 100% TP rate, that results in a speed of 1.5 mbps. Further increase in repetition codes only decreases the speed of the attack.

6.2 Attack with Differential Signaling

This section evaluates the baseline and ConNOC attack setup alongside encoder and decoder with repetition codes and differential signaling analysis under low, medium, and high noise conditions. The experiments are performed in the presence of three real applications that generate different levels of noise on shared NoC hardware. For low noise conditions, a *LYNX browser* (low noise)

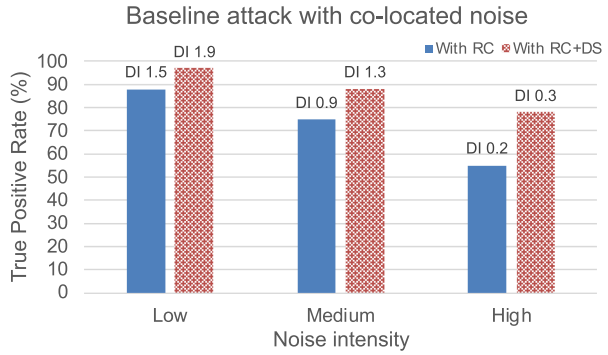

Fig. 9. Baseline attack TP Rate vs. DI under low, medium, and high noise.

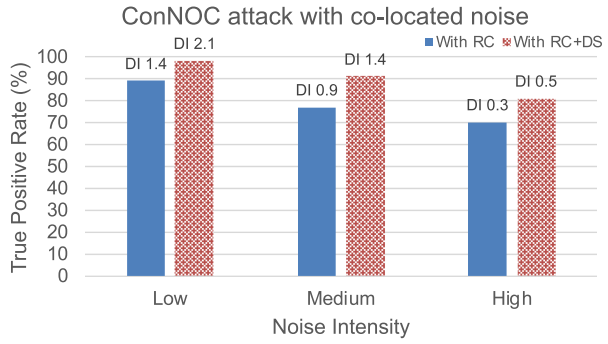

Fig. 10. ConNOC attack TP rate vs. DI under low, medium, and high noise.

application is executed on core 2 for baseline and ConNOC attack setup, the core that hosts transmitter data and traffic between attack setup. For medium noise, the *CURL* application is used that download a file from a remote server. To generate high external noise, an artificial noise is generated using *STRESS* application. The *STRESS* application is used with memory-intensive operations that involve NOC. Figure 9 shows results of baseline attack setup with repetition code only and repetition codes with differential signaling. These experiments use a repetition rate of 55. We observed that the TP rate drops to 88%, 78%, and 55% in the presence of the low, medium, and high noise, respectively, without differential signaling. In the presence of differential signaling, the TP rate stays 97%, 88%, and 78% in low, medium, and high noise, respectively.

Figure 10 shows the experimental results for the ConNOC setup. Here repetition codes of 5 are used, since they result in the target TP rate. The results show that information theory concepts not only make the attack practical but also makes it resilient to unwanted external noise and randomly added noise for obfuscation-based mitigations.

6.3 Covert-communication Attack

This section evaluates our practical attack implementation using baseline attack setup with 55 repetition codes that exploits underlying shared NoC hardware as a timing channel. Our attack consists of a transmitter application and a receiver application. The transmitter application loads the BMP format 240 × 180 pixel image (176-kilometer size). Each byte is transmitted bitwise using the contention (for bit 1) and no contention (bit 0) at a speed of 244 kbps speed under different conditions. Figure 11(a) shows original image, where Figure 11(b) show an image received using 55

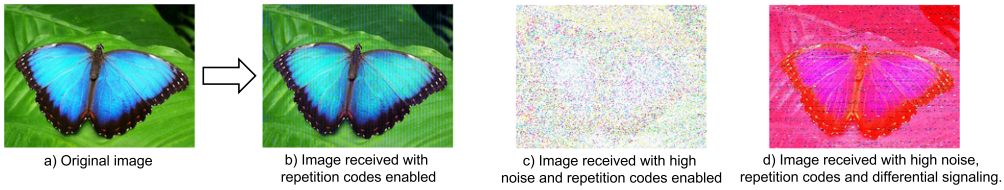

Fig. 11. Covert transmission of sample image with information theory attack.

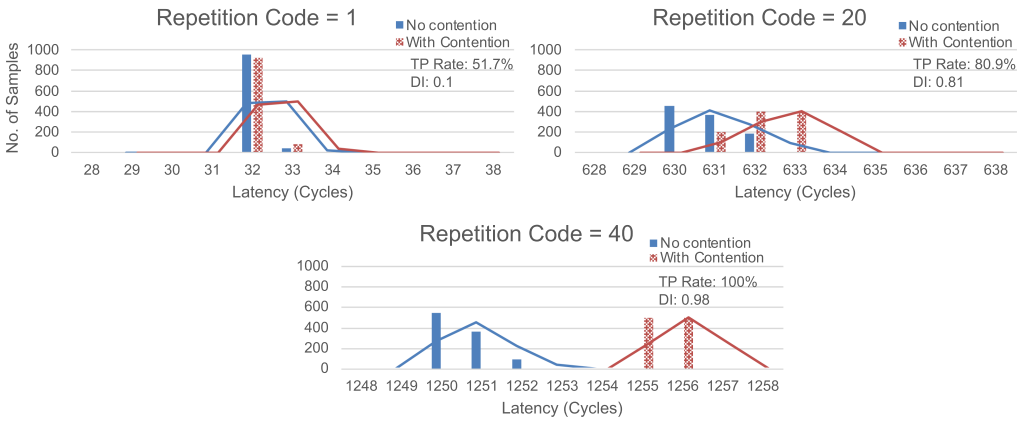

Fig. 12. Histograms showing fidelity of timing variations on simulator.

repetition codes implementation. Figure 11(c) shows retrieved image under high noise conditions with repetition codes only. The visual results confirmed that attack efficacy dropped sufficiently under high external noise. In the last experiment, we have used differential signaling and repetition code of 55 under high noise conditions. Figure 11(d) confirmed our hypothesis that information theory concepts can de-noise the external noise and thus can be used against obfuscation-based mitigation schemes.

6.4 Timing Variations Fidelity on Simulator

Study on performance implications of mitigation schemes requires a simulator. This section evaluates timing variation fidelity on the RISC-V simulator. Figure 12 shows a timing-variation study with contention and without contention using different repetition codes. For example, a histogram without repetition rate (i.e., baseline setup) and a threshold of 32 shows a negligible timing variation with a TP rate of 51.7% and DI of 0.1. If we increase the repetition rate to 20, then the histogram shows considerable timing variation under contention and no contention situations. The threshold of 631 cycles shows a TP rate of 80.9%, and DI increases to 0.81. Finally, a repetition rate of 40 shows that TP rate of 100.00% and DI of 0.98, and this configuration can be used to construct practical attacks. The results in Figure 12 reveal that this simulator has sufficient fidelity for timing-based SCA that targets the underlying NoC hardware.

6.5 Performance Implications of Mitigations Schemes

This work evaluates the temporal and spatial isolation schemes on a 64-core multicore simulator using 2-domains. Figure 13 summarizes the performance implications for six real-world workloads. The results are normalized to the default configuration of 64 cores without any mitigation scheme.

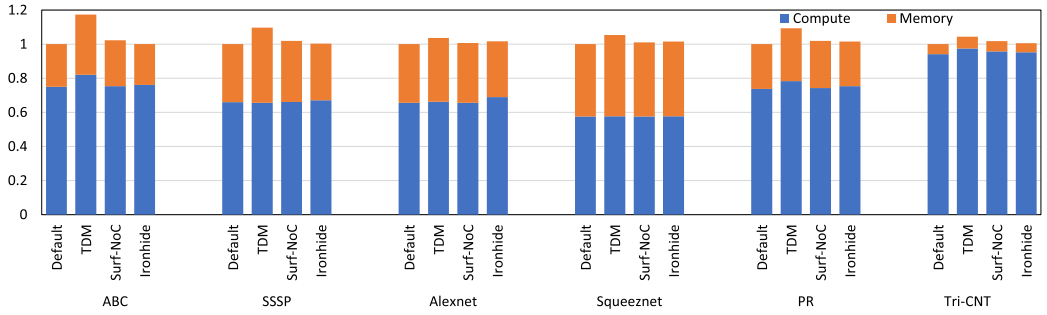

Fig. 13. Applications completion time breakdown for core and memory.

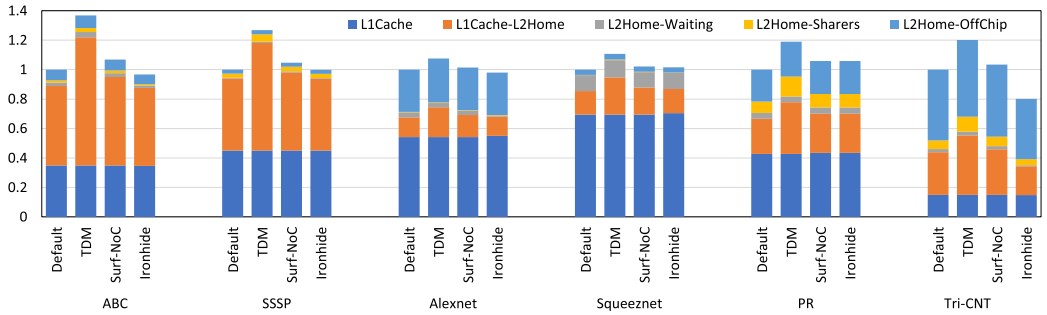

Fig. 14. Applications completion time breakdown of memory hierarchy.

The completion time is distributed in core computation time and memory communication time that involves the NoC hardware.

The TDM requires modification in NoC routers and causes a performance penalty on all NoC routers directly proportional to a number of domains and cores. Our performance evaluation shows an average additional latency of 12.5% compared to the default configuration of 64 cores without any mitigation scheme. Figures 13 and 14 shows the distribution of individual workload and time spent at each component on workload for the TDM scheme. The increase in computation time for TDM is due to stalls while waiting for the data, whereas memory components show an increase due to the additional penalty of TDM implementation. The TDM shows the worst performance compared to Ironhide and SurfNOC. The SurfNOC is an improvement over the TDM mitigation scheme and shows up to 75% reduction in latency overhead compared to TDM. The SurfNOC is implemented on individual NoC routers in 64 cores configurations. Figure 14 summarize the latency distributions of memory operations of six workloads using SurfNOC implementations. Similarly to TDM, the increase in computation time is due to stalls and, and additional latency in L1-to-L2 and L2-to-Off-chip is due to modifications to NoC routers scheduling that is required to implement SurfNOC. The SurfNOC increased completion time on average of 2.97%, which is a quarter of the TDM scheme and competitive to Ironhide scheme.

The Ironhide creates two clusters and dynamically distributes physical resources, including cores, caches, memory, and NoC hardware. The Ironhide dynamic allocation depends on the saturation point on the MPKI curve. Figure 15 shows MPKI curves with saturation points for all six workloads. For example, the performance of ABC saturates at 54 cores, and allocating more cores does not yield further improvements. Therefore, Ironhide can execute another process in parallel with ABC on the remaining 10 cores. However, workloadslike SSSP with high saturation points,

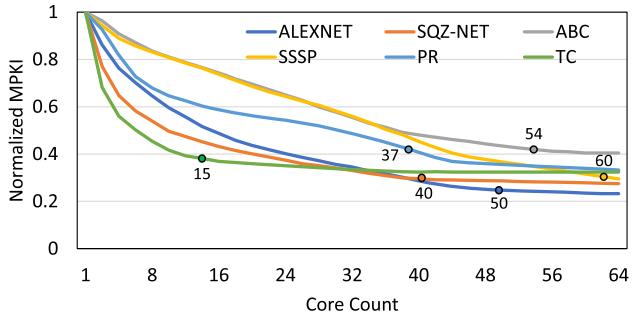

Fig. 15. Benchmark applications MPKI trends.

e.g., 60, result in fewer cores allocated for other clusters. Such resource distribution can affect the overall system performance. We have captured the completion time distribution for benchmarks mentioned in Section 5.4 by allocating the saturation point number of cores. Figure 13 shows that for Ironhide, the core component in completion time breakdown is slightly increased, because Ironhide allocates less number of cores to process the same input. The rest of the components show negligible variation over the default setup. Overall for all workloads, Ironhide shows an average performance overhead of 2.29% over default multicore setup without any mitigation scheme.

Although all isolation-based mitigation schemes protect against timing attacks, the Ironhide and SurfNOC show competitive performance implications of 2.29% and 2.97% for real six workloads and thus should be used in practice. The Ironhide works best if workload demanded cores are available in the cluster. Otherwise, SurfNOC works better.

6.6 Validation of Mitigation on Real Hardware

The simulator-based evaluation of the mitigation schemes shows that both Ironhide and SurfNoC are competitive in terms of performance implications. However, implementation of these mitigation schemes require changes in the processor architecture. SurfNoC needs to update NoC router microarchitecture for packet traversals in their respective waves. For a single virtual channel router design, the SurfNoC requires modifications for domain specific arbitration of crossbar. Moreover, a surf scheduling table needs to be pre-loaded in each router. However, Ironhide requires the creation of clusters of cores such that packets from a given domain do not violate their cluster boundary. Ironhide achieves strong isolation by (1) pinning threads to cores within the assigned cluster, (2) mapping shared cache accesses to the cache resources allocated within the assigned cluster, and (3) mapping memory controller accesses to the controllers assigned within the cluster. Thread pinning on dedicated cores is implemented in most commercial processors. However, mapping cache and memory controller accesses require modifications for hardware resource partitioning. This capability is commercialized, such as Intel Cache Allocation technology [9] and Tilera's hardware hashing functionality to map memory pages to dedicated shared cache slices and memory controllers [10]. We evaluate the Ironhide mitigation on the Tilera multicore processor. The *hash-for-home* hardware capability is used to configure the per-core TLBs. During the virtual to physical address translation in each core, the physical address uses a special hashing feature to determine a user specific tile for shared cache homing. The shared L2 cache misses of a cluster are routed to their respective DRAM regions via dedicated memory controllers. This capability is enabled by a hardware exposed API call. It uses a user specific bit-mask that assigns memory controllers to memory regions (physical addresses).

To validate the performance implications of the Ironhide mitigation scheme on a real machine, the Tilera capabilities are used to configure clusters of cores for the evaluated benchmarks.

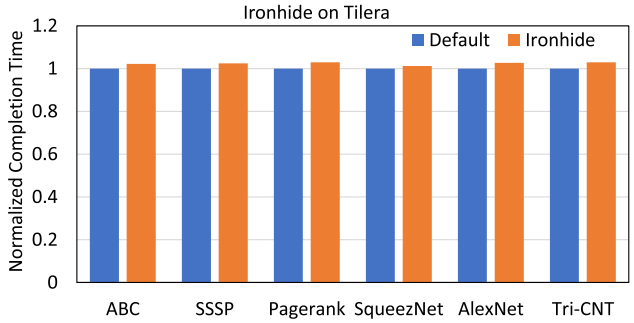

Fig. 16. Performance comparison of benchmark applications on Tilera.

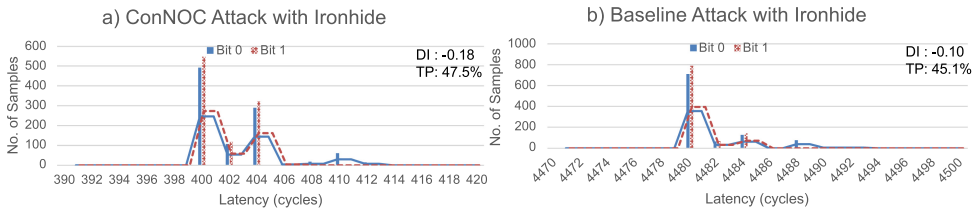

Fig. 17. Ironhide security validations on the real hardware.

The number of cores per cluster are matched with the saturation point of each workload. Figure 16 shows the completion time comparisons that are found to be correlated with the simulator-based evaluation from Section 6.5. Ironhide approach reduces the available core-level performance slightly and impacts performance by ∼2.4% as compared to the default setting (where all cores execute the application).

To validate the security guarantees of Ironhide scheme, the ConNOC attack setup with 5 repetition codes and the baseline attack setup with 55 repetition codes are used. Figure 17(a) shows timing-variations study for ConNoC attack setup where receiver application is pinned in cluster 0, and transmitter application is pinned in cluster 1. The TP rate of 47.1% and negative DI of −0.18 indicates the successful mitigation of the attack. Similarly, Figure 17(b) shows a timing-variations study for baseline attack setup with receiver and transmitter applications are pinned on different clusters. A 45.1% TP and negative DI of −0.10 show that the attack is successfully mitigated.

6.7 Validation of Mitigation on Simulator

The temporal isolation schemes (TDM and SurfNOC) guarantee protection against timing side channel attacks. However, both schemes requires modifications in NoC router architecture and these guarantees cannot be validated on real hardware. For example, the TDM implementations for two domain requires crossbar to have ability to route packets in even or odd cycles for respective domains. Similarly, the SurfNoC requires capabilities to generate a wavelike packet traversing mechanism. Modern commercialized hardware do not have these capabilities, and an application-class simulator is needed to model and validate security guarantees of these schemes. We modified the NoC architecture of our RISC-V simulator and implemented TDM and SurfNOC schemes. The security guarantees of TDM and SurfNOC schemes are validated against baseline attack setup and ConNOC attack setup. Figure 18 shows timing variations histogram for baseline attack. Figure 18(a) shows timing-variation histogram for TDM scheme, where contention and no contention distributions are completed overlapped. The TP rate of 50% shows that the probability of detection

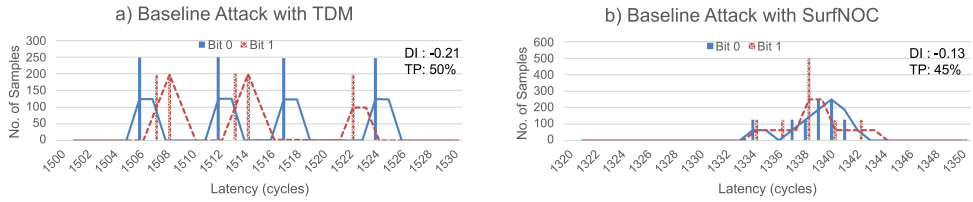

Fig. 18. TDM and SurfNOC mitigation schemes security validation against baseline attack setup on the simulator.

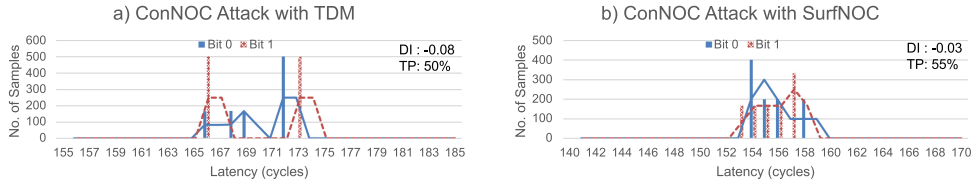

Fig. 19. TDM and SurfNOC mitigation schemes security validation against ConNOC attack setup on the simulator.

of contention and no contention is random guess. The negative DI os −0.21 quantify that both distributions are completed overlapped and attack is fully mitigated. Similarly, Figure 18(b) shows timing-variations histograms for SurfNOC scheme, where contention and without contention distributions are complete overlapped. The less than 50% TP and negative DI of −0.13 shows that baseline attack is fully mitigated. Figure 19 shows timing variations histogram for ConNOC attack setup. A negative DI and 50% TP rate shows ConNOC attack is successfully mitigated using both TDM and SurfNOC schemes.

7 CONCLUSION

This article studies the impact of unwanted noise on the timing-based baseline and ConNOC attack setups that target non-persistent shared NOC hardware in a multicore processor. The results show that a high noise situation is sufficient to drastically reduce the efficacy of timing-based SCA. It proposes an information theory attack that uses concepts of repetition codes and differential signaling to overcome challenges of unwanted noise. The evaluation shows that attack with information theory has efficacy of 97%, 88%, and 78% in low, medium, and high noise conditions, respectively. It implements and demonstrates a successful practical covert-communication attack in the presence of high noise. Based on these experiments, it concludes that the usage of noise as mitigation is inadequate for practical purposes, and isolation schemes are required to mitigate the attack successfully. This work implements state-of-the-art isolation-based mitigation schemes, including Ironhide, a spatial isolation scheme, and SurfNOC, a temporal isolation scheme on a multicore RISC-V simulator. Performance evaluation of Ironhide and SurfNOC reveals a performance impact of 2.29% and 2.97% on security-centric real-world workloads. It concludes that Ironhide works better with workloads that required resources are available in clusters. Otherwise, SurfNOC shows low-performance implications for security-centric workloads.

REFERENCES

[1] 1998. Curl: Command Line Tool and Library for Transferring Data with URLs. Retrieved from https://curl.se/.
[2] 2002. Linux numactl. Retrieved from https://linux.die.net/man/8/numactl.
[3] 2013. FreescaleADL: An Industrial-Strength Architectural Description Language for Programmable Cores. Retrieved from http://opensource.freescale.com/fsl-oss-projects/.

[4] 2017. QUARQ: A Novel General Purpose Multicore Architecture for Cognitive Computing. Retrieved from https://khan.engr.uconn.edu/pubs/quarq-techcon17.pdf.
[5] 2018. LYNX—The Text Web-Browser. Retrieved from https://lynx.invisible-island.net/.
[6] 2019. STRESS—Tool to Impose Load On and Stress Test Systems. Retrieved from https://linux.die.net/man/1/stress.
[7] Masab Ahmad, Farrukh Hijaz, Qingchuan Shi, and Omer Khan. 2015. CRONO: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'15)*.
[8] Usman Ali and Omer Khan. 2021. ConNOC: A practical timing channel attack on network-on-chip hardware in a multicore processor. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST'21)*.
[9] Intel Corporation. 2015. Improving Real-time Performance by Utilizing Cache Allocation Technology. Retrieved from https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf.
[10] Tilera Corporation. 2014. TILE-Gx72 Processor. Retrieved from http://www.mellanox.com/related-docs/prod_multi_core/PB_TILE-Gx72.pdf.
[11] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*.
[12] Halit Dogan, Masab Ahmad, Brian Kahne, and Omer Khan. 2019. Accelerating synchronization using moving compute to data model at 1,000-core multicore scale. *ACM Trans. Archit. Code Optim.* 16, 1, Article 4 (February 2019), 27 pages. https://doi.org/10.1145/3300208
[13] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2018. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security'18)*. USENIX Association, 955–972.
[14] F. Liu and R. B. Lee. 2014. Random fill cache architecture. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 203–215. https://doi.org/10.1109/MICRO.2014.28
[15] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang. 2021. A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography. *ACM Comput. Surv.* 54, 6, Article 122 (July 2021), 37 pages. https://doi.org/10.1145/3456629
[16] Yehia Massoud, Jamil Kawa, Don MacMillen, and Jacob White. 2001. Modeling and analysis of differential signaling for minimizing inductive cross-talk. In *Proceedings of the 38th Annual Design Automation Conference (DAC'01)*. Association for Computing Machinery, New York, NY, USA, 804–809. https://doi.org/10.1145/378239.379070
[17] Jason E. Miller, Harshad Kasture, George Kurian, Charles Gruenwald, Nathan Beckmann, Christopher Celio, Jonathan Eastep, and Anant Agarwal. 2010. Graphite: A distributed parallel simulator for multicores. In *Proceedings of the 16th International Symposium on High-Performance Computer Architecture (HPCA'10)*. 1–12. https://doi.org/10.1109/HPCA.2010.5416635
[18] H. Omar and O. Khan. 2020. IRONHIDE: A secure multicore that efficiently mitigates microarchitecture state attacks for interactive applications. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*. 111–122. https://doi.org/10.1109/HPCA47549.2020.00019
[19] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM addressing for cross-CPU attacks. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security'16)*. USENIX Association, 565–581.
[20] Stephen Röttger and Artur Janc. 2021. A Spectre proof-of-concept for a Spectre-proof web. Retrieved from https://security.googleblog.com/2021/03/a-spectre-proof-of-concept-for-spectre.html.
[21] Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W. Fletcher. 2019. MicroScope: Enabling microarchitectural replay attacks. In *Proceedings of the ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA'19)*. 318–331.
[22] Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W. Fletcher. 2020. MicroScope: Enabling microarchitectural replay attacks. *IEEE Micro* 40, 3 (2020), 91–98. https://doi.org/10.1109/MM.2020.2986204
[23] Dimitrios Skarlatos, Zirui Neil Zhao, Riccardo Paccagnella, Christopher W. Fletcher, and Josep Torrellas. 2021. Jamais vu: Thwarting microarchitectural replay attacks. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*. Association for Computing Machinery, New York, NY, 1061–1076. https://doi.org/10.1145/3445814.3446716
[24] Y. Wang and G. E. Suh. 2012. Efficient timing channel protection for on-chip networks. In *Proceedings of the IEEE/ACM 6th International Symposium on Networks-on-Chip*. 142–151. https://doi.org/10.1109/NOCS.2012.24
[25] H. Wassel, Y. Gao, J. Oberg, Ted Huffmire, R. Kastner, F. Chong, and T. Sherwood. 2013. SurfNoC: A low latency and provably non-interfering approach to secure networks-on-chip. In *Proceedings of the International Symposium on Computer Architecture (ISCA'13)*.

[26] Yu Xue, Jiongming Jiang, Binping Zhao, and Tinghuai Ma. 2018. A self-adaptive artificial bee colony algorithm for global optimization. *Soft Comput.* 22 (2018), 2935–2952.

[27] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security'14)*. USENIX Association, 719–732.

[28] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. 2017. Camouflage: Memory traffic shaping to mitigate timing attacks. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'17)*. 337–348. https://doi.org/10.1109/HPCA.2017.36

Received 15 March 2022; revised 26 September 2022; accepted 29 January 2023