# Detection of and Countermeasure against Thermal Covert Channel in Many-core Systems

Hengli Huang, Xiaohang Wang, *Member, IEEE,* Yingtao Jiang, Amit Kumar Singh, *Member, IEEE*
Mei Yang, *Member, IEEE* and Letian Huang

***Abstract*—The thermal covert channels (TCC's) in many-core systems can cause detrimental data breaches. In this paper, we present a three-step scheme to detect and fight against such TCC attacks. Specifically, in the detection step, each core calculates the spectrum of its own CPU workload traces that are collected over a few fixed time intervals, and then it applies a frequency scanning method to detect if there exists any TCC attack. In the next positioning step, the logical cores running the transmitter threads are located. In the last step, the physical CPU cores suspiciously engaging in a TCC attack have to undertake Dynamic Voltage Frequency Scaling (DVFS) such that any possible TCC trace will be essentially wiped out. Our experiments have confirmed that on average 97% of the TCC attacks can be detected, and with the proposed defense, the packet error rate (PER) of a TCC attack can soar to more than 70%, literally shutting down the attack in practical terms. The performance penalty caused by the inclusion of the proposed DVFS countermeasures is found to be only 3% for an 8×8 many-core system.**

***Index Terms*—Many-core Systems, Thermal Covert Channel, Defense against Covert Channel Attack.**

## I. Introduction

CHIP-LEVEL security can be compromised by the existence of covert channels through which an attacker can gain unauthorized access to sensitive information and the legitimate users are left in the dark. Of many communication media (*e.g.*, heat transfer, timing, or inaudible sound) available for the construction of covert channels, thermal covert channels (TCC) by means of heat transfer can be particularly dangerous [1], [2].

Like any other communication systems, a thermal covert channel involves a transmitter and receiver pair. Essentially, a transmitter admits sensitive data and transmits the thermal signals, while a receiver on the other end of data transmission reads its thermal sensor and recovers the original sensitive data transmitted. In the example illustrated in Fig. 1 featuring an eight-core chip, there exists a covert channel between core *A* and core *B*, and core *A* sits in a secured zone where sensitive information is not supposed to be shared with the cores outside this zone, while core *B* is in a non-secured zone. The transmitter on core *A* can convert the sensitive data (*e.g.*, user passwords) into temperature signals by regulating the core's activities and correspondingly its power consumption. In this example, bit '1' is represented as a sequence of rise and fall of temperature, while bit '0' is represented as no change in temperature. The temperature signals can travel through the chip by heat transfer among the processor cores, and eventually the thermal signals reach the destination, core *B*, where they can be picked up by core *B*'s local thermal sensor, and subsequently, the original data can be recovered.

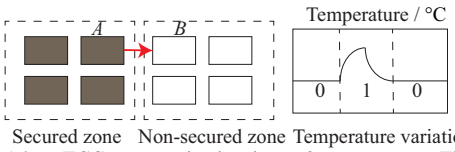



Fig. 1. A 1-hop TCC communication in an 8-core system. The arrow from core *A* to core *B* indicates the direction of heat flow between the two cores.

Since a thermal covert channel, as described in the above example, is committed to transmit thermal signals over a chip, it is prone to be disrupted/degraded by thermal noise and ambient temperature fluctuations. Recently, there have been new designs of thermal covert channels that have improved throughput and noise immunity. Masti *et al.* [1], for instance, built a 1-hop TCC channel (a channel's transmitter and receiver are one hop/core apart) that could achieve a throughput of 1.33 bps with a bit error rate (BER) of 11% on a commercially available platform. Later on, a separate study [12] demonstrated that an intra-core TCC channel (a channel that both the receiver and the transmitter have access to the same thermal sensors or the same thermal data files) has a throughput higher than 45bps, and it also demonstrated a 1-hop channel with a throughput of higher than 5bps with a BER less than 1%.

Apparently, the ever-improving performance of the thermal covert channels poses even greater risks and danger to data/information security in today's ubiquitous multi-core/many-core systems. Although it is stated in [12] that TCC can be countered by limiting the accessibility to thermal information or by decreasing the resolution of sensors, as in [28], the receiver can infer temperature by means of measuring the memory access delay. In fact, in most off-the-shell com-

H. Huang and X. Wang are with the School of Software Engineering, South China University of Technology, China. (e-mail: sehenry@mail.scut.edu.cn, xiaohangwang@scut.edu.cn). Xiaohang Wang is the corresponding author.

Y. Jiang and M. Yang are with the Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, USA. (e-mail: yingtao.jiang@unlv.edu, mei.yang@unlv.edu)

A. K. Singh is with the School of Computer Science and Electronic Engineering, University of Essex, UK. (e-mail: a.k.singh@essex.ac.uk)

L. Huang is with the University of Electronic Science and Technology of China, Chengdu 610054, China. (e-mail: huanglt@uestc.edu.cn)

This research program is supported by the Natural Science Foundation of Guangdong Province No. 2018A030313166, Pearl River S&T Nova Program of Guangzhou No. 201806010038, the Fundamental Research Funds for the Central Universities No. 2019MS087, Open Research Grant of State Key Laboratory of Computer Architecture Institute of Computing Technology Chinese Academy of Sciences No. CARCH201916, Natural Science Foundation of China No. 61971200, the Key Laboratory of Big Data and Intelligent Robot (South China University of Technology), Ministry of Education, and the National Key Research and Development Program of China No. 2019QY0705.

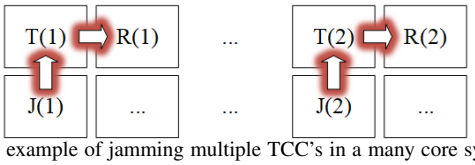



Fig. 2. An example of jamming multiple TCC's in a many core system. T($i$), R($i$) and J($i$) are the $i$-th transmitter, receiver and jammer cores, respectively. The red arrow is the direction of heat flow.

mercial computers, thermal sensor information can be accessed by user programs (receiver). A naive countermeasure to fight against TCC attack is to produce jamming noise with the same transmission frequency as TCC. However, it fails to jam a channel which is enhanced by exploiting frequency-hopping spread spectrum (FHSS) [32]. Using full band jamming or designing an enhanced jamming model to track the transmission frequency will cause huge implementation overhead. In addition, jamming tends to pause a normal task running in a neighbor core to generating thermal noise. As shown in Fig. 2, if a many-core system (*e.g.,* 8 $\times$ 8) has multiple TCCs and each TCC is away from the other, in this case, blocking one TCC needs one neighbor core to run the jamming, which wastes a lot of legitimate cores thus lead to poor performance. In this paper, we attempt to avoid these problems by proposing a countermeasure against thermal covert channel attack inspired by a DVFS-based observation illustrated in Fig. 3.

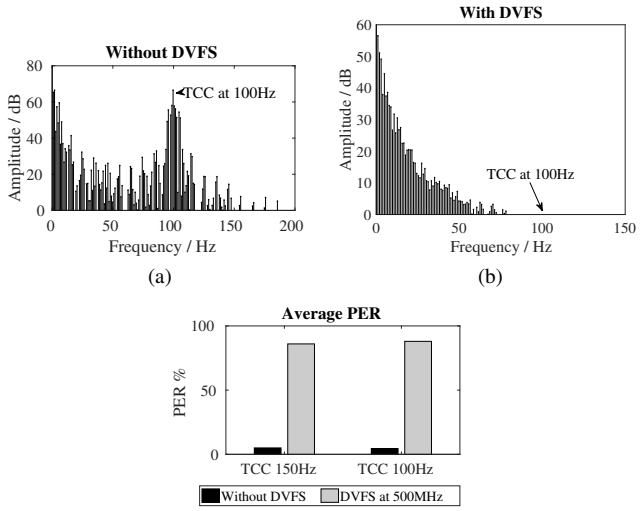



Fig. 3. The spectra of the thermal signals of the transmitter core with and without DVFS turned on. Note that the transmitter and the receiver are physically adjacent to each other in this example. (a) and (b) are the spectra of the thermal signals generated by the transmitter core when the transmission frequency is 100Hz, and the working CPU frequency of the transmitter core is 2000MHz and 500MHz, respectively; and (c) is the comparison of average packet error rate (PER) of the thermal signals received by the receiver when the DVFS is turned on and off at the transmitter core.

Assume that a many-core system in Fig. 3 has a thermal covert channel with a transmission frequency of 100Hz. The frequency level (CPU working frequency) of the transmitter core is set to be 2GHz, which is fairly typical in modern many-core systems. Also assume that the transmitter and the receiver run their dedicated programs for transmission and receiving, respectively, and their neighboring cores run a few threads created by a multi-threaded application (*e.g.*, Barnes from SPLASH-2). The thermal signals without DVFS or with DVFS applied on the tranmitter core, generated by the transmitter are shown in Fig. 3(a) and 3(b), respectively. One can see that the data transmitted over the thermal covert channel can be successfully recovered when no DVFS is applied (Fig. 3(c)), with an average packet error rate (PER) of 5%. However, if the CPU frequency of the transmitter core drops to 500MHz, as a result of dynamic voltage and frequency scaling (DVFS), the thermal signals received by the receiver suffer a high PER (88% in Fig. 3(c)), in this case, no signal around 100Hz can be detected. In addition, when we change the transmission frequency to 150Hz (Fig. 3(c)), the average PERs at the receiver side are 5% and 86% without and with DVFS turned on, respectively.

The comparison in Fig. 3(c) motivates us that once the DVFS is applied to the exact transmitter core, a TCC attack cannot work with a high PER.

Although DVFS can be used to countermeasure the thermal covert channel attack, as shown in Fig. 3, blindly applying this technique may lead to significant performance loss. In the worst case, when all the suspicious cores run at their lowest frequency levels (*e.g.*, 500MHz), the performance can drop to an unacceptably low level. This performance issue is thus addressed along with the proposed countermeasure that includes three major steps.

During the first step, a detection scheme is applied to a many-core system to check if it encounters any thermal covert channel attack.

In the second step, the threads involved in a thermal covert channel are determined.

In the third step, for all the found TCC threads identified engaging in a thermal covert channel from the previous step, a DVFS-based countermeasure is applied to the cores where those threads run to shut down any possible TCC signal transmission at a minimum cost to the system performance.

This detection and countermeasure method to thwart covert thermal channel attack marks a significant extension of our previous work [3], summarized below.

1) The detection algorithm is simplified by eliminating the module of bandpass filtering. Our previous work uses a bandpass filter to filter out the signals that are out of every band of interest, after which the signal's peak amplitude is determined. This process incurs substaintial amount of processing time to scan through the entire frequency band in one detection cycle. Instead, we only need to calculate the spectra of the signals once in a detection cycle with this improved detection algorithm.
2) The detection algorithm is also modified in the way that it can handle TCC whose transmitter and receiver communicate over two different channels. Our previous work restricted to deal with the transmitter and the receiver operating over one channel, and used the detected channel to locate a pair of TCC transceiver cores.
3) Intra-core channel is taken into consideration while designing the countermeasure scheme. There are two types of intra-core channels: i) The receiver and transmitter threads are both located at the same physical core and they share the same thermal sensors; and ii) The receiver

and the transmitter have access to the same thermal data files. An intra-core channel can leak sensitive data more effectively than other type channels. Detection of TCC is better performed at the logical core level to exactly pin down the threads of intra-core channels.

4) We introduce the CPU workloads per logical core as a better metric for the detection of TCC threads that are associated with logical cores. Previous work found out a pair of transmitter receiver cores by detecting the thermal signals, which fails when the transmitter and receiver threads run in the same physical core. Instead, detecting CPU workloads of each logical core can locate a TCC pair even for intra-core channels.
5) Experiments are substantially extended, including the experiments on a 3D many-core system and on a modern real machine. We also show our proposed countermeasure is suitable for shutting down the TCC enhanced by FHSS (Frequency Hopping Spread Spectrum).

The rest of this paper is organized as follows. Section II surveys the previous works and provides the preliminaries about the thermal covert channel attacks. Section III presents the details regarding the defense against TCC. In Section IV, the proposed countermeasure is experimentally verified, and the results are reported. Finally, Section V concludes this paper.

## II. RELATED WORK AND PRELIMINARIES

### A. Covert Channels

Covert channel attacks impose a major security threat against literally any part or aspect of our computer and information systems, ranging from cloud systems [15], operating system [16], to multi/many-core chips [17], *etc.* Covert channels can be established using a wide range of communication media that span from inaudible sound [18], inter-arrival timing of packets [19]–[21], light [22], [23], magnetic field [24], [25] to voltage [13]. At the chip level, covert channels can exploit cache-timing [17], dynamic frequency scaling [26], [27], *etc.*, for data transmissions.

Unlike many types of covert channels, a thermal covert channel does not rely on any shared resources such as cache and memory, which helps it easily circumvent modern system's defense. Relying on a simple on-off keying line coding scheme to encode bit '1' and bit '0', a thermal covert channel over an x86-based platform could transmit data with a BER of 11% and a throughput of 1.33 bps, as demonstrated in [1]. Bartolini *et al.* [12] adopted a Manchester encoding scheme, and they managed to improve the efficiency of a thermal covert channel over a real machine with a throughput of higher than 5bps and a BER lower than 1% for a 1-hop channel, and with a throughput of higher than 45bps for an intra-core channel. A different encoding scheme based on non-return-to-zero (NRZ) was adopted in [2], and it was shown that a properly high transmission frequency could significantly prevent the heat noise from interfering other active cores. Furthermore, in [28], without reading any thermal sensors, the receiver can infer temperature by means of measuring the access delay of DRAM memory which is on top of the system-on-chip (SoC).

Although thermal covert channel attacks are perceived to impose an increasingly severe security threat to today's multi-core systems, yet there has little work done to develop a defense or countermeasure against it, especially a TCC that can only detected at runtime and is turned on for a short period of time.

In [1], a possible countermeasure was proposed. By restricting processor cores' access to the thermal sensors, and particularly separating processes that run on the same core or two neighboring cores and isolating cores temporally and spatially, the effectiveness of a TCC, if it ever exists, was supposed to be constrained. One major drawback of this scheme also stems from its requirement to limit processor cores from accessing the thermal sensors that are critical for a many-core chip's power/thermal managment as well as its safe operation. As a matter of fact, limiting/restricting a processor core to access to the thermal sensors does not prevent TCC from obtaining the thermal information (thermal signal); such information can actually be estimated through other means, like measuring the memory access delay, which is the case in [28]. In addition, there is no guarantee that the transmitter and receiver of a TCC can be detected and mitigated by separating processes.

Another countermeasure against TCC was suggested in [33], with a set of rules specific to FPGA devices. Essentially, an FPGA device needs to be heated up or cooled down to a known temperature (*i.e.*, the device goes through a thermal reset) before the device is available to the next user. Another big piece of the countermeasure is to enforce a minimum idle period between users to allow the FPGA to cool down and stay at a steady temperature between users. All these rules in [33] cannot stop TCC. When a TCC is active during runtime, it generates its own thermal signals.

Johann and Ozgur [34] tended to mitigate thermal side channel (TSC) by decorrelating thermal patterns from power and program activities, and their scheme focused on optimizing the floorplans of heterogeneous 3D ICs at design time. However, this method has nothing to do with the runtime detection and mitigation of TCC.

### B. Key Components and Functionalities

As shown in Fig. 1, the attacker of a TCC relies on a transmitter-receiver pair to launch and sustain an attack. A transmitter in this context is a specially designed program running on a processing core sitting in the secured zone, while a receiver is a program running on a processing core outside the secured zone. Note that the secured zone in a multi-core platform provides secure environment and is commonly supported by technologies like ARM TrustZone [4] and Intel software-guard extensions (SGX) [5]. Bascially, Intel SGX relies on a container called 'Enclave' to protect user privacy code and data from being accessed outside the container, and it has a mechamism to allow users to specify which code should be made private. Unfortunately, both the ARM TrustZone and Intel SGX are found vulnerable to side channel attacks [6], [7]; it was demonstrated in [8] that the SGX could be compromised by exposing it to voltage-induced hardware faults. In this

study, we assume that any secured zone is vulnerable to attacks, and the transmitter program capable of launching a TCC can be injected into the secured zone through software updates or some other means. The receiver, since it is installed in a non-secured zone, can be easily implanted by the hacker.

To covertly transmit secret information from a secured zone to a non-secured one, the TCC programs need to create and measure temperature signals. Except very few cases as reported in [28], which measured the decay rate of the DRAM cells to identify heat transfer of a thermal covert channel, most of the TCCs require access to on-chip digital thermal sensors (DTS) to acquire the temperature information across the chip [1], [2], [12]. The attacker (both the transmitter and the receiver) of a TCC can access the local thermal sensors' data by reading some specific control registers (*e.g.*, Model Specific Register in Intel's x86 processors) [10]. Although only privileged instructions can access the MSR, the attacker can acquire such privilege indirectly, through installed temperature monitoring software, or through the sysfs system in Linux [11].

The two most important parameters for on-chip thermal sensors are the refresh rate and resolution of thermal sensors. For Intel MSR and coretemp [11], the refresh rates are 1000 times per second and 500 times per second, respectively, which correspondingly set the highest transmission frequencies to be 500Hz and 250Hz, as dictated by the Nyquist sampling theorem. In this paper, we thus let the transmission frequency of an TCC attack fall into the range of DC to 500Hz. As for the resolution, the on-chip DTS sensors of several off-the-shelf CPU cores have a resolution of 1°C [13] and the state-of-the-art thermal sensors can even reach a resolution of 0.12°C [14]. It has been anticipated that sensors with an even higher resolution (0.1°C [9] or better) will emerge in many-core systems, which will enable TCC to run at a higher transmission frequency.

Opposite to the attacker that relies on a transmitter/receiver pair stands the defender. The defender is a global manager which might be in the secured zone and has the privilege to read all the cores' temperatures. In addition, the defender shall be able to monitor all the cores' workloads (*e.g.*, instructions per cycle, IPC) in real time by reading their retired instructions and unhalted reference cycles from their performance counter registers [10]. To make its defence more efficient, the defender may distribute the necessary defence tools, such as detection and blocking programs, to run on any core that it deems necessary.

### C. TCC Communication Architecture

A thermal covert channel involves a transmitter and a receiver pair, and the signal flow is shown in Fig. 4. The transmitter reads the sensitive data and packetizes them with a padded Error Correcting Code (ECC). Besides data packets defined in the communication protocol, there are other types of packets, namely request (*i.e.*, Connect Request and End) and acknowledgement (*i.e.*, Connect ACK and Data ACK) packets.

Once a packet to be transmitted is created, it needs to be converted to the thermal signal for transmission. Basically,

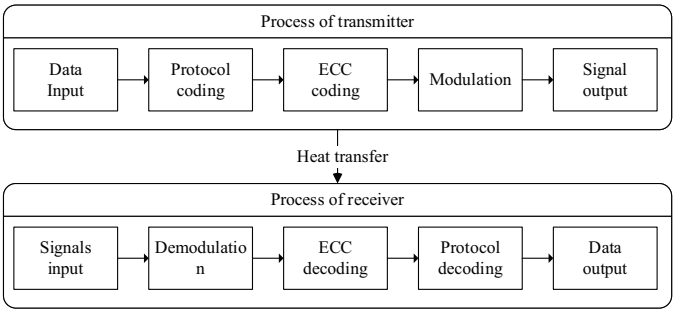



Fig. 4. The elements and signal flow of a communication system through a thermal covert channel: from the transmitter to the receiver.

the packet to be transmitted dictates the transmitter core's program to generate temperature signals by letting the core run computation-intensive code to boost the temperature of the core or insert idle CPU cycles to cool it down.

On the receiver side, it picks up the temperature signal from its local thermal sensor, and demodulates the thermal signal and converts it back into the binary bit streams. Next, the receiver examines the ECC code for packet integrity purpose. If the received packet is determined uncompromised during the transmission, the receiver extracts the data fields from the packet; otherwise, the receiver discards the binary bit streams and waits for retransmission.

### D. Channelization and Communication Flow

Hereandafter we consider that the TCC includes both intra-core channels and inter-core channels. As for an inter-core channel, the receiver can only read its local temperature sensor, and the receiver and the transmitter are at different cores. When the transmitter and the receiver are physically next to each other, they form a 1-hop channel, as the case between core *A* and core *B* shown in Fig. 1.

The transmitter and the receiver first need to reach an agreement on the transmission frequency for the thermal covert channel. The flow of a communication session, modified from the scheme introduced in [2], is shown in Fig. 5(a).

1. To initiate a session, the transmitter sends a connect request packet to the receiver and waits for a reply.

2. When the receiver identifies that the received packet is a legitimate request packet, it replies with an acknowledgement (*i.e.*, Connect ACK) packet to the transmitter. If the transmitter does not receive a Connect ACK packet within certain amount of time, it assumes the packet did not go through and then sends another request packet.

3. The transmitter transmits data packets to the receiver after receiving a Connect ACK packet.

4. The receiver keeps receiving data packets sent from the transmitter. Every time when a data packet is received, the receiver replies an acknowledgement (Data ACK) packet to the transmitter. If the transmitter does not receive a Data ACK packet within certain amount of time, it assumes the last packet got lost and thus resends the packet. If the receiver receives the same data packet more than once, it just discards the duplicate packet and replies with a Data ACK packet.

5.The transmitter sends a termination (End) packet to notify the receiver to end the current session.

6. If the receiver receives an End control packet, it terminates this session. If the End packet is corrupted during transmission, the receiver exits this session after its waiting time is expired.

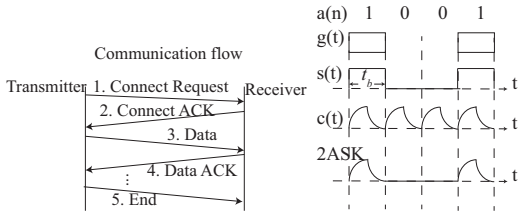



Fig. 5. (a) Illustration of a communication session, (b)The waveforms of the 2ASK modulation for Eqns. (1)-(3).

### E. Transmitter

A transmitter has the following modules.

1. Data input module, which runs on a core within a secured zone, collects sensitive data/information and then converts it into a bit stream.

2. Protocol coding module, where the transmitter generates data and control packets (*i.e.*, request or acknowledgement packets) that do not include ECC codes.

3. ECC coding module, where an ECC code is added to every data packet.

4. Modulation module, where the amplitude shift keying (ASK) modulation, illustrated in Fig. 5(b), is adopted to modulate the data packets with a transmission frequency $f_t$ given below,

$$e_{2ASK}(t) = s(t) \times c(t) \tag{1}$$

$$s(t) = \sum_n a_n \times g(t - n \times t_b) \tag{2}$$

$$t_b = 1/f_t \tag{3}$$

where $s(t)$ is the signal to be transmitted, $c(t)$ is the carrier signal, $a(n)$ is the logic value ('1' or '0') of the $n$-th bit, $t_b$ is the temporal duration of a signal duty cycle, and $g(t)$ is the baseband pulse waveform with a duration of $t_b$.

To make the carrier signal $c(t)$ periodical with a frequency of $f_t$, the NRZ encoding scheme in [2] is employed to avoid continuous temperature build-up when transmitting continuous '1's, as shown in Fig. 6(a). In this NRZ encoding, a '1' is represented as a rise in temperature followed by a fall, as shown in Fig. 6(b). A single bit '1' spans two times, $t_h$ and $t_l$:

$$t_b = t_h + t_l \tag{4}$$

where $t_h$ is the duration of the thermal signal at a high temperature level, and $t_l$ is the duration of the thermal signal at a low temperature level. Bit '0' is represented as a low temperature with a duration of $t_b$.

5. Signal output module, where the transmitter sends out the signal by controlling the power consumption of the transmitter core. That is, the transmitter runs computation-intensive code with the sole purpose to generate heat to create a high temperature, and it inserts idle cycles, which gives time for the running core to cool down, thus creating a low temperature.

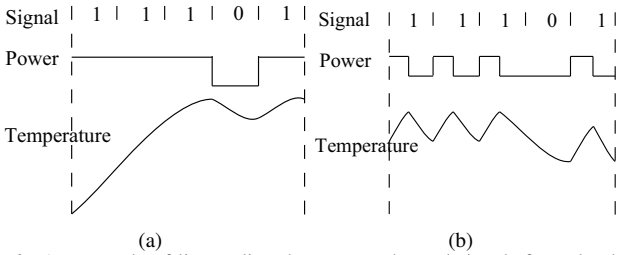



Fig. 6. An example of line coding that creates thermal signals from the data to be transmitted. (a) Waveform of temperature variations controlled by the traditional on-off keying [1], (b) waveform of temperature variations with the NRZ scheme in [2].

### F. Receiver

A receiver also has five modules, and their functionalities are descibed below.

1. Signal input module: This module is responsible to sample its local thermal sensor with a sampling frequency $k$ ($k >=2$) times higher than that of the highest transmission frequency, as dictated by the Nyquist sampling theorem.

2. Demodulation module: The receiver uses a bandpass filter with its center frequency tuned to the transmission frequency to filter out the signals out of band. After filtering, the decision-making module decides on the binary value of the signal by comparing the signal amplitude against preset thresholds. That is,

$$S_i = \begin{cases} 1 & A_i \geq A_0(f_t) \\ 0 & A_i < A_0(f_t) \end{cases} \tag{5}$$

where $S_i$ is the binary value of the $i$-th bit, $A_i$ is the amplitude of the $i$-th bit after filtering, and $A_0(f_t)$ is a preset threshold at frequency $f_t$.

3. ECC decoding module, where ECC of the received packet is calculated. If the ECC does not match the data in the packet and the errors are deemed not correctable, the packet is marked so. Otherwise, the data packet is considered uncompromised over the transmission and it will be processed further.

4. Protocol decoding module, where a received packet is inspected in search for a specific preamble (*e.g.*, 101010).

- If the packet doesn't contain such a preamble, which could get lost during the transmission, the packet is discarded. Once the packet is marked as 'incorrectable', the receiver sends an acknowledgement packet back to the transmitter and asks for a resend.
- If the received packet contains a correct preamble, the receiver sends a positive acknowledgement (*i.e.*, Data ACK or Connect ACK) packet to the transmitter.

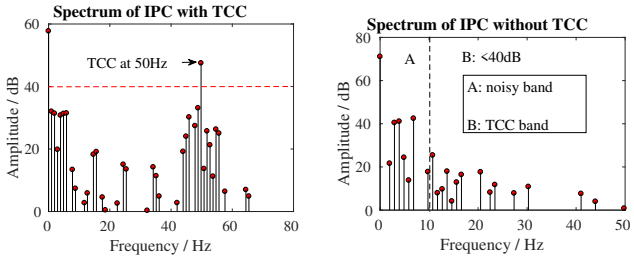
5. Data output module, where a recognized data packet's data fields get extracted.

## III. COUNTERMEASURES AGAINST THERMAL COVERT CHANNEL ATTACKS

To fight against the thermal covert channel attack, the proposed countermeasure essentially consists of three major steps: detection, positioning, and traffic blocking. All the three steps are initiated by a core designated as the global manager. We assume that the system supports hyper-threading

(simultaneous-multithreading, SMT), which is becoming commonplace in today's multi-core processors. The detection is performed at logical core level, and a physical core can host one or multiple logical cores.

### A. Spectrum analysis of TCC




(a) (b)

Fig. 7. (a) The spectrum of IPC values of a logical core where a transmitter thread runs. The transmission frequency is 50Hz. (b) The spectrum of IPC values of a normal logical core where a thread of a typical multi-threaded application (*i.e.*, Bodytrack from PARSEC) runs.

Since the encoding scheme of a TCC attack is often not known a priori, we have to rely on analyzing the thermal signals in frequency domain. First of all, the frequency band from DC to 10Hz (referred as band *A*) is treated as noisy channel and the frequency band higher than 10Hz is treated as a band suitable for TCC transmission, since the thermal noise from other cores is concentrated at band *A* and experiments show that the average PER of TCC in band *A* is higher than 50%. Thus, we set the detection frequencies of the signals fall into the range of 10Hz to 500Hz (referred as band *B*). Unfortunately, detecting the thermal signal of TCC is not suitable for separating a normal logical core from a TCC logical core since a thermal sensor is shared by multiple logical cores for systems that support hyper-threading. For example, if the transmitter and receiver threads both run in the same physical core in a hyper-threading system, our previous work [3] can't find out a pair of physical cores whose thermal signals are deemed as TCC signals. Instead, for the first time, we propose exploiting the instantaneous CPU workload (measured as instructions per cycle, IPC) to be a better metric to analyze the TCC's behavior.

Fig. 7 shows the spectrum of the IPC values of a logical core with and without a transmitter thread running on it. One can see that when a logical core runs a transmitter thread (as shown in Fig. 7(a)) or a normal thread (as shown in Fig. 7(b)), the signal amplitudes of the IPC values of that logical core are higher or lower than a threshold $\rho$ (*i.e.*, 40dB) in band *B*, respectively. One can easily find out the working frequency of TCC at which the signal amplitude is the highest (as shown in Fig. 7(a)) in band *B*.

To make the threshold $\rho$ in band *B* more general, that is, independent for different applications, the threshold is statistically determined using a method described in the Appendix. After the threshold $\rho$ is determined offline, We use a frequency-scanning-based detection that runs through band *B* to find out the suspicious threads, as summarized in Section III-B.

### B. Overview of the proposed countermeasure

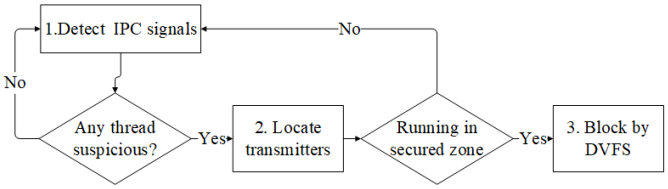



Fig. 8. The procedure of the proposed countermeasure.

Here a detection cycle refers to the time in the detection step and the positioning step. A global manager is a thread to control all these three steps. The three steps are shown in Fig. 8.

Step 1 (Detection): First, the global manager samples all cores' IPC values and it is assumed that the global manager has the privilege to read all cores' register values (*i.e.*, IPC related registers). Then the global manager commands all cores to use a frequency-scanning-based detection that runs through band *B* to find out all suspicious threads whose signal amplitudes of IPC values are higher than $\rho$. The global manager may run multiple detection cycles to identify an attack.

Step 2 (Positioning): In this step, the global manager tries to locate the threads (essentially the logical cores running the threads) that engage in data transmissions in the secured zone. Due to the physical protection nature of the secured zone (*e.g.*, an 'Enclave' is enabled by the processor reserved memory (PRM) [5]), if the global manager cannot access the address space of some threads, those detected threads are deemed as transmitters. Note that a legitimate application may well behave like a TCC that transmits data at some fixed frequency.

Step 3 (Traffic blocking): In the last step, if the previous step provides the positions of TCC logical cores, the physical cores which the detected logical cores belong to are applied DVFS with a sole purpose to block communications within a thermal covert channel. Note that once killing a legitimate thread, the entire multi-threaded application may be terminated.

The global manager runs these three steps repeatedly, and once it receives all cores' reports, it will ask the TCC cores to take traffic blocking step and continues to initiate a new detection cycle to sample all cores' IPC values. Assume that the times of sampling all cores' IPC values, all cores' frequency scanning detection, and positioning transmitter threads are $t_1$, $t_2$, and $t_3$, respectively, so the time of each detection cycle is $t_1+t_2+t_3$ which is shown in Fig. 9. When the global manager core is sampling IPC values, other cores have an interval, $t_1$ (*i.e.*, 2 seconds in our experiments), to run their normal tasks. Since the TCC almost only works during $t_1$, our proposed method can always detect TCC cores.

All of the notations used in Algorithm 1 are listed in Table I.

### C. Detection

As shown in Fig. 10 the global manager commands each logical core to detect whether there is a possible attack or not, and each logical core performs detection individually and reports to the global manager with its findings. In each

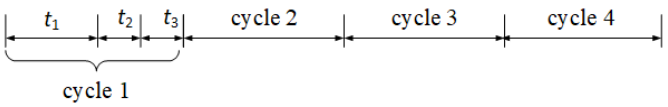



Fig. 9. The detection cycles.

TABLE I
NOTATIONS USED IN ALGORITHMS 1

| Notation | Description |
|---|---|
| $IPC_i$ | The CPU workload traces (IPC) of logical core $i$ recorded in a detection cycle. |
| $\rho$ | A preset amplitude threshold for the detection of IPC signal. |
| $\triangle f$ | Frequency incremental as needed in Fast Fourier transform (FFT) and it is set to be 1Hz in this study. |
| $n_c$ | Number of logical cores in the SoC. |
| $L$ | A list of the transmitter logical cores during a detection cycle with a length of $n_c$. This list is initialized to be empty. |
| $F$ | The available frequency set. The elements span from 10Hz to the upper bound of band $B$ with an incremental of $\triangle f$. |
| $p_{ipc_i}$ | A mapping set with each element made of a "key-value" pair corresponding to the mapping between a "key" and a "value". Here a "key" of an element is a signal frequency from the spectrum of $IPC_i$, and the corresponding "value" is the signal amplitude at the frequency "key". |
| $p_{ipc_i}(f)$ | The "value" of an element of $p_{ipc_i}$ at the frequency of $f$. |

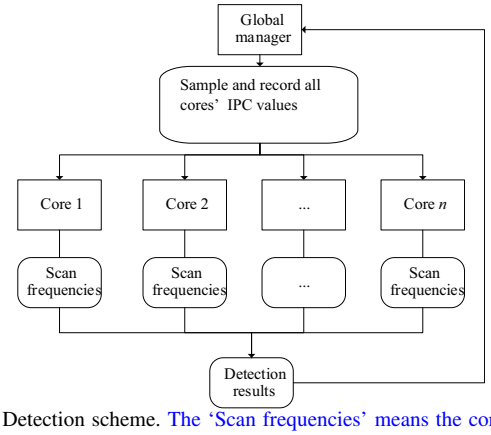



Fig. 10. Detection scheme. The 'Scan frequencies' means the core scans the entire power spectrum of IPC signal.

detection cycle, the global manager samples and records each logical core's IPC profiles over a time window, say $t_1$ seconds (*i.e.,* 2 seconds in our experiments), and it then asks each logical core to execute a frequency scanning process to search for any existing TCC channels. This detection task is set to supersede any other tasks of a logical core is performing.

*1) Frequency-Scanning-Based Detection of TCC Channels:* In a detection cycle, the system checks which signal frequencies are available for TCC channels. As listed in Alg. 1, the frequency–scanning-based detection algorithm works as follows.

Step 1. The global manager initiates a detection cycle to see if there is any possible TCC channel; upon receiving the command from the global manager to start a new detection cycle, each logical core extracts the spectrum of its IPC signals (see lines 3 and 4 in Alg. 1). Then each logical core scans the entire spectrum in a linear sweeping fashion with an incremental of $\triangle f$ (see lines 5 and 11 in Alg. 1).

**Algorithm 1:** Frequency-scanning-based detection at core $i (1 \leq i \leq n_c.)$

```
   Input: IPC_i, ρ, L, F, and △f.
1  Initialization: f ← 10Hz;
2  begin
3      calculate the spectrum of signal IPC_i using the
         Fast Fourier transform (FFT) algorithm;
4      fill p_ipc_i with all the "frequency-amplitude"
         elements from spectrum of IPC_i;
5      while f ∈ F do
6          if p_ipc_i(f) > ρ then
7              add i to L;
8              report logical core i to the global manager;
9              return.
10         end
11         f = f + △f.
12     end
13     send a message to the global manager that no TCC
         channel is found in core i.
14 end
```

Step 2. During the frequency scanning process, each logical core determines the amplitudes of the IPC signals (see lines 6 to 10 in Alg. 1). By comparing the signal amplitudes against the threshold $\rho$, one can decide whether the signals are actually from a covert channel or not (see line 6 in Alg. 1). Note that a normal application running on a core may be detected as a suspicious core; this false positive, with a low probability (typically 5% following the threshold computation method in Appendix), is acceptable in this step. Once a suspicious channel is detected in a logical core $i$, the position $i$ is added to list $L$ (see line 7 in Alg. 1).

Step 3. At the end of a detection cycle, if a logical core confirms that a TCC attack is present, it reports its findings, *i.e.,* the position of the detected logical core, to the global manager (see line 8 in Alg. 1). Otherwise, the logical core can conclude that no TCC attack has been found in the current detection cycle and reports so to the global manager (see line 13 in Alg. 1).

Step 4. If the global manager finds no TCC channel exists in any of the logical cores, it initiates a new detection cycle. Then, it will start all over from step 1 again.

If the global manager receives a notification of the existence of a TCC channel, it begins to locate all transmitter logical cores, as described in the next subsection.

### *D. Positioning the Logical Cores Affected by a Thermal Covert Channel Attack*

A message that a logical core sends to the global manager at the end of a detection cycle contains two parts of information: 1) a confirmation that one possible TCC channel is found,

and 2) the position of that logical core. If the global manager receives such a message, it begins to find the logical cores running in a secured zone.

This positioning step involves 2 steps and is explained below.

Step 1. For each detected thread in list $L$ (see Table 1), where $L$ stores all the detected logical cores during a detection cycle, the global manager checks whether the address space of that thread can be accessed or not. If the answer is yes, core $i$, the position of that thread, is considered as a normal logical core outside a secured zone, and as so, it is removed from list $L$. Note that in a short time (*e.g.,* 2 seconds), we assume a TCC thread keeps running in a dedicated logical core.

Step 2. At the end of each detection cycle, if the global manager decides to locate one or more logical cores that harbor a TCC attack (*i.e.,* list $L$ is not empty), it exits the positioning step and turns to the next traffic blocking step (Section III-E). If the global manager fails to locate a transmitter logical core, it initiates a new detection cycle as an attempt to collect extra information.

### E. Blocking the Thermal Covert Channel Attack

Once a core in list $L$ is identified to be associated with a TCC, a countermeasure shall follow to block any future TCC transmission. Here we assume that a TCC thread keeps running on a dedicated logical core throughout a full communication session or even beyond. We apply DVFS to the CPU core where the detected threads run to block the transmission. Scaling down the frequency level can effectively change the temperature signal drastically so that the the receiver will experience extremely high PER that, in the end, no meaningful TCC communication is possible (*i.e.*, no data can be faithfully received).

Note that in today's multi-core processors that hyper-threading is becoming commonplace, more than likely, the physical cores in a TCC may also run some legitimate threads other than the TCC threads. In this case, the proposed DVFS strategy should take into account of both the security requirements and system performance issues. As a tradeoff, a variable down-up rate, $\beta$, herein is introduced, which is defined as the ratio of the temporal duration of the low frequency level, denoted as $t_{down}$, to temporal duration of the normal frequency level, denoted as $t_{up}$. That is,

$$\beta = t_{down}/t_{up} \tag{6}$$

As shown in Fig. 11, when the transmitter core is running at its typical frequency level (*e.g.*, 2.5GHz), the temperature signal may see a sharp rise and fall when encoding signal '1'; however, the signal amplitude is much lower than before when the frequency of the transmitter core drops to a lower level (*e.g.*, 500MHz) for a duration of $t_{down}$ and then returns to its typical value for a duration of $t_{up}$. By enforcing the down-up rate, the frequency cycles through low (to 500 MHz in Fig. 11) and high levels (back to 2.5 GHz).

The traffic blocking involves the following major steps.

Step 1. The global manager calculates the duration $t_b(= 1/f_t)$ of one signal cycle. Here $f_t$ is fixed to 500Hz since

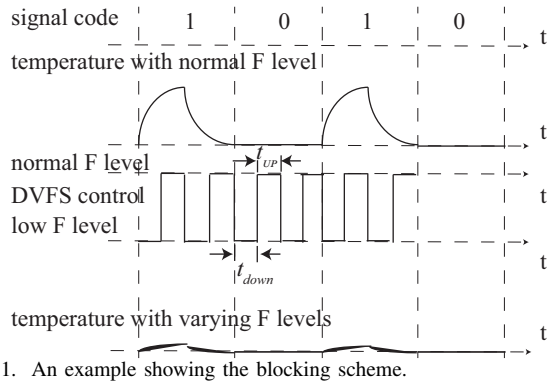



Fig. 11. An example showing the blocking scheme.

the transmitter and receiver may use different channels to communicate.

Step 2. During the period of $0.5t_b$, the global manager first demands the suspicious core to scale down its frequency to a randomly selected low frequency level for a duration of $t_{down}$, after which the frequency of that core will be allowed to restore to its normal level.

Step 3. The global manager always checks whether another TCC channel is present or not for each detection cycle, and also checks whether the supposedly blocked channels are still active or not for every $m$ detection cycles. Here $m$ is set to be an integer randomly selected from 1 to 10. If the global manager finds another TCC channel that is associated with a different core, it gets back to the positioning step while keeping all the currently detected TCC transmission(s) blocked. If the global manager does not find any TCC channels in currently blocked cores, it recovers the frequency levels of those cores to their normal ones to reduce the performance loss.

## IV. Experimental Evaluation

### A. Experimental Setup

Two sets of experiments are performed on respective 2D and 3D many-core processors. Mostly, these experiments are performed using a many-core simulator, Sniper-v7.2 [29], with McPAT-v1.0 integrated as the power model. We also adopt the Hotspot-v6.0 thermal model to dynamically generate temperatures for all the cores. We choose a few benchmarks from PARSEC and SPLASH-2 and they will be treated as the thermal noise to a TCC. The accuracy of simulation temperatures is set to be 0.12°C [14] and 1°C for simulations in both 2D and 3D many-core systems, respectively. A resolution of 0.12°C means that the temperature output from the Hotspot model has a precision of 0.12°C. The detailed configuration of Sniper, Hotspot, and TCC programs are tabulated in Table II.

*1) Experimental Configurations of the 2D Many-core System:* The channels include inter-core and intra-core channels. As for an inter-core channel, two physically separated cores are paired together as part of a TCC, while all the other cores are running legitimate threads from the selected benchmarks. Specifically, each physical core runs two simultaneous multithreading (SMT) threads, and both the transmitter core and the receiver core run a TCC thread and a thread spawned by the benchmarks. As for an intra-core channel, the two logical

TABLE II
SIMULATION CONFIGURATION

| Sniper configuration | |
|---|---|
| Instruction set architecture | x86-64 |
| 2D Network topology | Mesh |
| 3D Network topology | 3D mesh, with each layer has its own 2D mesh and the layers are connected vertically through Through-Silicon-Vias (TSV) [31]. |
| Number of cores (2D) | 3×3 / 4×4 / 8×8 |
| Number of cores (3D) | 3×3×3 / 4×4×3 / 8×8×3 |
| Number of SMT threads per core | 2 |
| Technology node (nm) | 22 |
| Frequency and voltage levels of CPUs (MHz/V) | 2500/1.0, 1000/0.7, 800/0.68, 700/0.66, 600/0.64, 500/0.6, 400/0.5 |
| Benckmarks of PARSEC | Blackscholes, Canneal, Fluidanimate, Streamcluster, Swaptions, X-264, Dedup, Freqmine. |
| Benckmarks of SPLASH-2 | Raytrace, Barnes. |
| Configuration for an integrated Hotspot | |
| Chip thickness | $0.15mm$ |
| Silicon thermal conductivity | $100W/(m \cdot K)$ |
| Silicon specific heat capacity | $1.75 \times 10^6 J/(m^3 \cdot K)$ |
| Heat sink side | $0.06m$ |
| Heat sink thickness | $6.9mm$ |
| Heat sink thermal conductivity | $400W/(m \cdot K)$ |
| Specific heat capacity of heat sink | $3.55 \times 10^6 J/(m^3 \cdot K)$ |
| Configuration for TCC programs | |
| Transmission frequency | 10Hz, 20Hz, 50Hz, 80Hz, 100 Hz, 150 Hz, *etc.* |
| Preamble of a packet | 101010. |
| Packet size in bits | 64. |
| Distance between a transmitter and a receiver | intra-core/1 hop. |
| Bandpass filter using by the receiver | window-based FIR (finite impulse response) filter. |
| bandwidth of bandpass filter | 4Hz. |

cores in the same physical core are able to run transmitter and receiver programs.

*2) Experimental Scenarios of the 2D Many-core System:* The TCC communications are set to be point to point. For each experiment, packets are transmitted randomly for 1000 times and the result is then averaged. Effectiveness of a TCC attack is measured in terms of the packet error rate (PER), as opposed to bit error rate (BER). The PER is defined as follows.

$$PER = N_e/N \times 100\% \quad (7)$$

where $N$ is the total number of packets transmitted, and $N_e$ is the number of packets failed to be correctly recognized. The experiments performed on the 2D many-core system include the following scenarios:

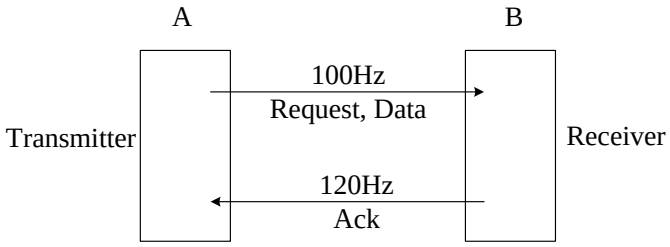



Fig. 12. An example showing the transmitter and receiver communicate using two different channels. *A* is the transmitter and *B* is the receiver.

- Measuring the PER's of a TCC communication under different thermal noise environments in a 4×4 many-core system.
- Measuring the PER's of a TCC communication under different system sizes.
- Measuring the PER's of a TCC communication without applying DVFS control to the transmitter core.
- Measuring the PER's of TCC communication when applying DVFS control to the transmitter core.
- Evaluating the performance loss of legitimate applications when exploiting DVFS control to the transmitter core.

For all experiments below, the transmitter and receiver of each TCC can communicate with each other in two different channels. As shown in Fig. 12, *A* is assumed the transmitter in a secured zone and *B* is the receiver in an unsecured zone. Note that both the transmitter and receiver can transmit information to each other according to the communication protocol, and each time we can only detect a thread (logical core) that is transmitting data. When *A* transmits data to *B*, *A* uses a transmission frequency of 100Hz, and *B* extracts data at the frequency of 100Hz. When *B* transmits data to *A* (*e.g.,* applies ACK or resend request to *B*), *B* uses a transmission frequency of 120Hz, and *A* extracts data at the frequency of 120Hz. Note that once the transmitter and receiver use two or more channels to communicate, our previous work [3] that positioning two cores by finding the same channel can not detect such TCC attacks.

*3) Measurements on the Real Machine:* The machine adopted in this study is a quad-core eight-thread Intel Core i7-7700HQ processor clocked at 2.8GHz. We fix the fan speed to the maximum and let other cores sleep, and only the transmitter core and receiver core are active like the work in [12]. The PER's of a TCC communication before and after applying DVFS control to the transmitter core are measured.

*4) Experimental Configurations of the 3D Many-core System:* We will evaluate both inter-core channels and intra-core channels of the 3D many-core system where its floorplan follows the one used in [30]. As for an inter-core channel, the receiver core is right below the transmitter core. The other configurations are set to be the same as those of the 2D many-core case. In a 3D many-core system, the vertical layers are connected by the TSV's (Through-Silicon-Vias). Note that the thermal correlation in the vertical direction is higher than that in the horizontal, thus, the vertically placed inter-core channels are more efficient than those inter-core channels in a 2D many-core system.

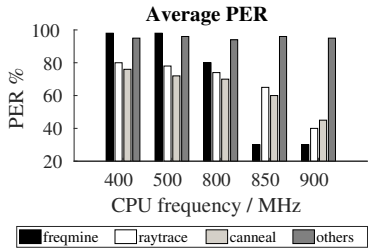



Fig. 13. The PER's of the 2D 1-hop channels running at different frequency levels under different thermal noise environments in a 4×4 many-core system. There are four cases: "raytrace" means the benchmark Raytrace from SPLASH-2 is running in the system, and the heat generated by running the threads of Raytrace is thermal noise to a TCC. 'freqmine' and 'canneal' mean the benchmarks Freqmine and Canneal from PARSEC are running in the system, respectively. 'others' include the benchmarks Barnes, Blackscholes, X-264, Fluidanimate, Swaptions, and Dedup.

### B. Selecting the Parameters for Defense

In order to balance between the security requirements (measured by the PER of a TCC communication) and system performance, the parameters of the defense scheme need to be selected first in the experiment.

Fig. 13 shows the simulation results of average PER's of the 1-hop channels as the transmitter core runs at different frequencies. Note that when the CPU frequency of the transmitter core is 900MHz or lower, all the PER's of the 2D 1-hop channels under thermal noise generated by running the benchmarks listed as 'others' in Fig. 13 are 95%±1%. One can see that when the frequency level of a transmitter core drops to 800MHz or even lower, in all cases, the TCC communication sees a high PER (*e.g.*, more than 70%) for the 1-hop channels. In addition, when transmitting a single bit, if we randomly choose the frequency level of the transmitter core to be 500MHz, 600MHz, 700MHz, and 800MHz, the average PER of the intra-core channels is significantly higher than 70%. In most cases, the PER reaches 95%. As for the real machine, since the allowed frequency spans from 800MHz to 2800MHz, we set the low frequency list to include 800MHz, 900MHz, and 1100MHz. The PER's of a TCC in a real machine, in all these cases, are higher than 60%. Therefore, in the following experiments, we include 500MHz, 600MHz, 700MHz, and 800MHz into the low frequency list (vs. the normal frequnency of 2500MHz) to defend against both inter-core and intra-core channels.

Once a TCC attack is confirmed, the CPU frequencies of the transmitter core and the reeiver core can be randomly selected from the low frequencies list. However, if we keep the frequency level of the CPU core at a low level all the time, other legitimate applications running on this same physical core will also be adversely affected, and the performance loss ($PL$) is defined as:

$$PL = \frac{\pi_{avg} - \pi_0}{\pi_{avg}} \times 100\% \tag{8}$$

where $\pi_{avg}$ is the average performance (IPC, instruction per cycle) of an application when no DVFS control is applied, and $\pi_0$ is the average performance of an application when DVFS control is in effect.

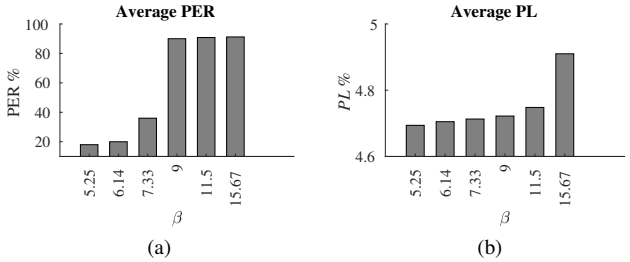



Fig. 14. (a) The average PER of TCC communication with respect to $\beta$. (b) The average performance loss ($PL$) of a normal application with respect to $\beta$. The TCC programs coexist with a mixture of 4 multi-threaded benchmarks serving as the noise source in a 4×4 many-core system.

To help balance the security requirements and performance, the value of $\beta$ needs to be properly selected. Fig. 14(a)-(b) show the PER of the TCC communication and the average performance loss with respect to different values of $\beta$. From Fig. 14(a), one can see that the average PER increases as $\beta$ increases. It remains below 36% until $\beta$ reaches 7.33. While $\beta$ is set to 9, PER soars to 90%. As shown in Fig. 14(b), the performance loss also increases as $\beta$ increases. In all the following experiments, $\beta$ is set to be 9 and $t_{up}$ is limited to 5% of one signal duty cycle.

### C. Experimental Results

*1) Results of TCC attacks:* The PER's of TCC are measured when there are two TCC logical cores in the system with all the other logical cores running legitimate applications from the benchmarks PARSEC and SPLASH-2 as the background noise to the TCC. Table III shows both simulations and real machine's results.

As for the simulations, the 1-hop channels are working with a transmission frequency of 100Hz and CPU frequency level of 2500MHz in a 4×4 many-core system. From Table III, one can see that the PER of the TCC receiver of the 2D 1-hop channels are below 1% for most situations. The reason why the 1-hop channels suffer a high PER (*i.e.,* 25%) with the thermal noise from PARSEC-Streamcluster is that the application temperature has frequency components in band *B*.

As for the TCC's in real machine, the average PER's of an intra-core channel and a 1-hop channel are below 5% with the transmission frequeny of 100Hz and 15Hz, respectively. Note that an intra-core channel does not need to transfer heat between two neighboring cores, and the transmission frequency of an intra-core channel can go much higher than that of inter-core channels, with the sensor resolution of 1°C.

From Fig. 15, one can see that the average PER's of TCC communications in all 2D and 3D many-core cases are below 7%, as long as the DVFS control is not applied. With such low PER, the TCC certainly poses a serious threat to any system.

*2) Evaluation of the Detection Scheme:* The detection step actually needs to identify whether there is a TCC attack or not, and if one attack is found, the transmitters associated with the TCC need to be positioned. The positioning accuracy is denoted as $P_{acc}$.

$$P_{acc} = \begin{cases} 1 & P_{detected} = P_{transmitter} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

TABLE III
PER'S OF TCC COMMUNICATION

| PER's of 1-hop TCC's in Simulations | | | |
|---|---|---|---|
| Benchmarks | PER | Benchmarks | PER |
| Barnes | 0.24% | Raytrace % | 0.2% |
| Freqmine | 0.12% | Blackschole | 0.15% |
| Canneal | 0.5% | Fluidanamite% | 0.32% |
| Swaptions | 0.71% | X264 | 1.25% |
| Dedup | 3% | Streamcluster% | 25% |

| PER's of TCC in real machine | | | |
|---|---|---|---|
| TCC types | PER | TCC types | PER |
| Intra-core | 3% | 1-hop | 5% |

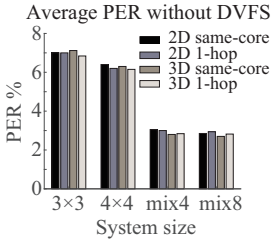



Fig. 15. The average PER's of a TCC communication under different system configurations and scenarios. Mix4 (Mix8) is the case that the thermal noise seen by a TCC is from a mixture of 4 (8) multi-threaded benchmarks. As for the 3D mesh, the 4×4 system size means that each layer consists of 4×4 cores, and so on.

TABLE IV
DETECTION ACCURACY OF $P_{acc}$

| System sizes | $P_{acc}$ |
|---|---|
| 2D 3×3 | 0.98 |
| 2D 4×4 | 0.975 |
| 2D 8×8 | 0.97 |
| 3D 3×3×3 | 0.98 |
| 3D 4×4×3 | 0.972 |
| 3D 8×8×3 | 0.96 |

where $P_{detected}$ is the position (core id) of the detected TCC cores, and $P_{transmitter}$ is the actual position of the transmitter cores.

From Table IV, one can see that the average accuracy of positioning transceiver is around 97%. By using the proposed detection strategy, we can almost always identify a TCC attack, should it ever exist, and correspondingly, the position(s) of the transmitter core(s) can be accurately determined.

*3) Defense Results:* With the proposed countermeasure, once a TCC attack is detected to transmit over a freqeuncy in band $B$, the TCC transmitter core's frequency level is changed with a down-up rate $\beta$ of 9.

As for the real machine (shown in Table. V), with our proposed countermeasure, the average PER's of an intra-core channel (with frequencies ranging from 10Hz to 150Hz) and a 1-hop channel (with frequencies ranging from 10Hz to 20Hz) in the real machine are higher than 70%. Note that with the resolution limitation of thermal sensors and low thermal correlation between cores, the 1-hop TCC's in our real machine can hardly transmit signals with frequency higher than 20Hz, even without applying DVFS.

TABLE V
PER'S IN REAL MACHINE WHEN APPLYING THE PROPOSED COUNTERMEASURE

| TCC types | PER |
|---|---|
| Intra-core | 72% |
| 1-hop | 100% |

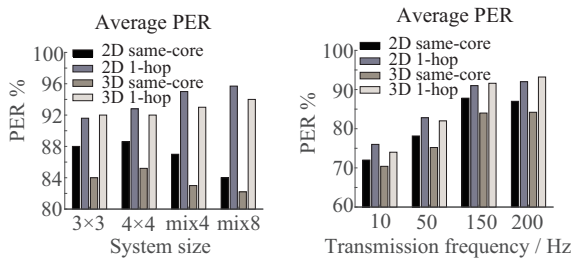



Fig. 16. (a) The average PER results of TCC communication with our proposed countermeasure with different system sizes. The transmission frequency of the TCC is 100Hz. (b) The average PER results of the TCC communication in a 4×4 system with different transmission frequencies within band $B$. The CPU frequency of TCC cores drops to 500MHz before TCC running.

As for the simulation results, from Fig. 16(a), one can see that the average PER's of the TCC attacks with a transmission frequency of 100Hz across all the systems with different sizes is higher than 82%. With such a high PER, no meaningful communications can be sustained; that is, our defense strategy can effectively shut down TCC attacks.

To evaluate how the proposed countermeasure affect the FHSS, we first drop the CPU frequencies of both the transmitter core and receiver core to 500MHz. The transmission frequency of the TCC dynamically changes with a range of 10Hz, 50Hz, 150Hz, and 200Hz. From Fig. 16(b), one can see that once the CPU core is applied our proposed countermeasure, the TCC cannot survive with the transmission frequencies in band $B$, with PER higher than 70% for all the situations. Therefore, we can draw the conclusion that once the TCC cores are detected and blocked by the proposed countermeasure, attempting to change transmission frequencies within band $B$ will not bring in any additional benefit.

*4) Average Performance Loss:* Note that scaling down the CPU frequency of the physical transmitter core will also negatively impact performance of a legitimate logical core. Fortunately, the TCC programs are not active all the time. After finishing transmission, they return to be inactive. Therefore, we apply DVFS countermeasure to block a TCC attack working in band $B$ only when an attack is identified being active. If the global manager does not detect a TCC channel on the same transmitter core next time, the frequency level of the core is tuned back to a normal one. We denote the ratio of the time of TCC being inactive to the time of TCC being active to be $\tau$.

We assume that a TCC thread may share the same physical core with a thread of a legitimate application, and the performance loss of the legitimate application is 0% if we do not apply DVFS to block TCC. When we apply DVFS

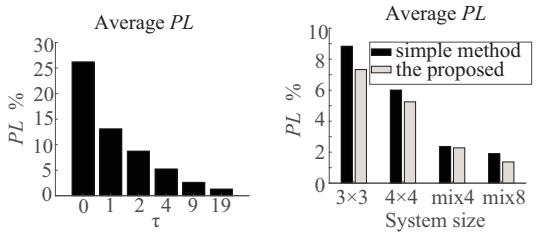



Fig. 17. (a) The average $PL$ results with different $\tau$'s. '0' means that the TCC keeps active all the time. (b) The average $PL$ results with different system sizes with $\tau$ set to 4.

to the physical cores of TCC, the performance loss of that legitimate application is calculated by Eqn. 8. Fig. 17 shows the average performance loss ($PL$) of normal applications, including all the false positives (*i.e.*, the cases that the normal cores are mistakenly treated as TCC cores). Our proposed countermeasure with a down-up rate $\beta$ of 9 ($\beta$=9) is compared against a simple countermeasure that keeps the transmitter core running at fixed voltage and frequency levels once a TCC attack is detected.

From Fig. 17(a), one can see that in a 4×4 many-core system, when $\tau$ increases, the average $PL$ decreases. As TCC stays inactive much longer than being active, the performance loss actually is less than 6% when $\tau$ is greater than 4. From Fig. 17(b), one can see that the average $PL$ result of our proposed countermeasure is always lower than that of the simple countermeasure for different system sizes. The average $PL$ of our prosposed countermeasure in a 3×3 many-core system is below 9%. As the system size or the number of benchmarks in a system increases, the average $PL$ of normal applications decreases since the threads trend to be scheduled to the cores that are not malicious. Therefore, in a large many-core system, the average performance loss of our proposed countermeasure is considered low (*e.g.*, below 3% for an 8×8 system).

From Figs. 17(a) and (b), one can see that the $PL$ of our proposed countermeasure is 12% lower than that of the simple countermeasure. Thus, our proposed countermeasure is better than the simple countermeasure at a modest cost of performance. The performance loss of our proposed countermeasure is quite low, *i.e.,* the $PL$ is below 3% for a large many-core system.

### D. Overhead of the Proposed Countermeasure

As shown in Fig. 9, a detection cycle spans multiple phases: $t_1$, $t_2$, and $t_3$.

During $t_1$ (2 seconds in our experiments), except the logical core functioning as the global manager, all the other cores are allowed to run their own tasks, both normal and potential TCC tasks. Thus, only the global manager thread incurs some overhead. When calculating each core's IPC value, the global manager needs to perform 2 subtract instructions and 1 divide instruction, which consumes a total of 28 clock cycles (based on the clock cycles of arithmetic operations of an architecture reported in [36]). Since the number of temperature traces recorded for 2 seconds with a sampling frequency of 1000Hz is fewer than 2048, for a system with $n_c$ logical cores and clocked at 2GHz (*i.e.,* $2 \times 10^9$ cycles per second), the global manager actually takes $57344 \times n_c$ (*i.e.,* $2048 \times 28 \times n_c$) clock cycles or $28 \times n_c \mu s$ (*i.e.,* $57344/(2 \times 10^9) \times 10^6 \times n_c$) to complete sampling all the cores' IPC values.

During $t_2$, an $N$-point fast Fourier transform needs to peform about $N_{fft}$ complex multiplications [35] where

$$N_{fft} = (N/2) \times \log_2 N \tag{10}$$

We assume each real number multiplication has 20 clock cycles [36] and each complex multiplication involves 4 (*i.e.*, 2 complex number contains 2 real numbers and 2 imaginary numbers) real number multiplications. For a 2048-point FFT calculation, 11264 multiplications of real numbers is needed, which corresponds to 901120 clock cycles, or run time of $0.45ms$ with a core clock running at 2GHz.

During $t_3$, the global manager needs to check whether the address space of each detected thread (thread's private stack address) can be accessed or not, which only includes memory access for at most $n_c$ times, and takes lower than $0.01 \times n_c \mu s$.

Putting all the phases together, one can see that although a detection cycle on average can last 2s, when $n_c \leq 100$, the runtime of the proposed detection accounts for lower than 0.17% (*i.e.*, $(28 \times n_c + 450 + 0.01 \times n_c)/(2 \times 10^6)$) of the execution time of the normal applications.

From McPAT, each core's average power consumption is around 9W and 13.5W during $t_1$ and $t_2$, respectively. When $n_c \leq 100$, the energy consumption overhead of the proposed countermeasure is lower than 3.4% (*i.e.*, $(x/(x+9\times2)$, where $x = 9 \times 28 \times n_c \times 10^{-6} + 13.5 \times 0.45 \times n_c \times 10^{-3}$) of the total energy consumption of the whole system.

Note that calculation of the threshold is typically done offline, following the procedure provided in Appendix. In general, the runtime overhead in terms of cycle count and energy consumption of our proposed countermeasure is fairly modest.

## V. CONCLUSION

In this paper, a three-step detection and countermeasure scheme was proposed to defend against TCC attacks that poses severe security threats to many-core systems. During the first detection step, a frequency scanning method is applied to examine the CPU-workloads to identify any possible TCC attack. In the following positioning step, the TCC logical cores are located. In the final traffic blocking step, a TCC attack can be blocked by applying DVFS to the cores participating in the TCC. Experimental results have confirmed that the proposed countermeasures could practically shut down TCC attacks by forcing their transmissions to undergo extremely high PER (>70%) at a very modest performance loss (< 3% in an 8×8 many-core system). With its low complexity and overhead, the proposed countermeasure is a suitable scheme that can be adopted by many-core systems to fight against TCC attacks.

## REFERENCES

[1] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms", in *Proc. USENIX Security Symp.*, 2015, pp. 865–880.

[2] Z. Long, X. Wang, Y. Jiang, G. Cui, L. Zhang, and T. S. T. Mak, "Improving the efficiency of thermal covert channels in multi-/many-core systems", in *Proc. Design, Automation & Test in Europe Conf. & Exhibition*, 2018, pp. 1459–1464.
[3] H. Huang, X. Wang, Y. Jiang, A. K. Singh, M. Yang and L. Huang. "On countermeasures against the thermal covert channel attacks targeting many-core systems", in *Proc. Design Automation Conf.*, 2020.
[4] ARM, "Building a secure system using TrustZone technology", http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.
[5] V. Costan and S. Devadas, "Intel SGX explained", *Cryptology ePrint Archive*, vol. 2016, pp. 86, 2016.
[6] B. Lapid and A. Wool, "Cache-attacks on the ARM TrustZone implementations of AES-256 and AES-256-GCM via GPU-based analysis", *Cryptology ePrint Archive*, vol. 2018, pp. 621, 2018.
[7] Y. Wu, W. Zheng, B. Mao and X. Wu, "Leaks or not: a framework for evaluating cache timing side channel attacks in SGX", in *Proc. Smart World, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, 2018, pp. 1467–1470.
[8] P. Qiu, D. Wang, Y. Lyu and G. Qu, "VoltJockey: breaching TrustZone by software-controlled voltage manipulation over multi-core frequencies", in *Proc. Computer and Communications Security Conf.*, 2019, pp. 195–209.
[9] Sining Pan, Cagri Gurleyuk, Matheus F. Pimenta, and Kofi A. A. Makinwa, "A 0.12mm$^2$ wien-bridge temperature sensor with 0.1°C ($3\sigma$) inaccuracy from -40°C to 180°C", in *Proc. IEEE Int'l Solid- State Circuits Conf.*, 2019, pp.184–186.
[10] "8th gen intel core processor family datasheet", https://www.intel.com/content/www/us/en/products/docs/processors/core/8th-gen-core-datasheet-vol-1.html.
[11] "CoreTemp", https://www.alcpu.com/CoreTemp/.
[12] D. B. Bartolini, P. Miedl, and L. Thiele, "On the capacity of thermal covert channels in multicores", in *Proc. European Conf. Computer Systems*, 2016, pp. 24:1–24:16.
[13] R. E. G. Dennis, C. D. K. Nguyen, S. H. Gillani and M. B. Tahoori, "Voltage-based covert channels in multi-tenant FPGAs", *Cryptology ePrint Archive*, vol. 2019, pp. 1394, 2019.
[14] S. Pan and K. A. A. Makinwa, "A 0.25 $mm^2$-resistor-based temperature sensor with an inaccuracy of 0.12 °C ($3\sigma$) from -55 °C to 125 °C", *J. Solid-State Circuits*, vol. 53, no. 12, pp. 3347–3355, 2018.
[15] Z. Wu, X. Zhang, and H. Wang, "Whispers in the hyper-space: high-speed covert channel attacks in the cloud", in *Proc. Usenix Security Symp.*, 2012, pp.159–173.
[16] Y. Xu, M. Bailey, F. Jahanian, K. R. Joshi, M. A. Hiltunen and R. D. Schlichting, "An exploration of L2 cache covert channels in virtualized environments", in *Proc. Cloud Computing Security Workshop*, 2011, pp. 29–40
[17] Z. Wang and R. B. Lee, "Covert and side channels due to processor architecture", in *Proc. Computer Security Applications Conf.*, 2006, pp. 473–482.
[18] L. Deshotels, "Inaudible sound as a covert channel in mobile devices", in *Proc. Workshop on Offensive Technologie*, 2014.
[19] S. Cabuk, C. E. Brodley and C. Shields, "IP covert timing channels: design and detection", in *Proc. ACM Conf. Computer and Communications Security*, 2004, pp. 178–187.
[20] C. Jie and G. Venkataramani, "CC-Hunter: uncovering covert timing channels on shared processor hardware", in *Proc. IEEE/ACM Int'l Symp. Microarchitecture*, 2014, pp. 216–228.
[21] X. Zhang, Y. Tan, C. Liang, Y. Li and J. Li, "A covert channel over VoLTE via adjusting silence periods", *IEEE Access*, vol. 6, pp. 9292–9302, 2018.
[22] M. Guri, O. Hasson, G. Kedma and Y. Elovici, "VisiSploit: an optical covert-channel to leak data through an air-gap", *CoRR*, abs/1607.03946, 2016.
[23] W. Liu, X. Zhou, J. Huo and K. Yan. "Modeling of visible light channel based on matrix reconstruction, in *Proc. Int'l Conf. Wireless & Optical Communications*, 2016.
[24] N. Matyunin, J. Szefer, S. Biedermann and S. Katzenbeisser, "Covert channels using mobile device's magnetic field sensors", in *Proc. Asia & South Pacific Design Automation Conf.*, 2016, pp. 525–532.
[25] M. Guri, M. Monitz and Y. Elovici, "USBee: air-gap covert-channel via electromagnetic emission from USB", in *Proc. Privacy, Security & Trus*, 2016, pp. 264–268.
[26] M. Alagappan, J. Rajendran, M. Doroslovacki and G. Venkataramani, "DFS covert channels on multi-core platforms", in *Proc. IFIP/IEEE Int'l Conf. Very Large Scale Integration*, 2017, pp. 1–6.
[27] E. M. Benhani and L. Bossuet, "DVFS as a security failure of TrustZone-enabled heterogeneous SoC", *CoRR*, abs/1902.08517, 2019.
[28] S. Chen, W. Xiong, Y. Xu, B. Li and J. Szefer, "Thermal covert channels leveraging package-on-package DRAM", in *Proc. IEEE Int'l Conf. Trust, Security and Privacy in Computing and Communications*, 2019, pp. 319–326.
[29] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur and L. Eeckhout, "An evaluation of high-level mechanistic core models", *ACM Trans. Architecture and Code Optimization*, vol. 11, no. 11, pp. 28:1–28:25, 2014.
[30] K. Y. Jheng, C. H. Chao, H. Y. Wang and A. Y. Wu, "Traffic-thermal mutual-coupling co-simulation platform for three-dimensional network-on-chip", in *Proc. IEEE Int'l Symp. VLSI Design Automation and Test*, 2010.
[31] B. Li, X. Wang, A. K. Singh and T. S. T. Mak, "On runtime communication- and thermal-aware application mapping in 3D NoC", in *Proc. IEEE/ACM Int'l Symp. Networks-on-Chip*, 2017, pp. 16:1–16:8.
[32] M. Strasser, C. Pöpper, and S. Capkun, "Efficient uncoordinated FHSS anti-jamming communication", in *Proc. Int'l Symp. Mobile Ad Hoc Netw. Comput.*, 2009, pp. 207-218.
[33] Shanquan Tian, and Jakub Szefer, "Temporal thermal covert channels in cloud FPGAs", in *Proc. Int'l Symp. Field-Programmable Gate Arrays*, 2019, pp. 298-303.
[34] Johann Knechtel, and Ozgur Sinanoglu, "On mitigation of side-channel attacks in 3D ICs: decorrelating thermal patterns from power and activity", in *Proc. Design Automation Conf.*, 2017, pp. 12:1-12:6.
[35] "FFT (fast fourier transform) waveform analysis", https://www.dataq.com/data-acquisition/general-education-tutorials/fft-fast-fourier-transform-waveform-analysis.html.
[36] "80x86 instruction set", http://www.penguin.cz/ literakl/intel/intel.html.
[37] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo, "CLKSCREW: exposing the perils of security-oblivious energy management", in *Proc. Usenix Security Symp.*, 2017, pp. 1057-1074.

## APPENDIX

The threshold $\rho$ needs to be determined for different many-core architectures or system sizes. Herein we provide a generic method for calculating $\rho$.

Step 1. Sample IPC values of the noise (*i.e.*, IPC of the normal applications) and get the average noise amplitude $\rho_l$ .

Step 2. Sample IPC values of TCC and get the maximum signal amplitude $\rho_h$ of TCC.

Step 3. For a threshold $\rho$ ranging from $\rho_l$ to $\rho_h$, apply the proposed detection and countermeasure to get the detection accuracy as well as PER.

Step 4. From the results in Step 3, assign $\rho$ with a value that leads to a detection accuracy higher than 95% and the maximum PER.

Note that based on the signal amplitude, $\rho$ is obtained for each individual core offline.

The procedures to verify whether the selected threshold $\rho$ is higher than the noise or not are given below.

Step 1: Identifying distribution of signal amplitude:

Assume that the signal in the range of 10Hz to 500Hz follows the Gaussian distribution with a probability density function (pdf), $f(x)$,

$$f(x) = a \times e^{-\frac{(x-b)^2}{kc^2}} \tag{11}$$

where $a$ and $k$ are coefficients, $b$ is the mean value of $x$, and $c^2$ is the variance of $x$.

This pdf function in Eqn. (11) is used as a regression model for the signal amplitude. As in the experiment, the R-squared values (coefficients of determination) are found to be close to

1, the assumption that the signal amplitudes over the frequency of interest obeys the Gaussian distribution is well justified.

Step 2: Performing the hypothesis testing:

To prove that our selection of signal amplitude thresholds, $\rho$, is almost always higher than normal signal amplitude, two hypotheses are used:

- $H_0$, the null hypothesis that the expected mean value $\mu$ of signal amplitudes is greater than or equal to an assumed value, $\mu_0$; and
- $H_1$, the alternative hypothesis which is the complement to $H_0$. $\mu_0$ is set to be $\rho$, that is, $\mu_0$=$\rho$.

$$H_0 : \mu \geq \mu_0.$$
$$H_1 : \mu < \mu_0.$$

A significance level $\alpha$ is chosen to be $\alpha$=0.05, and $n$ signal amplitudes, are sampled to form a random variable $X$. Its sample mean $\bar{X}$ and sample variance $s^2$ are given below:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{12}$$

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2 \tag{13}$$

where $X_i$ is the $i$-th sample value of $X$.

Since the amplitudes, $X$, obeys the Gaussian distribution:$X \sim N(\mu, \sigma^2)$, the chosen test statistic $q$ is set to be

$$q = \frac{\bar{X} - \mu_0}{s/\sqrt{n}} \tag{14}$$

Note that $q$ obeys $t$-distribution with $n-1$ degrees of freedom. That is, $q \sim t(n-1)$ when $H_0$ is true, with the sample standard deviation $s$ and $n$ samples.

As the acceptance region of $q$ is given as $q > -t_{\alpha/2}(n-1)$. We have

$$\frac{\bar{X} - \mu_0}{s/\sqrt{n}} > -t_{\alpha/2}(n-1) \tag{15}$$

where $t_\alpha(n)$ is the $\alpha$-th quantile of $t$-distribution and $\alpha$ is the chosen significance level.

$$P\{q > -t_{\alpha/2}(n-1)\} = 1 - \alpha \tag{16}$$

where $P(x)$ is the probability of $x$ and $\alpha$ is the chosen significance level.

Then the confidence interval of $\mu_0$ is given by:

$$[-\infty, \bar{X} + \frac{s}{\sqrt{n}} \times t_{\alpha/2}(n-1)] \tag{17}$$

It is found that $\mu_0$ is far away from the confidence interval defined in Eqn. (17), and thus, we reject hypothesis $H_0$ at the significance level of 0.05. That is, when the signal frequency falls into (10,500Hz], our selection of $\rho$, which is equals to $\mu_0$ mentioned above, is higher than normal signal amplitudes with a probability of 95% at the significance level of of 0.05.

**Hengli Huang** Hengli Huang received his bachelor degree in software engineering from South China University of Technology (SCUT), Guangzhou, China. He is pursuing his master degree in the school of software engineering, SCUT. His research interests include system security and NoC-based systems.

**Xiaohang Wang** Xiaohang Wang received the B. Eng. and Ph. D. degree in communication and electronic engineering from Zhejiang University, in 2006 and 2011, respectively. He is currently an associate professor at South China University of Technology. He was the receipt of PDP 2015 and VLSI-SoC 2014 Best Paper Awards. His research interests include many-core architecture, power efficient architectures, optimal control, and NoC-based systems.

**Yingtao Jiang** Yingtao Jiang received his Ph. D. in Computer Science from the University of Texas at Dallas in 2001. Upon graduation, he immediately joined the Department of Electrical and Computer Engineering (ECE), University of Nevada, Las Vegas, where he was promoted to full professor in 2013, and subsequently served as the ECE Department Chair between 2015 and 2018. Currently, he is the associate dean of the college of engineering at the same university. His research interests include algorithms, computer architectures, VLSI, networking, nanotechnologies, *etc.*

**Amit Kumar Singh** Amit Kumar Singh (M'09) is a Lecturer at University of Essex, UK. He received the B.Tech. degree in Electronics Engineering from Indian Institute of Technology (Indian School of Mines), Dhanbad, India, in 2006, and the Ph. D. degree from the School of Computer Engineering, Nanyang Technological University (NTU), Singapore, in 2013. He was with HCL Technologies, India for year and half until 2008. He has a post-doctoral research experience for over five years at several reputed universities. His current research interests are system level design-time and runtime optimizations of 2D and 3D multi-core systems for performance, energy, temperature, reliability and security. He has published over 80 papers in reputed journals/conferences, and received several best paper awards, *e.g.* ICCES 2017, ISORC 2016 and PDP 2015. He has served on the TPC of prestigious IEEE/ACM conferences DAC, DATE, CASES and CODES+ISSS.

**Mei Yang** Mei Yang received her Ph. D. in Computer Science from the University of Texas at Dallas in Aug. 2003. In Aug. 2004, she joined in the Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, where she was promoted to full professor in 2016. Her research interests include computer architectures, interconnection networks, machine learning, and embedded systems.

**Letian Huang** Letian Huang received the MS and Ph. D. degrees in communication and information system from the University of Electronic Science and Technology of China (UESTC), Chengdu, China in 2009 and 2016, respectively. He is an associate professor with UESTC. His scientific work contains more than 40 publications including book chapters, journal articles and conference papers. His research interests include heterogeneous multi-core system-on-chips, network-on-chips, and mixed signal IC design.