IChannels: Exploiting Current Management Mechanisms to Create Covert Channels in Modern Processors

Jawad Haj-Yahya Jeremie S. Kim A. Giray Yağlıkçı Ivan Puddu
Lois Orosa Juan Gómez Luna Mohammed Alser Onur Mutlu
ETH Zürich

To operate efficiently across a wide range of workloads with varying power requirements, a modern processor applies different current management mechanisms, which briefly throttle instruction execution while they adjust voltage and frequency to accommodate for power-hungry instructions (PHIs) in the instruction stream. Doing so 1) reduces the power consumption of non-PHI instructions in typical workloads and 2) optimizes system voltage regulators' cost and area for the common use case while limiting current consumption when executing PHIs.

However, these mechanisms may compromise a system's confidentiality guarantees. In particular, we observe that multi-level side-effects of throttling mechanisms, due to PHI-related current management mechanisms, can be detected by two different software contexts (i.e., sender and receiver) running on 1) the same hardware thread, 2) co-located Simultaneous Multi-Threading (SMT) threads, and 3) different physical cores.

Based on these new observations on current management mechanisms, we develop a new set of covert channels, IChannels*, and demonstrate them in real modern Intel processors (which span more than* 70% *of the entire client and server processor market). Our analysis shows that IChannels provides more than* 24× *the channel capacity of state-of-the-art power management covert channels. We propose practical and effective mitigations to each covert channel in IChannels by leveraging the insights we gain through a rigorous characterization of real systems.*

1. Introduction

Modern high-performance processors support instructions with widely varying degrees of *computational intensity*, due to different instruction types (e.g., add, multiply, fused-multiply-add, integer, floating point, vector) and data widths (from 32 to 512 bits).[1] This large range of computational intensity across different instructions results in an instruction set with a wide range of power (current) requirements. For example, a power-hungry instruction (PHI) that performs a 256-bit fused-multiply–add (FMA256) can consume approximately 100× the power of a 32-bit register move (MOV32) instruction [32]. This wide range of power requirements feeds into the many design constraints of modern processors (e.g., thermal limits, electrical limits, energy consumption), resulting in current management mechanisms that allow the processor to be energy efficient while dynamically meeting the performance demand.

Current management mechanisms dynamically adjust the processor's voltage and frequency based on the power consumption of the instructions that are being executed while making sure that design constraints are not violated. For example, 256-bit AVX2 instructions [34] exhibit significantly larger supply voltage fluctuations (*di/dt*)[2] compared to lower power instructions (e.g., 64-bit scalar, 128-bit SSE), which can lead to voltage drops. To prevent the voltage from dropping below the minimum operating voltage limit (Vcc_{min}), the processor increases the voltage guardband of the core before running AVX2 instructions [15, 27, 32]. However, changing the voltage/frequency of the processor can take several microseconds [20, 22, 25, 29, 39, 71], whereas a program can change from lower-power instructions to PHIs within a few clock cycles (i.e., a few nanoseconds, *three* orders of magnitude lower latency than that needed to change the voltage). To address this disparity, current management mechanisms throttle the instruction stream to limit the power (current) of the instructions while adjustments to the voltage/frequency are being made. As the throttling side-effects of these mechanisms are observable across different software contexts, they can be used to break the confidentiality guarantees of the whole system. Recent works [57, 91] demonstrate methods for creating covert channels via side-effects that arise due to the two different throttling techniques employed by the current management mechanisms of modern processors: 1) the *voltage emergency avoidance* mechanism [20, 29, 54, 83, 84] and 2) the *voltage and current limit protection* mechanism [10, 20, 29, 32, 46, 49, 68, 74, 81, 107, 114, 116], both of which we explain in Section 2. Unfortunately, these recent works are limited in creating covert channels and proposing countermeasures due to the limited or inaccurate observations they are built on, as we describe in Section 3.

Our **goal** is to develop a thorough understanding of current management mechanisms by rigorously characterizing real modern systems. This allows us to gain several deep insights into how these mechanisms can be abused by attackers, and how to prevent such vulnerabilities. Our experimental characterization yields three new observations about the throttling side-effects of current management mechanisms. These observations enable high capacity covert channels between otherwise isolated *execution contexts*: that is, between different processes, threads, and in general logical partitions of an application with different privilege levels (such as browser tabs). These covert channels can be established even if the communicating execution contexts are located 1) on the same hardware thread, 2) across co-located Simultaneous Multi-Threading (SMT) threads, and 3) across different physical cores. We demonstrate these throttling side-effects in real modern Intel processors, which span more than 70% of the entire client and server processor market [13]. We refer to our three new observations respectively as Multi-Throttling-Thread, Multi-Throttling-SMT, and Multi-Throttling-Cores.

Observation 1: Multi-Throttling-Thread. We find that executing PHIs results in a multi-level reduction of the instructions per cycle (IPC) performance of the core.[3] Effectively, the IPC reduction manifests as a multi-level throttling period

[1] We loosely define computational intensity as the amount of logic and wire resources an instruction would use to perform the computation it specifies.

[2] Supply voltage fluctuations, known as the *di/dt*, occur when the processor demands rapid changes in load current over a relatively small time scale, due to large parasitic inductance in power delivery [20, 29, 32, 54, 65].

[3] Instruction supply rate into the backend of the core gets reduced to one of several discretized levels.

length proportional to the instructions' computational intensity. For example, executing a 256-bit vector bit-wise OR instruction (e.g., VORPD-256) results in a throttling period shorter than executing a 512-bit vector multiplication instruction (e.g., VMULPD-512). A longer throttling period is primarily due to the higher voltage guardband requirement of instructions with higher computational intensity, which requires more time for the voltage regulator to ramp up the voltage level.

Observation 2: Multi-Throttling-SMT. We find that in a processor with Simultaneous Multi-Threading (SMT) [105], when a thread is throttled due to executing PHIs, its co-located hardware thread running on the same physical core, is *also throttled*. We experimentally conclude that this side-effect is due to the core's throttling mechanism halting the execution of all SMT threads for *three-quarters* of the throttling period. The throttling period length is proportional to the computational intensity of the PHI any of the SMT threads executes.

Observation 3: Multi-Throttling-Cores. We find that when two cores execute PHIs at similar times, the throttling periods are exacerbated proportionally to the computational intensity of each PHI across the two cores. We specifically find that this exacerbation of throttling periods occurs when two cores execute PHIs within a few hundred cycles of each other. This side-effect is because the processor power management unit (PMU) waits until the voltage transition of one core completes before starting the voltage transition of the next core. Therefore, one core can infer the computational intensity of instructions that are being executed on another core.

Based on our three new observations on current management mechanisms in modern processors, we develop a set of covert channels, which we call *IChannels*. IChannels consists of covert channels between various execution contexts that communicate via controllable and observable throttling periods. These covert channels vary slightly depending on where the two covertly-communicating software contexts are executing (same thread, same core, or different cores). In particular, the three covert channels work as follows: 1) IChannels exploits the Multi-Throttling-Thread side-effects to establish a covert channel between two execution contexts *within the same hardware thread*. We call this covert channel **IccThreadCovert**. 2) IChannels exploits the Multi-Throttling-SMT side-effects to create a covert channel between two execution contexts running on the same core but *within two different SMT threads*. We call this covert channel **IccSMTcovert**. 3) IChannels exploits Multi-Throttling-Cores to create a covert channel between two execution contexts running on *two different physical cores*. We call this covert channel **IccCoresCovert**.

We demonstrate IChannels on real modern Intel processors and find that IChannels provides 3*Kbps* of covert channel capacity, which is more than 24× the capacity of state-of-the-art power management-based covert channels [5, 57, 59] and 2× the capacity of state-of-the-art PHI-latency-variation-based covert channels [91].

Based on our deep understanding of the architectural techniques used in current management mechanisms to throttle the system when executing PHIs, we propose three practical mitigation mechanisms to protect a system against known covert channels caused by current management mechanisms implemented in modern processors.

This work makes the following **contributions**:

- We rigorously characterize real systems to develop thorough and accurate explanations for variable execution times and frequency changes that happen when running PHIs on modern Intel processors (e.g., Intel Haswell [58], Coffee Lake [113], and Cannon Lake [45]).
- We present IChannels, a new set of covert channels that exploits multi-level throttling mechanisms used by the current management mechanisms in modern processors. These covert channels can be established between two execution contexts running 1) on the same hardware thread, 2) on Simultaneous Multi-Threading (SMT) threads, and 3) across different physical cores.
- We demonstrate that IChannels increases the channel capacity of state-of-the-art covert channels that exploit the variable latency of PHIs by 2× and covert channels that leverage power management mechanisms by more than 24×.
- We propose a set of practical mitigation mechanisms to protect a system against known covert channels caused by current management mechanisms.

2. Background

We provide a brief background into modern processor architectures[4] and their power delivery networks and electrical design limits.

Client Processor Architecture. A high-end client processor is a system-on-chip (SoC) that typically integrates three main domains into a single chip: 1) compute (e.g., CPU cores and graphics engines), 2) IO, and 3) memory system. Figure 1 shows the architecture used in recent Intel processors (such as Coffee Lake [113] and Cannon Lake [45]) with a focus on CPU cores.

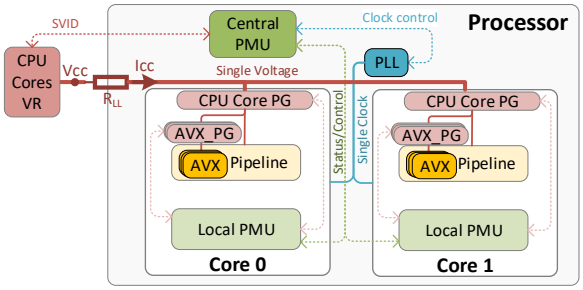

Figure 1: Architecture overview of recent Intel client processors with a focus on CPU cores. All cores share the same voltage regulator (VR) and clock domain. The central power management unit (PMU) controls 1) the VR using an off-chip serial voltage identification (SVID) interface, and 2) the clock phase-locked loop (PLL) using an on-chip interface. Each CPU core has a power-gate (PG) for the entire core. Each AVX unit (e.g., AVX512 [69, 85]) has a separate PG.

Power Delivery Network (PDN). There are three commonly-used PDNs in recent high-end client processors: motherboard voltage regulators (MBVR [14, 37, 52, 87]), integrated voltage regulators (IVR [7, 9, 93, 94, 96, 103]), and low dropout voltage regulators (LDO [10, 48, 75, 100]). We present the MBVR PDN here. Section 7 discusses the other two PDNs. As shown in Figure 1, MBVR PDN of a high-end client processor includes 1) one motherboard voltage regulator (VR) for all CPU cores, 2) a *load-line* resistance (R_{LL}), and 3) power-gates for each entire core and separately for each AVX unit (e.g., AVX512 [69, 85]) inside a CPU core. All CPU cores share the same VR [14, 28,

[4] We present the Intel client processor architecture (e.g., Skylake [6, 33], Kaby Lake [112], Coffee Lake [113], Cannon Lake [45]) to simplify discussions and descriptions. Sections 6.4 and 7 discuss the applicability of our work to Intel server processors and other processors.

37, 52, 87]. Our earlier work [27] provides more background on state-of-the-art PDNs.

Clocking. A phase-locked loop (PLL) supplies the clock signal to all CPU cores. All CPU cores have the same maximum clock frequency[5] [28, 40, 41, 48].

Power Management. The processor includes one central power management unit (PMU) and one local PMU per CPU core. The central PMU is responsible for several power-management activities, such as dynamic voltage and frequency scaling [18, 26, 29, 87]. The central PMU has several interfaces with on-chip and off-chip components, such as 1) the motherboard VR, called serial voltage identification (*SVID*) [20, 29, 88, 90], to control the voltage level of the VR, 2) the PLL to control the clock frequency, and 3) each core's local PMU for communicating power management commands and status reporting. The local PMU inside the CPU core is responsible for core-specific power management, such as clock gating, power gating control, thermal reporting.

Load-line. Load-line or *adaptive voltage positioning* [46, 47, 99, 104] is a model that describes the voltage and current relationship[6] under a given system impedance, denoted by R_{LL}. Figure 2(a) describes a simplified power delivery network (PDN) model with a voltage regulator (VR), load-line (R_{LL}), and load (CPU Cores). R_{LL} is typically 1.6m–2.4$m\Omega$ for recent client processors [41]. The voltage at the load is defined as: $Vcc_{load} = Vcc - R_{LL} \cdot Icc$, where Vcc and Icc are the voltage and current at the VR output, respectively, as shown in Figure 2(b). From this equation, we can observe that the voltage at the load input (Vcc_{load}) decreases when the load's current (Icc) increases. Due to this phenomenon, the PMU increases the input voltage (Vcc), i.e., applies a *voltage guardband*, to a level that keeps the voltage at the load (Vcc_{load}) above the minimum functional voltage (i.e., Vcc_{min}) under even the most intensive load (i.e., when all active cores are running a workload that exercises the highest possible dynamic capacitance (C_{dyn})). This workload is known as a *power-virus* [15, 27, 32] and results in the maximum possible current (Icc_{virus}). A typical application consumes a lower current $Icc_{typical}$ than Icc_{virus}. The minimal current that the processor can consume is the leakage current (Icc_{lkg}) once the clocks are gated (while the supply voltage is *not* power-gated). In all cases where the current is lower than Icc_{virus}, the voltage drop (i.e., $Icc \cdot R_{LL}$) is smaller than when running a power-virus, which results in a higher load voltage Vcc_{load} than necessary (as shown in Figure 2(b)), leading to a power loss that increases quadratically with the voltage level.

Adaptive Voltage Guardband, Icc_{max}, and Vcc_{max}. To reduce the power loss resulting from a high voltage guardband when *not* running a power-virus, due to the load-line effect, modern processors define *multiple levels* of power-viruses depending on the maximum dynamic capacitance (C_{dyn}) that an architectural state (e.g., number of active cores, the computational intensity of running instructions) can draw. For each power-virus level, the processor applies a *different* voltage guardband. Figure 2(c) illustrates the load-line model behavior of a processor with three power-virus levels denoted by $VirusLevel_1$, $VirusLevel_2$, and $VirusLevel_3$ (where $VirusLevel_1 < VirusLevel_2 < VirusLevel_3$). The three power-virus levels represent multiple scenarios. For example, $VirusLevel_1$, $VirusLevel_2$, and $VirusLevel_3$ can represent one, two, or four active cores, respectively, for a processor with four cores. When the processor moves from one power-virus level to a higher/lower level, the processor increases/decreases the voltage by a voltage guardband (ΔV). For example, when moving from $VirusLevel_1$ to $VirusLevel_2$, the processor increases the voltage by ΔV_2 as shown (in blue text) in Figure 2(c).

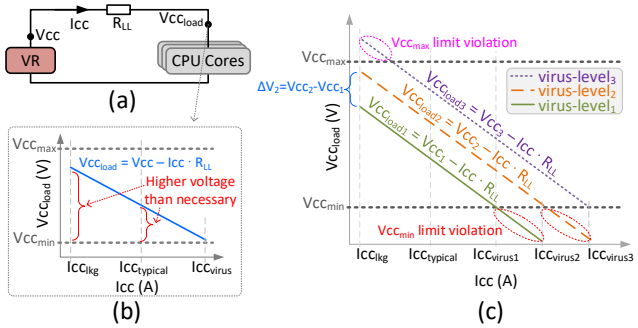

Figure 2: Adaptive voltage guardband on modern processors. (a) Simplified Power Delivery Network (PDN) model with a load-line. (b) Voltage at the load is defined as: $Vcc_{load} = Vcc - R_{LL} \cdot Icc$, where Vcc and Icc are the voltage and current at the VR output, respectively. (c) Multi-level load-line with three power-virus levels. The voltage guardband is adjusted based on the power-virus level corresponding to the architectural state of the processor (e.g., number of active cores and instructions' computational intensity).

Voltage and Current Limit Protection. When dynamically increasing the voltage guardband due to moving to a higher power-virus level, the processor may reduce the cores' frequency 1) to keep the voltage within the maximum operational voltage (i.e., Vcc_{max} in Figure 2(c), pink dashed ellipse), and 2) to keep the current (Icc in Figure 2(b)) consumed from the VR within the maximum current limit (Icc_{max}[7]) [10, 20, 29, 46, 49, 74, 81, 114, 116].

Voltage Emergency (di/dt) Avoidance. When dynamically moving to a higher power-virus level, the processor increases the cores' voltage guardband to prevent the voltage from dropping below the minimum operating voltage (i.e., Vcc_{min} in Figure 2(c), red dashed ellipses) due to high *di/dt* voltage fluctuations [20, 29, 54, 83, 84].

Parameters Affecting the Voltage Guardband. When moving from the power-virus level, $VirusLevel_1$ to a higher level, $VirusLevel_2$, the voltage guardband (ΔV), that should be added to the supply voltage level corresponding to $VirusLevel_1$ (Vcc_1), is correlated with the difference in current consumption between the two levels, ($Icc_2 - Icc_1$). As shown in Equation 1, ΔV is proportional to 1) the CPU core frequency, F, 2) the supply voltage level corresponding to $VirusLevel_1$, Vcc_1, 3) the load-line impedance, R_{LL}, and 4) the difference between core dynamic capacitances of the two power-virus levels, $C_{dyn2} - C_{dyn1}$.

$$\begin{aligned} \Delta V &= Vcc_2 - Vcc_1 \approx (Icc_2 - Icc_1) \cdot R_{LL} \\ &= (C_{dyn2} \cdot Vcc_1 \cdot F - C_{dyn1} \cdot Vcc_1 \cdot F) \cdot R_{LL} \\ &= (C_{dyn2} - C_{dyn1}) \cdot Vcc_1 \cdot F \cdot R_{LL} \end{aligned} \quad (1)$$

The dynamic capacitance of the CPU cores at a given point in time is correlated with the 1) number of *active cores*, 2) the

[5] Intel client processors, including Haswell and Ice Lake processors, which use fully-integrated voltage regulator (FIVR) power delivery, have the same clock frequency domain for all cores [48, 58].

[6] In this model, short current bursts result in voltage droops [11, 83, 84], which are filtered out by the decoupling capacitors [80], while long current bursts are detected by the motherboard VR.

[7] Icc_{max} limit is the maximum current the VR must be electrically designed to support. Exceeding the Icc_{max} limit can result in irreversible damage to the VR or the processor chip [20, 50].

capacitance, *C*, of each core, and 3) the node *activity factor* of each core. The capacitance, C, of a core is a function of the computational intensity (e.g., MOV, OR, ADD, MULL, FMA) and width (e.g., 64-bit scalar, 128-bit vector, 256-bit vector, 512-bit vector) of the instructions executed by the cores.

Power Gating. Power gating is a circuit-level technique to eliminate leakage power of an idle circuit [20, 29, 38]. Typically, the wake-up latency from the power-gated state can take a few cycles to tens of cycles [20, 56]. However, to reduce the worst-case peak current and voltage noise on the power delivery (e.g., di/dt noise [20, 29, 65]) when waking up a power-gate, the power-gate controller applies a *staggered* wake up technique [2] that takes tens of nanoseconds (typically, 10–20*ns*) [4, 55, 56].

3. Limitations of Prior Work

We describe two state-of-the-art covert channels, NetSpectre [91] and TurboCC [57], and their major limitations based on our rigorous characterization (which is provided in Section 5).

Schwarz *et al.* propose NetSpectre [91], a mechanism that builds a covert channel by exploiting the variation in AVX2 [34] instructions' execution times due to throttling mechanisms. We observe that NetSpectre has three main limitations. First, the NetSpectre covert channel can be established *only* between two execution contexts on the *same hardware thread*, whereas our rigorous characterization demonstrates that the throttling side-effect can be observed across both threads of different cores (Section 5.5) and Simultaneous Multi-Threading (SMT) threads (Section 5.6). Therefore, NetSpectre has a narrower and more limited attack vector compared to our work. Second, NetSpectre uses a single-level throttling side-effect (i.e., whether or not the thread is throttled) to communicate confidential data. Our rigorous characterization shows that the throttling side-effect can result in multi-level (up to five) throttling periods based on the power requirements of the PHIs that are being executed (see Sections 5.5 and 5.6). IChannels utilizes the multi-level throttling periods to transmit two bits per communication transaction while NetSpectre sends only a single bit per transaction. Therefore, NetSpectre can use only *half* of the covert channel bandwidth provided by the throttling side-effect of current management mechanisms. Third, NetSpectre does *not* identify the true source of throttling. As a result, NetSpectre does not propose any mitigation techniques. NetSpectre hypothesizes that the source of throttling is the power-gating of the AVX2 execution unit, while we demonstrate, in Section 5.4, that ∼99% of the throttling period is due to voltage transitions initiated by current management mechanisms. Thus, we propose, in Section 7, practical and effective mitigation techniques.

Kalmbach *et al.* [57] propose TurboCC, a mechanism that creates cross-core covert channels by exploiting the core frequency throttling when executing PHIs (e.g., AVX2) at Turbo frequencies [89, 90]. We identify two main limitations in TurboCC. First, TurboCC focuses *only* on the slow side-effect of frequency throttling that happens when executing PHIs at *Turbo* frequencies, which takes tens of milliseconds. In our work, we observe and utilize a cross-SMT-thread and cross-core immediate side-effect (that occurs in tens of microseconds, i.e., three orders of magnitude faster than the frequency changes TurboCC relies on) that happens at *any* frequency (not only Turbo frequencies), as we discuss in Sections 5.5 and 5.6. Second, TurboCC does not uncover the real reason behind the vulnerability. The authors hypothesize that the side-effect of frequency throttling is due to thermal management mechanisms [8, 87–89, 95], while we observe that this side-effect is due to current management mechanisms [10, 20, 29, 46, 49, 74, 81, 114, 116] that also exist in thermally-unconstrained systems or when thermals are not a problem during execution in a system (Section 5.3).

To our knowledge, no previous work 1) provides and analyzes the fundamental reasons why each type of throttling occurs when executing PHIs (Section 5); 2) comprehensively exploits the multi-level throttling side-effects of the current management mechanisms within a thread, across SMT, and across cores (Section 4); and 3) proposes practical and effective mitigation techniques (Section 7).

4. IChannels Overview

Attacker Model. We assume a standard threat model for a covert channel attack [5, 57, 59, 63, 72, 91, 108, 115]. Such a threat model consists of two malicious user-level attacker applications, sender and receiver, that *cannot* communicate legitimately through overt channels. The sender has legitimate access to sensitive information (such as a secret key). However, the sender does not have access to any overt channel for data communication (e.g., system calls or inter-process communication). On the other hand, the receiver has access to an overt channel with the attacker, but does not have access to sensitive information. The common goal of the sender and the receiver is to exfiltrate sensitive data despite the lack of an overt communication channel between them.

Attack. Based on our characterization of the throttling side-effects of the current management mechanisms (Section 5), we build three high-throughput covert channels. These covert channels exploit *three* side-effects (on the same hardware thread, across SMT threads, and across cores) of core execution throttling mechanisms on modern Intel processors when executing power-hungry instructions (PHIs) at any core frequency. We first briefly discuss these key side-effects in the context of IChannels covert channels and later present our full real system characterization results in Section 5.

Figure 3 demonstrates the *common pseudo-code* of the three covert channels. `Sender` code can send two bits of a secret (`send_bits[i+1:i]`) per communication transaction. The `Sender` executes a PHI in a loop (e.g., a few thousand loop iterations). The PHI's computational intensity level depends on the value of the two secret bits. In particular, there are four computational intensity levels (L1–L4) corresponding to four instruction types, i.e., `128b_Heavy`, `256b_Light`, `256b_Heavy`, and `512b_Heavy`. We define *Heavy* instructions to include any instruction that requires the floating-point unit (e.g., ADDPD, SUBPS) or any multiplication instruction, while *light* instructions include all other instructions (e.g., non-multiplication integer arithmetic, logic, shuffle and blend instructions).

The `Receiver` can then read the secret data by simply executing one of three instruction types in a loop (e.g., a few thousand loop iterations) while measuring its own *throttling period* (`TP`) using the `rdtsc` instruction. Depending on the `Sender` and `Receiver` locations (e.g., same hardware thread, across SMT threads, or across cores) the `Receiver` executes one of three instruction types (i.e., `512b_Heavy`, `64b`, or `128b_Heavy`). The `Receiver` decodes the transmitted bits by the `Sender` based on the `TP` range it measures. Since the `TP` range is predictably dependent on the instruction executed by the `Sender`, the `Receiver` can infer the instruction executed by the `Sender`.

Sender

```
case (send_bits[i+1:i])
  00: 128b_Heavy loop()  //L4
  01: 256b_Light_loop()  //L3
  10: 256b_Heavy_loop()  //L2
  11: 512b_Heavy_loop()  //L1
```

Receiver

```
start = rdtsc
if (same-thread) 512b_Heavy_loop()
if (across-SMT)  64b_loop()
if (across-cores) 128b_Heavy_loop()
TP = rdtsc - start
case ( TP )
  L4_range: received_bits[1:0] = 00
  L3_range: received_bits[1:0] = 01
  L2_range: received_bits[1:0] = 10
  L1_range: received_bits[1:0] = 11
```

Figure 3: Common pseudo-code of two execution contexts (`Sender` and `Receiver`) that can be located on the same hardware thread, across SMT threads, and across cores, communicating via IccThreadCovert, IccSMTcovert, and IccCoresCovert, respectively.

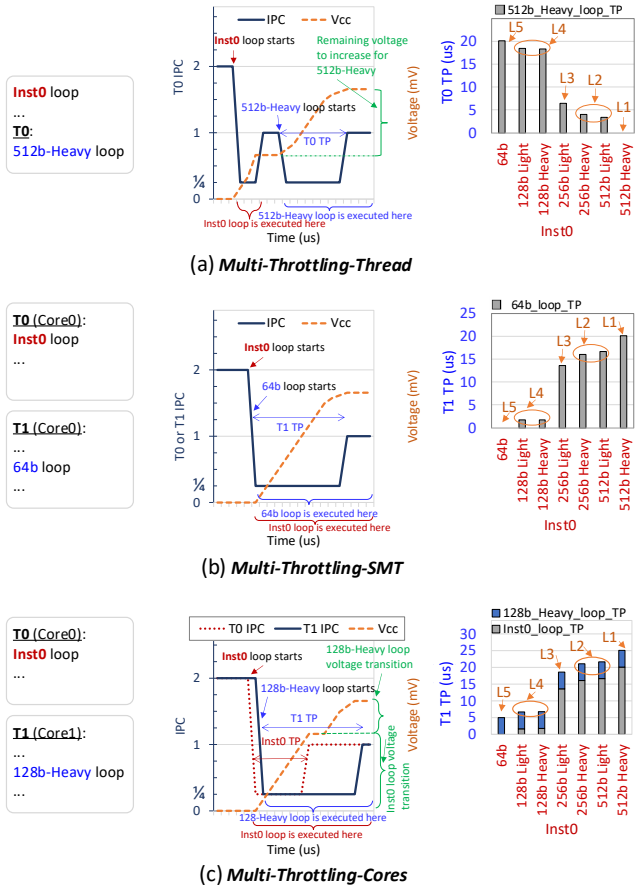

Figure 4: Illustrating the three throttling side-effects: (a) In Multi-Throttling-Thread, the `Inst0` loop increases the voltage based on the power-level required to execute `Inst0` and the subsequent `512b_Heavy` loop, which requires the worst-case voltage, increases the voltage further beyond the level reached by `Inst0` loop. Thus, the throttling period (`TP`) is dependent on the type of `Inst0`. (b) In Multi-Throttling-SMT, when executing the `Inst0` loop on thread `T0`, the IPC of both SMT threads is reduced by 75%, therefore, the `TP` depends on the type of `Inst0`. (c) In Multi-Throttling-Cores, running the `Inst0` loop on `Core0` results in throttling the core and requesting a voltage increase. Subsequent execution of the `128b_Heavy` instruction on `Core1` results in throttling the core and requesting a voltage increase. Since the voltage regulator (VR) is busy with `Core0` voltage transition, `Core1` remains throttled until both cores complete their voltage transitions.

4.1. Covert Channel 1: IccThreadCovert

The first covert channel is IccThreadCovert, which exploits the Multi-Throttling-Thread side-effect to establish a covert channel.

4.1.1. Side-Effect 1: Multi-Throttling-Thread. We find that executing PHIs results in a multi-level (i.e., one of several discretized) throttling period lengths proportional to the computational intensity of the instructions. Observing the variation in instructions per cycle (IPC) during execution allows the `Receiver` to determine the throttling period. Based on this, the `Receiver` can determine the computational intensity of recently executed instructions by the `Sender`. We find that the throttling period is primarily due to and dependent on the higher voltage guardband requirement of instructions with higher computational intensity, which requires more time to ramp up the voltage level of the voltage regulator. For example, executing a 512-bit vector multiplication instruction results in a longer throttling period than executing a 256-bit vector bitwise OR instruction.

We find that executing an instruction with high computational intensity results in a throttling period proportional to the difference in voltage requirements of 1) the currently executing instruction and 2) the previously executed instruction. Figure 4(a) illustrates this. At the beginning, the processor executes scalar instructions with *IPC* = 2. Later `Inst0` loop starts executing (we assume IPC of `Inst0` is 1). `Inst0` loop is throttled (*IPC* = ¼) while ramping the voltage (*Vcc*) to accommodate `Inst0`'s computational intensity. Once the target voltage is reached, the throttling is stopped (*IPC* = 1). When `512b_Heavy` (we assume *IPC* = 1) loop is executed, it is first throttled (*IPC* = ¼) while ramping the voltage (*Vcc*) to accommodate `512b_Heavy`. The throttling period (`T0 TP` in Figure 4(a)) before executing the `512b_Heavy` instructions with full IPC (i.e., *IPC* = 1) is dependent on the computational intensity of `Inst0` loop, since the remaining voltage required to execute a `512b_Heavy` instruction depends on the previous voltage level that was reached when `Inst0` loop was executed. The bar chart on the right part of Figure 4(a) shows how the throttling period (`T0 TP`) changes with the computational intensity of the `Inst0` loop. For instructions (`Inst0`) with lower computational intensity that require lower voltage level to execute, the required voltage increase needed to next execute a `512b_Heavy` loop is larger and hence the `TP` is longer (see L5 and L1 in the bar chart). An observer of the `TP` of the `512b_Heavy` loop can thus infer the computational intensity of the previously executed instructions in the `Inst0` loop.

4.1.2. IccThreadCovert. *IccThreadCovert* exploits the Multi-Throttling-Thread to build a covert channel between two execution contexts, `Sender` and `Receiver`, running on the same hardware thread. Figure 3 demonstrates how `Sender` code can send two bits of a secret (`send_bits[i+1:i]`) per communication transaction. The `Sender` executes PHIs with a computational intensity level (one out of four levels, L1–L4) depending on the value of the two secret bits (as shown in Figure 3). While the current management mechanisms adjust the supply voltage to the core to provide enough power to execute the PHI loop of the `Sender`, the IPC is reduced to ¼th of its baseline value (i.e., *IPC* = 1). This IPC throttling lasts for a length of time proportional to the computational intensity of the PHI that the `Sender` executes.

The `Receiver` can then read the secret data by simply executing a `512b_Heavy` loop (i.e., same-thread in Figure 3)

and measuring its throttling period (TP). The higher the power required by the PHI loop executed by the Sender, the shorter the TP experienced by the Receiver will be, as the voltage supplied to the core was already partially ramped up to execute the PHI loop of the Sender. The Receiver can identify the two bits of secret sent by the Sender based on the length of the TP. The Sender can initiate sending additional data after waiting a certain period of time. We find that a waiting period of approximately 650*us* is required after the last time the thread executes a PHI. We refer to this waiting period as *reset-time*. The need for a waiting period is due to the fact that the processor keeps a hysteresis counter that keeps the voltage at a high level corresponding to the highest power PHI executed within the reset-time frame. If no executed PHIs are within a 650*us* time frame, the processor reduces the voltage to the baseline voltage level. After the reset-time passes, subsequent executions of PHIs result in the three side-effects that we describe in this section (Section 4).

4.2. Covert Channel 2: IccSMTcovert

The second covert channel is IccSMTcovert, which exploits the Multi-Throttling-SMT side-effect.

4.2.1. Side-Effect 2: Multi-Throttling-SMT. We find that, in a processor with Simultaneous Multi-Threading (SMT) [105], when a thread is throttled due to executing PHIs, the co-located (i.e., running on the same physical core but different SMT context) hardware thread is *also throttled*. Depending on the length of the throttling period, a co-located hardware thread can infer the computational intensity of instructions that have recently been executed in the throttled thread. We discover that co-located hardware threads are throttled *together* because the throttling mechanism in the core pipeline blocks the front-end to back-end interface during *three-quarters* of the TP for the *entire core*, as we show in Section 5.6. Figure 4(b) demonstrates how two SMT threads, T0 and T1, are throttled by a single-core executing PHIs. T0 executes the Inst0 loop, while T1 executes a scalar 64b instruction loop. The IPCs of both T0 and T1 drop once the co-located thread, T0, starts executing Inst0 loop. The throttling period depends on the computational intensity of Inst0 (executed by T0), which determines the voltage level to which the processor needs to increase the supply voltage. The higher the voltage level required, the longer the throttling period required (as shown by the bar chart that depicts our measurements for TP).

4.2.2. IccSMTcovert. *IccSMTcovert* exploits the Multi-Throttling-SMT side-effect to provide a covert channel between two SMT threads. Figure 3 shows Sender code that sends two bits of a secret (send_bits[i+1:i]) per communication transaction. The Sender executes a PHI loop with a computational intensity level (L1–L4) depending on the two secret bits' values. This code results in throttling of the entire core (frond-end to back-end interface) in which the IPCs of both SMT threads drop to ¼th of their usual values (i.e., *IPC* = 1). The throttling period (TP) is proportional to the computational intensity of the executed PHI by the Sender, similarly to as described for IccThreadCovert.

The Receiver code in Figure 3 on a co-located hardware thread (i.e., *across-SMT* in Figure 3), executes a scalar loop (denoted by 64b_loop) while measuring the TP of the loop. Based on the TP time, the Receiver determines the value of the two bits sent by the Sender code. The Sender can initiate sending additional two bits after waiting for a *reset-time*, which is the same as the one used for IccThreadCovert.

4.3. Covert Channel 3: IccCoresCovert

The third covert channel is IccCoresCovert, which exploits the Multi-Throttling-Cores side-effect of PHIs.

4.3.1. Side-Effect 3: Multi-Throttling-Cores. We find that when two cores execute PHIs at similar times, the throttling periods are exacerbated proportionally to the computational intensity of each PHI executed in each core. One core can thus infer the computational intensity of instructions being executed on another core. We specifically find that this exacerbation of the throttling period occurs when two cores execute PHIs within a few hundred cycles of each other. This increase in the throttling period is because the processor's central PMU waits until the voltage transition for core A to complete before starting the voltage transition for core B.

Figure 4(c) shows how the throttling period across two threads, T0, and T1, running on two different cores, Core0 and Core1, can affect each other. When T0 initiates the execution of Inst0 loop, and T1 shortly after initiates the execution of a 128b_Heavy PHI loop, the throttling period (TP) of T1 depends on the computational intensity of the instructions executed by the T0 thread.

4.3.2. IccCoresCovert. *IccCoresCovert* exploits the Multi-Throttling-Cores side-effect of PHIs to provide a covert channel between two threads in two different physical CPU cores. Figure 3 shows how a Sender thread can send two bits of secret information (send_bits[i+1:i] per transaction with six steps. 1) Synchronize (not shown in Figure 3) Sender and Receiver threads (see detail below). 2) The Sender thread executes a PHI loop with a computational intensity level (L1–L4) depending on the two secret bits' values. When executing a PHI, the processor throttles the execution and initiates a voltage ramp-up command to the voltage regulator. 3) A Receiver thread executes its PHI loop (i.e., the 128b_Heavy_loop, corresponding to *across-cores* in Figure 3) code while the Sender code is throttled (waiting for the voltage transition to complete). 4) The Receiver continues to be throttled until the voltage transition of the Sender is complete. The duration of the Sender voltage transition depends on the computational intensity level of the PHI the Sender executes. 5) Once the voltage transition of the Sender is complete, the processor starts a voltage transition corresponding to the computational intensity of the PHI loop that the Receiver executes (i.e., 128b_Heavy_loop). This process effectively throttles the Receiver code by a TP time proportional to the computational intensity of the PHI loop executed by the Sender. 6) The Receiver measures its TP using the rdtsc instruction to determine the two bits sent by the Sender, as shown in the Receiver code of Figure 3.

4.3.3. Thread Synchronization. To correctly transfer data between the Sender and the Receiver threads, it is essential to synchronize their operations precisely. One common way to perform this synchronization is by using the wall clock [79], where each thread can obtain the wall clock using rstsc instruction. To accurately synchronize the Sender and Receiver threads, each thread waits (e.g., executes rdtsc in a busy loop at the beginning of its code) for fixed points in time before starting to execute its actual covert-channel code.

5. Throttling Characterization

To understand the true architectural techniques used by a current management mechanism to throttle the system when executing PHIs, we experimentally study three modern Intel processors (i.e., Intel Haswell [58], Coffee Lake [113], and Cannon

Lake [45]). First, we describe our experimental methodology. Second, we study the side-effects of running PHIs on supply voltage, current, and frequency. Third, we investigate whether the real cause of throttling is power-gating as hypothesized in prior work [91]. After eliminating power-gating as a potential cause, we explore whether thread throttling occurs due to micro-architectural throttling techniques implemented inside the core and examine the throttling side-effects. Finally, we reveal a multi-level throttling side-effect between cores due to the shared voltage regulator (VR) between cores.

5.1. Experimental Methodology

We characterize current management mechanisms using real systems with modern Intel processors: Intel Haswell (Core i7-4770K [58], four cores), Coffee Lake (Core i7-9700K [113], eight cores), and Cannon Lake (Core i3-8121U [45], two cores).

Voltage and Current Measurements. We measure the voltage and current of the CPU core while executing PHIs with a National Instruments Data Acquisition (NI-DAQ) card (NI-PCIe-6376 [76]), whose sampling rate reaches up to 3.5 Mega-samples-per-second (MS/s). Differential cables transfer multiple measurement signals from the CPU cores' voltage regulator output wires and sense resistors on the motherboard to the NI-DAQ card in the host computer that collects the power measurements, as depicted in Figure 5. The power measurement accuracy of the NI-PCIe-6376 is 99.94% [76]. For more detail, we refer the reader to the National Instruments manual [76].

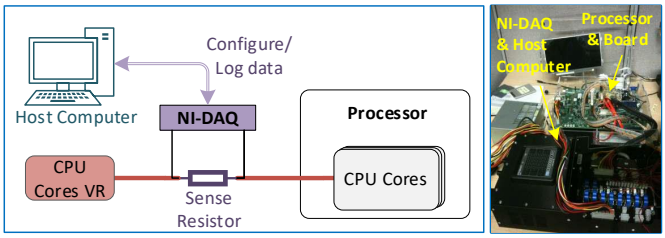

Figure 5: Voltage and current measurement infrastructure using a National Instruments Data Acquisition (NI-DAQ) card. Differential cables transfer multiple measurement signals from the CPU cores' voltage regulator output wires and sense resistors on the motherboard to the NI-DAQ card in the host computer that collects the measurement data.

Performance Counters and Micro-benchmarks. We customize multiple micro-benchmarks of the Agner Fog measurement library [3] and implement Performance Monitoring Counters (PMCs) that we track in our experiments.

5.2. Voltage Emergency Avoidance Mechanism

The voltage emergency avoidance mechanism prevents the voltage from dropping below the minimum operating voltage (i.e., Vcc_{min}) due to high *di/dt* voltage fluctuations, as discussed in Section 2. We study the impact of PHI instructions on the supply voltage of the CPU cores (i.e., *Vcc*). To do so, we track the change in *Vcc* during *two* experiments on a two-core Coffee Lake system (i7-9700K): 1) two CPU cores executing code that includes AVX2 phases, and 2) two CPU cores executing 454.calculix from SPEC CPU2006 [97], compiled with auto-vectorization to AVX2 [44]. In both experiments, the CPU core clock frequency is set to $2GHz$, which is significantly lower than the baseline frequency ($3.6GHz$). Figure 6(a) shows the change in *Vcc* (i.e., *Vcc* delta, shown on the left y-axis), relative to the starting *Vcc*, (i.e., $788mV$), when multiple cores execute code with AVX2 instructions, and Figure 6(b) shows the change in *Vcc* when executing 454.calculix under the same setup.

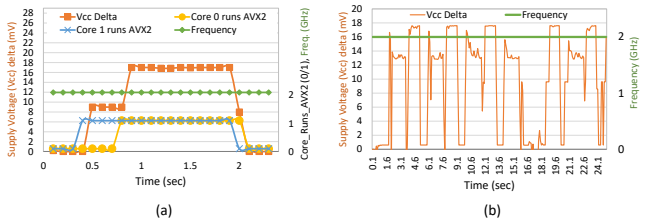

Figure 6: Change in supply voltage (*Vcc*, left y-axis) and cores' frequency (right y-axis) when (a) multiple cores execute code with AVX2 instructions (as shown by Core_Runs_AVX2, on the right y-axis), and (b) executing 454.calculix from SPEC CPU2006 [97] with AVX2 on two cores. Both (a) and (b) run at 2GHz clock frequency, which is significantly lower than the baseline frequency (3.6GHz).

We make five key observations from Figure 6(a). First, when core 1 (blue plot) begins executing AVX2 instructions (*time* = 0.4 s), supply voltage increases by approximately 8 mV. Second, when core 0 (yellow plot) also begins executing AVX2 instructions (*time* = 0.8 s), supply voltage increases by an additional 9 mV. Third, when core 1 stops executing AVX2 instructions (*time* = 2.0 s), supply voltage reduces by $8mV$. Fourth, when core 0 also stops executing AVX2 instructions (*time* = 2.1 s), supply voltage returns to its original value (i.e., $788mV$). Fifth, the clock frequency of the cores (green plot) remains the same throughout the entire execution of code on both cores. This is also reflected in Figure 6(b), where throughout the execution of the 454.calculix workload, the processor only adjusts the supply voltage depending on the code phases (Non-AVX vs. AVX2) of each of the two cores.

We conclude that 1) the processor adjusts the supply voltage proportionally to the number of cores executing PHIs (e.g., AVX2) to prevent voltage emergencies (i.e., di/dt noise) due to the instantaneous high current that PHIs consume (as discussed in Section 2), and 2) core frequency is *not* affected when the processor runs at a low clock frequency relative to its baseline frequency, since such a frequency does not violate the processor power delivery current and voltage limits.

Key Conclusion 1. *A modern processor employs the voltage emergency (i.e., di/dt noise) avoidance mechanism, a current management mechanism that prevents the core voltage from dropping below the minimum operational voltage limit (Vcc_{min}) when executing PHIs.*

5.3. Icc_{max} and Vcc_{max} Limit Protection Mechanisms

In order to demonstrate the impact of executing PHIs (e.g., AVX2) on supply current (i.e., *Icc*) and supply voltage (i.e., *Vcc*), we study two systems: 1) a single-core Coffee Lake (i7-9700K) desktop CPU operating at Turbo frequencies (i.e., $4.9GHz$ and $4.8GHz$), and 2) a two-core Cannon Lake (i3-8121U) mobile CPU operating at Turbo frequencies (i.e., $3.1GHz$ and $2.2GHz$). We run two workloads on each of the two systems with each of the two Turbo frequencies: 1) a loop that runs scalar instructions, denoted by *Non-AVX*, and 2) a loop that runs *AVX2* instructions.

Figure 7(a) shows the *Vcc* (top) and *Icc* (bottom) measurements from each system when running each workload at each Turbo frequency. All values shown in Figure 7(a) are experi-

mentally measured except the bars with green borders, which are projected values.[8]

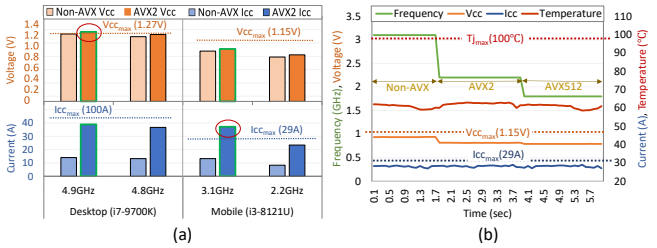

Figure 7: (a) *Vcc* (top), and *Icc* (bottom) when running non-AVX/AVX2 workloads at two Turbo frequencies on desktop/mobile systems. (b) *Vcc*, *Icc*, core junction temperature, and core clock frequency for the mobile system when executing code with three distinct execution phases (Non-AVX, AVX2, and AVX512). Tj_{max} is the maximum allowed core junction temperature.

We make two key observations from Fig 7(a). First, in the desktop system, the supply current consumption when operating at either frequency (4.9 GHz and 4.8 GHz) is below the system limit (Icc_{max} =100 A), while the supply voltage *exceeds* the system limit (Vcc_{max} =1.27 V) when executing AVX2 code at a frequency of 4.9 GHz. In contrast, when the processor operates at 4.8 GHz while running AVX2 code, the supply voltage stays within the Vcc_{max} limit. Second, the supply voltage of the mobile system when running AVX2 code at either frequency (3.1 GHz and 2.2 GHz) is below the system limit (Vcc_{max} =1.15 V), while the supply current *exceeds* the limit (Icc_{max} =29 A) when running AVX2 code at 3.1 GHz. In contrast, when the processor operates at the lower 2.2 GHz frequency while running the same AVX2 code, the supply current stays within the Icc_{max} limit.[9]

Fig 7(b) plots the supply current, supply voltage, core junction temperature, and core clock frequency for the Cannon Lake mobile system when executing code with three distinct execution phases consisting of instructions with different computational intensities: 1) Non-AVX, 2) AVX2, 3) AVX512.

We make three key observations: 1) when the instructions of the current phase require more power (current) than the instructions of a previous phase, the processor reduces its frequency (e.g., *time* = 1.7 s and 4.0 s) to a level that maintains its supply current below the Icc_{max} limit, 2) the voltage is set to a level corresponding to the new frequency based on the voltage/frequency curves, which is significantly lower than Vcc_{max}, and 3) the junction temperature (which is between 58°C and 62°C) is much lower than the maximum allowed junction temperature of the processor, Tj_{max} (100°C).

The Intel architecture provides three Turbo frequency licenses (LVL{0,1,2}_TURBO_LICENSE) that the processor operates at. This depends on the instructions that are being executed and the number of active cores [50]. Kalmbach *et al.* [57] exploit this side-effect of throttling the core's frequency when executing PHIs on multiple cores at Turbo frequency to create a cross-core covert channel. The authors hypothesize that the side-effect is due to *thermal* management mechanisms, while we observe that the side-effect is due to *current* management mechanisms, which affect the system even though temperature is not a problem (as seen in Figure 7(b)).

We make two major conclusions on the core frequency reduction that follows (within tens of microseconds) the execution of PHIs (e.g., AVX2 and AVX512), based on experimental observations from Figure 7. First, this frequency reduction is mainly due to the *maximum instantaneous current limit* (i.e., Icc_{max}) and *maximum voltage limit* (i.e., Vcc_{max}) protection mechanisms, which keep the processor within its maximum current and maximum voltage design limits when executing PHIs. Second, the frequency reduction is *not* caused by immediate thermal events or thermal management mechanisms, which typically take tens of milliseconds to tens of seconds to develop or react after an increase in processor power [8, 87–89, 95].

Key Conclusion 2. *Contrary to the state-of-the-art work's hypothesis [57], we observe, on real processors, that the core frequency reduction that directly follows the execution of PHIs at the maximum (Turbo) frequency is due to the maximum instantaneous current limit (i.e., Icc_{max}) and maximum voltage limit (i.e., Vcc_{max}) protection mechanisms, and not due to thermal events or thermal management mechanisms.*

5.4. AVX Throttling is Not Due to Power Gating

We next measure the throttling period of AVX2 instructions on three generations of Intel processors (i.e., Haswell, Coffee Lake, and Cannon Lake) to analyze the impact of the AVX unit's power-gating on the throttling time of AVX instructions. We plot the distribution of the throttling period for each processor in Fig 8(a). We observe that the throttling side-effect occurs even in the Haswell processor[10] (Intel's fourth generation client processor), *although* Intel reports that AVX power-gating is a *new* feature introduced in the later Skylake processor (Intel's sixth generation client processor) [69] to reduce core leakage power when the AVX unit is *not* in use. Furthermore, as discussed in Section 2, prior works [4, 55, 56] have shown that opening a power-gate takes up to tens of nanoseconds, whereas the throttling period lasts several microseconds. Therefore, we *hypothesize* that the source of throttling is *not* the newly-added power-gating mechanism in the Skylake processor.

To support our hypothesis, we want to find the time it takes to open the AVX2 power-gate by comparing the execution time of multiple subsequent iterations of AVX2 instructions. We track the core clock cycles (CPU_CLK_UNHALTED) on the Coffee Lake and Haswell processors when running AVX2 instructions in a loop (consisting of 300 *VMULPD* instructions that use registers) with a 3*GHz* core clock frequency. Figures 8(b) and (c) plot the deltas of the execution time (measured with the CPU_CLK_UNHALTED performance counter) of the first iteration of the loop (in which the power-gate is being opened) and subsequent iterations (in which the power-gate is already opened) against the expected execution times (based on operating frequency and instructions' IPC) on Coffee Lake and Haswell processors, respectively, for the first three iterations (all remaining iterations are similar to the third iteration). All iterations run under the throttling side-effect, i.e., at a *quarter* of the expected IPC.

We observe that the first iteration of the loop running on Coffee Lake (shown in Figure 8(b)) is more than 8*ns* longer than the other two iterations, while for the Haswell processor (shown in Figure 8(c)), all iterations have nearly the same latency. We

[8] We extrapolate the current and voltage based on our real system measurements described in Section 5.1.

[9] Intel allows exceeding Icc_{max} and Vcc_{max} limits when overclocking a system (e.g., via the BIOS [42] or the XTU tool [51]). This process is out of the processor's specification and can shorten the processor's lifetime [42, 51].

[10] The Haswell processor uses a faster voltage regulator (FIVR [10]) than Coffee Lake and Cannon Lake (MBVR [30]), and therefore has a shorter throttling period.

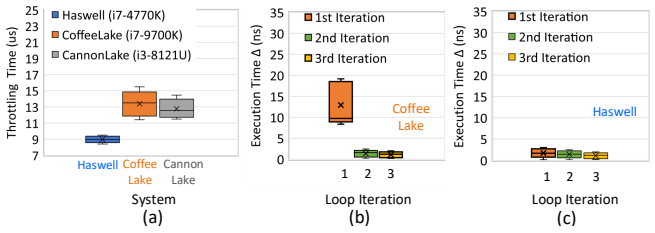

Figure 8: (a) Distribution of throttling period for Haswell, Coffee Lake, and Cannon Lake client processors. (b) and (c) show the execution time delta from the expected execution time of AVX2 loop iterations on the Coffee Lake and Haswell processor, respectively.

conclude that the AVX power-gating feature, implemented in all processors since Skylake, has approximately 8–15 nanoseconds of wake-up latency, which is only ∼0.1% of the AVX throttling period (12–15 microseconds, as demonstrated in Fig 8(a)).

Figure 9 depicts how AVX power-gating, IPC, frequency, and *Vcc* changes over time during the AVX2 PHI execution on a Cannon Lake system. The current management mechanisms throttle the CPU core to either 1) increase the *Vcc* voltage guardband to prevent di/dt noise, (as illustrated orange plot of Figure 9(a)), or 2) decrease the core frequency to keep the Vcc/Icc within limits (as illustrated in orange and green plots of Figure 9(c)). In the case the power-gate is closed, the processor first opens the power-gate quickly (i.e., within several nanoseconds) and starts executing the PHIs in throttled mode (as shown in Figure 9(b), which zooms into the appropriate parts of Figures 9(a) and 9(c)).

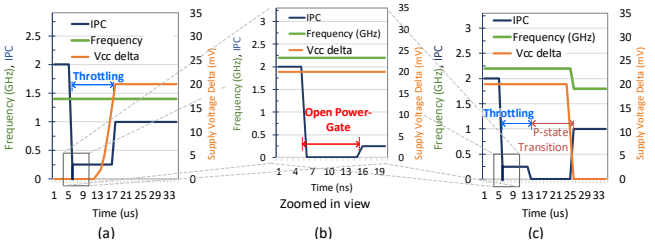

Figure 9: Illustration of the AVX power-gate, *Vcc*, CPU core frequency, and IPC behavior when executing the AVX2 instruction loop that activates two current management mechanisms: (a) *di/dt* noise prevention by throttling the core execution while increasing the voltage guardband, and (c) Vcc_{max}/Icc_{max} design limit protection by throttling the core execution while initiating a P-state transition to reduce the voltage and frequency. (b) Zoomed in (nanosecond-granularity) view of the opening of the AVX2 power-gate. Core's local PMU opens the AVX2 power-gate when the first AVX2 instruction is executed.

Key Conclusion 3. *Contrary to the state-of-the-art work's hypothesis [91], power-gating execution units (e.g., AVX2 power gating [69]) accounts for an insignificant portion (∼0.1%) of the total throttling time observed when executing PHIs.*

5.5. Multi-level Throttling

To further understand the side-effects of the throttling technique, we conduct experiments on the Cannon Lake processor. We execute one of seven instruction types (i.e., 64b, 128b_Light, 128b_Heavy, 256b_Light, 256b_Heavy, 512b_Light, and 512b_Heavy, discussed in Section 4) in a loop using three experiments. In the first and second experiments, we execute the seven instruction types on one and two cores, respectively, at one of three different frequencies (1*GHz*, 1.2*GHz* and 1.4*GHz*). We measure the throttling period of each core. Figure 10(a) plots the results of these two experiments. In the third experiment, we execute the seven instruction types in a loop followed by a loop of 512b_Heavy (the heaviest instruction type). We measure the throttling period of the 512b_Heavy loop. Figure 10(b) plots the results of the third experiment with an operating frequency of 1.4*GHz*.

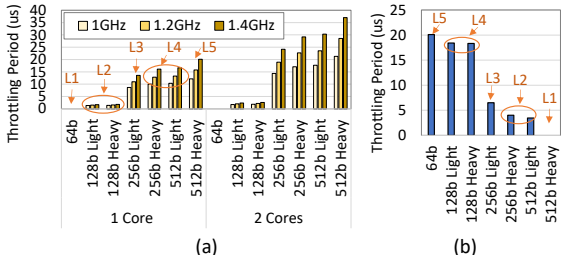

Figure 10: (a) Effect of executing different instruction types at various frequencies on one and two cores on the length of the throttling period in the Cannon Lake system. (b) Throttling period of a 512b_Heavy loop when the loop is preceded by different instruction types (x-axis). Different voltage guardband levels result in a multi-level (L1-L5) throttling period.

We make two observations from Figure 10(a). First, the throttling period increases as the 1) computational intensity (i.e., heaviness of the operation type and/or width) of the executed instructions increases, 2) frequency of the core increases, and 3) number of cores that are concurrently-executing PHIs increases. Increasing any of these parameters increases current consumption, thereby increasing the supply voltage guardband (i.e., ΔV), as we explain in Section 2 (Equation 1). Second, when running PHIs on two cores, the throttling period increases significantly (e.g., 256b_Heavy throttling period is 9*us* on two cores at 1*GHz* compared to 5*us* in only one core). The longer throttling period is due to the higher voltage guardband, which requires more time for the processor to increase the voltage to the required level. The processor PMU stops throttling the cores once the shared VR is settled at the required level by *both* cores.

We observe from Figure 10(b) that the throttling period of the 512b_Heavy loop increases when the computational intensity of the instructions executed in the preceding loop (shown on the x-axis) decreases. This is because the lower the computational intensity of the instructions in the preceding loop, the lower the voltage guardband, and thus, executing the 512b_Heavy loop requires more time for the processor to increase the voltage to the required level.

We observe from Figures 10(a) and (b) that there are at least *five* throttling levels (L1–L5) corresponding to the computational intensity of instruction types, as shown Figure 10(a) and 10(b) (indicated by the orange text).

We conclude that 1) the throttling period of a CPU core when running PHIs has at least five levels,[11] which depends on multiple parameters such as frequency, voltage, and the computational intensity of the instructions, and 2) there is a *cross-core* throttling side-effect due to the shared voltage regulator between cores.

Key Conclusion 4. *Current management mechanisms result in a multi-level throttling period depending on the computational intensity of the PHIs. This multi-level throttling period has a*

[11] These five throttling levels, which affect the core throttling period after executing PHIs at any frequency (even if Turbo frequencies are disabled), are different from the three Turbo licenses (i.e., LVL{0,1,2}_TURBO_LICENSE) discussed in Section 5.3.

cross-core side-effect, in which the throttling period depends on how many cores are running PHIs.

5.6. Throttling Affects SMT Threads

To understand the source of throttling and its microarchitectural impact on the core, we track the number of micro-operations (uops) that the core pipeline delivers from the front-end to the back-end during throttled and non-throttled loop iterations when running AVX2 PHIs on the Cannon Lake processor. To do so, we read the `IDQ_UOPS_NOT_DELIVERED` and `CPU_CLK_UNHALTED` performance counters at the beginning and end of each iteration. `IDQ_UOPS_NOT_DELIVERED` counts the number of uops *not delivered* by the Instruction Decode Queue (IDQ) to the back-end of the pipeline when there were *no back-end stalls* [43]. We *normalize* the counter by the maximum possible number of uops that can be delivered during the iteration, i.e., `UOPS_NOT_DELIVERED` = `IDQ_UOPS_NOT_DELIVERED/(4·CPU_CLK_UNHALTED)`. Figure 11(a) shows the distribution of the normalized undelivered uops for throttled and non-throttled loop iterations. We observe from Figure 11(a) that for a throttled loop iteration, the IDQ does *not* deliver any uop in approximately three-quarters (∼75%) of the core cycles even though the back-end is *not* stalled. On the other hand, for a non-throttled loop iteration, we observe that there are almost *no* cycles in which the IDQ does not deliver uops to the back-end.

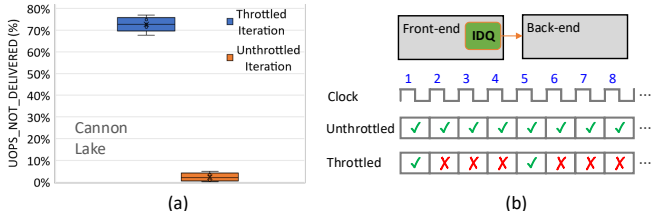

Figure 11: (a) Normalized IDQ_UOPS_NOT_DELIVERED performance counter during throttled and unthrottled iterations. (b) Pipeline behavior during throttling.

We conclude that the core uses a throttling mechanism that limits the number of uops delivered from the IDQ to the back-end during a certain time window, as illustrated in Figure 11(b). During a time window of four core clock cycles, the IDQ delivers uops to the back-end in only one cycle, while in the remaining three cycles, the throttling mechanism blocks the IDQ (i.e., no uops are delivered). We found that this throttling mechanism affects *both threads in Simultaneous Multi-Threading* (SMT). This means that both threads are throttled when one thread executes a PHI because the IDQ-to-back-end interface is shared between the threads.

Key Conclusion 5. *Contrary to the hypothesis of a state-of-the-art work [91], we observe on a real system that the processor front-end (IDQ) to back-end uop delivery is blocked during 75% of the time, rather than a reduced core clock frequency of 4×. This throttling mechanism affects both threads in Simultaneous Multi-Threading (SMT).*

5.7. Software-level Power Management Policies

To examine whether software-level power management policies affect the underlying mechanisms of IChannels, we observe the underlying mechanism under three separate power management policy governors [77]: userspace, powersave, and performance. We find that the underlying mechanism of IChannels persists across all three policies. We also find that software-level power management policies do not affect the hardware throttling mechanisms. This is because hardware throttling is implemented inside the core for fast response (i.e., nanoseconds), and we are not aware of any software control that allows to disable this mechanism.

6. IChannels Evaluation

This section evaluates our three IChannels covert channels, IccThreadCovert, IccSMTcovert, and IccCoresCovert, using our proof-of-concept (PoC) implementations. We compare our three IChannels covert channels to recent works [5, 57, 59, 91] that exploit PHIs and power management vulnerabilities to create covert channels. In this evaluation, we build covert channels between execution contexts running on: (1) the same hardware thread, (2) the same physical core but using different SMT threads, and (3) different physical cores.

6.1. Setup

We evaluate IChannels on Coffee Lake (Core i7-9700K [113]) and Cannon Lake (Core i3-8121U [45]). We test IccThreadCovert and IccCoresCovert on both processors, but we test IccSMTcovert only on Cannon Lake as Coffee Lake does not support SMT.

6.2. IChannels Throughput

The throughput (i.e., the channel capacity) of our three IChannels side channels is ∼2.9*Kbps*. Each channel sends 2 bits in each communication transaction cycle. The actual sending of the 2 bits takes less than < 40*us*, during which the `Sender` code executes the appropriate PHI while the `Receiver` code measures the throttling-period (TP) to decode two transmitted bits. Before sending the next two bits, each covert channel should wait for *reset-time* (∼650*us*), as we discuss in Section 4.1.2. Therefore, each covert channel's cycle time is the sum of the latencies for reset-time and transmitting two bits, which is less than 690*us*. Next, we compare each covert channel's throughput to its most relevant prior work.

IccThreadCovert. We compare IccThreadCovert against NetSpectre [91], the state-of-the-art work that exploits the variable latency of PHIs to create a covert channel between two execution contexts running on the *same hardware thread*. Similarly to IccThreadCovert, NetSpectre exploits the throttling of AVX2 instructions to establish a covert channel within the same hardware thread. The NetSpectre covert channel can send *one bit* per communication transaction, while the IccThreadCovert covert channel can send *two bits* per communication transaction. Figure 12(a) compares the IccThreadCovert throughput to that of NetSpectre. Our results show that the IccThreadCovert throughput is 2× that of NetSpectre.[12]

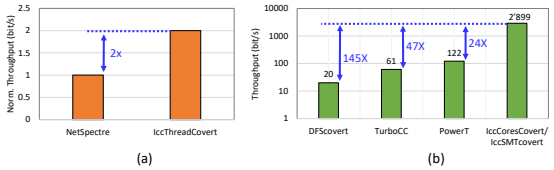

Figure 12: (a) Normalized throughput of IccThreadCovert and NetSpectre [91]. (b) Throughput of IccSMTcovert, IccCoresCovert, DFScovert [5], TurboCC [57] and PowerT [59].

IccSMTcovert and IccCoresCovert. The three most related works to IccSMTcovert and IccCoresCovert, which create

[12] We compare IccThreadCovert to NetSpectre's main gadget (i.e., throttling of PHIs on the same hardware thread), and not to the end-to-end NetSpectre implementation, which includes network components.

covert channels across SMT threads and across cores, are DFScovert [5], TurboCC [57], and PowerT [59]. These works exploit different power management mechanisms of modern processors to build covert channels across cores and SMT threads. DFScovert manipulates the power governors that control the CPU core frequency. TurboCC utilizes CPU core frequency changes due to the frequency boosting mechanism of modern Intel processors (i.e., Intel Turbo [89, 90]). PowerT utilizes CPU core frequency changes due to the thermal management mechanisms in modern processors. These three mechanisms are slow (i.e., can take several milliseconds) compared to the current management mechanisms that our IChannels covert channels utilize. Figure 12(b) compares the throughput of our IccSMTcovert and IccCoresCovert mechanisms to DFScovert, TurboCC, and PowerT. IccSMTcovert/IccCoresCovert throughput is 145× (2899/20), 47× (2899/61), and 24× (2899/122) the throughput of DFScovert, TurboCC, and PowerT, respectively.

6.3. System Noise Effect on IChannels Accuracy

Similar to all recent covert channels (e.g., Spectre [63], Meltdown [66], NetSpectre [91], DFScovert [5], TurboCC [57], and PowerT [59]), IChannels covert channels' accuracy is sensitive to system noise. System noise can occur in two main scenarios. First, due to system activity, such as interrupts and context switches, which can extend the execution time measured by the `Receiver`, causing errors in decoding of the received bits. Second, an application running concurrently with the covert channel's `Sender`/`Receiver` processes can reduce the covert channel's accuracy by, for example, executing PHIs. In this section, we analyze the IChannels error rate under these two scenarios and propose approaches to mitigate the effects of noise.

Low-Noise System. Figure 13 shows the distribution of the throttling period (TP) counter (in cycles) of the `Receiver` code for a client system with relatively low noise (i.e., interrupt and context-switch rates below 1000 events per second) while other *non-AVX* applications (besides the `Sender/Receiver`) are running on the system. The TP values are centered around the threshold of each of the four-level ranges marked in the green dashed line in Figure 13. The ranges do not overlap since there is a significant difference in the values ($> 2K$ cycles). Therefore, the error rate of the IChannels covert channel is close to zero when there is low noise in the system.

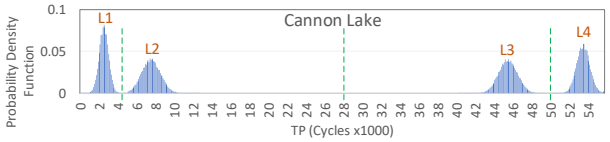

Figure 13: Distribution of throttling period (TP) values in the `Receiver` when detecting each of the four throttling levels (L1-L4) in a low-noise system.

High-Noise from Interrupts and Context Switches. Interrupts and context-switches that occur while the `Receiver` is decoding the received bits (i.e., measuring the execution time of instructions in green in Figure 3) can significantly increase the measured time (on the order of several microseconds), thereby causing errors in decoding. Interrupt and context-switch latencies are typically within few microseconds [36, 106] to few tens of microseconds [21, 109], respectively, with occurrence frequencies of a few hundred (e.g., in a noisy system) to thousands (e.g., in a highly noisy system) of events per second. Figure 14(a) shows the measured bit-error-rate (BER) as a function of system event (i.e., context-switch or interrupt) occurrence rates. The results show that the BER is low even when the system is highly noisy. This is because these events have a low probability of occurring during the decoding interval (only several microseconds) at the `Receiver`.

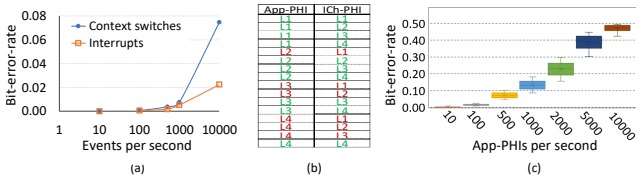

Figure 14: (a) Bit-error-rate (BER) as a function of system event (context-switch or interrupt) rates. (b) Cases of erroneous decoding (red) due to a noise that occurs when an application executes PHIs (App-PHI) concurrently with the IChannels `Sender/Receiver` processes that execute PHIs (ICh-PHI). (c) Distribution of BER when running a synthetic Application (App) concurrently with the `Sender/Receiver`, which executes PHIs at different rates.

Noise from Concurrent Applications. An application that executes PHIs while running concurrently with the IChannels' `Sender/Receiver` processes can cause bit errors in the covert channel. In particular, the error can occur if the power level of the application's PHI is higher than that of IChannels PHIs executed by the `Sender/Receiver` processes as shown in Figure 14(b) (in red). Figure 14(c) shows the measured distribution of channel BER when running a synthetic application (App) concurrently with the IChannels processes. App injects PHIs with a random power level (from the four levels) using different rates (10–10,000 App-PHIs per second). The figure shows that the BER significantly increases when App executes PHIs at a higher rate.

We evaluate the effect of concurrently running the 7-zip [1] application (which uses AVX2 instructions but not AVX-512) and observe a BER of less than 0.07 across the three IChannels covert channels when sending data during 60 seconds.

Mitigating the Effects of System Noise. We discuss three approaches to mitigating the effects of system noise on covert channels. First, an attacker can send the secret value many times and the `Receiver` can average the received value over a large number of measurements to obtain the secret value with an acceptable error rate. Second, we can use error detection and correction codes. This approach is used by several recent covert channel works [17, 24, 57, 70, 92]. Third, the `Sender/Receiver` can initiate a communication transaction only during low-noise periods [86]. In case of IChannels, the sender can track the system status and only transmit data when no other PHIs are executing. Client processors (i.e., our main target systems) are often idle ($> 80\%$ of the day [12, 31, 98]) due to low utilization, which provides many opportunities for the sender to reliably transmit data with little or no system noise.

6.4. IChannels on Server Processors

We have already shown that *all* Intel client and server[13] processors from the last decade (from Sandy Bridge client, 2010, to Ice Lake client, 2020 and Sandy Bridge server, 2011, to Cascade Lake server, 2019), which can be found in hundreds

[13] An Intel CPU core has nearly the same microarchitecture for client and server processors. Intel CPU core design is a single development project, leading to a master superset core. The project has two derivatives, one for server and one for client processors [111].

of millions of devices in use today [102], are affected by at least one of our three proposed covert-channels.

6.5. Side-channel Attacks Exploiting IChannels Side-effects

IChannels throttling side-effects (e.g., Multi-Throttling-SMT and Multi-Throttling-Cores) can also be used as a side-channel to leak data from victim code. Attacker code can infer the instruction types (e.g., 64bit scalar, 128bit vector, 256bit vector, 512bit vector instructions) of victim code that is running 1) on another SMT thread by utilizing the Multi-Throttling-SMT side-effect, or 2) on another core by utilizing the Multi-Throttling-Cores side-effect. In fact, our proof-of-concept code of the three IChannels covert channels can be used, with minimal changes, to demonstrate a synthetic side-channel between attacker code and victim code. However, it would be challenging and requires significant additional effort to conduct a real-world attack, which can extract sensitive information using the leaked data (i.e., the instruction type executed by the victim code). We leave such side-channel attacks to future work.

7. Mitigations for IChannels

In this section, we propose practical hardware or software techniques for the mitigation of IChannels covert channels.

Fast Per-core Voltage Regulators. As explained in Section 4.3.1, the largest side-effect that IChannels exploits is the cross-core side-effect, which occurs when two cores execute PHIs simultaneously (IccCoresCovert). When this happens, the throttling period of one core is extended because the other core also requires a voltage increase (to execute PHIs at the same time). To eliminate this cross-core side-effect, we propose to implement a PDN with *per-core* voltage regulators, such as *low dropout voltage regulators* (LDO [7, 9, 93, 94, 96, 103]) or *integrated voltage regulators* (IVR [10, 48, 75, 100]). Doing so allows each core that executes PHIs to handle its individual voltage transitions using its dedicated VR.

On the other hand, to mitigate IccThreadCovert and IccSMTcovert, a fast voltage ramp is required. This is because the root cause of the relatively long throttling period ($> 10us$) is the long time it takes for the VR to increase the voltage, while the throttling due to power-gating is only approximately $15ns$ ($\sim$0.1% of the throttling time), as explained in Section 5.4. We also show in Section 5.4 that processors with IVR, such as Haswell [10], still have a long throttling period (e.g., $\sim 9us$), even though the IVR PDN has a faster voltage ramp time relative to the motherboard voltage regulators (MBVR [14, 37, 52, 87]) used in some of the systems we evaluate (i.e., Coffee Lake and Cannon Lake). To mitigate this issue, we propose to implement an LDO PDN, which exists in most recent AMD processors [7, 9, 93, 94, 96, 103]). The LDO PDN allows fast voltage transitions (e.g., $< 0.5us$) [67], thereby significantly reducing the throttling time. Reducing the throttling time from $> 10us$ to $< 0.5us$ does not *completely* eliminate the covert channel. However, it makes establishing such a covert channel is much more difficult.

The LDO PDN incurs an area overhead of 11% – 13% [61, 82] of the core. However, this PDN can significantly benefit the system by reducing board area (e.g., by reducing the number of off-chip VRs), enabling fast voltage transitions, and improving energy efficiency [27, 30].

Improved Core Throttling. The IccSMTcovert covert channel exploits the throttling side-effect that we illustrate in Figure 11. Once a thread executes a PHI, the processor aggressively throttles the entire core, i.e., blocks delivering uops from the front-end to back-end for three cycles in a window of four cycles, which affects *both threads* in an SMT core. We propose to implement a more efficient throttling mechanism in two stages. First, instead of blocking all the uops sent from the front-end (IDQ) to the backend, the core blocks only the uops that belong to the thread that executes the PHI. Second, our enhanced throttling mechanism does not block non-PHI uops.

This mitigation option should not incur area, power, or performance overhead if carefully implemented. In fact, it would likely improve performance by increasing instruction-level concurrency. However, it may add more implementation, design, and verification effort.

A New Secure Mode of Operation. While current management mechanisms can improve system energy efficiency, we have shown that these mechanisms can result in security vulnerabilities in the system. To avoid these vulnerabilities, especially when handling sensitive data, we propose to add a new mode of system operation to the processor: *secure-mode*. The user can enable secure-mode at any point in time, during which the processor begins to operate at its highest voltage guardband corresponding to the worst-case power virus. While in secure-mode, throttling does not occur as a result of increased power requirements (e.g., executing PHIs) since the voltage guardband is already set to the maximum level. While secure-mode results in increased power consumption (by up to 4%/11% for a system with AVX2/AVX512 due to the additional voltage guardband), it can significantly improve the security of the overall system by mitigating the three covert channels of IChannels. In contrast, existing security techniques incur significant performance and energy overheads. For example, the well-known Intel SGX mechanism can incur significant performance degradation (e.g., up to 79% [110]) and an increase in energy consumption (up to 67% [19]) compared to running in native (i.e., non-SGX) mode. In addition, SGX increases the area of the memory controller since it requires 1) a memory-encryption-engine (MEE [23]) and 2) additional area to maintain SGX metadata [23].

Table 1 summarizes the effectiveness and overhead of our proposed mitigation techniques with respect to each one of the three covert channels of IChannels.

Mitigation	IccThreadCovert	IccSMTcovert	IccCoresCovert	Overhead
Per-core VR	Partially	Partially	✓	11%-13% more area
Improved Throttling	✗	✓	✗	Some design effort
Secure-Mode	✓	✓	✓	4%-11% additional power

Table 1: Effectiveness and overhead of our mitigation techniques

IChannels on other Microarchitectures. The greater part of a processor's microarchitecture is usually not publicly documented. Thus, it is virtually impossible to know the differences in the implementations (e.g., different throttling techniques, power delivery, resource sharing in SMT, current management at the circuit level) that allow or prevent IChannels in other processor architectures (e.g., AMD or ARM) without very careful examination. In fact, we confirmed that naively porting IChannels to recent AMD and ARM processors does not work. However, we believe the insights provided by our study (e.g., the multi-level throttling side-effect due to voltage changes, interference between SMT threads while throttling, interference between cores due to sharing of the voltage-regulator) on Intel processors can be applied to effectively adapt IChannels to other processors and inspire experts on other microarchitectures to perform the necessary studies for revealing similar covert channels.

8. Related Work

To our knowledge, this is the first paper that demonstrates how the multi-level throttling side-effects of current management mechanisms can be used to build covert channels across execution contexts running on the same thread, across SMT threads, and across cores in modern processors. We discuss and compare to state-of-the-art covert channels [57, 91] due to throttling techniques of current management mechanisms in Table 2 and Section 6. We also compare to state-of-the-art power [59] and frequency covert channels [5, 57] in Section 6. Other recent works on covert channels tend to follow one of the below directions.

Power Management Vulnerabilities. There are many works that exploit different power management vulnerabilities [5, 17, 17, 24, 53, 92, 101, 117]. JayashankaraShridevi *et al.* [53] create two attacks that significantly degrade power efficiency by inserting a hardware Trojan in the power management unit. Tang *et al.* [101] propose an attack that infers secret keys by enforcing anomalous frequencies and voltages in the DVFS, and inducing timing errors. Alagappan *et al.* [5] create a covert channel between a Trojan and a spy process by modulating the processor operating frequency, with the help of power governors, to stealthily exfiltrate data. Giechaskiel *et al.* [17] create a covert channel between independent FPGA boards in a data center by using the power supply unit, without physical access to boards. Compared to these previous works, IChannels is the first work that exploits multi-level throttling side-effects of current management mechanisms to develop a new set of covert channels, providing more than 24× higher channel capacity than the state-of-the-art power management-based covert channel works (see Section 6).

Also compared to these previous works, IChannels is 1) *faster* as it uses a mechanism with a response time of only a few hundreds of microseconds (i.e., throttling due to current management mechanisms), while CPU core thermal and governor-controlled [5, 77] frequency changes normally take several milliseconds, and 2) more *robust* as the processor gives priority to handling issues due to current emergencies (e.g., throttling starts in a few nanoseconds) while thermal and frequency changes are considered less urgent and can be handled on the order of milliseconds [5, 20, 26, 35, 77].

Other Covert Channels. There are many other types of covert channels that exploit other components of the system, such as CPU caches [70, 78, 115], DRAM row buffers [79], CPU functional units [108], or the RowHammer phenomenon in DRAM [16, 60, 62, 64, 73]. None of these exploit current management mechanisms like IChannels does.

9. Conclusion

Based on our rigorous characterization of current management mechanisms in real modern systems, we develop a deep understanding of the underlying mechanisms related to power management for PHIs (power-hungry instructions). In particular, we notice that these mechanisms throttle the execution for a period of time whose length is related to other instructions and PHIs executing on the system. We exploit these throttling side-effects to develop a set of covert channels, called *IChannels*. These high-throughput covert channels can be established between two execution contexts running on 1) the same hardware thread, 2) the same simultaneous multi-threaded (SMT) core, and 3) different physical cores. We implement a proof-of-concept of IChannels on recent Intel processors. We show that IChannels can reach more than 24× the channel capacity of other recent covert channels leveraging power management techniques. We propose multiple practical mitigations to protect against IChannels in modern processors. We conclude that current management mechanisms in modern processors can lead to malicious information leakage and hope our work paves the way for eliminating the confidentiality breaches such processor mechanisms lead to.

Acknowledgments

We thank the anonymous reviewers of ASPLOS 2021 and ISCA 2021 for feedback. We thank the SAFARI Research Group members for valuable feedback and the stimulating intellectual environment they provide. We acknowledge the generous gifts provided by our industrial partners: Google, Huawei, Intel, Microsoft, and VMware.

References

[1] 7-Zip, "7-Zip: File Archiver with a High Compression Ratio," online, accessed April 2020, https://www.7-zip.org/history.txt.
[2] K. Agarwal *et al.*, "Power Gating With Multiple Sleep Modes," in *ISQED*, 2006.
[3] Agner Fog, "Test Programs for Measuring Clock Cycles and Performance Monitoring," online, accessed August 2020, http://www.agner.org/optimize/testp.zip.
[4] C. J. Akl *et al.*, "An Effective Staggered-Phase Damping Technique for Suppressing Power-Gating Resonance Noise During Mode Transition," in *ISQED*, 2009.
[5] M. Alagappan *et al.*, "DFS Covert Channels on Multi-core Platforms," in *VLSI-SoC*, 2017.
[6] I. Anati *et al.*, "Inside 6th Gen Intel® Core™: New Microarchitecture Code Named Skylake," in *HotChips*, 2016.
[7] N. Beck *et al.*, "Zeppelin: An SoC for Multichip Architectures," in *ISSCC*, 2018.
[8] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," in *HPCA*, 2001.
[9] T. Burd *et al.*, "Zeppelin: An SoC for Multichip Architectures," *JSSC*, 2019.
[10] E. A. Burton *et al.*, "FIVR - Fully Integrated Voltage Regulators on 4th Generation Intel® Core™ SoCs," in *APEC*, 2014.
[11] M. Cho *et al.*, "Postsilicon Voltage Guard-band Reduction in a 22 nm Graphics Execution Core using Adaptive Voltage Scaling and Dynamic Power Gating," *JSSC*, 2016.
[12] ComScore, "Cross Platform Future in Focus ," https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/2017-US-Cross-Platform-Future-in-Focus, 2017.
[13] CPUbenchmark, "AMD vs Intel Market Share," https://bit.ly/3kV6kWY. accessed Nov 2020.
[14] E. Fayneh *et al.*, "4.1 14nm 6th-Generation Core Processor SoC with Low Power Consumption and Improved Performance," in *ISSCC*, 2016.
[15] E. Fetzer *et al.*, "Managing Power Consumption in a Multi-core Processor," US Patent 9,069,555. 2015.
[16] P. Frigo *et al.*, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *S&P*, 2020.
[17] I. Giechaskiel *et al.*, "C^{3}APSULe: Cross-FPGA Covert-Channel Attacks through Power Supply Unit Leakage," in *S&P*, 2020.
[18] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *JSSC*, 1996.
[19] C. Göttel *et al.*, "Security, Performance and Energy Trade-offs of Hardware-assisted Memory Protection Mechanisms," in *SRDS*, 2018.
[20] C. Gough *et al.*, "CPU Power Management," in *Energy Efficient Servers: Blueprints for Data Center Optimization*, 2015, pp. 21–70.
[21] S. Gravani *et al.*, "IskiOS: Lightweight Defense Against Kernel-level Code-reuse Attacks," *arXiv preprint arXiv:1903.04654*, 2019.
[22] E. Grochowski *et al.*, "Best of both Latency and Throughput," in *ICCD*, 2004.
[23] S. Gueron, "A Memory Encryption Engine Suitable for General Purpose Processors," *IACR*, 2016.
[24] M. Guri *et al.*, "PowerHammer: Exfiltrating Data From Air-Gapped Computers Through Power Lines," *TIFS*, 2020.
[25] D. Hackenberg *et al.*, "An Energy Efficiency Feature Survey of the Intel Haswell Processor," in *IPDPS*, 2015.
[26] J. Haj-Yahya *et al.*, "SysScale: Exploiting Multi-Domain Dynamic Voltage and Frequency Scaling for Energy Efficient Mobile Processors," in *ISCA*, 2020.

Proposal	Same Core	Cross-SMT Threads	Cross-Core	BW	User/ Kernel	Underlying Mechanisms	Turbo-Independent	Root Cause	Effective Mitigations
NetSpectre [91]	✓	✗	✗	1.5 kb/s	U	Single-level Thread Throttling	✓	✗	✗
TurboCC [57]	✗	✗	✓	61 b/s	K	Turbo Frequency Change	✗	✗	✗
IChannels	✓	✓	✓	3 kb/s	U	Multi-level Thread, SMT, and Core (VR) Throttling	✓	✓	✓

Table 2: Comparison to state-of-the-art covert channels utilizing throttling effects of current management mechanisms.

[27] J. Haj-Yahya *et al.*, "FlexWatts: A Power-and Workload-Aware Hybrid Power Delivery Network for Energy-Efficient Microprocessors," in *MICRO*, 2020.
[28] J. Haj-Yahya *et al.*, *Energy Efficient High Performance Processors: Recent Approaches for Designing Green High Performance Computing*. Springer, 2018.
[29] J. Haj-Yahya *et al.*, "Power Management of Modern Processors," in *Energy Efficient High Performance Processors*, 2018, pp. 1–55.
[30] J. Haj-Yahya *et al.*, "A Comprehensive Evaluation of Power Delivery Schemes for Modern Microprocessors," in *ISQED*, 2019.
[31] J. Haj-Yahya *et al.*, "Techniques for Reducing the Connected-Standby Energy Consumption of Mobile Devices," in *HPCA*, 2020.
[32] J. Haj-Yihia *et al.*, "Compiler-directed Power Management for Superscalars," *TACO*, 2015.
[33] J. Haj-Yihia *et al.*, "Fine-grain Power Breakdown of Modern Out-of-order Cores and its Implications on Skylake-based Systems," *TACO*, 2016.
[34] P. Hammarlund *et al.*, "Haswell: The Fourth-Generation Intel Core Processor," *IEEE Micro*, 2014.
[35] H. Hanson *et al.*, "Thermal Response to DVFS: Analysis with an Intel Pentium M," in *ISLPED*, 2007.
[36] W. He *et al.*, "SGXlinger: A New Side-channel Attack Vector Based on Interrupt Latency Against Enclave Execution," in *ICCD*, 2018.
[37] B. Howse and R. Smith, "Tick Tock On The Rocks: Intel Delays 10nm, Adds 3rd Gen 14nm Core Product Kaby Lake," 2015.
[38] Z. Hu *et al.*, "Microarchitectural Techniques for Power Gating of Execution Units," in *ISLPED*, 2004.
[39] S. Huang *et al.*, "Measurement and Characterization of Haswell Power and Energy Consumption," in *E2SC*, 2015.
[40] Intel, "6th Generation Intel Processor Families for S-Platforms," online accessed Aug 2020 https://intel.ly/2XVdORo.
[41] Intel, "8th and 9th Generation Intel Core Processor Families Datasheet, Volume 1 of 2," online, accessed Aug 2020 https://intel.ly/30VcShP.
[42] Intel, "How to Overclock Your CPU from BIOS," online, accessed Aug 2020 https://www.intel.com/content/www/us/en/gaming/resources/bios-overclocking.html.
[43] Intel, "Intel Microarchitecture Code Named Skylake Events," online, accessed August 2020, https://download.01.org/perfmon/index/skylake.html.
[44] Intel, "Intel System Studio: Intel AVX2 Support in the Intel C++ Compiler," online, accessed August 2020 https://software.intel.com/content/www/us/en/develop/articles/intel-system-studio-avx2-support.html.
[45] Intel, "Intel® Core i3-8121U Processor," online, accessed Aug 2020 https://ark.intel.com/content/www/us/en/ark/products/136863/intel-core-i3-8121u-processor-4m-cache-up-to-3-20-ghz.html.
[46] Intel, "Voltage Regulator Module (VRM) and Enterprise Voltage Regulator-Down (EVRD) 11.1."
[47] Intel, "Module, Voltage Regulator and Enterprise Voltage Regulator-Down (EVRD) 11.1 Design Guidelines," *Intel Corp., Santa Clara, CA*, 2009.
[48] Intel, "Ice Lake, 10th Generation Intel® Core™ Processor Families," https://intel.ly/3frvxpK", July 2019.
[49] Intel, "Skylake-X, 6th Generation Intel Core X-series Processors Families," https://intel.ly/30SP8uX, July 2019.
[50] Intel, "Intel 64 and IA-32 Architectures Optimization Reference Manual," 2020.
[51] Intel, "Overclocking Intel® Core Processors: Taking Overclocking to the Next Level," online, https://bit.ly/3iITafa. accessed Aug 2020.
[52] S. Jahagirdar *et al.*, "Power Management of the Third Generation Intel Core Micro Architecture Formerly Codenamed Ivy Bridge," in *HotChips*, 2012.
[53] R. JayashankaraShridevi *et al.*, "Catching the Flu: Emerging Threats from a Third Party Power Management Unit," in *DAC*, 2016.
[54] R. Joseph *et al.*, "Control Techniques to Eliminate Voltage Emergencies in High Performance Processors," in *HPCA*, 2003.
[55] A. B. Kahng *et al.*, "TAP: Token-based Adaptive Power Gating," in *ISLPED*, 2012.
[56] A. B. Kahng *et al.*, "Many-Core Token-Based Adaptive Power Gating," *TCAD*, 2013.
[57] M. Kalmbach *et al.*, "TurboCC: A Practical Frequency-Based Covert Channel with Intel Turbo Boost," *arXiv preprint arXiv:2007.07046*, 2020.
[58] D. Kanter, "Haswell FIVR Extends Battery Life," *Microprocessor Report, The Linley Group*, 2013.
[59] S. K. Khatamifard *et al.*, "POWERT Channels: A Novel Class of Covert Communication Exploiting Power Management Vulnerabilities," in *HPCA*, 2019.
[60] J. S. Kim *et al.*, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *ISCA*, 2020.
[61] W. Kim, "Reducing Power Loss, Cost and Complexity of SoC Power Delivery Using Integrated 3-Level Voltage Regulators," Ph.D. dissertation, Harvard University, 2013.
[62] Y. Kim *et al.*, "Flipping Bits in Memory without Accessing them: An Experimental Study of DRAM Disturbance Errors," *ISCA*, 2014.
[63] P. Kocher *et al.*, "Spectre Attacks: Exploiting Speculative Execution," in *SP*, 2019.
[64] A. Kwong *et al.*, "RAMBleed: Reading Bits in Memory without Accessing Them," in *S&P*, 2020.
[65] P. Larsson, "di/dt Noise in CMOS Integrated Circuits," in *Analog Design Issues in Digital VLSI Circuits and Systems*. Springer, 1997.
[66] M. Lipp *et al.*, "Meltdown: Reading Kernel Memory from User Space," in *USENIX Security*, 2018.
[67] K. Luria *et al.*, "Dual-mode Low-drop-out Regulator/Power Gate with Linear and On–off Conduction for Microprocessor Core on-die Supply Voltages in 14 nm," *JSSC*, 2016.
[68] R. M. Ma *et al.*, "Maximum Current Throttling," US Patent. 13/537,319. Jan. 2014.
[69] J. Mandelblat, "Technology Insight: Intel's Next Generation Microarchitecture Code Name Skylake," in *Intel Developer Forum, San Francisco*, 2015.
[70] C. Maurice *et al.*, "Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud," in *NDSS*, 2017.
[71] A. Mazouz *et al.*, "Evaluation of CPU frequency transition latency," *Computer Science-Research and Development*, 2014.
[72] J. K. Millen, "Covert Channel Capacity," in *S&P*, 1987.
[73] O. Mutlu and J. S. Kim, "RowHammer: A Retrospective," *TCAD*, 2019.
[74] S. Naffziger, "Integrated Power Conversion Strategies Across Laptop Server and Graphics Products," in *Proc. Power Supply Chip*, 2016.
[75] A. Nalamalpu *et al.*, "Broadwell: A Family of IA 14nm Processors," in *VLSI Circuits*, 2015.
[76] National Instruments, "NI-DAQ PCIe-6376," online accessed 2019 http://www.ni.com/pdf/manuals/377387c.pdf.
[77] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor," in *Proceedings of the Linux Symposium*, 2006.
[78] C. Percival, "Cache Missing for Fun and Profit," 2005.
[79] P. Pessl *et al.*, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *USENIX Security*, 2016.
[80] A. V. Peterchev and S. R. Sanders, "Load-line Regulation with Estimated Load-current Feedforward: Application to Microprocessor Voltage Regulators," *IEEE TPEL*, 2006.
[81] C. Piguet, *Low-Power CMOS Circuits: Technology, Logic Design and CAD Tools*. CRC Press, 2005.
[82] A. Quelen *et al.*, "LDO-assisted Voltage Selector over 0.5-to-1V VDD Range for Fine Grained DVS in FDSOI 28nm with 200ns/V Controlled Transition," in *ESSCIRC*, 2018.
[83] V. J. Reddi *et al.*, "Voltage Emergency Prediction: Using Signatures to Reduce Operating Margins," in *HPCA*, 2009.
[84] V. J. Reddi *et al.*, "Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-guided Thread Scheduling," in *MICRO*, 2010.
[85] J. Reinders, "Intel AVX-512 Instructions," *Intel Software Developer Zone, Jun*, 2017.
[86] H. Ritzdorf, "Analyzing Covert Channels on Mobile Devices," Master's thesis, ETH Zürich, Department of Computer Science, 2012.
[87] E. Rotem *et al.*, "Power Management Architecture of the 2nd Generation Intel® Core Microarchitecture, Formerly Codenamed Sandy Bridge," in *HotChips*, 2011.
[88] E. Rotem and S. P. Engineer, "Intel Architecture, Code Name Skylake Deep Dive: A New Architecture to Manage Power Performance and Energy Efficiency," in *Intel Developer Forum*, 2015.
[89] E. Rotem *et al.*, "Power and Thermal Constraints of Modern System-on-a-Chip Computer," in *THERMINIC*, 2013.
[90] E. Rotem *et al.*, "Power-management Architecture of the Intel Microarchitecture Code-named Sandy Bridge," *IEEE MICRO*, 2012.
[91] M. Schwarz *et al.*, "NetSpectre: Read Arbitrary Memory Over Network," in *ESORICS*, 2019.
[92] N. Sehatbakhsh *et al.*, "A New Side-Channel Vulnerability on Modern Computers by Exploiting Electromagnetic Emanations from the Power Management Unit," in *HPCA*, 2020.
[93] T. Singh *et al.*, "3.2 Zen: A Next-generation High-performance $\times$86 Core," in *ISSCC*, 2017.
[94] T. Singh *et al.*, "Zen: An Energy-Efficient High-Performance –x86 Core," *JSSC*, 2018.
[95] G. Singla *et al.*, "Predictive Dynamic Thermal and Power Management for Heterogeneous Mobile Platforms," in *DATE*, 2015.
[96] A. A. Sinkar *et al.*, "Low-cost Per-core Voltage Domain Support for Power-constrained High-Performance Processors," *VLSI*, 2013.
[97] Standard Performance Evaluation Corporation, "SPEC," online, accessed March 2018, www.spec.org. Mar 2018.
[98] R. Stedman, "Reducing Desktop PC Power Consumption Idle and Sleep Modes," *DELL Corp, Technical Presentation*, 2005.
[99] J. Sun *et al.*, "A Novel Input-side Current Sensing Method to Achieve AVP for Future VRs," *IEEE Transactions on Power Electronics*, 2006.
[100] S. M. Tam *et al.*, "Skylake-SP: A 14nm 28-Core Xeon® Processor," in *ISSCC*, 2018.
[101] A. Tang *et al.*, "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management," in *USENIX Security*, 2017.
[102] Techradar, "Intel Shipped More than 100 Million Microprocessors during Q3," online, accessed Feb 2019 https://www.techradar.com/news/computing/intel-shipped-more-than-100-million-microprocessors-during-q3-1269051, 2017.
[103] Z. Toprak-Deniz *et al.*, "5.2 Distributed System of Digitally Controlled Microregulators Enabling per-core DVFS for the POWER8 TM Microprocessor," in *ISSCC*, 2014.
[104] C.-H. Tsai *et al.*, "Switching Frequency Stabilization Techniques for Adaptive on-time Controlled Buck Converter with Adaptive Voltage Positioning Mechanism," *IEEE TPEL*, 2015.
[105] D. M. Tullsen *et al.*, "Simultaneous Multithreading: Maximizing on-chip Parallelism," in *ISCA*, 1995.
[106] J. Van Bulck *et al.*, "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic," in *CCS*, 2018.
[107] A. Varma *et al.*, "Power management in the Intel Xeon E5 v3," in *ISLPED*, 2015.
[108] Z. Wang and R. B. Lee, "Covert and Side Channels due to Processor Architecture," in *ACSAC*, 2006.
[109] V. M. Weaver, "Linux perf_event Features and Overhead," in *FastPath*, 2013.
[110] O. Weisse *et al.*, "Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves," *ISCA*, 2017.
[111] Wikichip, "Skylake (server) - Microarchitectures - Intel," online, accessed August 2019 https://bit.ly/2MHEWkj.
[112] Wikipedia, "7th Generation Intel Core Processor Family (Kaby Lake)," online, accessed August 2020 https://en.wikipedia.org/wiki/Kaby_Lake.
[113] Wikipedia, "8th and 9th Generation Intel Core Processor Family (Coffee Lake)," online, accessed August 2020 https://en.wikipedia.org/wiki/Coffee_Lake.
[114] S. Wright *et al.*, "Characterization of Micro-bump C4 Interconnects for Si-carrier SOP Applications," in *ECTC*, 2006.
[115] Z. Wu *et al.*, "Whispers in the Hyper-space: High-bandwidth and Reliable Covert Channel Attacks inside the Cloud," *TON*, 2014.
[116] R. Zhang *et al.*, "Architecture Implications of Pads as a Scarce Resource," in *ISCA*, 2014.
[117] S. Zhang *et al.*, "Blacklist Core: Machine-Learning based Dynamic Operating-Performance-Point Blacklisting for Mitigating Power-management Security Attacks," in *ISPED*, 2018.