POWERT Channels: A Novel Class of Covert Communication Exploiting Power Management Vulnerabilities

S. Karen Khatamifard*, Longfei Wang†, Amitabh Das‡, Selçuk Köse§ and Ulya R. Karpuzcu*
*Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, Minnesota, USA
Email: {khatami, ukarpuzc}@umn.edu
†Department of Electrical and Computer Engineering, University of South Florida, Tampa, Florida, USA
Email: longfei@mail.usf.edu
‡Austin, Texas, USA Email: dashamitabh@gmail.com
§Department of Electrical and Computer Engineering, University of Rochester, Rochester, New York, USA
Email: skose@ur.rochester.edu

***Abstract*—To be able to meet demanding application performance requirements within a tight power budget, runtime power management must track hardware activity at a very fine granularity in both space and time. This gives rise to sophisticated power management algorithms, which need the underlying system to be both highly *observable* (to be able to *sense* changes in instantaneous power demand timely) and *controllable* (to be able to *react* to changes in instantaneous power demand timely). The end goal is allocating the power budget, which itself represents a very critical shared resource, in a fair way among active tasks of execution. Fundamentally, if not carefully managed, any system-wide shared resource can give rise to covert communication. Power budget does not represent an exception, particularly as systems are becoming more and more observable and controllable. In this paper, we demonstrate how power management vulnerabilities can enable covert communication over a previously unexplored, novel class of covert channels which we will refer to as POWERT channels. We also provide a comprehensive characterization of the POWERT channel capacity under various sharing and activity scenarios. Our analysis based on experiments on representative commercial systems reveal a peak channel capacity of 121.6 bits per second (bps).**

***Keywords*-covert channels; power management; power headroom modulation.**

I. Introduction

Modern computing platforms are fundamentally power limited [1]. This gives rise to sophisticated runtime power management – spanning several software and hardware layers of the system stack – in order to meet diverse and demanding runtime performance needs within the stringent power budget. Effective power management requires a highly *observable* and *controllable* system, at a very fine granularity in both space and time. Observability is necessary to be able to timely *sense*; controllability, to be able to timely *react*, to changes in the instantaneous power consumption of the overall system. Activity monitors in the form of performance counters or sensors dispersed across chip serve the purpose. Exposing fine grain hardware knobs for power management to the software layers of the system stack can also help, as, for example, it is the case for Intel's Running Average Power Limit (RAPL) interface [2].

By distributing the power budget carefully among active tasks of execution, runtime power management has to guarantee that the system-wide power consumption never exceeds the system-wide power budget. The power budget itself represents a very critical shared resource. If not carefully managed, any shared (hardware or software) resource can easily enable information leakage via covert communication [3], [4], [5]. As a fundamental shared resource, power budget unfortunately does not represent an exception. The abundance of specialized activity monitors, their exposure to software layers, and the need for tight global control to avoid power budget overshoots, exacerbate the situation.

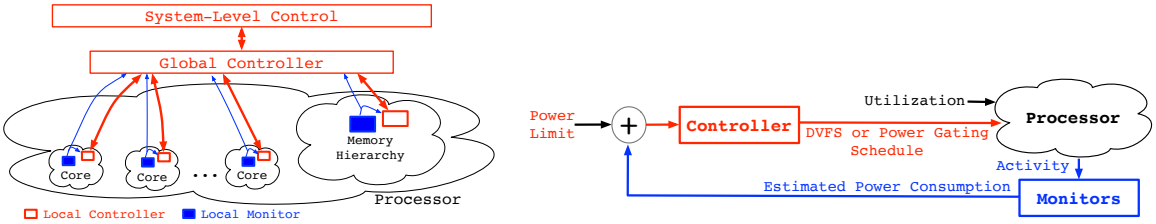
In this study, we introduce, demonstrate and characterize a novel class of covert communication over previously unexplored channels triggered by power management vulnerabilities. In the following, we will refer to this novel class as POWERT (POWER + c ovERT) channels. Key contributions of this study include:

- Introduction and detailed analysis of covert communication over POWERT channels; a novel, previously unexplored class of covert channels induced by power management vulnerabilities;
- Demonstration of POWERT communication on two representative commercial systems;
- Comprehensive, analytical and experimental characterization of the POWERT channel capacity.

In the following, Section II covers the basics of POWERT communication; Section III, channel specifics; Sections IV and V, evaluation; Section VI, countermeasures; Section VII, related work; and Section VIII, a summary of our findings.

II. POWERT Communication Basics

We start the characterization with basic definitions and the threat model in Section II-A and continue in Section II-B with an overview of inevitable power management practices that enable POWERT channels. Section II-C provides a conceptual explanation of how emerging power management practices facilitate covert channels. We conclude by demonstrating a proof-of-concept POWERT attack on a representative commercial system in Section II-D.



(a) Macroscopic view (b) Microscopic view

Figure 1: Hierarchical on-chip power management and control, adapted from [6].

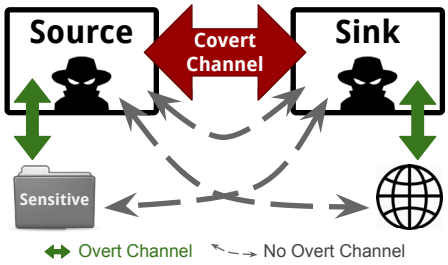


Figure 2: Threat model.

A. Threat Model

Fig. 2 depicts two malicious entities (the *source* and the *sink*, respectively), not permitted to communicate legitimately through *overt* channels, performing a *covert* channel attack. Without loss of generality, these entities may correspond to hardware (such as cores or functional units), or software sharing hardware resources [7]. The transmitting end, the *source*, has access to sensitive information (such as a secret key), however, not to any overt channel for data communication. The receiving end, the *sink*, on the other hand, has access to an overt channel (for potential data communication), but not to sensitive information. By communicating with the source over a *covert* channel, the sink can not only acquire access to sensitive information, but also can subsequently send this information to third parties over the overt channel. As a representative example, the source can be a contacts manager, and the sink, a weather application, in a mobile system [7]. By construction, such covert communication is hidden from other hardware or software entities sharing the same system.

B. Power Budget: A Critical Shared Resource

Modern power-limited computing platforms benefit from sophisticated power management in two distinct ways:

(i) Prevention of serious power budget overshoots, which can physically damage the system;

(ii) Optimal distribution of the shared power budget among active tasks of execution, to satisfy possibly conflicting performance requirements in a fair and efficient manner.

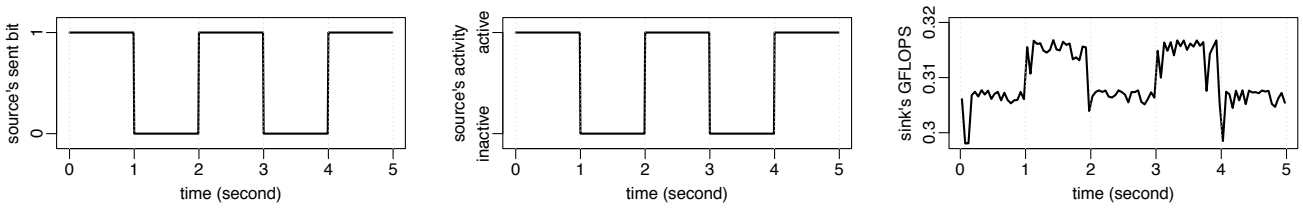
Fig. 1a illustrates an overview of inevitable power management practices in modern systems. System-level control at the hardware-software interface directs a global controller, which in turn orchestrates local controllers dispersed across chip. Local controllers periodically evaluate monitored local activity to adjust the operating point (i.e., the operating voltage or frequency). This type of distributed control is becoming the norm, as local controllers can react to locally confined changes in the instantaneous power demand much faster. If a reallocation of the system wide power budget becomes necessary, system-level control alerts the global controller, which in turn makes the local controllers adjust the local operating voltage and frequencies accordingly.

Fig. 1b depicts a generic control loop, which is equally applicable to both global and local controllers in Fig. 1a. Be it local or global, the controller modulates the operating point as a function of the power limit provided at its input. The goal always is delivering the maximum possible performance without violating the power limit, which reflects (instanteneous) power budget induced constraints. Various options exist for operating point modulation, including adjustments to the operating voltage and frequency (via DVFS, Dynamic Voltage and Frequency Scaling), selective shut-down of idle resources (via power gating) or both.

To summarize, system-level control imposes an instantaneous power budget, which the global controller has to meet via orchestrating local controllers. Local controllers in turn enforce necessary adjustments to local operating points. In order to prevent power budget overshoots, controllers periodically monitor the impact of these operating point adjustments on the instantaneous power consumption.

C. POWERT Communication via Power Headroom Modulation (PHM)

Following the threat model from Fig. 2, let us assume that a source application, which has access to sensitive information, shares processor resources with a sink application (possibly along with other applications). Controllers usually modulate the operating point (by, e.g., DVFS) periodically. This is because activity monitors time-sample the system at regular intervals. The period of operating point adjustments, t_{PM}, is a function of the period of activity monitors along with the time it takes to perform the actual change in the operating point (which typically incurs the latency across the power/clock distribution networks and of voltage regulators [6]). The period t_{PM} is usually in the order of several processor clock cycles. By construction, the source and the sink are very well aware of this period. Moreover, typical power management algorithms are of predictive nature and extrapolate predictions



(a) bits sent by source (b) source's activity (c) sink's performance (GFLOPS)

Figure 3: POWERT attack example.

from a history of monitored activity which are t_{PM} apart in time from each other.

If no other application resides in the system, but the source and the sink, the source can easily modulate the share of the power budget available to the sink, to encode binary information. This is possible simply because the chip-wide power budget is a shared resource. In the following, we will refer to the sink's share of the power budget as the *power headroom*. The source can easily control its own activity, and thereby, its own power consumption. Then, what is left to the sink is the power budget minus the source's power consumption, which forms the power headroom.

The procedure is straight-forward: To encode a logic 1, the source can run a power hungry virus. In this manner, the source can reach its own power budget limit, and taint its local activity history to trick the controllers. To prevent the instantaneous (system-wide) power consumption exceed the available budget as a result of excessive consumption at the source, the controllers will likely trigger emergency power throttling at the sink. The power headroom becomes practically zero. Under regular activity at the source (including no activity), on the other hand, such throttling events at the sink become much less likely. Therefore, by tracking the power headroom, the sink can clearly distinguish extremely high activity from other activity levels at the source. If, for example, the source chooses to encode a logic 0 (1) by no (extreme) activity, the sink can decode a sizable positive power headroom as a logic 0, and any power headroom change (caused by source's activity) as a logic 1.

Other applications sharing the same processor resources can challenge this type of covert communication between the source and the sink. Inevitably, sharing induces noise in the covert channel, which can reduce the correlation between distinct activity patters at the source (used to encode information by the source) and the power headroom (used to decode information by the sink). At the same time, power consumption of sharing applications can also cause a faster onset of throttling at the sink (as the source is attempting to send a logic 1, following the previous example). Therefore, if the sink chooses to quantify its available power headroom (i.e., to decode the source's message bit by bit) by tracking throttling events, sharing can render faster covert communication.

D. Anatomy of a POWERT Attack

In the following, we will refer to the entire hierarchy of the controllers in charge of the power management, as depicted in Fig. 1, as the *Power Manager* (PM). In a typical multi/many-core, the power budget cannot accommodate all cores operating at the peak performance point at the same time. Hence, when multiple cores run compute-intensive workloads simultaneously, PM has to assign a lower operating frequency to them than the rated peak frequency, in order to meet the power budget. We will next characterize a POWERT attack exploiting this inevitable behavior of PM.

Usually, when only one compute-intensive application is running on one of the cores, PM lets that core run at the rated maximum frequency. The common outcome for two compute-intensive applications running at the same time on two different cores is a lower frequency than the rated maximum, enforced on both cores. Thereby, a given application's activity pattern can directly affect the performance of other applications running on the system. Applications like the source and the sink from Fig. 2 can rely on this phenomenon to communicate with each other covertly, by affecting the operating frequency and/or voltage, hence the performance, of each other.

Let us next take a closer look into an example POWERT attack: The source and the sink are both compute-intensive applications. The source sends a "1" through the covert channel by running a compute-intensive workload, and a "0", by going into sleep. In order to capture source's activity pattern, the sink constantly runs a compute-intensive workload, as well. As a result, PM slows down the sink when the source is sending a "1" (i.e., running a compute-intensive workload), compared to when the source is sending a "0" (i.e., going into sleep). The sink therefore can retrieve bits sent by the source by just tracking its own performance. To measure its own performance, the sink can simply periodically check its own progress. Neither the sink, nor the source does need any system level privilege to this end, which challenges detecting (and potentially blocking) the attack.

Fig. 3 demonstrates a POWERT attack[1], where the source sends 5 bits through the covert channel at a communication rate of 1bit/s[2], as shown in Fig. 3a. The source becomes

[1]As we will detail in Section IV, this attack is performed on an Intel platform.

[2]We pick a relatively low communication frequency here to ease illustration and explore higher frequency ranges in Section V.

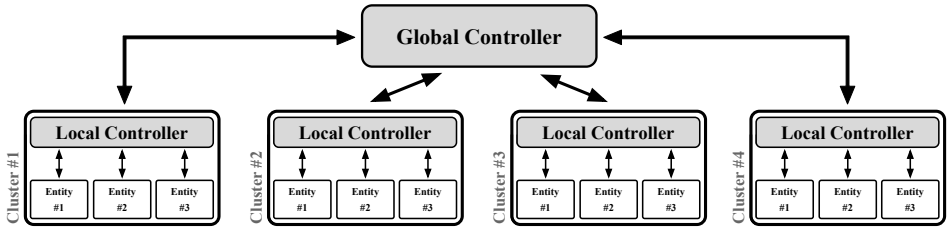


Figure 4: Two layer hierarchical power management.

active when sending a "1", and goes to sleep otherwise. Fig. 3b captures source's activity pattern corresponding to the sent bits. Finally, Fig. 3c depicts sink's performance, as measured by the sink itself. Y-axis represents GFLOPS (Giga Floating Point Operations per Second); the x-axis, time. In this particular example the compute-intensive sink application comprises a floating point heavy loop, therefore, GFLOPS is a good proxy for checking the rate of forward-progress at the sink. We observe that the GFLOPS rate of the sink decreases by around 2.5% on average when the source is active (sending "1"), compared to when the source is sleeping (sending "0"). The sink can therefore retrieve information from this covert channel by simply checking its GFLOPS rate. We fully specify and characterize similar POWERT attacks on different representative commercial platforms in Section V.

III. Communication Rate (or Frequency)

Power Manager, PM, constituting the entire hierarchy of controllers from Fig. 1, orchestrates power consumption of different entities, where each entity (e.g., cores, caches or routers) can operate at a different voltage and frequency. Let us assume that PM is in charge of N different entities, where P_i depicts the maximum possible power consumption for entity i, i ranging from 1 to N. Following this definition and our observations from Section II-D, if the instantaneous power consumed by each entity i reaches P_i simultaneously, a power budget overshoot becomes inevitable. In other words,

$$\sum_{i=1}^{N} P_i > power\ budget$$

applies. However, when the system is not highly utilized, most of the entities become idle and can be power-gated. In this case, PM can let the few active entities operate at their peak power consumption, P_i, as long as the overall power consumption stays below the power budget. As utilization increases, more entities have to become active at the same time in order to meet performance goals. Even under less than 100% utilization, the overall power consumption with only the active subset of the entities running at P_i may violate the power budget. To avoid budget violation, PM has to force all active entities to a lower-power operating point.

As an example, let us assume a four-core processor where each core consumes 15 Watts at peak and the overall power budget is 40 Watts. If only two of the cores are active, PM can let them run at their peak power consumption. If more than two cores are active, this is not possible anymore. PM has to distribute the available power budget evenly between the cores to avoid any budget overshoot. In the case of four active cores, this translates into each core consuming at most 10 Watts, which inevitably renders a relative slow-down (with respect to the peak rated performance point corresponding to 15 Watts) in all four cores.

It is this type of inevitable PM decisions that give rise to POWERT channels via power headroom modulation (Section II-C). A malware (such as the source from Fig. 2) can send information covertly to another malware (such as the sink from Fig. 2) by modulating the receiving side's share of the power budget, i.e., power headroom, and consequently, performance. The source can also activate a number of entities, enough to violate the power budget, in order to send a "1". In this case, PM has to lower the share of the power budget of all entities, including the sink at the receiving end. The sink in turn can retrieve the sent bit (a "1"), by sensing a slow-down in its own performance. Similarly, to send a "0", the source can put multiple entities to sleep to minimize the chance of violating the power budget (hence, of the sink being throttled). At the receiving end, the sink does not sense a slow-down in this case, and translates this to having received a "0".

POWERT communication via power headroom modulation has four phases:

- The first phase spans the time window t_{Util}, over which the source enforces changes in the activity, i.e., utilization, by modifying the number of active entities (or simply its own activity level).
- The second phase covers the time window $t_{Monitor}$, over which activity monitors sense the corresponding change in the power consumption of the affected entities.
- In the third phase, PM senses the change in the activity by reading monitors and makes a decision about throttling. Similar to the second phase, this phase usually takes place periodically, with a period of t_{PM} (Section II-C).
- The last phase comprises two steps: The first step is the time it takes for the affected entities to adjust their operating voltage and frequency to meet the enforced power budget, t_{Adjust}. The second step is the time it takes for the sink to sense the changes in its own performance, t_{Sense}.

These four phases together span the duration of communication, starting from sending a single bit until sensing it at the receiving end. Pipelined communication by overlapping

phases is also possible: For example, after PM reads the monitors and makes a new decision (phase three), the source can start sending the next bit (phase one). This is because the sink has enough time to sense the changes (phase four) until the next PM decision (corresponding to the next bit) takes place. Therefore, the first two phases together, the third phase, and the fourth phase can form three distinct stages of a pipeline, to accelerate POWERT communication. In Section V, we will assume such pipelined POWERT communication in deriving an upper-bound for the communication rate (or frequency), r_{MAX} as

$$(max(t_{Util} + t_{Monitor}, t_{PM}, t_{Adjust} + t_{Sense}))^{-1}.$$

PM has to manage N different entities. What distinguishes an *entity* is PM's capability to adjust its operating point independently. This implies observability and controllability on a per *entity* basis. Increasing degrees of system-wide observability and controllability refine the granularity for independent operating point adjustment. In other words, considering the same system, the span (which often translates into size) of each controllable/observable *entity* decreases. This, in turn, leads to a higher *entity* count N. As an example, N becomes 48 for POWER8 processors [8]. In this case, a single centralized PM is very likely to result in sub-optimal power management, as the latency of collecting data from all monitors and the complexity of solving a larger optimization problem both increase drastically with increasing N. This mandates a hierarchical PM as depicted in Fig. 1.

Without loss of generality, Fig. 4 provides a two-layer example, which closely mimics the general structure from Fig. 1. A global controller manages the power budget of 4 different clusters, each containing 3 entities, where the respective local controller of each cluster distributes the assigned power budget among the three entities.

The sink and the source can reside in the same cluster. Under *intra-cluster* covert communication, the source forces the local controller of the cluster to throttle the performance of other entities within the cluster according to the local power budget. The upper-bound of intra-cluster covert communication rate (or frequency), $r_{MAX,intra-cluster}$, hence becomes

$$(max(t_{Util} + t_{Monitor}, t_{PM_{local}}, t_{Adjust} + t_{Sense}))^{-1}$$

where $t_{PM_{local}}$ is the local controller's decision period.

The sink and the source can be in separate clusters, as well. Under *inter-cluster* covert communication, the source should increase the corresponding cluster's power consumption (or even, the power consumption of multiple clusters) to the point where the global controller has to limit the power budget of all active clusters, including the one containing the sink. Then, the local controller of sink's cluster adjusts the share of the power budget of each entity of the cluster accordingly, which inevitably leads to noticeable performance degradation at sink. Clearly, inter-cluster covert communication is slower than intra-cluster. An upper-bound for inter-cluster covert communication rate (or frequency), $r_{MAX,inter-cluster}$ hence becomes

$$max(t_{Util} + t_{Monitor} + t_{Comm}, t_{PM_{global}}, t_{Adjust} + t_{Sense} + t_{Comm}))^{-1}$$

where t_{Comm} is the local to global controller communication latency; and $t_{PM_{global}}$, global controller's decision period.

Table I: Evaluated systems.

	Intel Xeon E3-1505M v5	**Samsung Exynos-5422**	
μarchitecture	Skylake family	Cortex-A15 (big)	Cortex-A7(little)
# of cores (threads)	4 (8)	4 (4)	4 (4)
technology node	14 nm	28 nm	
frequency	(0.8-2.80) GHz	(0.2-2.0) GHz	(0.2-1.4) GHz
L1 Inst.	32KB 8-way	32KB 2-way	32KB 2-way
L1 Data	32KB 8-way	32KB 2-way	32KB 2-way
L2	256KB 4-way	2MB 16-way	512KB 8-way
L3	8MB 16-way	NA	

IV. Evaluation Setup

A. Evaluated Systems

As a proof-of-concept, we characterize POWERT communication on two representative commercial platforms (Table I): a laptop machine featuring an Intel Xeon E3-1505M v5 and Ubuntu 14.04.5; and an ODROID-XU4 board, featuring a Samsung Exynos-5422 with a processor based on ARM's big.LITTLE architecture [9] and Ubuntu 16.04.3 LTS. Both source and sink represent floating-point heavy applications (Section IV-B). We compile the sink and the source using GNU GCC version 4.8.4 on the laptop platform, and GNU GCC version 5.4.0 on the ODROID board, with all optimizations disabled. To maximize energy efficiency, the default mode of the operating system's power manager, *on demand*, allocates (by consulting hardware PM) the maximum possible frequency to the source (during active phases) and the sink. We do not change this default throughout the experiments to keep OS induced noise at bay, and to make sure that indeed hardware PM is making the throttling decisions. Besides, to minimize the impact of background noise (in order to better characterize the channel capacity), we turn off unnecessary OS services.

B. Malware Codes

Fig. 5 depicts the sink code, which resides on the receiving end of the covert channel. The sink constitutes an infinite floating-point heavy loop. The sink has full-fledged control over the mix and count of the executed floating-point instructions within `Run_Float()`. The sink samples the channel every `t_Sample` seconds and calls `sigalrm_handler` function. We set `t_Sample` to be 20 times smaller than the (known-to-both-sides) communication period `t_Covert`, which leads to 20 samples per bit [3]. Periodic interrupt timer overflows invoke `sigalrm_handler` function, with a period of `t_Sample`. Inside `sigalrm_handler` the sink

[3]We set this parameter empirically to maximize decoding accuracy.

```
// gets called when timer overflows.
sigalrm_handler ( ) {

  Print_to_file ( Loop_Counter );

  //sets interrupt timer.
  setitimer ( t_Sample );
}

Main ( ) {

  //sets interrupt timer.
  setitimer ( t_Sample );

  //infinite while loop
  while (1) {
    Loop_Counter++;

    //runs multiple fp instructions.
    Run_Float ();
  }
}
```

Figure 5: Sink application's code.

```
// gets called when timer overflows.
sigalrm_handler ( ) {

  Data_Index++;

  // activates the Power Virus, to send a 1
  if (Data[Data_Index]=1 && Active=0) {
    system (   kill -CONT PVirus_PID   );
    Active = 1;
  }

  // stops the Power Virus, to send a 0
  if (Data[Data_Index]=0 && Active=1) {
    system (   kill -TSTP PVirus_PID   );
    Active = 0;
  }

  //sets interrupt timer.
  setitimer ( t_Covert );
}

Main ( ) {

  //sets interrupt timer.
  setitimer ( t_Covert );

  //infinite while loop
  while (1);
}
```

Figure 6: Source application's code.

dumps the loop counter variable, `Loop_Counter`, which serves as a proxy for the rate of forward progress, to an output file. The sink extracts its GFLOPS rate periodically from `Loop_Counter`. A third-party application after receiving this file can also retrieve the data communicated over the POWERT channel.

On the other end of the POWERT channel, as explained in Section II-D, the source runs a highly power hungry application, which we will refer to as the power virus. On the Intel platform we use the latest version (v2810) of MPrime [10] to this end, specifically, the `Torture Test` mode to maximize the power consumption in active phases. Similarly, cpuburn-a7 [11] constitutes the power virus to maximize power consumption on the ARM platform. Fig. 6 depicts the code of the source application. Similar to the sink, the source sets an interrupt timer for accurate timing of communication. Every `t_Covert` seconds, which represents the communication period (known to both sides), `sigalrm_handler` function gets invoked. Inside `sigalrm_handler`, the source changes the power virus's status from running to idle or vice versa, according to the value of the next bit to be sent.

C. Communication Protocol

The source sends data in 100-bit long packets, at a rate known to the sink – every `t_Covert` seconds following Section IV-B. At the same time, each packet starts with a 5-bit preamble which is known to both sides, and to the third party receiving data from the sink, as well. The preamble specifically is used by the third party application to synchronize with the sink and to remove noise from retrieved information (Section IV-D). To synchronize with the sink at the start of communication, the source, on the other hand, sends a long bit-stream of interleaved ones and zeros followed by a short bit-stream of zeros only (both of known lengths). The sink periodically probes the channel with a shorter period than the duration of the ones and zeros in this header. During probing, when the sink receives these ones and zeros, it remains active and waits for the short bit-stream of zeros to arrive. At the end of the short bit stream of zeros, the sink starts to dump `Loop_Counter` information to the output file (according to Fig. 5), as the following bits sent by the source are the actual data packets.

D. Communication with Third Party

As explained in Section II-A, the sink has access to the network (an overt channel) to send the retrieved information to a third party application. The third party is responsible for decoding this information (which entails 100-bit long data packets) from a signal similar to the one shown in Fig. 3c. As explained in Section IV-B, the sink records 20 loop count samples per each bit sent by the source. The third party application therefore can use the median of every 20 samples to represent each bit, in order to remove noise from the data, and subsequently compare each median value to a threshold, Bit_{THR}, to find the actual bit value.

The third party application can extract retrieved information from the sink bit by bit, even if the sampling rate deviates from the expected 20 samples per bit. To this end, the third party application can simply try a few potential sample rates to decode the known preamble (the first 5 bits of every 100 bit data package) and record the corresponding error, to settle at the sample rate which results in the minimum error. A similar method applies for extracting the Bit_{THR}. The third party in turn can use this sample rate and Bit_{THR} to decode the actual information. We implement the third party application in R version 3.4.0.

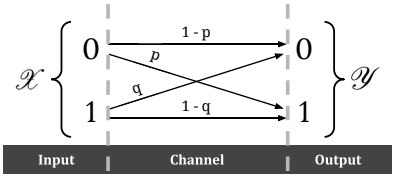


Figure 7: Binary asymmetric channel model.

E. Design Space Exploration

We explore different POWERT attack scenarios to find the maximum achievable covert communication rate on the evaluated systems, including inter- and intra-cluster communication. To minimize simulation noise, we pin the source and the sink applications to specific cores during the entire execution. We also experiment with different number of source and sink applications, and characterize the channel capacity for each case.

F. Channel Capacity by Shannon Theorem

We next look into how to derive an upper-bound for channel capacity from measurements on a given system. Following the methodology from [12], we quantify channel capacity as the maximum possible communication bit rate (in bits per second, bps) under noise according to Shannon's theorem [13]. To this end, we model the POWERT channel as a binary asymmetric channel, as shown in Fig. 7. $\mathcal{X} = \{0, 1\}$ represents the input alphabet; $\mathcal{Y} = \{0, 1\}$, the output alphabet. We send one bit at a time through the channel, which assumes a value $\in \mathcal{X}$ at the channel input; and $\in \mathcal{Y}$ at channel output. The exit and entry values are the same with probability $1 - p$ and $1 - q$, respectively, for input values 0 and 1. Otherwise, a bit flip is the case. From actual measurements we can find estimates for p, the probability of sending a 0 and receiving a 1; and q, the probability of sending a 1 and receiving a 0. POWERT channels are asymmetric by construction as p and q are not necessarily always equal.

The maximum possible channel capacity *per channel use* under noise, C, evolves as a function of p and q, over all possible probability distributions over the input alphabet [13]. *Per channel use* in this case corresponds to each bit transfer attempt through the channel. In other words, C is the theoretical maximum possible number of bits that we can send per each bit transfer attempt through a noisy channel. Hence, the theoretical maximum value of C itself is 1. We can translate C into the overall channel capacity (over multiple channel uses, i.e., bit transfer attempts) simply by multiplying by f, the frequency (or rate) of communication, which reflects the frequency of bit transfer attempts by definition. That said, p and q evolve as a function of f (as impact of noise changes with f), hence, C depends on f, as well.

For the derivation, we first estimate p and q from measurements over different values of f. We then calculate C by plugging in p and q into Shannon's model, and finally find $C \times f$ for each f considered.

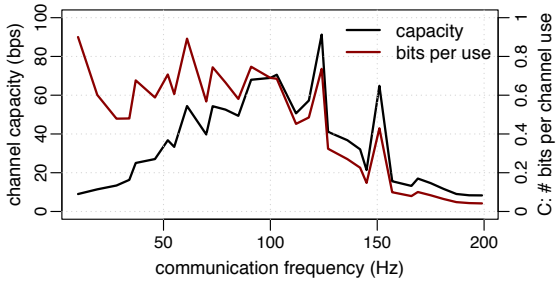


Figure 8: Single source to single sink communication.

V. Evaluation

We will next examine POWERT channel characteristics on the two representative commercial platforms.

A. Case Study I

In this section, we evaluate POWERT communication on the Intel Xeon system (Section IV-A).

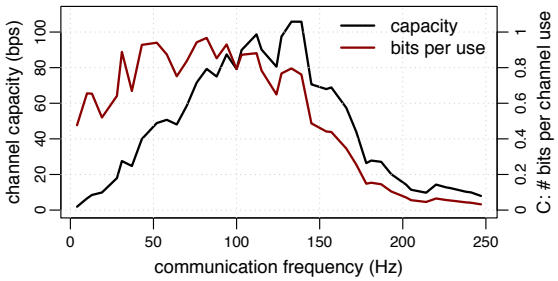
Analytical Upper Bound: We can directly apply the analytical model from Section III to this 4-core system to derive an upper bound for communication frequency, as follows (we will revisit this upper-bound using actual measurements later in this section): First we extract t_{Util} of the evaluated system. We find t_{Util} by altering the activity status of MPrime from running to idle and the other way around continuously for a fixed time window, t_{test}. Then, we count how many status changes happens during the t_{test} window, N_{Util}. We extract t_{Util} from the ratio t_{test}/N_{Util}, which is around 709.2μs for the evaluated system. We estimate $t_{Monitor}$ to be around 250μs [14]; and t_{PM}, around 1ms [15]. This system features on-chip voltage regulation with a t_{Adjust} of around 100-200ns [16]. Finally, t_{Sense} depends on the sink application's timer precision, which for the evaluated system is 1μs. Based on these parameters, the upper-bound for channel capacity, for C=1 and using $C \times f_{MAX}$ becomes

$$(max(709.2\mu s + 250\mu s, 1ms, 200ns + 1\mu s))^{-1} = 1\ Kbps.$$

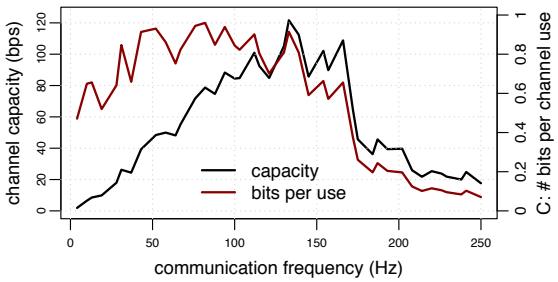
This upper bound applies irrespective of the number of active source and sink instances, as all copies operate simultaneously.

Measurement-based Characterization for Single Source, Single Sink: We next characterize covert communication through POWERT channels using the methodology from Section IV-E. We start with single source to single sink communication, each running on a separate core.

Fig. 8 shows the channel capacity on the left y-axis for different communication frequencies (x-axis). The right y-axis, on the other hand, represents C (Section IV-F): the actual number of bits we can send per each bit transfer attempt through the channel, which has a continuous range of [0,1] bits. For instance, a C of 0.5 indicates that we can send only 0.5 bit of information per each bit transfer attempt. Therefore, channel capacity (left y-axis) simply corresponds to C (right y-axis) multiplied by per bit communication frequency (x-axis), i.e., the frequency of bit transfer attempts.



(a) First Sink



(b) Second Sink

Figure 9: Single source to two sink communication.

As Fig. 8 indicates, for lower than approx. 100Hz frequencies, we see a relatively constant C of around 0.65 on average. However, for larger frequencies than 100Hz, C starts to drop since many of the OS background tasks, the main source of the background noise for POWERT channels, have similar activity rates. The peak channel capacity in this case is around 91.3 bps (which, as expected, is less than the channel capacity cap of 1Kbps we derived previously) at a communication frequency of around 124.1Hz. In this case, sink's GFLOPS rate differs by 2.56% depending on the bit value sent by the source; i.e., logic 0 (idle source) vs. logic 1 (active source).

Measurement-based Characterization for Single Source, Multiple Sink: A higher number of active cores decreases the available power headroom, and consequently, increases the likelihood of throttling. To quantify this effect, we increase the number of active cores by instantiating multiple copies of the sink. This, by construction, cannot disturb information sent by the source through the POWERT channel, in the form of activity change. Each of the sink instances stays *constantly* active, running the same floating point heavy loop. Therefore, after initialization, GFLOPS rate of each sink instance can only change, primarily, as a function of source's activity, and not of other sinks'.

Fig. 9 characterizes the channel for a single source communicating with two sinks. Each of the sinks and the source run on three separate cores. Fig.s 9a and 9b characterize POWERT communication for the two sinks separately. We observe that C of both sink instances remains above 0.8 for frequencies lower than 130Hz, and starts to fall for frequencies greater than 150Hz. One sink instance achieves a peak channel capacity of approx. 121.6bps; the other one, of approx. 105.8bps; both at a similar communication frequency of around 133.0Hz. The difference between GFLOPS levels of each sink instance becomes approx. 2.67%, which is 4.6% larger than the single sink outcome. As a larger gap between GFLOPS levels of each sink (leading to easier decoding by the third party application) makes communication more robust to noise, we observe the peak channel capacity at a higher communication frequency in this case. At the same time, C at low frequencies is higher on average when compared to the single sink case.

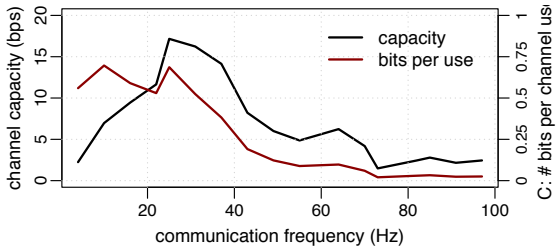
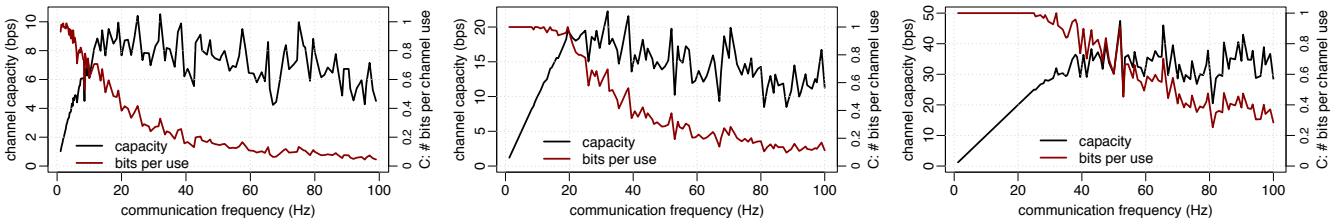


Figure 10: Two source to single sink communication.

When we increase the number of sink instances to 3, malware code (including the source) occupies all 4 cores of the evaluated system. In this case, we observe a significant drop in C, irrespective of the communication frequency. This is because, *full-load* pushes the system to its limits – be it power or thermal budget, which triggers throttling. As all cores become active at the same time, PM has to enforce a strict power budget across the board. In the end, the source can only modulate the power headroom of the sink instances by controlling its own activity, hence power consumption. Under *full-load*, the source does not have much room left to control its own consumption to start with. Generally, even if the malware code does not occupy the entire system, *full-load* can be the case due to utilization by other applications running on the system. Under *full-load*, covert communication through POWERT channels becomes, by construction, infeasible.

Measurement-based Characterization for Multiple Source, Single Sink: We conclude the first case study with channel characterization when multiple instances of the source send the exact same data to a single sink. Having multiple instances of power virus getting activated and deactivated simultaneously increases the gap between the power demand when the source instances are sending a logic 0 vs. when the source instances are sending a logic 1. This can result in a more pronounced difference between the two GFLOPS levels (corresponding to logic 0 and logic 1 respectively) at the sink.

Fig. 10 characterizes the channel for two source instances communicating with a single sink, each running on a separate core. The peak channel capacity reaches approx. 17.2bps at a communication frequency of around 24.9Hz, which is significantly lower compared to the single source scenario. Although the difference between GFLOPS levels at the sink is about 15.2% on average, channel capacity remains relatively



(a) single source (b) two sources (c) three sources

Figure 11: Channel characterization within little cluster: single (a), two (b), three (c) source to sink communication.

low, since – similar to under *full-load* – we observe more information loss in the channel due to throttling. As the MPrime (the power virus at source) is extremely power hungry, even having two instances of MPrime running on the system (along with an instance of the floating point heavy sink application) causes throttling. Therefore, increasing number of source instances does not improve the POWERT communication rate on this platform.

B. Case Study II

In this section, we evaluate POWERT communication on the ARM system (Section IV-A).

Analytical Upper Bound: For this 8-core heterogeneous platform, as well, we can apply the analytical model from Section III to estimate an upper bound for communication rate, and hence, channel capacity: First we extract t_{Util} of the evaluated system, mimicking the same method as the first case study (Section V-A). We keep activating and deactivating cpuburn-a7 (the power virus at source in this case) continuously for a fixed time window, t_{test}. Then, we count the number of status changes during the t_{test} window, N_{Util}, and calculate t_{Util} using t_{test}/N_{Util}. For this platform, t_{Util} is around 2.1ms. We estimate $t_{Monitor}$ to be around 250μs [14]; and t_{Sense}, the platform's timer precision, to be around 1μs. Using conservative estimates for the rest of the parameters which were not explicitly reported in the literature, $C \times f_{MAX}$, for $C = 1$, becomes 0.42Kbps.

We will next look into POWERT channel characteristics using the methodology from Section IV-E.

Measurement-based Characterization for Little-to-Little POWERT Communication: We first characterize POWERT channels within the little cluster only. Fig. 11a provides channel characteristics for single source to single sink communication, where the source and the sink run on distinct little cores. For lower communication rates (x-axis), C, the actual number of bits sent per channel use (right y-axis), is close to 1, indicating almost perfect communication. However, as the communication rate increases and matches background noise, C drops with a sharp slope. We observe a peak channel capacity of 10.5bps for a communication rate of around 32.0Hz, in this case.

For the single source case, the gap between the GFLOPS levels of the sink (corresponding to logic 0 vs. logic 1) is around 0.13%. To increase this gap and consequently, increase the POWERT communication channel capacity, we can increase the number of sources. Fig.s 11b and 11c demonstrate channel capacity for two and three source instances running on the little cores, respectively. We observe that by increasing the number of source instances, POWERT channel capacity increases notably.

For two sources (and a single sink), we observe a peak channel capacity of 22.3bps at a communication frequency of around 32.0Hz (Fig. 11b). For communication rates below 20Hz, the channel becomes almost noise-free (i.e., $C \approx 1$). For three sources (and a single sink), on the other hand, communication remains almost noise-free for communication rates below 25Hz (Fig. 11c). In this case, the peak channel capacity is around 47.5bps. The gap between GFLOPS levels of the sink (when all sources are active vs. inactive) for the two and three source cases are 0.25% and 0.49% respectively. Unlike the first case study, we do not see throttling events, as POWERT communication in this scenario affects only the 4 little cores of the processor (out of 8). As result the communication rate is significantly higher for a higher number of source instances.

We also explore POWERT communication for multiple sinks and a single source. In this case, the overall power consumption is higher, but still, primarily the single source's activity modulates the power headroom of the sinks. We observe that for two and three sinks the gap between the GLOPS levels stays in the same range, 0.12% and 0.11%, respectively. Peak channel capacity assumes a lower value than the single sink case (10.5bps) for both – 7.2bps and 8.4bps, respectively – which indicates that multiple sinks do not improve channel capacity.

Measurement-Based Characterization for Big-to-Big POWERT Communication: We next analyze POWERT communication when both the sink and the source only use big cores. Fig. 12 provides the characteristics for a single source and a single sink. We observe that C remains close to zero for almost any communication rate. In other words, POWERT communication is not feasible. However, as we increase the number of sources, we observe that C increases. We observe the peak channel capacity of around 5.8bps at a communication rate of near 48.5Hz, for three sources running on big cores at the same time. Although the communication

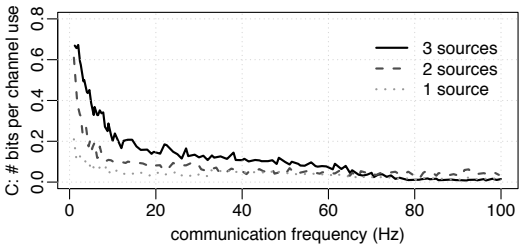
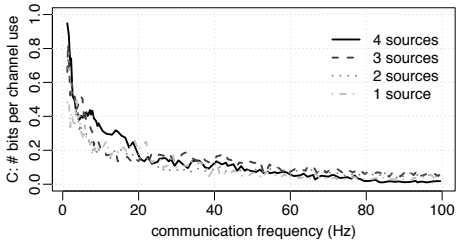
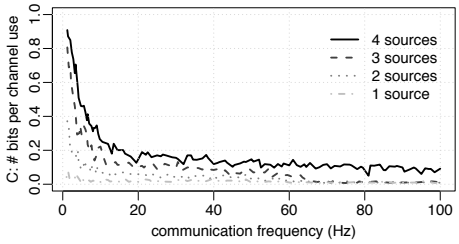


Figure 12: Channel characterization within big cluster.



(a) big-to-little



(b) little-to-big

Figure 13: Inter-cluster channel characterization.

rate is still very low, this analysis provides proof of existence of POWERT channels on big cores. We should also note that the power virus at the source, cpuburn-a7, is optimized for Cortex-A7 (little) cores, therefore, we expect a much higher channel capacity in the big cluster for a properly tailored power virus. Otherwise, as it was the case for the little cluster, we do not observe practical benefits by increasing the number of sink applications.

Measurement-Based Characterization for Big-to-Little & Little-to-Big POWERT Communication: We next look into inter-cluster POWERT communication, where the sink and the source instances run on cores in different types of clusters (i.e., big or little). First, we run source(s) on big cores, and a sink on a little core. As Fig. 13a depicts, C increases when we have multiple source instances running on the big cores at the same time. We observe a peak channel capacity of 5.5bps at a communication frequency of around 40.5Hz when four source instances are running on all four big cores. Fig. 13b provides the symmetric analysis for having different number of sources running on little cores, and a sink on a big core. Similar to the previous case, we observe that more number of sources increases C. This renders a peak channel capacity of 8.7bps for four sources at a communication rate

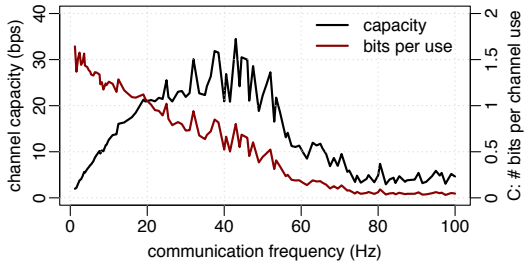


Figure 14: 2-bit encoding on little cores.

of around 61.0Hz. While the channel capacity for this type of inter-cluster POWERT communication is not as large as the (intra-cluster) little-to-little communication, this analysis indicates that inter-cluster channels exist, and hence need to be considered when designing effective counter-measures.

Multi-level Encoding on Little Cores: We next characterize POWERT communication for a more effective data encoding: Using four (instead of two) distinct levels of source's activity, enabling us to send two bits of information per channel use, instead of one (as we covered so far). While we do not have full-fledged control over the activity level of the power viruses (MPrime and cpuburn-a7, which form the workload of the source), we can enforce different levels of activity by changing the number of active sources. For example, to send a binary value of "10", we can activate two sources. In other words, we can encode a two-bit value into the number of active sources. This way, we can send more than one bit in each channel use, which can improve channel capacity.

As a proof-of-concept, we apply this multi-level encoding scheme to little cores, where we have observed the best channel profile when using multiple sources for POWERT communication (Fig. 11). As depicted in Fig. 14, we observe a peak channel capacity (left y-axis) of 34.5bps at a communication rate (x-axis) of around 43.0Hz in this case. While at low frequencies we observe around 1.5bits sent per channel use (right y-axis), C quickly goes down to near zero (i.e., no information sent) at frequencies above 70Hz. This is not unexpected as we have observed a similar trend for only one or two sources running in Fig.s 11a and 11b, respectively. Hence, we cannot reach a channel capacity as high as the three sources case alone, as depicted in Fig. 11c.

VI. CounterMeasures

A. Avoiding Power Budget Sharing

One way to avoid POWERT attacks is to assign a separate, fixed and safe power budget to each entity and thereby to exclude any power budget sharing. In this case, independent local power management is necessary to keep power consumption of each entity under its respective, constant power budget. Let us assume that an individual power budget of E_i applies per entity. By construction, E_i is lower than the maximum possible consumption of each entity, P_i, as $\sum_i E_i \leq power\ budget$ must be the case. Consequently, even if an entity is the only active entity in the system, it

will not be able to operate at a higher performance point which would consume more power than E_i. This can lead to significant performance loss and degrade overall power efficiency.

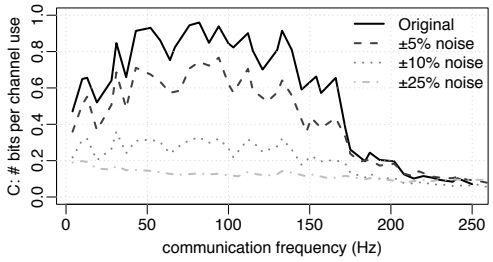
To quantify the overhead of this countermeasure, i.e., the performance loss caused by fixing (and thereby practically decreasing) per-entity power budget, we run each power virus on all cores of its respective platform and compare the performance to when only one core is running the power virus. When all cores are active, each core inevitably consumes less power at the peak (corresponding to E_i), to meet the overall system-wide power budget. On the other hand, when only one core is active, the active core can run at maximum performance, and consume by itself the entire effective budget for all cores being active. We use the performance difference under both scenarios as a quantitative estimate for the overhead of this countermeasure. Overall, we observe a performance degradation of over 30.1% for the Intel; and 7.8%, for the ARM framework. The degradation for the first platform is more meaningful, as the corresponding power virus is more effective in pushing the system to its limits. We conclude by noting that this countermeasure incurs a high performance penalty which may not always be acceptable.

B. Operating Frequency Randomization

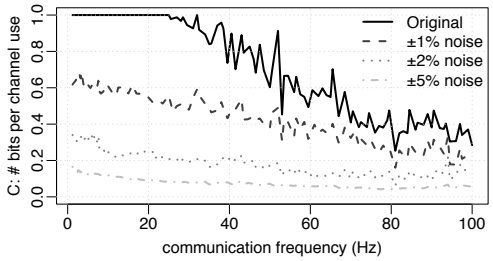
As reported in Section V-A, the difference between the sinks' GFLOPS levels for the first case study, when communicating at the peak rate, is around 2.7%. The same gap between GFLOPS levels is around 0.5%, for the fastest covert communication on the second case study, as Section V-B reveals. Hence, on both systems we observe a slim gap that needs to be carefully sensed to be able to accurately decode the leaked information.

Based on this observation, one way that PM can limit the bandwidth for POWERT communication is by imposing random noise on the GFLOPS signal, simply by adding random noise to the operating frequency of each core. In other words, when PM finds the optimal operating frequency for a core based on power demand, workload behavior, and other parameters, it can add random noise (e.g., in the range of [-2,+2]%) to it before actually tuning the respective core's frequency. While this countermeasure would inevitably degrade power-efficiency and performance, it can significantly complicate the decoding process, potentially to a point where covert communication becomes impossible.

Fig. 15 depicts how adding random uniform noise to GFLOPS affects C, on both platforms in the attack scenario where we observe the highest channel capacity. To limit C to less than 0.1 bits per channel use, we have to add a random uniform noise with the magnitude of [-25,+25]% of the GFLOPS signal on the Intel platform. On the other hand, this magnitude is around [-5,+5]% on the ARM platform, since, as reported in Section V-B, the gap between GFLOPS levels is smaller on the ARM platform.



(a) Intel



(b) ARM

Figure 15: Impact of adding noise to GFLOPS signal on C.

C. Slowing Down Communication

Section III provides an analytical model to estimate an upper-bound for POWERT communication rate, as a function of different system parameters. We can utilize this model to find ways to lower the upper-bound, to a desirable safe level. For instance, one easy way to slow-down POWERT communication is by increasing the decision-making period of the power manager, t_{PM}. While this, as well, degrades overall power efficiency, it can effectively limit POWERT communication rate, and thereby, the channel capacity. We can manipulate other parameters, as well, all at the cost of perturbing power management and consequently, degrading overall power efficiency.

VII. Related Work

Covert channels: Resource sharing, be it in hardware or software, is inevitable for power or area efficiency, however, almost exclusively brings up security concerns. Since one of the first mentions of covert channel attacks in 1973 [17], a variety of covert channels have been revealed [18], [5], [19], [20], [4], [3], [21], [22], [23]. The vast majority of these works covers cache-based covert channels [3], [18]. Not only higher level caches, but also the main memory and functional units shared by different threads under simultaneous multithreading (SMT) can be subject to information leakage through covert channels [17]. Recent work has also shown how thermal sensors (as a key component of on-chip thermal management) can enable similar covert communication [5]. Thermal effects can also lead to clock skew changes, which attackers can exploit for covert communication [19]. Covert channels which need special privilege, for instance to access hardware monitors like thermal sensors, can be blocked simply by restricting access to those resources. This does not apply to POWERT

attacks, since no special privilege is needed to perform these attacks. Other hardware resources such as the memory bus [4], random number generator [20], magnetic field sensors [22], USB charging cable [23], and general purpose graphics processing units [21] are vulnerable, as well. The condition that the sender and receiver need to reside at the same place is necessary for covert communication in earlier studies, while recent studies demonstrate that this requirement can be relaxed if the timing of sender activities can be measured remotely [17]. Similar to many of these covert channels including cache-based covert channels, blocking POWERT attacks inevitably degrades system performance and power efficiency (as explained in Section VI). It becomes even more challenging as power management is getting more crucial in preserving power efficiency of even more power limited platforms of the future.

On the modeling side, Hunger et al. proposed a simple mathematical abstraction to capture common characteristics of all microarchitectural channels [12]. While the model is applicable to many contention-based microarchitectural channels, it does not directly apply to POWERT channels. This is because the model assumes that probing the channel perturbs the data. However, in POWERT channels, multiple receivers can listen to the covert channel, without affecting the transmitted data itself, as shown in Section V.

Power management vulnerabilities: Power management vulnerabilities can result in a variety of security issues. For example, JayashankaraShridevi et al. analyze two types of attacks enabled by hardware Trojans embedded in the power management unit (PMU) of a mobile system on chip [24]. The first attack leads to higher operating voltages than necessary. The second one delays the activation of power-gated blocks. In both cases, power efficiency degrades. Tang et al. demonstrate another type of vulnerability due to DVFS [25], where an attacker can enforce lower (higher) than safe voltages (frequencies) to induce timing errors. Physical access is not necessary, as software controls voltage regulators and phase-locked loops (PLLs). The authors show how to infer 128-bit AES keys via overclocking the processor. Zhang et al. recently proposed a mitigation technique for such power-management based fault injection attacks, by dynamically "blacklisting" unsafe operating points [26]. The recently revealed DVFS Channel [27] exploits the fact that a core's frequency can be dynamically controlled using DVFS (and not that cores share the same power budget). This type of attack is easier to block by limiting the access to the files containing current frequency information, while POWERT attacks do not need any privilege, making them harder to block. Finally, due to low rate of updates to frequency information files, DVFS Channel demonstrates much lower bit rates, compared to POWERT. PMU Trojan [28] is similar to DVFS Channel, but attacks happen in hardware. Contrary to POWERT communication, PMU Trojan, as well, does not exploit the fact that cores share the same power budget. Similar to DVFS Channel, this attack also can be blocked by limiting access to operating frequency information of the cores, on the receiving side. Besides, POWERT attacks do not rely on hardware Trojan, and can be performed on regular PM hardware. The available instantaneous power budget itself represents a shared resource, therefore, power management practically entails finding the optimal allocation of the power budget among active tasks of execution. To the best of our knowledge, our study is the first to cover covert channel communication enabled by power headroom modulation, without the need for any sort of privileged access to shared hardware or software resources.

VIII. Conclusion

As power-limited computing platforms of today are getting more and more observable and controllable to facilitate sophisticated power management operating at very fine granularity in both space and time, a novel class of covert communication by power headroom modulation becomes possible. In this study, we characterize this novel class of covert channels triggered by power management vulnerabilities. Only the characterization of these vulnerabilities can enforce security as a power management design parameter as important as (or more important than) performance, power, or accuracy.

Acknowledgment

This work was supported in part by the National Science Foundation CAREER Award under Grant CCF-1350451, in part by the National Science Foundation/Semiconductor Research Corporation SaTC:STARSS Award under Grant CNS-1715286, and in part by a Cisco Research Award. The authors would like to thank numerous anonymous reviewers, and particularly Meisam Razaviyayn for constructive feedback.

References

[1] M. Horowitz, "Computing's Energy Problem (and what we can do about it)," *Keynote at International Conference on Solid State Circuits*, 2014.

[2] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: memory power estimation and capping," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pp. 189–194, IEEE, 2010.

[3] C. Percival, "Cache missing for fun and profit," 2005.

[4] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: High-speed covert channel attacks in the cloud.," in *USENIX Security symposium*, pp. 159–173, 2012.

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

[6] P. Bose, A. Buyuktosunoglu, J. A. Darringer, M. S. Gupta, M. B. Healy, H. Jacobson, I. Nair, J. A. Rivers, J. Shin, A. Vega, *et al.*, "Power management of multi-core chips: Challenges and pitfalls," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 977–982, EDA Consortium, 2012.

[7] H. Ritzdorf, "Analyzing Covert Channels on Mobile Devices," *M.S. Thesis, ETH*, 2012.

[8] Z. Toprak-Deniz, M. Sperling, J. Bulzacchelli, G. Still, R. Kruse, S. Kim, D. Boerstler, T. Gloekler, R. Robertazzi, K. Stawiasz, *et al.*, "5.2 distributed system of digitally controlled microregulators enabling per-core dvfs for the power8 tm microprocessor," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 98–99, IEEE, 2014.

[9] P. Greenhalgh, "Big. little processing with arm cortex-a15 & cortex-a7," *ARM White paper*, vol. 17, 2011.

[10] *MPrime*. https://aur.archlinux.org/packages/mprime/.

[11] "cpuburn-a7 for arm cortex-a7." https://github.com/ssvb/cpuburn-arm/blob/master/cpuburn-a7.S. Accessed: 2010-09-30.

[12] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari, "Understanding contention-based channels and using them for defense," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 639–650, IEEE, 2015.

[13] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[14] "POWER8 On Chip Controlle: Measuring and Managing Power Consumption: https://hpm.ornl.gov/Archives/HPM15/documents/HPM2015_Rosedahl.pdf."

[15] J. Doweck, W.-F. Kao, A. K.-y. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-generation intel core: new microarchitecture code-named skylake," *IEEE Micro*, no. 2, pp. 52–62, 2017.

[16] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," pp. 123–134, February 2008.

[17] J. Szefer, "Survey of microarchitectural side and covert channels, attacks, and defenses," *IACR Cryptology ePrint Archive*, pp. 1–28, 2016.

[18] Z. Wang and R. B. Lee, "Covert and side channels due to processor architecture," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pp. 473–482, IEEE, 2006.

[19] S. J. Murdoch, "Hot or not: Revealing hidden services by their clock skew," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 27–36, ACM, 2006.

[20] D. Evtyushkin and D. Ponomarev, "Covert channels through random number generator: Mechanisms, capacity estimation and mitigations," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 843–857, ACM, 2016.

[21] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and characterizing covert channels on gpgpus," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 354–366, 2017.

[22] N. Matyunin, J. Szefer, S. Biedermann, and S. Katzenbeisser, "Covert channels using mobile device's magnetic field sensors," in *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 525–532, 2016.

[23] R. Spolaor, L. Abudahi, V. Moonsamy, M. Conti, and R. Poovendran, "No free charge theorem: A covert channel via usb charging cable on mobile devices," in *Applied Cryptography and Network Security - ACNS 2017*, pp. 83–102, 2017.

[24] R. JayashankaraShridevi, C. Rajamanikkam, K. Chakraborty, and S. Roy, "Catching the flu: emerging threats from a third party power management unit," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2016.

[25] A. Tang, S. Sethumadhavan, and S. Stolfo, "Clkscrew: exposing the perils of security-oblivious energy management," in *USENIX Security Symposium*, 2017.

[26] S. Zhang, A. Tang, Z. Jiang, S. Sethumadhavan, and M. Seok, "Blacklist core: machine-learning based dynamic operating-performance-point blacklisting for mitigating power-management security attacks," in *Proceedings of the 23rd IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, 2018.

[27] M. Alagappan, J. Rajendran, M. Doroslovački, and G. Venkataramani, "Dfs covert channels on multi-core platforms," in *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*, pp. 1–6, IEEE, 2017.

[28] M. N. Islam and S. Kundu, "Pmu-trojan: on exploiting power management side channel for information leakage," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, pp. 709–714, IEEE Press, 2018.