This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2018.2860006, IEEE Computer Architecture Letters



# A New Class of Covert Channels Exploiting Power Management Vulnerabilities

S. Karen Khatamifard* Longfei Wang† Selçuk Köse† Ulya R. Karpuzcu*
* University of Minnesota † University of South Florida

**Abstract**—Effective runtime power management requires hardware activity to be tracked at a very fine granularity in both space and time in order to meet diverse workload performance requirements within a tight power budget. As the avaliable instanteneous power budget itself represents a shared resource, this practically translates into finding the optimal allocation of the power budget among active tasks of execution. Covert communication over a previously unexplored class of channels thereby becomes possible, which forms the focus of this paper.

## 1 Problem Statement & Threat Model

*Covert* channel attacks involve two malicious entities which are not allowed to communicate legitimately through *overt* channels. Fig. 1 depicts an example. These entities can represent hardware blocks (such as cores or functional units) or pieces of software sharing hardware resources [1]. The transmitting end, the *source*, has access to sensitive information (such as a secret key), however, not to any overt channel for information transfer. The receiving end, the *sink*, has access to an overt channel for potential data transfer, but not to sensitive information. Therefore, by having the data transmitted over a covert channel by the source, the sink can get access to sensitive information (to possibly communicate it to third parties subsequently). This covert communication, by construction, is not obvious to other hardware or software entities within the same system.

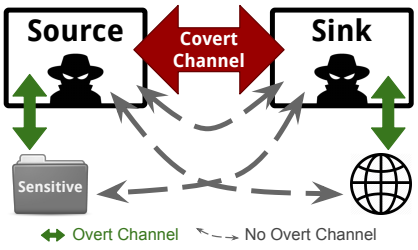



Fig. 1: Threat model.

Modern power-limited computing platforms feature sophisticated power management – spanning several software and hardware layers of the system stack – in order to meet diverse runtime performance needs within the stringent power budget. Controlling power consumption necessitates hardware activity to be closely tracked and evaluated at a very fine granularity in both space and time. Activity monitors in the form of sensors or performance counters dispersed across chip serve the purpose. At the same time, such monitors are exposed to the software layers.

The abundance of specialized activity monitors, their exposure of control to software layers, and the need for tight control to avoid power budget overshoots, unfortunately, render power delivery and management particularly vulnerable to covert communication over a previously unexplored class of covert channels. In this work, we characterize this novel class, communication over which closely follows the generic model from Fig. 1.

The avaliable instanteneous 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. As explored previously, many other shared resources such as caches [2], memory bus [3] or thermal sensors/monitors [4] are vulnerable to covert channel attacks. To the best of our knowledge, our study is the first to cover covert communication enabled by power management vulnerabilities, without the need for any sort of priviledged access to shared hardware or software resources.

Power management vulnerabilities can give rise to other types of security problems, as well: For example, JayashankaraShridevi et al. analyzed two attacks by a malicious third party power management unit embedded in a mobile system on chip, which enforces operation at sub-optimal voltages and frequencies, and thereby degrades the power efficiency [5]. Tang et al., on the other hand, demonstrated how to infer 128-bit AES keys by fault injection via overclocking the processor on a real machine [6].

## 2 Covert Communication by Power Headroom Modulation

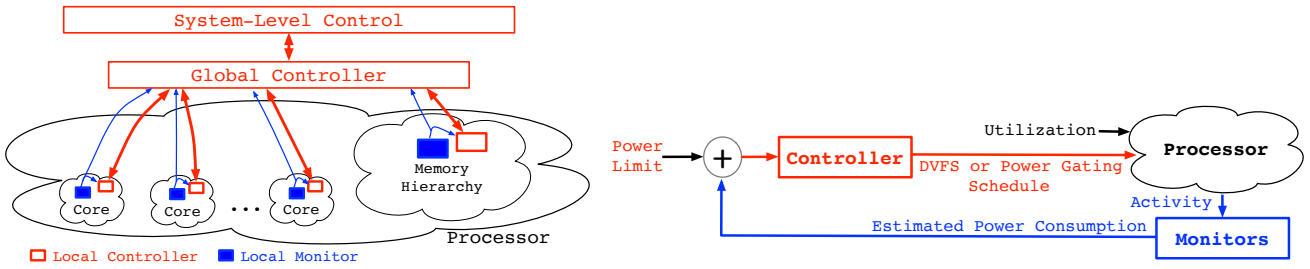
Power management serves two main purposes: (1) Providing (at least statistical) guarantees that the system does not exceed the power budget; (2) Optimal allocation of shared processor resources in meeting possibly conflicting performance requirements. Fig. 2a provides a generic overview: a global controller (often directed by system-level control at the hardware-software interface) orchestrates a dispersed set of local controllers across chip, which modulate the operating point by evaluating continuously monitored local activity. Local controllers feature a much faster response time to locally-confined power emergencies. When directed by system-level control to adjust chip-level power consumption, the global controller reallocates the power budget among the local controllers by enforcing local operating point adjustments.

The control loop from Fig. 2b applies to both global and local controllers from Fig. 2a, without loss of generality. The controller accepts a power limit as the input and modulates the operating

*Manuscript submitted: 27-Mar-2018. Manuscript accepted: 29-Jun-2018. Final manuscript received: 11-Jul-2018. This work is supported in part by the NSF CAREER Award under Grant CCF-1350451, in part by the NSF Award under Grant CNS-1715286, and in part by the Cisco Systems Research Award.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2018.2860006, IEEE Computer Architecture Letters






(a) Macroscopic view (b) Microscopic view

Fig. 2: Hierarchical on-chip power management and control (adapted from [7]).

point accordingly to maximize (overall) performance subject to the (input) power limit. Typical knobs for control are the operating voltage and frequency (modulated via dynamic voltage frequency scaling, DVFS) or selective cut-off of idle resources from the power supply (power gating).

Putting it all together, the global controller imposes a power budget on the local controllers on-chip, as dictated by the system-level control. The local controllers, in response, may adjust the local voltages/frequencies or power gating schedules accordingly, which in turn get closely monitored to prevent power budget overshoots.

Following the threat model from Fig. 1, let us assume that a source application with access to sensitive information resides on-chip along with a sink application. We can envision both applications running on a mobile chip, possibly along with other applications. For example, the source can be a contacts manager, and the sink, a weather application [1]. The adjustments to voltage and frequency usually happen periodically, subject to the speed of on-chip activity monitors (along with the power distribution network, voltage regulators, and the clock distribution network) [7]. This is because typical monitors rely on periodic time-sampling. The period $P$ – usually in the order of several processor clock cycles – is by construction known to the source and the sink. At the same time, typical power management algorithms are of predictive nature and try to learn from a history of monitored activity which are $P$ apart in time from each other.

If the source and the sink are the only applications running on the system, the source can easily modulate the available power budget to (i.e., the *power headroom* of) the sink to encode digital data as follows: To encode logic 1, the source can run a very power hungry virus periodically, with a period (known to both sides) equal to an integer multiple of $P$. In this manner, the source can reach its power budget limit, and taint its local activity history to trick the global controller. The sink can distinguish this case from no activity (hence, low power consumption) at the source, as the controllers will very likely trigger emergency power throttling at the sink – to prevent the instantaneous (system-wide) power consumption exceed the available budget as a result of excessive consumption at the source. Under no activity at the source, on the other hand, such throttling events at the sink become much less likely. In this particular example, the sink decodes an almost constant, sizable positive power headroom, as logic 0; and any power headroom change (periodic at a fixed frequency, as a result of source's activity) as logic 1.

The existence of other applications sharing the same processor resources can challenge such covert communication between the source and the sink. Sharing inevitably introduces noise to the covert channel. However, power consumption of sharing applications can also cause a faster onset of throttling at the sink as the source is sending a logic 1. This is because the sink quantifies its available power headroom, i.e., decodes the source's message bit by bit, by tracking the frequency of throttling events.

## 3 Proof-of-Concept Attack Scenario

Following the threat model from Fig. 1, we assume that the source and the sink occupy two different cores in a multi-core system. The source encodes the data to be communicated (to the sink) as fixed-frequency changes in its activity. The controllers from Fig. 2 naturally track such activity changes to adjust the operating point (i.e., voltage and frequency) of all cores in the system (including the sink) accordingly. Therefore, changes in the activity of the source can easily lead to changes in the available power budget to (hence, performance of) the sink. The sink can in turn track such changes using various performance metrics. We will use Floating Point Operations Per Second (FLOPS), and show how, by decoding the changes in its FLOPS, the sink can extract the information sent by the source.

The source encodes bits by modulating its activity, specifically by going to sleep and waking up periodically. The period (or frequency) of such activity changes is fixed and known to the both sides. The source becomes inactive (for a fixed duration) to send a "0"; periodically active, to send a "1". During the active phase, the source executes a highly compute-intensive task. During the inactive phase, the source remains idle in order to maximize the difference in power demands between the active and inactive phases. To start the communication, the source sends a long bitstream of ones followed by a short bitstream of zeros (of known lengths). The sink uses this header to synchronize with the source.

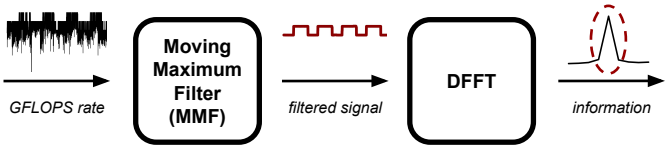
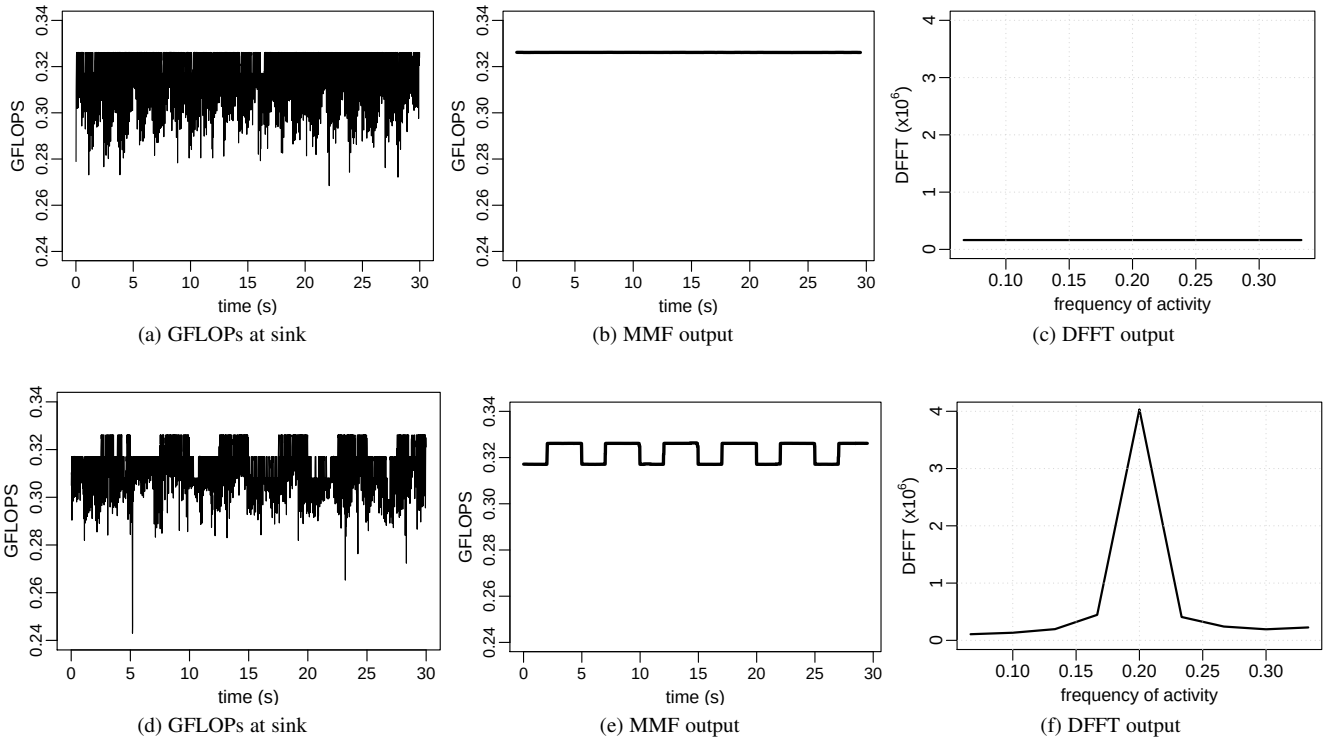



Fig. 3: Information retrieval at the sink.

The sink recognizes the header string by periodically probing the channel. The probing period is shorter than the duration of the "1"s in the header. Upon recognizing the header, the sink starts listening to the actual activity in the channel, by running a floating-point heavy loop. By tracking loop iterations, the sink samples its own Giga FLOPS (GFLOPS) rate. As an example, the sink can extract the leaked information from its own GFLOPS rate by simply following the methodology from Fig. 3: First step is removing noise from the GFLOPS rate by using a Moving Maximum Filter (MMF). MMF is a low-pass filter which assigns

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2018.2860006, IEEE Computer Architecture Letters






(a) GFLOPs at sink (b) MMF output (c) DFFT output

(d) GFLOPs at sink (e) MMF output (f) DFFT output

Fig. 4: Information retrieval under two scenarios: the source is not active (top row); the source is active (bottom row).

the maximum value (of all points) around a given point $p$ to $p$. Among many alternatives, we pick MMF based on the shape of the signal constituting temporal changes in GFLOPS, as shown in Fig. 4d. After filtering the noise, the sink takes the discrete Fourier Transform (DFFT) of the signal. Next, in frequency domain, the sink checks for a peak at the activity frequency of the source – recall that the activity frequency is fixed, known to both parties, and captures the sleep/wake-up period of the source (and not the actual operating frequency). If a peak exists, the source communicated a "1", otherwise, a "0". To distinguish a "1" from potential noise, a threshold in the frequency domain can help. To extract this threshold, we profile the DFFT output when the source is inactive, to capture the impact of background noise, which evolves as a function of the workload. Only if the peak exceeds the threshold, the sink records a "1".

## 4 Evaluation Setup

As a proof of concept, we next characterize covert communication between the source and the sink on a laptop machine featuring an Intel Xeon E3-1505M v5 processor (Table 1) and Ubuntu 14.04.5. To maximize power efficiency, as both the source and the sink are floating-point heavy, the operating system's power manager, *governor* (in its default mode *on demand*) tries to allocate the maximum possible frequency (subject to the available power budget) to the source (during active phases) and the sink.

On the receiving side of the covert channel resides the sink application constituting a floating-point heavy loop, which is able to periodically measure its own GFLOPS rate. Using system interrupts, the sink periodically dumps the loop counter to a file, which is then used to decode leaked information. We compile this application using GNU GCC version 4.8.4, with all optimizations disabled. On the source side resides, as explained in Sect. 3, a highly power hungry application. We use the latest version

| | |
|---|---|
| **number of cores** | 4 (8 threads) |
| **technology node** | 14nm |
| **frequency** | 2.80GHz (3.70GHz Turbo) |
| **L1 instruction** | 32 KB 8-way set associative |
| **L1 data** | 32KB 8-way set associative |
| **L2** | 256 KB 4-way set associative |
| **L3 (shared)** | 8 MB 16-way set associative |

TABLE 1: Baseline processor.

of MPrime [8] (v2810) to this end, specifically the `Torture Test` mode to maximize the power consumption in active phases. We periodically pause or resume MPrime to mimic sleep/wake-up phases of the source, by `kill -TSP` and `kill -CONT`, respectively.

To minimize simulation noise, we pin both the source and the sink applications to cores during the entire execution, and we experiment with various core combinations. We sweep the activity frequency of the source (running MPrime) in the range of 10, 2, 1, and 0.2Hz, respectively, to quantify the sensitivity of the covert channel capacity to the source activity. We empirically set the frequency at which the sink measures its own GFLOPS rate to at least one tenth of the activity frequency of the source. We implement the information retrieval steps from Fig. 3 in MATLAB R2016a.

To find a threshold for DFFT in order to distinguish between the signal (i.e., a logic 1) and the noise (i.e., a logic 0 – as the source is idle in this case, only background noise would be visible to the sink), we characterize the background noise by running a typical workload including web browser, file explorer, text editor, terminal, etc. Such a workload, as well as kernel tasks, form the main sources of noise in this communication channel. As a proof-of-concept, we only analyze covert communication under typical use and leave further exploration of channel capacity under worst-case background noise to future work. On a system running numerous power hungry applications with activity rates similar to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2018.2860006, IEEE Computer Architecture Letters



the source's, we expect to see lower channel capacity.

## 5 Evaluation

We start the evaluation with GFLOPS characterization on the sink side, considering two scenarios: the source being inactive (Fig. 4a–4c) vs. the source being periodically active (Fig. 4d– 4c). For this experiment, the source's activity frequency is 0.2Hz. Fig. 4a (when the source is inactive) and Fig. 4d (when the source is periodically active) depict the actual GFLOPS measured at the sink. All figures capture 30 seconds of activity, without loss of generality. Fig. 4d reveals how the peak GFLOPS change with the source's activity, periodically. Otherwise, as Fig. 4a indicates, the peak GFLOPS remains mostly constant when the source is not active. For both cases, the moving maximum filter effectively eliminates noise from the respective GFLOPS signals, as demonstrated by Fig. 4b and Fig. 4e. Comparing Fig. 4f to Fig. 4c, we can identify the peak at the source's activity frequency of 0.2Hz. As Fig. 4c reveals, the peak vanishes if the source is inactive. The DFFT output increases by 24.7× when the source becomes periodically active.

Fig. 4 provides a proof of concept for covert communication exploiting power management vulnerabilities. The analysis spans 30 seconds of execution. In other words, we need 30 seconds of communication to extract one bit of information, which translates into nearly 0.033 bits per second. Although a relatively high confidence applies when it comes to the retrieval of the leaked information at the sink (Fig. 4f), the bit rate is low. We next look into how much we can increase the channel bit rate by trying to extract information from GFLOPS data over a shorter time window, considering the accuracy of communication.

We pin the sink and the source applications to two cores, and experiment with different core combinations for tens of minutes to collect GFLOPS data at the sink. We repeat the same experiments as before, just for a longer time window. We adjust the DFFT threshold to characterize the background noise in the channel when the source is inactive. For each time window (with each implying a specific bit rate), we pick 1000 samples where the source is periodically (in)active. Following the methodology of Fig. 3, we first find the DFFT of the GFLOPS signal for the cases when the source is not active, to quantify the background noise. The maximum DFFT among all (background noise) samples gives us a worst-case threshold, which we use to distinguish between the actual signal (due to the source's activity) and the (background) noise. Then, we compare the DFFT output of all samples (when the source is periodically active) to this threshold to detect a logic "1". We quantify communication accuracy as the ratio of the samples (with the source being active) which have a DFFT larger than the threshold.

Fig. 5 depicts how the communication accuracy degrades as the bit rate increases. The X-axis captures bit rate (bits per second); the Y-axis, communication accuracy (%). This figure corresponds to an activity frequency of 0.5Hz at the source, at which we recorded the fastest error-free communication. Fig. 5 reflects the worst case among all different core pinning options. Communication accuracy starts to degrade when we try to send more than 0.17 bits per second, as captured by the dashed line.

| Activity Frequency (Hz) | 0.2 | 0.5 | 1 | 5 |
|---|---|---|---|---|
| Communication rate (bits/s) | 0.13 | 0.17 | 0.09 | 0.02 |

TABLE 2: Maximum error-free communication rate for different source activity frequencies.

We sweep the activity frequency at the source to extract the fastest error-free (i.e., of 100% accuracy) communication.

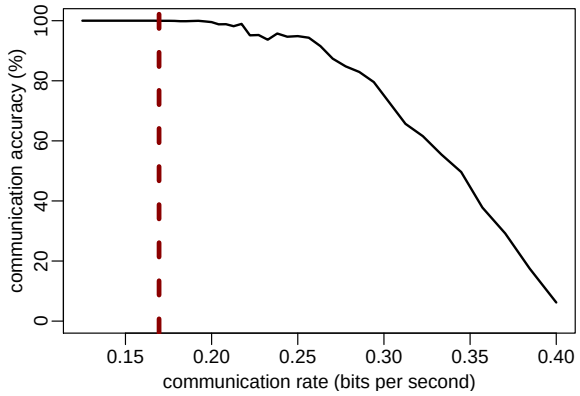



Fig. 5: Communication accuracy at different bit rates.

We observe a similar pattern as shown in Fig. 5 for different activity frequencies. For each activity frequency, considering the worst case core-pinning combination, we find the maximum error-free communication rate (i.e., the correspondent of the dashed line of Fig. 5). Table 2 summarizes these maximum error-free communication rates for different source activity frequencies. We observe that the fastest communication happens if we set the activity frequency of the source to 0.5Hz. As we try to increase the activity frequency to increase the communication rate, the channel becomes more noisy, since kernel background tasks cover approximately the same frequency range. These high frequency changes are already visible in Figs. 4a and 4d. In other words, by increasing the activity frequency, we get closer to the frequency range that we are trying to eliminate from the GFLOPS signal using the maximum moving filter. Therefore, for higher activity frequencies at the source, such as 1Hz and 5Hz, we do not get a better communication rate, while by increasing the activity frequency from 0.2Hz to 0.5Hz, communication becomes faster.

## 6 Conclusion

This paper introduces a novel class of covert channels enabled by power management vulnerabilities. Only the characterization of such vulnerabilities can fundamentally change how the power delivery and management is accomplished in modern systems, where security has emerged as a design parameter as important as (or more important than) performance, power or reliability.

## References

[1] H. Ritzdorf, "Analyzing Covert Channels on Mobile Devices," *M.S. Thesis, ETH*, 2012.
[2] C. Percival, "Cache missing for fun and profit," 2005.
[3] 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.
[4] 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.
[5] 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.
[6] A. Tang, S. Sethumadhavan, and S. Stolfo, "Clkscrew: exposing the perils of security-oblivious energy management," in *USENIX Security Symposium*, 2017.
[7] 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.
[8] *MPrime*. https://aur.archlinux.org/packages/mprime/.