High-Speed In-Kernel Network Periodicity Detection: A Discrete Wavelet Transform Implementation via eXpress Data Path

Nicolas Kolling Ribas
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
nicolas.ribas@inf.ufrgs.br

Marco Aurélio Spohn
Federal University of Fronteira Sul
Chapecó, Brazil
marco.spohn@uffs.edu.br

Lisandro Zambenedetti Granville
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
granville@inf.ufrgs.br

***Abstract*—Periodicity in network traffic often indicates anomalies; yet, analyzing high-speed streams is challenging due to the volume of data. While signal processing techniques such as the Discrete Wavelet Transform (DWT) are effective, their real-time application in the network data plane remains non-trivial. This paper presents the design and implementation of an online DWT-based method for detecting periodicity in network traffic using the eXpress Data Path (XDP). To this end, we develop an algorithm that bypasses key constraints of the extended Berkeley Packet Filter (eBPF), such as the lack of floating-point arithmetic, and performs DWT decomposition entirely in the XDP data plane. The algorithm operates on a time-series signal generated from packet-per-second counts. We validate our approach by applying it to real-world, security-related traffic captures containing known periodic patterns, which our approach successfully identifies. We also assess the performance of our solution by comparing its throughput against a baseline scenario on high-performance hardware, specifically a 100 Gbit/s BlueField-2 DPU, thereby demonstrating its suitability for high-traffic environments.**

***Index Terms*—Data plane, discrete wavelet transforms, eBPF, network security, periodicity detection, traffic analysis.**

I. Introduction

Due to the growing volume of network traffic, analyzing it to identify malicious activity has become increasingly challenging for network operators. In addition, in many cases, storing network measurement data for later analysis is not feasible or implies prohibitive delays. As a result, analysis of streaming data is required. From this data, we can compute a signal based on its characteristics (e.g., packet count per time interval). By applying signal processing techniques to streaming data, periodicity within the network can be detected, potentially indicating either malicious or benign activities [1]. Furthermore, periodicity detection can notify operators or automation software, prompting an investigation to determine the cause of the activity.

Signal processing using the Discrete Wavelet Transform (DWT) [2] provides an efficient method for analyzing time-series data at low computational cost. Compared to the $O(n \log n)$ complexity of the Discrete Fourier Transform (DFT), DWT's $O(n)$ complexity and time-localized nature make it well-suited to the strict latency requirements of the network data plane. DWT can analyze signal components in both the time and frequency domains [3]. It provides higher time resolution for high-frequency components than for lower-frequency ones, thereby enabling simultaneous analysis across different scales. Despite its low overhead, achieving effective processing at line rate requires supplemental technologies. The extended Berkeley Packet Filter (eBPF) [4] is particularly promising, as it allows for efficient packet processing at the kernel level. In conjunction with the eXpress Data Path (XDP) [5], eBPF enables processing directly on network hardware. However, eBPF poses implementation challenges, as it lacks support for key operations, such as floating-point arithmetic.

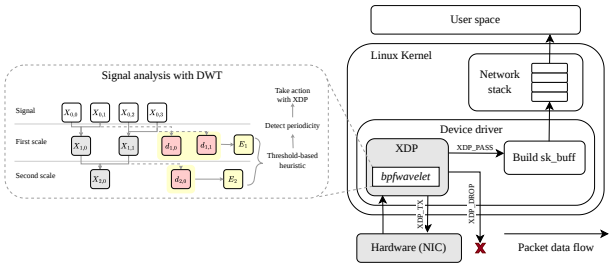


Figure 1: Overview of the *bpfwavelet* architecture

Although the challenge of real-time, low-overhead periodicity detection in streaming network traffic has been addressed in previous research, such as through P4 [6], no prior work has explored the potential of eBPF and XDP for this purpose. Existing approaches often rely on specialized hardware or programming languages designed for programmable switches, which can impose higher complexity or hardware requirements. From a network management perspective, offloading signal processing to the data plane enables autonomous, real-time monitoring without the overhead of forwarding traffic to the user space or requiring specialized P4-enabled hardware. In contrast, eBPF provides a flexible, kernel-level solution that

operates directly on general-purpose network hardware, offering an alternative that could reduce complexity and enhance accessibility without compromising performance.

In this paper, we present *bpfwavelet*, a solution that implements DWT-based periodicity detection directly within the XDP data plane, as illustrated in Figure 1. *bpfwavelet* performs the transform, identifies signals in both the time and frequency domains, and analyzes the decomposition levels by calculating the energy function. This energy function is then applied in a threshold-based heuristic to detect periodicity. To overcome eBPF limitations, we made mathematical adjustments to eliminate unsupported operations, such as floating-point arithmetic and division. Our solution has a small memory footprint and exhibits negligible overhead for 256-byte and larger packets. Importantly, we show that throughput remains effectively independent of the number of decomposition levels, establishing DWT as a scalable primitive for in-kernel analytics.

The remainder of this paper is organized as follows. Section II provides an overview of the key concepts, starting with an explanation of the discrete wavelet transform, followed by a discussion of eBPF limitations. In Section III, we present our proposed method for performing online wavelet decomposition and detail its implementation using eBPF. In Section IV, we evaluate our solution, while Section V discusses related work. Finally, Section VI concludes the paper.

II. Background

A. Discrete Wavelet Transform

The fundamental idea behind DWT involves the use of functions called *wavelets*, which are small waves or pulses localized in time. To decompose a signal, DWT uses a low-pass filter (also known as a scaling function or father wavelet) and a high-pass filter (also known as a wavelet function or mother wavelet). These filters are convolved with k data points at a time (depending on the filter size), encoding high- and low-frequency information into two distinct decomposition sets and effectively downsampling the original signal by a factor of 2. The data points encoded by the high-pass and low-pass filters are referred to as detail and approximation coefficients, respectively. We can apply DWT decomposition recursively m times, using the approximation coefficients at level $j-1$ as input for level j $(1 \leq j \leq m)$ to analyze frequencies at a finer granularity. The original signal corresponds to level zero.

Among the available wavelets, we use the Haar wavelet, which is one of the simplest and often serves as an introduction to the concept of DWT [7]. Thus, we define its high-pass and low-pass filters as $(1/\sqrt{2}, -1/\sqrt{2})$ and $(1/\sqrt{2}, 1/\sqrt{2})$, respectively. Considering the time series $X_{0,k}, k = 0, 1, 2, \ldots$ corresponding to the input signal, the approximation and detail coefficients are calculated using (1) and (2), respectively.

$$X_{j,k} = 1/\sqrt{2}(X_{j-1,2k} + X_{j-1,2k+1}) \tag{1}$$

$$d_{j,k} = 1/\sqrt{2}(X_{j-1,2k} - X_{j-1,2k+1}) \tag{2}$$

To analyze recurring patterns in the network, the energy function of the detail coefficients has been utilized [8]. The energy function E_j is defined as:

$$E_j = \frac{1}{N_j} \sum_k |d_{j,k}|^2, \quad j = 1, 2, \ldots, m \tag{3}$$

Here j represents the decomposition level, and N_j is the number of coefficients at level j. Calculating the energy of the detail coefficients at each decomposition level enables us to examine the signal's temporal properties, progressing from higher to lower frequencies as the decomposition level increases.

Moreover, [9] demonstrated that plotting the function $g(j) = \log_2(E_j)$ can be used as a method to detect periodicities in a signal. As each decomposition level filters a specific frequency range in the original signal, a particular periodicity manifests as a sudden decrease in $g(j)$. This dip indicates that the signal's energy is concentrated at a specific scale, effectively revealing the underlying period.

B. eBPF and XDP

eBPF is a Linux kernel technology that enables the execution of custom, sandboxed programs at various kernel hook points, thereby allowing dynamic extension of kernel functionality. To ensure kernel stability, all eBPF programs are first subjected to a verifier that statically analyzes them to prevent unsafe operations, but this also imposes significant constraints. Most notably, the verifier disallows unbounded loops and, most critically for signal processing, lacks any support for floating-point arithmetic.

The XDP is a specific eBPF hook located at the earliest possible point in the network processing path—directly within the Network Interface Controller (NIC) driver. By running an eBPF program attached to an XDP hook, packets can be inspected, processed, forwarded, or dropped at line rate before the kernel's main networking stack even allocates them, making it ideal for high-performance networking, observability, and security.

III. Online DWT Decomposition

To detect periodicity without floating-point support, we propose a streaming algorithm adapted for integer arithmetic. The process operates in three stages: (1) *Sampling* packet counts into time intervals; (2) *Decomposition* via recursive integer operations; and (3) *Thresholding*, where energy dips are detected using a ratio-based heuristic.

For our application of detecting periodic network activity, we consider a signal in which each sample represents the number of packets in a network flow over a fixed time interval. A flow refers to all packets that match a rule specified by the network operator, such as those with a particular destination port or IP address. Implementing the DWT transformation for this type of signal in eBPF poses a significant challenge, as eBPF does not support all the arithmetic operations required to calculate (1), (2), and (3). Additionally, calculating and storing

the signal and approximation coefficients across different levels may incur significant processing and storage overhead.

At first glance, (3) presents additional challenges for its implementation in eBPF, including division by a number N_j that varies over time for each decomposition level and floating-point operations (division by $\sqrt{2}$). However, by expanding the equation for different levels, we can simplify it to the point where these operations are no longer necessary. Initially, we assume that the signal length is a power of two, i.e., $N = 2^n$ for some n, so that $N_j = N/2^j$, where j represents the decomposition level, or the signal when $j = 0$. In this case, the equation is represented as:

$$NE_j = 2^{j-1} \sum_k (X_{j-1,2k} - X_{j-1,2k+1}) \tag{4}$$

To detect periodicity in the signal, we use the heuristic of dividing two adjacent energy levels and checking whether the difference exceeds a threshold. If the energy suddenly drops from one level to another, then E_{j-1}/E_j is greater than the limit. Specifically, we check if $E_{j-1}/E_j > \alpha/\beta$. Defining $S_j = \sum_k (X_{j-1,2k} - X_{j-1,2k+1})$. To avoid floating-point divisions, we can rewrite the equation that detects periodicity as:

$$\beta S_{j-1} > 2\alpha S_j \tag{5}$$

Algorithm 1 Decompose signal for every new sample

```
1:  Initialize w, s, c, i ← 0
2:  procedure COUNT
3:      c ← c + 1
4:      if is new flow then
5:          Schedule(Decompose)
6:      end if
7:  end procedure
8:  procedure DECOMPOSE
9:      x ← c
10:     k ← i
11:     for j ← 0 to L do
12:         if k is even then
13:             w_j ← x
14:             break
15:         end if
16:         s_j ← s_j + (w_j − x)^2
17:         if j ≥ 1 then
18:             if β · s_{j−1} > 2 · α · s_j then
19:                 NotifyPeriodicity(j)
20:             end if
21:         end if
22:         x ← x + w_j
23:         k ← k >> 1
24:     end for
25:     i ← i + 1
26:     c ← 0
27:     Schedule(Decompose)
28: end procedure
```

To implement the DWT online, we created an algorithm that stores only the last approximation coefficient (X_j) calculated at each level, the accumulated sum S_j for each level, the index of the current sample i, and a packet counter c. Thus, our algorithm stores only $2L + 2$ integer values per flow, where L is the number of decomposition levels applied consecutively. Although we chose to use the packet count in our algorithm, the method is signal-independent and can be applied to any statistic that can be computed in eBPF.

The pseudocode for our implementation is shown in Algorithm 1. The first procedure, *Count*, is triggered by the XDP hook; thus, it is executed upon each packet's arrival. In it, the packet counter is incremented. Upon receiving the first packet of a flow, it schedules the second procedure, *Decompose*, to execute after the sampling interval. In our implementation, we use *bpf_timers* from the eBPF API to schedule procedures.

In the procedure *Decompose*, the packet count is collected, effectively sampling the signal. The decomposition starts for all levels (Line 10), until the index k is even (a pair of coefficients—or samples—is needed to calculate the next decomposition). Thus, if the index of the last coefficient is even, a new decomposition cannot be calculated. While the index is odd, the decomposition advances one level. At each level, the sum S_j is computed (Line 16), and in Line 17 it is combined with S_{j-1} to identify periodicity via (5). In Line 22, the approximation coefficient (X_j) is calculated for the next level. The bit-shift in Line 23 effectively maps the index of the current calculated approximation coefficient to the index in the next-coarser time scale (level $j + 1$) for the subsequent approximation coefficient.

At the end of the procedure, the sample index is incremented, the packet count is reset, and the procedure is scheduled to execute again after the sampling interval has passed. This mechanism enables asynchronous signal sampling. The first arriving packet triggers its first execution, and subsequent executions are scheduled by itself (Line 26).

IV. Evaluation

This section presents a two-fold evaluation of the proposed DWT-based periodicity detection solution, *bpfwavelet*. First, a performance analysis quantifies the processing overhead introduced by the solution in the data plane, demonstrating its suitability for high-throughput environments. Second, an applicability evaluation is performed using real-world network traffic captures to validate the accuracy of the DWT-based detection heuristic. To foster reproducibility, the source code of our work, datasets, results, and benchmark scripts are available in a repository [10].

A. Performance Evaluation

The primary goal of this evaluation is to precisely measure the computational overhead of *bpfwavelet*. The experiment seeks to assess the performance impact as a function of key operational parameters, namely the sampling interval, the number of decomposition levels, and the packet size.

1) Setup: The experiments are conducted on two servers connected back-to-back. Each server is equipped with an AMD EPYC 7282 2.80 GHz CPU, 128 GiB (32×4) of DDR4 RAM, and dual-port BlueField-2 100 GbE SmartNICs. One server acts as a traffic generator and receiver, while the other serves as the Device Under Test (DUT) and runs our *bpfwavelet* eBPF program attached to the XDP hook. Traffic is generated, and the DUT processes and reflects it through the same interface.

Traffic generation and performance measurement are conducted using Cisco TRex [11], a DPDK-based specialized tool for generating packets at line rate. We implemented this tool to perform successive transmissions and adjust the packet-generation rate to identify the rate at which packet loss remained below 1%. The found "optimal" rate was then used to perform a 60-second transmission, which was repeated 30 times. The data shown in this evaluation are the averages of these measurements.

To accurately quantify the overhead introduced by *bpfwavelet*, the results are compared against a minimal baseline XDP program, *xdp-bench* [12]. We used this program to perform packet redirection without any additional processing. *xdp-bench* is the *de facto* standard tool for benchmarking and regression analysis of XDP implementations of "in-tree" drivers [13]. This baseline establishes the hardware's fundamental forwarding capacity.

The evaluation measures the maximum forwarding rate in both megapackets per second (Mpps) and Gigabits per second. Three key parameters are systematically varied:

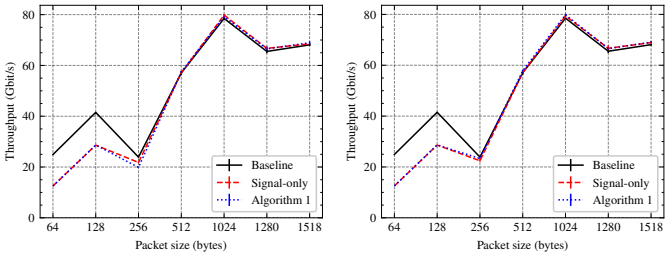
a) Packet Size: A range of packet sizes from 64 bytes to 1518 bytes is used to simulate diverse network conditions, from control-plane-heavy traffic dominated by small packets to large data transfers characterized by a maximum transmission unit (MTU)-sized packets.

b) Decomposition Levels: The number of DWT decomposition levels varies across values of 0, 1, 2, 4, 8, 16, and 32, allowing for a precise measurement of the marginal computational cost associated with increasing the analytical depth of the wavelet transform.

c) Sampling Interval: Two distinct sampling intervals are tested: a fine-grained 250 µs interval and a coarser 1 s interval. This variation is designed to assess the impact of signal granularity on processing performance.

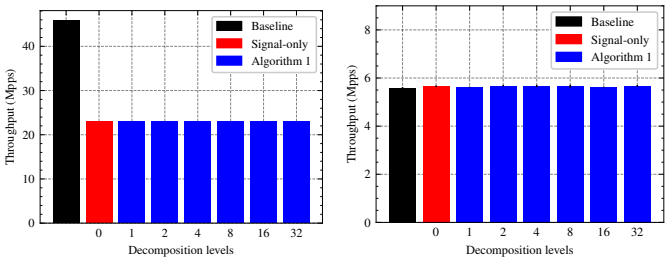
The analysis considers three distinct scenarios to decompose the performance costs: **Baseline**, where the *xdp-bench* program is used, representing the hardware's maximum theoretical throughput; **Signal-Only**, *bpfwavelet* configured with 0 decomposition levels. This configuration isolates the overhead of the initial signal construction phase—namely, packet counting and timestamp-based interval calculation—without performing any DWT computation. **Full Algorithm**, where *bpfwavelet* operates with a variable number of decomposition levels (1–32).

2) Results: In Figure 2a, we show the throughput obtained when varying the packet size while using 1 s as the sampling interval. We observe overheads of 31.0% and 49.7% for 64-byte and 128-byte packet sizes, respectively. We attribute this



(a) (b)

Figure 2: Impact of packet size on throughput for sample interval of (a) 1 s and (b) 250 µs



(a) (b)

Figure 3: Impact of the number of decomposition levels on throughput for packets of (a) 64 bytes and (b) 1518 bytes (250 µs sampling interval)

to the fact that, at these smaller sizes, the number of packets is higher, leading to higher contention within *bpfwavelet*. In our implementation, all cores counting packets compete to write to a single global atomic counter. *xdp-bench* uses a special per-CPU eBPF map; this way, the CPUs have exclusive access to a local counter. Using the same map for our implementation proved infeasible, as the only way to aggregate values across all counters is in userspace code. We opted to maintain all processing in the kernel, as we want periodicity detection to serve as a foundational data-plane primitive.

For more realistic traffic (e.g., packets of 256 bytes or larger), the *bpfwavelet* overhead becomes negligible, reaching the same throughput as the baseline, since the counting overhead is no longer the bottleneck; the throughput is then limited only by hardware capacity.

One might expect that, as packet size increases, throughput should increase monotonically until it reaches line rate. However, our results show a non-monotonic behavior, with significant performance dips at 256 and 1280 bytes. As these anomalies were observed in the baseline measurements and our 30 test repetitions confirmed this behavior with a very low standard deviation, we attribute them to platform-specific testbed artifacts, such as suboptimal alignment with NIC buffers or PCIe DMA batching.

Similar results are observed when using a shorter, fine-grained sampling interval of 250 µs (Figure 2b). More frequent reading of the counter for sampling the signal did not impose significant overhead, as there are very few operations relative

to the writes by the counting cores, and thus did not increase contention.

Figure 3a shows the throughput in millions of packets per second when varying the number of decomposition levels for a packet size of 64 bytes. Similarly, in Figure 3b, the Mpps are shown when varying the number of decomposition levels for a packet size of 1518 bytes. The performance remained unchanged across decomposition levels. This result demonstrates that *bpfwavelet* can be configured to detect both fine-grained periodicities (using small sampling intervals) and coarse-grained, long-running periodicities (using more decomposition levels) without incurring an additional performance penalty.

B. Applicability Evaluation

Having established the low performance overhead of *bpfwavelet*, this section evaluates its effectiveness in detecting periodic behavior in real-world scenarios. The validation focuses on security use cases, where periodicity is a known and often subtle indicator of malicious activity. To this end, we selected a set of publicly available network captures containing traffic associated with well-known malware and exploits. Periodic communications characterize these datasets. The specific use cases, their identified periodicities, and the trace lengths and sampling intervals are summarized in Table I. Given the lack of standard benchmarks for periodicity detection, we prioritized scenarios in which the ground-truth period could be unambiguously manually verified.

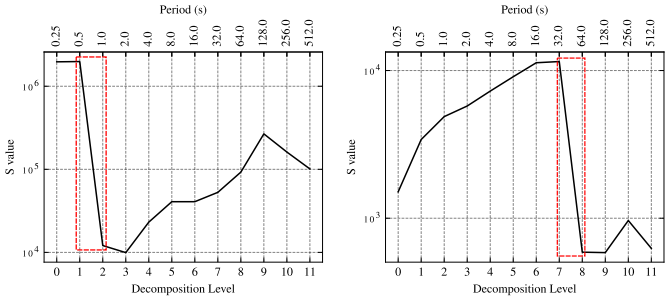
Table I: Use cases and their periodicities

#	Use Case	Period (s)	Trace Length	Interval (ms)
1	Heartbleed [14]	1	20 min 27 s	250
2	Cobalt Strike [15]	60	17 min 12 s	250
3	Trickbot (a) [16]	300	303 min 47 s	600
4	Trickbot (b) [17]	300	166 min 5 s	600

We replayed the captures and used *bpfwavelet* to analyze the traffic. The values of α and β, which we used for the threshold, were empirically determined beforehand by an analysis of the ratio between energy in adjacent levels for multiple periodic and non-periodic signals mixed with noise [6]. We show the energy values for each decomposition level (S_j) calculated by *bpfwavelet*, highlighting the dips that exceed the threshold, thereby characterizing periodicity in the signal.

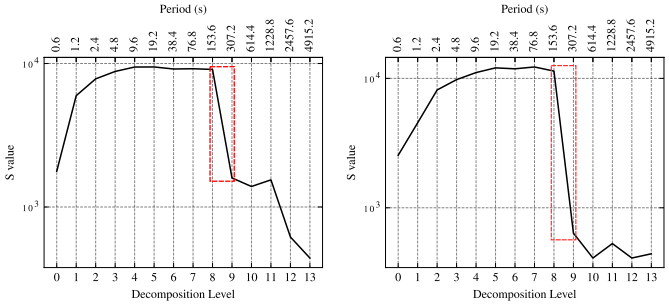
The analysis of these plots is guided by the theoretical foundation established by [9], which demonstrates that periodicity in a signal manifests as a sudden, sharp decrease or "dip" in the log-scale energy plot. For each use case, we identify this characteristic dip and verify that its location corresponds to the known period of the malicious traffic.

For instance, in the case of the trace in Figure 4a, an attacker uses OpenSSL's Heartbleed vulnerability [18] to retrieve the contents from the victim's machine memory by issuing a Heartbeat request approximately every 1 second. This trace was analyzed by sampling the packet count at 250 ms intervals; the energy plot shows a distinct drop between decomposition



(a) (b)

Figure 4: Energy values for (a) Heartbleed and (b) Cobalt Strike traces



(a) (b)

Figure 5: Energy values for the two Trickbot traces

levels 1 and 2. This location corresponds to a period of $2^2 \times 250\,\text{ms} = 1\,\text{s}$. Similarly, for the Cobalt Strike capture (Figure 4b), which has a known command-and-control (C2) period of 60 s. The dip can be observed between levels 7 and 8, which corresponds to periods of $2^8 \times 250\,\text{ms} = 64\,\text{s}$.

In the two traces in which the Trickbot malware performs C2 communication (Figures 5a and 5b), both exhibit a period of 300 s. We can see a sharp dip between decomposition levels 8 and 9, which corresponds to $2^9 \times 600\,\text{ms} = 307.2\,\text{s}$.

Secondary dips observed in the energy plots are attributed to harmonics or background traffic noise. However, these dips do not cross the threshold, effectively filtering them out as false positives, demonstrating that *bpfwavelet* correctly localizes the periodicity's timescale and achieves accurate detection.

V. Related Work

Existing eBPF monitoring systems, such as ePPing [19] or those with adaptive control planes [20], [21], are limited to exporting simple packet- or flow-level statistics (e.g., RTTs, counters, rates) for control-plane processing. To our knowledge, no existing work performs complex temporal-statistical analyses, such as periodicity detection, directly within the XDP data plane.

While the DWT is used for periodicity detection in offline frameworks such as RobustPeriod [22], adapting this method to online, real-time data-plane execution remains challenging. Some in-dataplane wavelet-based systems, like μMON [23],

exist. Still, they use wavelets for lossy compression rather than for the real-time statistical analysis on which our work focuses.

The most closely related work implements DWT in P4 for periodicity detection [6]. That approach, however, targets specialized programmable network devices. Our work is distinct because we address the challenge of implementing DWT-based periodicity detection using eBPF/XDP on commodity hardware with standard Linux kernels.

VI. Conclusion

This paper demonstrates that complex signal processing, specifically DWT-based periodicity detection, is feasible to implement entirely within the network data plane using XDP. We developed *bpfwavelet*, an algorithm that operates within the constraints of the eBPF environment by successfully eliminating unsupported operations, including all floating-point arithmetic and division, and performs decomposition. The evaluation confirmed that this approach is practical and successfully identified known periodic behaviors in real-world security captures, such as the 1-second Heartbleed exploit and the 300-second C2 cycle of the Trickbot malware.

The performance evaluation of $100\,\mathrm{Gbit/s}$ hardware shows that this analysis is achieved with negligible throughput overhead for 256-byte packets or larger. A key finding is that performance is independent of the number of DWT decomposition levels. This capability is crucial, as it enables the detection of very long-period events at line rate without introducing a performance bottleneck, thereby directly addressing a central challenge in high-speed network monitoring.

Further developments can expand on the foundations presented in our work. The current implementation uses the Haar wavelet; subsequent work could explore other wavelet types to address its known analytical limitations [24]. Also, given the experimental and system-oriented nature of this paper, we didn't fully explore detection accuracy or robustness under diverse traffic conditions, which could be addressed in future work. Additionally, the multiresolution analysis could be applied to other signal metrics beyond packet counts, such as throughput. A significant extension would be to integrate the detection primitive with XDP's packet-handling capabilities, allowing for automated traffic redirection or blocking based on a detected periodic signature.

Acknowledgments

This work was partially supported by the São Paulo Research Foundation (FAPESP) under grant number 2020/05152-7, the PROFISSA project, and is part of the CNPq process 316662/2021-6. It is also part of the INCT of Intelligent Communications Networks and the Internet of Things (ICoNIoT), funded by CNPq (proc. 405940/2022-0) and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) Finance Code 001 and 88887.954253/2024-00.

References

[1] A. Mahboubi, K. Luong, H. Aboutorab, H. T. Bui, S. Camtepe, K. Ansari, and B. Barry, "The evolving threat landscape of botnets: Comprehensive analysis of detection techniques in the age of artificial intelligence," *Internet of Things*, vol. 33, p. 101728, 2025.

[2] M. J. Shensa, "The discrete wavelet transform: wedding the a trous and mallat algorithms," *IEEE Trans. Signal Process.*, vol. 40, pp. 2464–2482, 1992.

[3] M. Roughan, D. Veitch, and P. Abry, "On-Line Estimation of the Parameters of Long-Range Dependence," in *Proc. IEEE GLOBECOM*, 1998.

[4] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[5] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path: Fast programmable packet processing in the operating system kernel," in *Proc. ACM CoNEXT*, 2018, pp. 54–66.

[6] B. R. Huaytalla, A. S. Jacobs, M. V. Silva, F. B. Carvalho, R. A. Ferreira, W. Willinger, and L. Z. Granville, "DWT in P4: Periodicity Detection in the Data Plane," in *Proc. IEEE GLOBECOM*, 2022.

[7] G. Bartlett, J. Heidemann, and C. Papadopoulos, "Low-Rate, Flow-Level Periodicity Detection," in *Proc. IEEE INFOCOM WKSHPS*, 2011.

[8] P. Huang, A. Feldmann, and W. Willinger, "A non-instrusive, wavelet-based approach to detecting network performance problems," in *1st ACM SIGCOMM Workshop on Internet Measurement*, ser. IMW '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 213–227. [Online]. Available: https://doi.org/10.1145/505202.505229

[9] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *Proc. ACM SIGCOMM*, 1999.

[10] "Source Code." [Online]. Available: https://github.com/nicolaskribas/bpfwavelet

[11] "Cisco TRex Realistic Traffic Generator," Accessed: Sep. 24, 2025. [Online]. Available: https://trex-tgn.cisco.com/

[12] "xdp-bench - an XDP benchmarking tool," Accessed: Sep. 24, 2025. [Online]. Available: https://github.com/xdp-project/xdp-tools

[13] M. Molè, "Driver Support for Programmable End-Host Networking," Master's thesis, Politecnico di Milano, 2023.

[14] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *Proc. ICISSP*, 2018.

[15] B. Duncan, "2021-05-13 - Hancitor infection with Ficker Stealer and Cobalt Strike," Accessed: Sep. 24, 2025. [Online]. Available: https://www.malware-traffic-analysis.net/2021/05/13/index.html

[16] ——, "2019-10-31 - Data dump: IcedID infection with Trickbot," Accessed: Sep. 24, 2025. [Online]. Available: https://www.malware-traffic-analysis.net/2019/10/31/index.html

[17] ——, "2019-12-26 - Data dump: IcedID infection with Trickbot," Accessed: Sep. 24, 2025. [Online]. Available: https://www.malware-traffic-analysis.net/2019/12/26/index.html

[18] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey *et al.*, "The Matter of Heartbleed," in *Proc. ACM IMC*, 2014.

[19] S. Sundberg, A. Brunstrom, S. Ferlin-Reiter, T. Høiland-Jørgensen, and J. D. Brouer, "Efficient Continuous Latency Monitoring with eBPF," in *International Conference on Passive and Active Network Measurement*, 2023.

[20] S. Magnani, F. Risso, and D. Siracusa, "A Control Plane Enabling Automated and Fully Adaptive Network Traffic Monitoring With eBPF," *IEEE Access*, vol. 10, pp. 90 778–90 791, 2022.

[21] M. Abranches, O. Michel, E. Keller, and S. Schmid, "Efficient Network Monitoring Applications in the Kernel with eBPF and XDP," in *Proc. IEEE NFV-SDN*, 2021.

[22] Q. Wen, K. He, L. Sun, Y. Zhang, M. Ke, and H. Xu, "RobustPeriod: Robust Time-Frequency Mining for Multiple Periodicity Detection," in *Proc. ACM SIGMOD*, 2021.

[23] H. Zheng, C. Huang, X. Han, J. Zheng, X. Wang, C. Tian, W. Dou, and G. Chen, "μMon: Empowering Microsecond-level Network Monitoring with Wavelets," in *Proc. ACM SIGCOMM*, 2024.

[24] I. M. Dremin, O. V. Ivanov, and V. A. Nechitailo, "Wavelets and their uses," *Physics-Uspekhi*, vol. 44, no. 5, p. 447, 2001.