DWT in P4: Periodicity Detection in the Data Plane

Briggette R. Huaytalla* Arthur S. Jacobs* Marcus V. B. Silva* Fabrício B. Carvalho†
Ronaldo A. Ferreira† Walter Willinger‡ Lisandro Z. Granville*
*UFRGS, Brazil †UFMS, Brazil ‡NIKSUN, USA

***Abstract*—This paper presents a P4 implementation of the (1-D) Discrete Wavelet Transform (DWT) method. As a mathematical tool for analyzing signals such as packet-level traces, the DWT divides a given signal into different frequency components and analyzes each component with a resolution matched to its scale. We develop an efficient online algorithm that circumvents various limitations of existing P4-programmable data plane devices and performs the DWT decomposition entirely in the data plane. Our evaluation of a hardware implementation (*i.e.,* Netronome NFP-4000 SmartNIC) of the algorithm shows that it results in only minimal throughput overhead (less than 1% for average-sized packets) and operates within constraints imposed by the limited available data plane resources. As an application, we use our lightweight P4 implementation of the DWT and describe a novel threshold-based approach for detecting periodic behavior in a signal in real-time, at line rate in the data plane (40 Gbps). We illustrate our approach with different examples of synthetic and real-world packet-level traffic traces that exhibit periodic patterns of either benign or malicious origins.**

I. Introduction

The recent proliferation of Internet-connected devices, systems, and services and dramatic changes in Internet usage [1] are among the main reasons for the continued exponential growth in Internet traffic. To carry out tasks such as detecting nefarious network activities or distinguishing these activities from benign behavior, network operators are required to collect and analyze enormous amounts of network measurement data. The analysis of such data may impose timing constraints (*e.g.,* non-real-time vs. real-time), determining the type of methods at the operators' disposal, such as traditional statistical analysis techniques [2]–[5], information theory-based approaches [6], [7], and machine learning algorithms [2]. These methods can be further separated into time-domain [8], frequency-domain [2], [9], and wavelet-domain techniques, and some of them can be adapted for streaming data analysis. Analyzing streaming data allows operators to consider different features (*e.g.,* packet counts in a time interval) required for inferring certain network activities, computing them in high throughput scenarios [5] and at line rate [2] without having to store the data under analysis.

One practical use of signal processing for analyzing streaming data is to infer periodic activities within network traffic. Such activities are often indications of recurring patterns in network usage and can be either malicious or benign. Inferring periodic activities of unknown origins will typically trigger a detailed post-mortem and offline forensic analysis by the network operator to identify the observed periodic activities' root cause(s). Examples of such efforts include detecting anomaly

978-1-6654-3540-6/22/$31.00 © 2022 IEEE

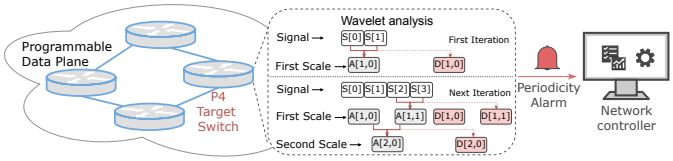


Fig. 1: Wavelets analysis for periodicity detection in P4.

behavior [3], [10], reconstructing the signal of a network communication [11], and analyzing the energy spectrum [12]. However, traditional signal processing techniques such as the Discrete Fourier Transform (DFT) are known to have a high computational overhead that prevents them from being used for real-time periodicity detection in high throughput scenarios [2], [10]. Their practical use in this context is therefore limited to performing post-mortem analysis tasks [9], [11].

In contrast, the Discrete Wavelet Transform (DWT) method can be used to analyze time-series data with low computational overhead by leveraging their intrinsic ability for time-frequency localization, *i.e.,* dividing the data into different frequency components and analyzing each component with a resolution matched to its scale. Used in prior works [3]–[5], [13] to analyze networking traffic data, the DWT method is an especially promising technique for analyzing streaming data where the "signal" is given in the form of packet-level network traces. Not only does it allow for the simultaneous analysis of the signal at different scales, but the method can be naturally parallelized and performed at line rate to enable real-time signal analysis.

Despite its low computational overhead, implementing the DWT method to process high-volume traffic streams at line rate poses significant challenges. Using off-the-shelf commodity hardware to perform the necessary time-frequency localization of the incoming traffic is, in general, inefficient as it can introduce additional overhead that offsets the benefits of using the DWT. In turn, recent advances in programmable data plane technologies (*e.g.,* P4 [14]) present unique opportunities to use techniques such as the DWT for line rate traffic processing in the data plane. However, P4's limited support for commonly used arithmetic operations makes it difficult to implement the DWT method in P4 and run it on actual hardware.

This paper presents a P4 implementation of the DWT method to perform a signal's time-frequency localization directly in the data plane. Figure 1 provides an overview of our solution, in which a P4-enabled programmable device is running an efficient algorithm that we designed to perform the DWT decomposition. We compute the energy function to analyze each decomposition level of the DWT, and use a

threshold-based heuristic to automatically alert the network operator of identified periodic behavior. To circumvent the limitations imposed by existing P4-enabled devices, we rely on a number of mathematical modifications that reduce the need for complex arithmetic operations. Our solution has a small memory footprint in the data plane and results in only minimal throughput overhead (less than 1% for average-sized packets). Finally, we show with several performance and security use cases how our proposed solution correctly identifies periodic behavior in both synthetic and real-world packet traffic traces.

II. Background

The Discrete Wavelet Transform (DWT) method uses waveforms to "localize" a signal in both frequency and time. In addition to pinpointing the specific times when each frequency occurs in the signal [15], the DWT has lower computational complexity than the more traditional Fourier transform method—$O(n)$ vs. $O(n \log n)$ [2], [15], where n is the number of samples in the signal—making it more appealing for analyzing signals in high-throughput settings such as network traffic.

To decompose a signal, the DWT uses a low-pass filter (*a.k.a.*, *scaling function* or *father wavelet*) and a high-pass filter (*a.k.a.*, *wavelet function* or *mother wavelet*). These filters are convolved with k data points at a time (depending on the size of the filters), encoding high- and low-frequency information into two distinct levels of decomposition and effectively sub-sampling the original signal by half. The encoded data points generated by the high-pass and low-pass filters are referred to as the *detail* and *approximation* coefficients, respectively. We can apply the DWT decomposition recursively m times using the approximation coefficients at level $j-1$ as input to level j ($1 \leq j \leq m$) to analyze frequencies at a finer granularity. The original signal corresponds to level zero.

The original DWT method was proposed alongside a simple set of wavelet filters known as the Haar wavelet [13]. Here, we also use the Haar wavelet and leave the application of other wavelets filters such as the Daubechies wavelets [3] for future work. We define the Haar wavelet low-pass and high-pass filters as $(1/\sqrt{2}, 1/\sqrt{2})$ and $(1/\sqrt{2}, -1/\sqrt{2})$, respectively [16] and consider a time series $X_{0,k}, k = 0, 1, 2, \ldots$ representing the input signal. For scale one of the DWT decomposition, we multiply these values with consecutive samples in the input signal and then add the resulting products to compute the approximation and detail coefficients, respectively. More generically, we describe the approximation and detail coefficients for scale j, $j \geq 1$, at position k by Equations 1 and 2, respectively.

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

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

A. Periodicity Detection: An Energy-Function Analysis

Most of the existing methods for analyzing recurring patterns in networking require storing a high volume of network traffic data and mining it post-mortem. Due to the high computational complexity of these methods [15], this type of analysis can take a long time. However, mining traffic in today's high-speed networks to detect patterns in communication requires analyzing measurements at line rates with methods that have low computational complexity. Compared to most existing approaches, the DWT method with its low computational complexity and broad applicability to different problems is especially well suited for the high-throughput conditions and real-time requirements imposed by modern-day networks.

In particular, the energy function of the detail coefficients has been used to detect periodic signals in different scenarios (*e.g.,* for studying network congestion and its impact on TCP retransmissions [16]). The energy function E_j is defined as

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

where j is the decomposition level, and N_j is the number of coefficients at level j. Computing the energy of the detail coefficients at each decomposition level allows us to examine the temporal properties in the signal from high to low frequencies as the level of the wavelet decomposition increases.

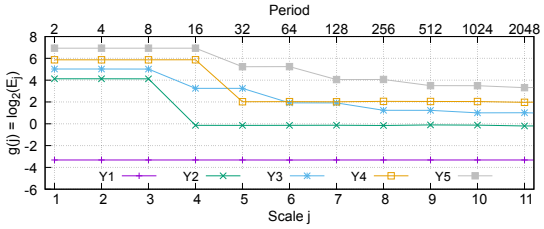


Fig. 2: Using the energy function to detect periodicity.

Feldmann *et al.* [17] show that plotting the function $g(j) = \log_2(E_j)$ can be used as a means to detect periodicities in a signal. As each decomposition level filters out a specific frequency range in the original signal, a particular periodicity manifests as a sudden decrease in $g(j)$. Figure 2 shows an example with five different signals. Y1 is white noise with no periodicity. Y2–Y5 are mixtures of white noise with a periodic signal of period 8, 10, 16, and 20, respectively. The figure shows that while $g(j)$ decreases for Y2–Y5 near the point marking the signal's period, $g(j)$ remains flat for Y1.

III. Online DWT Decomposition

We consider a signal where each sample is indexed and represents the number of packets of a flow in a fixed time interval. A flow refers to all the packets that match a rule specified by the network operator, *e.g.,* all packets with a given destination port or IP address. Implementing the DWT transform for such signals in P4 presents significant challenges. First, P4 does not allow loops and does not support all the arithmetic operations required to evaluate Equations 1, 2 and 3. Second, processing in P4 is asynchronous, meaning that we need a packet arrival to trigger a computation. Finally, computing and storing the signal and the approximation coefficients at different levels may impose significant processing and storage overheads.

A. Arithmetic Operations

At first sight, Equation 3 presents additional hurdles for its implementation in P4, including the division by a number N_j that varies over time for each level of the decomposition and floating-point operations (division by $\sqrt{2}$). However, by expanding the equation to different levels, we can simplify it to the point where these operations are no longer needed. First, we assume without loss of generality that the length of the signal is a power of two, *i.e.,* $N = 2^n$ for some n, so $N_j = N/2^j$, where j represents the level of the decomposition or the signal when $j = 0$. Then, we can write Equation 3 as

$$NE_j = 2^{j-1} \sum_k \left| X_{j-1,2k} - X_{j-1,2k+1} \right|^2 \tag{4}$$

For conciseness, let $A_{j,w} = X_{j-1,2w} + X_{j-1,2w+1}$, where X_j is the signal for $j = 0$ or the approximation coefficients for $j \geq 1$ as defined in Section II. $A_{j,w}$ is a parameterized version of the expression inside the parenthesis of Equation 1, *i.e.,* $A_{j,w}$ is the approximation coefficient without the $\sqrt{2}$ normalization factor. For $j \geq 2$, we can expand Equation 4, using the definition of $A_{j,w}$ and Equation 1 as

$$NE_j = 2^{j-2} \sum_k \left| A_{j-1,2k} - A_{j-1,2k+1} \right|^2 \tag{5}$$

Also, since we have $A_{j-1,2k} = X_{j-2,4k} + X_{j-2,4k+1}$ and $A_{j-1,2k+1} = X_{j-2,4k+2} + X_{j-2,4k+3}$, we can compute Equation 5 using only additions, subtractions, and multiplications. Moreover, we can efficiently implement multiplication of a number by itself or by a power of two using only shift and addition operations. Finally, since we are only interested in finding points where $g(j)$ decreases, we do not need to divide the right-hand side of Equation 5 by N.

B. Asynchronous Computation and Resource Usage

To build a signal on a P4 switch, we need to execute an action to count the number of packets in a sampling interval. However, the switch runs an action only when a packet arrives and matches an operator-defined rule. If no packet arrives during long periods, we have a signal with multiple consecutive samples equal to zero. Since P4 does not allow loops, we cannot simply iterate over these samples to compute the approximation and detail coefficients of the DWT transform and propagate them to calculate the other decomposition levels. Also, storing the signal's samples and the approximation and detail coefficients for each decomposition level would consume a large amount of memory in the switch and limit the length of the signal and the number of flows we could analyze.

We leverage the simplification of the energy function we developed in Section III-A to develop an efficient algorithm that stores for each signal only a sliding window W_j with two approximation coefficients and the cumulative sum $S_j = \sum_k |A_{j-1,2k} - A_{j-1,2k+1}|^2$ for each level j. We also store the index (L) of the sample that was last processed by the switch, totaling $3*$MAX_LEVEL$+1$ integers per flow, where MAX_LEVEL is the maximum number of decomposition levels. To compute the k-th approximation coefficient at level j, we need the coefficients (or signal samples if $j = 1$) with indices $2k$ and $2k+1$ at level $j-1$ (See Equations 1 and 5). We can map this dependence of the approximation coefficients to a tree structure where signal samples are the leaves, and the coefficients are the internal nodes of a binary tree, as illustrated in Figure 3. Note that to calculate the coefficient of an internal node, we always need the values of its two children. Since our algorithm performs online computations, some samples or coefficients might not be available when trying to compute the approximation coefficient and the cumulative sum of the energy function at the next level at a specific time. Also, to deal with the case where the switch does not receive any packet for a flow during several sampling intervals, we introduce samples with a value set to zero. Therefore, the arrival of a packet at the switch may trigger the computation of the coefficients at several levels as a new sample may complete an entire subtree. For example, the availability of sample 7 in Figure 3d allows the computation of the approximation coefficients 3 at level 1, 1 at level 2, and 0 at level 3.

Algorithm 1 Online DWT Decomposition and Energy Function.

```
1:  procedure OnlineDWT(
        L: Index of the previously measured sample,
        R: Index of the currently measured sample,
        s: Number of packets in the current sample)
2:      if IsEven(L) and IsEven(R) then PropagateL(L, R, 1)
3:      else if IsEven(L) and IsOdd(R) then PropagateLR(L, R, 1, s)
4:      else if IsOdd(L) and IsEven(R) then PropagateL(L+1, R, 1)
5:      else if IsOdd(L) and IsOdd(R) then
6:          PropagateLR(L + 1, R, 1, s)
7:      W_{0,IsEven(R) ? 0 : 1} ← s
8:      L ← R
9:  procedure PropagateL(L, R, j)
10:     if R − L < 2 then
11:         if R − L = 1 and IsOdd(L) then
12:             BranchOdd(L, j)
13:     else
14:         Decompose(L/2, j)
15:         if j < MAX_LEVEL then PropagateL(L/2, R/2, j + 1)
16: procedure PropagateLR(L, R, j, s)
17:     if R − L = 1 then
18:         if j = 1 then W_{0,1} ← s
19:         Decompose(L/2, j + 1)
20:         if IsOdd(L) then BranchOdd(L/2, j + 1)
21:     else
22:         BranchEven(L, R, j)
23:         W_{0,1} ← s
24:         BranchOdd(R, j)
25: procedure BranchOdd(index, j)
26:     Decompose(index/2, j)
27:     if j < MAX_LEVEL and IsOdd(index/2) then
28:         BranchOdd(index/2, j + 1)
29: procedure BranchEven(L, R, j)
30:     if R − L < 2 then
31:         if IsOdd(L) then BranchOdd(L, j)
32:     else
33:         Decompose(L/2, j)
34:         if j < MAX_LEVEL then BranchEven(L/2, R/2, j + 1)
35: procedure Decompose(index, j)
36:     W_{j,IsEven(index) ? 0 : 1} ← W_{j−1,0} + W_{j−1,1}
37:     S_j ← S_j + (W_{j−1,0} − W_{j−1,1})^2
38:     if α(S_{j−1} ≪ (j − 3)) − β(S_j ≪ (j − 2)) > 0 then
39:         GenerateAlarm(j − 1)
```

Algorithm 1 shows the pseudocode to decompose a signal using the DWT transform, compute the energy function for

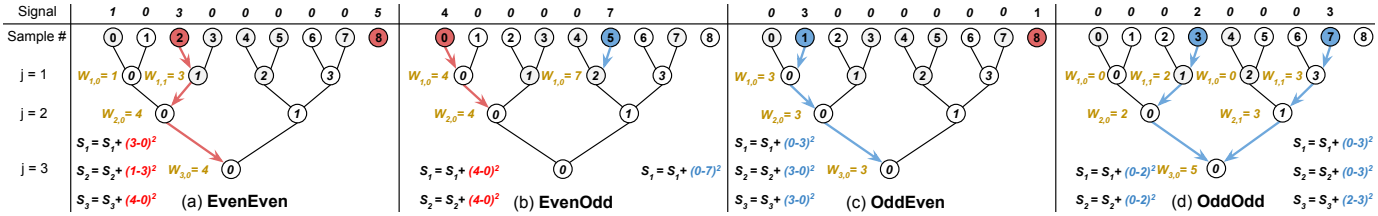


Fig. 3: Examples of signal decompositions and computations of the energy function.

each decomposition level, and generate an alarm when it detects a periodicity in a signal. To simplify the explanation, we present the algorithm with recursive functions to traverse the decomposition tree. Although P4 does not support recursion, we can implement the algorithm by simply expanding the recursive functions MAX_LEVEL times. We evaluate the maximum number of levels we can support on a programmable NIC in Section IV. Line 1 shows the main function of the algorithm that is invoked every time the switch receives a packet that marks the end of a sampling interval. Figure 3a illustrates a signal with nine samples indexed from 0 to 8 that triggers the call to function `OnlineDWT` with $L = 2$, $R = 8$, and $s = 4$. We use L and R and the fact that all samples between them are zero to define four different cases depending on whether L and R are even or odd: EvenEven, EvenOdd, OddEven, and OddOdd (Figures 3a-d).

Due to limited space, we only describe the EvenEven case, but note that similar arguments apply when considering the remaining cases as detailed by Algorithm 1. We use Figure 3a to illustrate how the algorithm works with an example where $L = 2$ and $R = 8$. Function `OnlineDWT` calls `PropagateL` (Line 9) to propagate sample L (saved in $W_{0,0}$) to the next levels because its sibling (*e.g.,* sample with index 3 in Figure 3a) for computing the approximation coefficient was not available in the previous invocation of `OnlineDWT`. As R is even and its sibling is not available yet, the algorithm does not propagate sample R to the next levels and just updates $W_{0,0}$ with the current sample (Line 7) and L with R (Line 8) for a future invocation of `OnlineDWT`. The `PropagateL` function recurses with L and R divided by two (Line 15) to map them to the proper nodes of the decomposition tree at the next level. This recursion stops when j reaches MAX_LEVEL (Line 15) or when the difference between L and R becomes less than two (Line 10). In this last case, the propagation has reached a level where either L is equal to R, and there is nothing more to propagate, or we call `BranchOdd` if L is odd at that level of the decomposition. When $index$ at `BranchOdd` is odd at a level, it means we have a sample that completes a pair of children for computing an approximation coefficient, so the `BranchOdd` function recurses while $index$ is odd in the subsequent levels, computing the approximation coefficients until j reaches MAX_LEVEL or the function reaches a node with $index$ even that does not have its sibling yet to compute the next approximation coefficient.

To detect periodicity in a signal, we use a heuristic that divides the energy at level $j-1$ by the energy at level j and checks if the result is greater than a threshold. If the energy decreases suddenly from level $j-1$ to level j, then E_{j-1}/E_j will be greater than one. More specifically, we check if $\frac{E_{j-1}}{E_j} > \frac{\alpha}{\beta}$, where α and β are integers and $\alpha > \beta$. Using Equation 5 and the definition of S_j, we rewrite this formula as $\frac{2^{j-3}S_{j-1}}{2^{j-2}S_j} > \frac{\alpha}{\beta}$. Substituting the multiplication by powers of two with shift operations, we can rewrite the equation as $\beta(S_{j-1} \ll (j-3)) - \alpha(S_j \ll (j-2)) > 0$, which is the same as Line 38 of Algorithm 1. Line 39 generates an alarm the first time the condition in Line 38 becomes true. In Section IV, we determine α and β empirically for a set of use cases we evaluate in this paper.

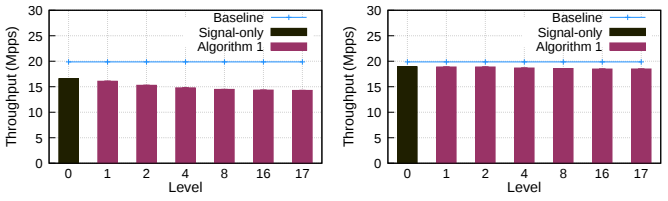
IV. Evaluation

To evaluate the performance and applicability of our approach, we first implement our algorithm in P4 with Micro-C and load it into a Netronome NFP-4000 SmartNIC to assess its overhead. Next, we determine the threshold for our periodicity heuristic and use publicly available traces of periodic traffic to analyze the energy function plot. To support reproducibility, the artifacts of our work are available at [18].

A. Performance

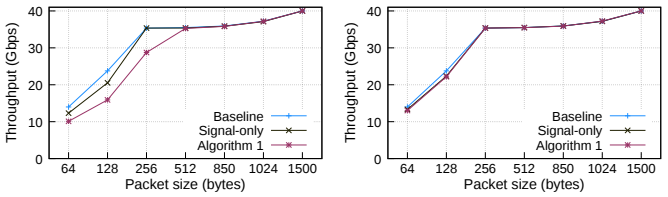
Setup. We use two servers with two Intel(R) Xeon(R) Silver 4114, each with 128 GB of RAM and a dual-port Netronome NFP-4000 40 GbE NIC directly connected on a sender-receiver configuration. The SmartNICs have a limit of 8k instructions per flow-processing core, allowing us to use at most 17 levels of decomposition. The receiver's SmartNIC receives packets, processes them according to Algorithm 1, and forwards them back to the sender. The sender sends packets and collects the throughput results. To quantify the throughout overhead of our implementation, we consider different packet sizes (from 64 to 1500 bytes) and different number of decomposition levels (from 1 to 17), and report experimental results for two different sampling intervals (250μs and 1s).

Results. Starting with the extreme case where all packets are of minimal size (*i.e.,* 64 bytes), Figures 4a and 4b show the SmartNIC throughput (in Mpps) for up to 17 decomposition levels (in red) for the sampling intervals 250μs and 1s, respectively. In both cases, level 0 (in black) denotes the throughput for basic packet capture without any decomposition (*i.e.,* constructing per-sample signal), and the baseline (in blue) is measured by simply forwarding packets to their destination. The SmartNIC provides 64-bits integer packet timestamps composed of two 32-bits variables, one for seconds and the other for nanoseconds. To compute accurate sampling intervals, we implemented a set of shift and multiplication operations to perform integer divisions using constants. As a result, when using a 250μs sampling interval, the throughput



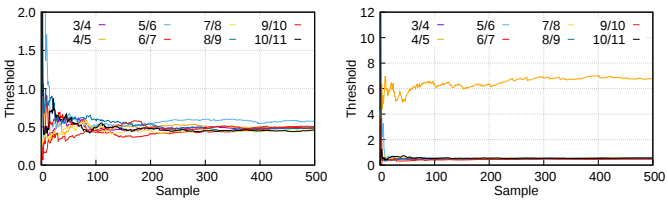
(a) 250μs. (b) 1s.

Fig. 4: Performance results varying levels using 64b packets.



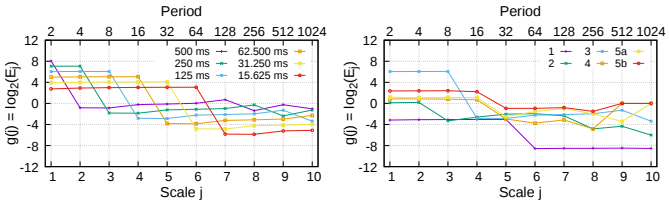
(a) 250μs. (b) 1s.

Fig. 5: Performance results varying packet size using 17 levels.



(a) Y1. (b) Y4.

Fig. 6: Threshold analysis.



(a) Heartbleed. (b) All use cases.

Fig. 7: Energy function plots for use cases from Table I.

overhead for construction of the signal is roughly 17% when compared to the baseline (see level 0 in Figure 4a). However, this overhead drops to less than 5% when the sampling interval is 1s (see level 0 in Figure 4b), mainly because in this case, calculating accurate timestamps requires fewer operations.

Figure 4 also shows that our algorithm manages to compute the DWT using very few resources. In particular, we observe that the performance overhead for the different decomposition levels (red bars) is small – roughly 12% compared to constructing the signal (black bar). This result indicates that accurately constructing the input signal (*i.e.,* placing samples in the correct interval) is responsible for most of the throughput overhead. When the sampling interval is 1s, the overhead is negligible because the SmartNIC provides timestamps already at the required granularity and there are fewer cases when the algorithm has to propagate coefficients to the subsequent levels.

All previous results assume a worst-case scenario where all packets are of minimal size (*i.e.,* 64 bytes). In Figure 5, we consider more realistic scenarios and analyze the throughput overhead of our algorithm with 17 levels of decomposition for packet sizes that vary from 64 to 1500 bytes. When the sampling interval is 250μs (Figure 5a), we notice that for packets of size 512 bytes and larger, the throughput overhead of our algorithm becomes negligible. In fact, assuming averaged-sized packets in the Internet to have 900 bytes [19], we observe a throughput overhead of less than 1% when compared to the baseline. For a sampling interval of 1s (Figure 5b), the throughput overhead is practically zero when compared to the baseline behavior of the SmartNIC.

B. Applicability

Setup. To apply our algorithm in practice, it is first necessary to define an appropriate threshold to be used in Line 38. To this end, we examine the examples considered in Figure 2 (see Section II), dividing the energy value at one level by that at the next level. Figure 6a shows that for a non-periodic signal such as Y1, after some initial phase, the ratios between successive scales converge to a value of around 0.5. In contrast, for periodic signals such as Y4 (Figure 6b), the ratios between levels 4/5 (dip in energy evident in Figure 2) converge to a value larger than 6. Based on those observations, we consider a heuristic threshold value of 1.5 ($\alpha = 3$ and $\beta = 2$ in Algorithm 1).

TABLE I: Network traffic use cases.

#	Use Cases	Period	Trace Length	Sampling Interval
1	Normal link RTT [17]	≈24ms	14’	750μs
2	Congested link RTT [17]	≈1.3s	33’	325ms
3	Heartbleed [20]	≈1s	20’	125ms
4	Cobalt strike [21]	≈60s	17’	4s
5	Trickbot (a) [22], (b) [23]	≈300s	291’, 161’	20s

To experiment with our algorithm, we next surveyed the literature for security and performance use cases with intrinsic periodic behavior and summarize the examples we found in Table I. Performance use cases (1-2) capture the RTT of packets in benign traffic over a normal link and a congested link, respectively. The security use cases (3-5) consist of different attacks that generate malicious traffic such as Heartbleed and Cobalt strike. We analyze each use case’s traffic separately. Also, for each use case we rely on a different sampling interval to facilitate presentation. While a larger sampling interval limits the applicability for fine-grained periodicity, it simply shifts the dips in the energy function to smaller levels. We illustrate this behavior for the Heartbleed use case in Figure 7a.

Results. Figure 7b shows the energy function plots for all use cases. Each use case corresponds to a different line, and the relevant sampling interval is as specified in Table I. As shown in Section II, the second X-axis at the top of the plot represents the period of each decomposition level when multiplied by the sampling interval. In all use cases, the drop in the energy plot happens in the correct periodic interval where it generates an alarm. For instance, the Heartbleed attack (use case 3 in 7b) has a period of 1s and a sampling interval of 125ms. Hence, we see a decrease in the energy function plot between levels 3-4, which indicates a periodicity of $8 \times 125\text{ms} = 1\text{s}$.

These results highlight that our algorithm is both accurate and versatile with respect to detecting periodic behavior. For

example, it can capture fine-grained periodic behavior such as the 24ms RTT pattern in a regular link (use case 1) using a 750μs sampling interval as well as coarser periodicities such as the 300s cycle of the Trickbot attack (use case 5) using a 20s sampling interval. Moreover, because of its demonstrated efficiency (see Section IV.A), the algorithm is highly effective in leveraging modern data plane technologies to detect periodic patterns in real-time.

V. Related Work

Periodicity detection in network traces. Prior works use real-world network traces with periodic components to evaluate Hurst parameter estimation methods [24], focus on detecting botnets using time- and frequency-based metrics to capture the very coarse-grained periodicity [25], and rely on the autocorrelation of their network traffic signals, producing periodic-based features for a random forest classifier [26]. Another study [27] examines periodicity in network traffic by tokenizing flows and looking for cycles in tokens from individual packets. These efforts are mainly concerned with performing post-mortem analyses of the traffic, and the proposed methods are typically not amenable to processing streaming-type data in programmable data planes.

Signal processing in networking. Several works use signal processing techniques to analyze network traffic traces. Some works use DWT coefficients to propose an anomaly detection classification model [3] and analyze changing patterns in time series with the entropy of IP addresses [4]. The BAlet [5] framework detects anomalous BGP updates by leveraging the DWT to decompose a signal of localized BGP update counts. Whisper [2] is a machine learning-based intrusion detection system that leverages frequency domain-based input features (*i.e.,* using the DFT). These efforts do not detect periodic behavior and do not consider any data plane implementation.

Statistical analysis in programmable data planes. Several recent papers propose statistical analysis approaches directly in the data plane. [28] relies on a P4 implementation of entropy to detect DDoS attacks and [6] presents methods for estimating logarithmic and exponential functions in P4 to track traffic entropy. Gao *et al.* [29] implement several statistical techniques (*e.g.,* mean, variance) in P4 and gather them in a library. Our work contributes to these ongoing efforts and adds a P4 implementation of the DWT and a data plane-based periodicity detection technique to this library.

VI. Conclusions and Future Work

This paper presents a P4 implementation of the DWT method and uses it to develop an energy function-based method for detecting periodic patterns in an incoming signal in real-time, at line rate in the data plane. Our P4 implementation of the DWT is based on an efficient online algorithm that overcomes the limitations of existing P4-programmable data plane targets by exploiting mathematical properties of the DWT decomposition. Using this new data plane capability afforded by our P4 implementation of the DWT, our novel energy function-based method for detecting periodic behavior in signals such a packet-level network traffic traces in real-time can be used to automatically alert network operators. Determining appropriate thresholds for our method and developing a dashboard for facilitating an efficient post-mortem analysis whereby operators can determine the alerts' root cause(s) so as to decide whether the automatically detected periodic patterns are malicious or benign is part of our future work.

Acknowledgements

This work was supported in part by CNPq procs. 423275/2016-0, 316662/2021-6, 142089/2018-4 and 465446/2014-0, by FAPESP procs. 2020/05152-7, 2015/24494-8, 2020/05183-0, 14/50937-1 and 15/24485-9, by FAPERGS proc. 16/2551-0000488-9, and by CAPES Finance Code 001.

References

[1] Cisco, "Cisco Annual Internet Report (2018–2023)," Cisco, White Paper.
[2] C. Fu *et al.*, "Realtime Robust Malicious Traffic Detection via Frequency Domain Analysis," in *ACM CCS*, 2021.
[3] S. Y. Ji *et al.*, "Designing an Internet Traffic Predictive Model by Applying a Signal Processing Method," *JNSM*, 2015.
[4] C. Callegari *et al.*, "Combining Wavelet Analysis and Information Theory for Network Anomaly Detection," in *ACM ISABEL*, 2011.
[5] J. Mai, L. Yuan, and C.-N. Chuah, "Detecting BGP Anomalies with Wavelet," in *IEEE/IFIP NOMS*, 2008.
[6] D. Ding *et al.*, "Estimating Logarithmic and Exponential Functions to Track Network Traffic Entropy in P4," in *IEEE/IFIP NOMS*, 2020.
[7] L. González *et al.*, "BUNGEE: An Adaptive Pushback Mechanism for DDoS Detection and Mitigation in P4 Data Planes," in *IEEE IM*, 2021.
[8] S. M. Popa and G. M. Manea, "Using Traffic Self-Similarity for Network Anomalies Detection," in *IEEE CSCS*, 2015.
[9] X. Luo, D. Li, and S. Zhang, "Traffic flow prediction during the holidays based on DFT and SVR," in *Journal of Sensors*, 2019.
[10] Z. Du *et al.*, "Network Traffic Anomaly Detection Based on Wavelet Analysis," in *IEEE SERA*, 2018.
[11] D. Jiang *et al.*, "A Compressive Sensing-Based Approach to End-to-End Network Traffic Reconstruction," in *IEEE TNSE*, 2020.
[12] M. Yue *et al.*, "Identifying LDoS attack traffic based on wavelet energy spectrum and combined neural network," in *IJCS*, 2018.
[13] G. Bartlett, J. Heidemann, and C. Papadopoulos, "Low-rate, flow-level periodicity detection," in *IEEE INFOCOM WKSHPS*, 2011.
[14] T. Dargahi, A. Caponi *et al.*, "A Survey on the Security of Stateful SDN Data Planes," in *IEEE Communications Surveys Tutorials*, 2017.
[15] M. Roughan, D. Veitch, and P. Abry, "On-line estimation of the parameters of long-range dependence," in *IEEE GLOBECOM*, 1998.
[16] P. Huang *et al.*, "A Non-Instrusive, Wavelet-Based Approach to Detecting Network Performance Problems," in *ACM IMW*, 2001.
[17] A. Feldmann *et al.*, "Dynamics of IP Traffic: A Study of the Role of Variability and the Impact of Control," *SIGCOMM CCR*, 1999.
[18] "Code," https://github.com/ComputerNetworks-UFRGS/p4wavelets.
[19] CAIDA, "CAIDA Data Monitors," 2021, https://www.caida.org/catalog/datasets/monitors/.
[20] I. Sharafaldin. *et al.*, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *ICISSP*, 2018.
[21] "2021-05-13 - Hancitor with Ficker Stealer and Cobalt Strike." 2021, https://www.malware-traffic-analysis.net/2021/05/13/index.html.
[22] "2019-10-31 - IcedId Infection with Trickbot." 2021, https://www.malware-traffic-analysis.net/2019/10/31/index.html.
[23] "2019-12-26 - IcedId Infection With Trickbot," 2021, https://www.malware-traffic-analysis.net/2019/12/26/index.html.
[24] T. Akgül *et al.*, "Periodicity-Based Anomalies in Self-Similar Network Traffic Flow Measurements," *IEEE TIM*, 2011.
[25] M. Eslahi *et al.*, "Periodicity classification of HTTP traffic to detect HTTP Botnets," in *IEEE ISCAIE*, 2015.
[26] L. Bilge *et al.*, "Disclosure: Detecting Botnet Command and Control Servers through Large-Scale NetFlow Analysis," in *ACM ACSAC*, 2012.
[27] R. Barbosa, R. Sadre, and A. Pras, "Exploiting traffic periodicity in industrial control networks," *IJCIP*, 2016.
[28] A. C. Lapolli *et al.*, "Offloading Real-time DDoS Attack Detection to Programmable Data Planes," in *IFIP/IEEE IM*, 2019.
[29] S. Gao, M. Handley, and S. Vissicchio, "Stats 101 in P4: Towards In-Switch Anomaly Detection," in *ACM HotNets*, 2021.