This paper was presented as part of the 14th IEEE Global Internet Symposium (GI) 2011 at IEEE INFOCOM 2011

Low-Rate, Flow-Level Periodicity Detection

Genevieve Bartlett[1] John Heidemann[1] Christos Papadopoulos[2]
[1] *USC/Information Sciences Institute; Marina del Rey, CA* [2] *Colorado State University; Ft. Collins, CO*

***Abstract*—As desktops and servers become more complicated, they employ an increasing amount of automatic, non-user initiated communication. Such communication can be good (OS updates, RSS feed readers, and mail polling), bad (keyloggers, spyware, and botnet command-and-control), or ugly (adware or unauthorized peer-to-peer applications). Communication in these applications is often regular, but with very long periods, ranging from minutes to hours. This infrequent communication and the complexity of today's systems makes these applications difficult for users to detect and diagnose. In this paper we present a new approach to identify low-rate periodic network traffic and changes in such regular communication. We employ signal-processing techniques, using discrete wavelets implemented as a fully decomposed, iterated filter bank. This approach not only detects low-rate periodicities, but also identifies approximate times when traffic changed. We implement a self-surveillance application that externally identifies changes to a user's machine, such as interruption of periodic software updates, or an installation of a keylogger.**

I. Introduction

As computer systems become more complicated, their maintenance and inter-machine coordination has become increasingly automated.As a result of this automation, communication is no longer strictly driven by user actions. Instead, computer-initiated communication is now common. Such automatic network communication means that users are increasingly unaware of when and with whom their machine communicates, and what information is being shared. Positive examples of this automatic communication include automatic updates to operating systems and applications, and automated tracking of information for later consumption, such as RSS feeds, email checks and auction bots. Finally, long-running applications such as peer-to-peer file sharing coordinate periodically to share data and maintain an overlay network.

While much of this communication is beneficial to users, not all automatic communication is desirable. Some applications may share more information than a user may like, such as assistants that report the user's click-stream to advertisers or keyloggers that share a user's passwords. Other applications inject ads into user browsing, periodically checking a control site for new ad content and updates. Finally, compromised computers poll increasingly sophisticated control networks, often using decentralized, peer-to-peer schemes to form botnets of hundreds of thousands of computers.

Hidden communication is often periodic at timescales of minutes to hours. Application and OS update checks often happen at regular intervals, usually hourly or weekly. Some applications regularly poll for new information, such as weather, stock, or news updates. For example, cnn.com's web page automatically reloads every 30 minutes. For one-time communications, such as reports of a new operating system or application installation, personal firewalls such as ZoneAlarm often alert users to some of this non-user-driven communication. For behind-the-scenes periodic communication however, a personal firewall does not typically alert a user every time the communication occurs, and such alerts would be quickly ignored if they could not be correlated with an activity.

These challenges point to a need for a self-surveillance tool that will warn users when suspicious network activity is detected from their machines. Such an application could run on the user's machine, or on a network monitoring appliance to be insulated from compromise of end user machines. Network administrators could apply our techniques to aggregate network traffic to find unexpected periodic traffic. While a full evaluation of these and other applications is outside the scope of this paper, here we identify the problem and show our first steps towards a system for detection of these phenomena.

In this initial look at low-rate periodic communication, our contribution is three-fold. First, we identify *low-rate flow periodicity* as a phenomena of interest in network traffic. Second, we present a novel method to identify low-rate periodic communications in aggregate traffic (Section III). We use full wavelet expansion to identify periodicities from several minutes to hours. Although wavelets are widely used for compression [16], and sometimes traffic analysis [1], [5], [9], we are the first to apply full expansion to identify low-rate periodic traffic. Last, we present a self-surveillance application which can detect the addition or removal of periodic communication (Section IV).

II. Related Work

Our work builds on identifying and classifying applications via network traffic using behaviors unique to specific applications, such as patterns of communication with other computers. Examples of such schemes include work by Karagiannis et al. [10], [11], Constantinou and Mavrommatis [4], and Bartlett et al. [2] where behaviors such as port usage, protocol usage, and number of connections made aid in classifying traffic. Our work also looks at network behavior to identify applications; however, we look at the previously unexplored behavior of *low-rate* periodic communication.

We are also related to detection tools which identify traffic *anomalies*. These systems characterize normal traffic and then watch for unexpected divergence using traffic entropy [6], [14] or through signal-processing techniques [1]. Unlike this prior work, we focus on low-rate periodicity generated by applications and changes in such behavior, and not characteristics

This work is partially supported by the United States Department of Homeland Security contract number NBCHC080035. Conclusions here are those of the authors and do not necessarily reflect the views of DHS.

of aggregate network traffic and anomalies which dominate aggregate traffic at some time scale.

Giroire et al. detect botnet command-and-control communication by finding persistent communication between the end-host and other destinations [**?**]. Both their work and ours focus on repeated communication, but we use signal processing to automatically detect periodic, persisent communication. Their work also requires fine-grain tracking, with rules to aggregate destinations into groups, with indivudally tracked end-hosts. We instead use signal processing to extract periodic traffic from aggregate traffic. Their success at detecting botnets from regular communication lends credence to our claim that identifying periodic communication can reveal malicious traffic.

Host-based malware detection is another class of application-detection schemes related to our work. Virus and spyware scanners run on a host and use signatures specific to each malware program to either identify unwanted files. Part of our work looks at detecting new installations of programs such as keyloggers and adware, but unlike typical malware detection, we are the first to look at general network *behavior* to identify some of these applications.

We apply spectral techniques to network traffic. Spectral techniques have been employed to identify bottleneck links [7], [8] and routing information [15] as well as a range of network anomalies [1], [12]. Magnaghi et al. detect anomalies within TCP flows using a wavelet-based approach to identify network misconfigurations [12]. Barford et al. also use wavelets to analyze SNMP and flow-level information to identify events which dominate the network at some time scale such as DoS floods, flash crowds, and routing failures [1].

We differ from previous work in three main ways. First, our focus is different from prior work. Our work looks at periodicity between flows—*not within a flow*—to identify hosts which maintain regular contact. Second, most prior work searches for specific frequency bands; we instead explore a very large number of narrow frequency ranges and achieve this with full decomposition using iterated filtering. Finally, most prior work looks at high frequency behavior; we instead consider events which occur at very low frequencies (sub-1Hz) and use long observation windows (hours to days) to see such events. Our work is a specific example of applying signal processing to network traffic [13].

III. Methodology

We use wavelets implemented as an iterated filter bank to identify periods of time when a periodic series of connections is present. In this section we discuss how we go from network events to identifying a change in periodic communication.

Although wavelets provide a well developed mathematical theory, and there has been some work applying wavelets to network traffic before, discovering *infrequent* periodic traffic is particularly demanding because of the long-timescales and sparse signals involved. Here we describe the four main parts of our approach extracting a timeseries of events from network traffic, decomposing the timeseries using an iterated filter bank, visualizing the resulting multi-resolution representation, and detecting the presence of a periodic signal. Our focus on long-timescales influences each of these steps.

A. Timeseries Extraction

To apply signal processing to network traffic we first must generate a timeseries of events that represent network traffic [13]. We begin by discarding all traffic that is not of interest, then map events of interest into a fixed-interval timeseries of events per time period.

Our first step is to subset traffic. This step is important because the signal-to-noise ratio governs our ability to detect behavior of interest, and any irrelevant traffic that can be discarded decreases background noise and improves sensitivity. In most cases we consider all TCP traffic. However, in certain applications we can improve our detection by discarding traffic based on other characteristics. For example, when looking for malware known to hide in web traffic, we can discard all traffic other than TCP connections to port 80.

Next, we define an event of interest. We are interested in long-duration interactions, so we monitor TCP *flows* rather than individual packets. The arrival of a SYN-ACK packet in the input packet stream defines the time of an event denoting a new TCP flow. (We use SYN-ACK since it indicates a complete three-way handshake with two active parties.) Malicious traffic with forged SYN-ACK packets may taint this data, but is unlikely to show long-term periodicity and so does not alter our results.

After extracting and sampling events, we create a timeseries covering the duration of analysis. We use fixed-duration bins, and count the number of new connection events per bin, resulting in a time series X of length n, where $n = \frac{duration}{timebinsize}$.

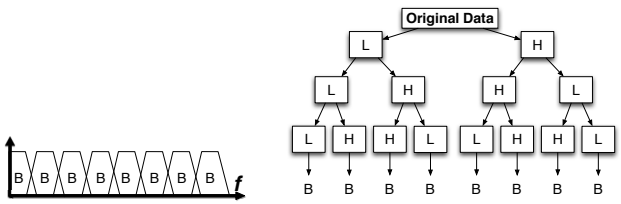
Since our target events are infrequent (minutes or hours apart), we study durations of at least 12 hours to observe multiple instances of an event. We use a bin size of 1s, chosen to support a large range of periods (from several seconds to several hours). We must pick a sampling rate high enough to keep precision for high frequency events, but low enough to avoid excess filtering. Binning events into 1s bins acts as a low-pass filter, and reduces the number of filtering iterations needed to focus in on lower frequencies, while still allowing us to look at periods as short as 2s.

B. Multi-resolution Analysis

Once we obtain a timeseries of events from network data, we next use signal processing to identify any periodic behavior in these events. We have chosen Haar wavelets, implemented as an iterated filter bank of low- and high-pass filters.

For each iteration, given a timeseries X, we get two resulting timeseries:X^H—the result of the high-pass filter—and X^L—from the low-pass filter, where the length of X^L and X^H is half that of the original timeseries X. We iteratively apply the low- and high-pass filters to the resulting timeseries and with each pass, we gain resolution in the frequency domain, and lose resolution in the time domain.

If we consider all the combinations of low- and high-pass filters, the full set can be viewed as a complete binary tree,



(a) Frequency split from filters in Figure *b*

(b) Filters resulting in Figure *a*

Fig. 1. Full decomposition of a filter tree, with "flip" in covered frequency bands.

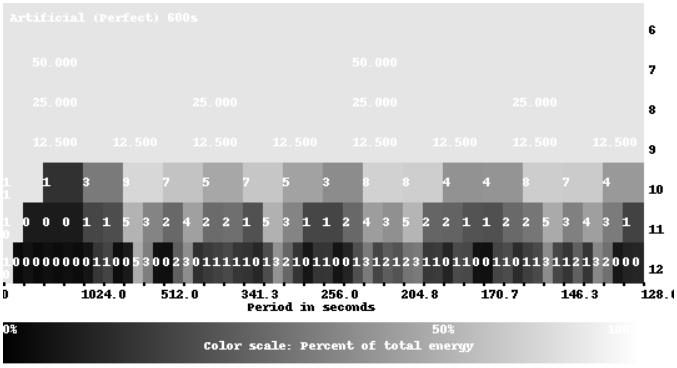


Fig. 2. Long-duration artificial period of 600s.

which we will refer to as a *filter tree*. Each node in the filter tree covers a range of frequencies, and—with the exception of the very left and right most nodes on each level—each node represents a band-pass filter defined by the path of filters used to get from the root to the node. We can—as prior work has done—focus in on a specific frequency range by dynamically choosing a path or using a fixed path from the root to a single leaf in the filter tree. However, unlike prior work, we do not have a pre-determined, specific range of target frequencies we wish to focus in on. Instead, we want to look for *all possible* low-rate periods, from a few seconds to a few hours. Therefore, we perform a *full decomposition* and explore all paths through the filter tree. Figure 1 shows a depiction of a filter tree with full decomposition.

Unlike other signal-processing approaches, our choice of multi-resolution analysis allows us to explore a large range of frequencies without fine-tuning the analysis for each frequency range. For example, a windowed Fourier transform requires choice of window size, and choice of window size optimizes detection of some frequencies while penalizing others. Wavelet analysis avoids early optimization on particular frequencies, allowing us to analyze a wide range of activity.

We select Haar as our basis function for both theoretical and practical factors. Mathematically, the simple square shape of the Haar wavelet is a good match for the sharp changes that occur when new connections are represented as unit impulses. Practically, the Haar wavelet admits a very simple and fast implementation where all operations are differencing or averaging. Although we do not explore hardware implementations, Haar is attractive because its simplicity makes it a good candidate for a hardware to operate at very high line rates. Although we find Haar a good match, exploration of alternative wavelets is part of future work.

Full decomposition requires multiple operations on a single timeseries and despite the use of simple operations appears quite expensive. To reduce the total number of operations we perform we prune certain paths through the tree, which significantly reduces the total number of filtering operations we perform. We discuss this pruning in Section III-E3.

C. Periodic Events and Energy

Given a multi-scale decomposition of observations, we must determine how to identify periodic events. The wavelet coefficients define the *energy* for a given timeseries X at some path P in the decomposition tree:

$$\begin{aligned} e(X^P) &\equiv \text{ss}(X^{PL}) + \text{ss}(X^{PH}) \\ \text{ss}(X) &\equiv \sum_{i=0}^{n} x_i^2 \end{aligned}$$

A concentration of energy in a narrow frequency range indicates the presence of a periodic signal. We show how we use energy to automate detection in Section III-E1.

One benefit of the Haar wavelet is that energy is conserved at each level of decomposition, $e(X) = e(X^L) + e(X^H)$. We can therefore normalize energy and evaluate the energy of each decomposition as a percentage of total energy.

Finally, it is possible to undershoot or overshoot a given frequency in the filter tree. With insufficient levels of decomposition, energy is spread uniformly across large frequency ranges. With excessive decomposition, imperfections in real-world periods cause energy from periodic events to be dispersed across several ranges. These constraints again motivate our desire to adaptively expand the tree until we find periodic behavior.

D. Filter to Frequency

While energy identifies the presence of periodic behavior, we also must know the frequency of the behavior. We therefore must map a position in the filter tree to a specific range of periods.

At first glance, filter mapping seems easy: low- and high-pass filters each separate the low and high frequency bands. Unfortunately, because filters are symmetric, repeated application of high pass filters "flip" the covered frequency bands. If we define $<$ as "covers lower frequencies than", and X^{ab} as applying filter a then b to timeseries X, then $X^L < X^H$ and $X^{LL} < X^{LH}$, but $X^{HH} < X^{HL}$. This flip can be seen in Figure 1, where the filters have been ordered by the frequency ranges they cover.

The correct mapping of filters to frequencies is essential for proper detection, and it also supports visualization of energy

over the frequency space as well. Figure 2 shows decomposition levels 6–12 of a signal with a 600s period. Each row in the visualization represents a level of decomposition, and the number of filter iterations is indicated by the row number to the right. We typically omit the first few levels since periodic behavior only becomes apparent with the finer resolutions offered after several levels of decomposition. Each row is divided into several blocks, showing increasing frequency from left to right. The top row shows 100% of the energy across all frequencies at the 6th level of decomposition. Each lower row shows twice as many blocks, each representing energy over bands of half the previous frequency.

Finally, we represent energy on the z-axis, using both color (white is large amounts of energy, black little) and a numeric value representing percentage of total energy. Because frequency bands become narrower at each level, we scale color in each row to the maximum energy in any band of that row. Thus in Figure 2, we see brighter white areas near 600s where two ranges show 5% and 3% of the total energy across the 12th level of decomposition. Energy is not in one exact range because of window alignment, and there is energy at harmonics as well (Sections III-E1).

While we use visualization to assist our intuition, differences can be subtle, particularly in real data and when traffic with different periods is mixed. To handle these subtleties, we next present a quantitative detection method.

E. Energy and Frequency to Detection

We have shown how periodic events correspond to energy (Section III-C), and how to relate that energy to frequencies (Section III-D). We now combine these to describe our detection algorithm, exploiting the temporal structure of wavelets to identify the start and stop times of a periodic behavior.

1) Detection: We detect events by comparing the energy in a given frequency range to an *energy threshold*. Energy from non-periodic events will disperse as we perform further decomposition, and narrow in on smaller and smaller frequency ranges. Conversely, energy from periodic events will remain concentrated around a specific frequency throughout decomposition. Therefore, to identify a periodic set of events, we must look for strong energy in a narrow range of frequencies.

We ignore detections when the frequency range is too wide. A range is narrow enough to consider detection if the frequency range is within a set percentage of its center period. From low to high frequencies we linearly decrease this percentage from 10% to 1%. This relaxes the definition of wide for lower frequencies since we expect lower frequency periods to have more jitter (a few seconds of jitter on a half-hour period is not as significant as a few seconds off on a 5 second period).

Our energy threshold is dependent on where in the filter tree we are making a detection decision. Further down in the tree at higher levels of decomposition, we lower the energy threshold since each bin represents a narrower frequency band and overall energy will be dispersed over more bins. Specifically, we exponentially decrease the energy threshold, such that the threshold for node n is $t_{energy} = (c_\ell/n)$, where the empirically derived constant c_ℓ, is set based on the tree depth. We currently start c_ℓ at 0.6 for the first 5 levels and then increase it linearly.

Although we expect events occurring at interval p to provide energy at frequency $f = 1/p$, they also give energy to harmonics at small integer multiples of f. We therefore choose the lowest frequency range in a harmonic set as the period of an identified frequency.

We can misdetect the true frequency for several reasons. We see energy at half the base frequency, or at half of a harmonic of the base frequency. Noise and window misalignment (frequencies that are not a power of two) also affect the strength of signals; we look at these effects in our technical report [3]. Typically we correctly find the base frequency, but occasionally a harmonic is stronger.

2) Locating Events in Time: Once we have identified the frequency range of a periodic series of events, we can estimate when the events started and stopped by looking at the time-series of coefficients corresponding to that frequency range. Recall that each node in path P in the decomposition contains a timeseries X^P. Each element i in this timeseries indicates a time x_i^P. To find the beginning and ending of an event in time, we look for a consecutive series of strong coefficients x_i^P. Our current simple approach is to compute the mean $\mu = E[X^P]$, then search backwards in time to find the first $x_b^P < \mu$ as the beginning, and forwards through the signal $x_i^P > \mu$ to find the next $x_e^P < \mu$, giving a period $b \leq i \leq e$. Often, the level of decomposition which identified the frequency range contains too little information in the time domain to make any useful statements about timing. In these cases, we can back up the filter tree two or more levels and examine coefficients at a level with better temporal resolution.

3) Pruning to Reduce Computation: Although our filters reduce to a simple set of additions and subtractions, we can reduce the amount of work done by *pruning* out certain paths through the filter tree. We prune branches from the filter tree for two reasons. First, we cease expansion in frequencies that exceed our sampling rate. Second, we stop expansion if there is minimal energy in the node. In practice, we find that frequency-based pruning is very effective, eliminating 70% of expansion after three levels. Energy-based pruning saves a few percent more; details are in our technical report [3].

F. Sensitivity to Noise

In practice traffic of interest, the *signal*, is always mixed with other traffic, *noise*. We therefore define the signal-to-noise ratio (SNR) as the ratio of periodic TCP flows to all TCP flows, and we measure how SNR affects our ability to detect traffic. Full details (found elsewhere due to space constraints) show in controlled and real-world experiments that we can find periodic traffic when it is at least 5–10% of the overall events [3]. Pre-filtering is important to maximize SNR; for example, if we can narrow the signal to periodic RSS traffic, then we can filter from all TCP flows to just HTTP flows to reduce the noise. We also find that long-term periodic traffic

is often persistent overnight, while human-induced noise is sporadic and diurnal, so 24- or 48-hour analysis can often detect periodic signals that might be missed mid-day.

IV. Applications

With an understanding of the underlying fundamentals of detection of low-rate periodic behavior (Section III), we next discuss applications of our methodology. We show experiment results of keylogger detection, and describe detection of OS update polling and other applications.

A. Self-surveillance

While often periodic communication is beneficial to users (for example, OS update checks), the addition or removal of periodic behavior indicates a change which may warrant further investigation. In this section we show that identifying *changes* in periodic behavior, and the time *when* these changes occur can be useful in self-surveillance.

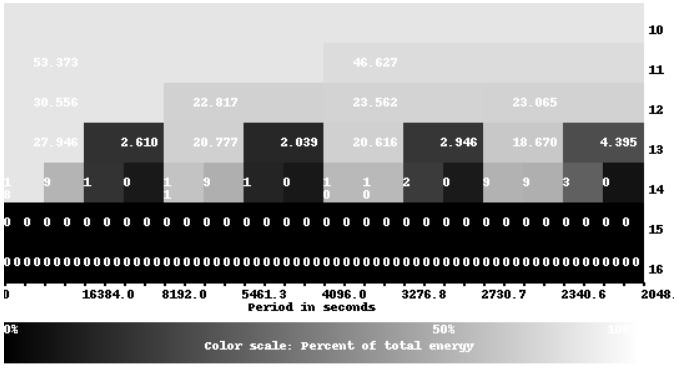
Identifying changes in the periodic behavior of a given host helps users better understand activities on their computer Many operating systems and applications automatically poll for updates. Benign applications sometimes do not disclose automatic polling, or what information they transmit. Undesirable applications, such as spyware and adware, often report back to or request new information from the external master; this contact is often hidden. Additionally, after infection, some malware will *terminate desired* OS or application updates. In all of these cases, a change in periodic traffic is something a user will want to know, but this change may be obscured or concealed. We reveal these changes.

Our approach will alert users when communication patterns change, but we depend on users or other security software to provide context to understand the cause of the change. For example, a user may know that she recently installed Kazaa, and so could trace a new addition of periodic communication to the Gator adware component bundled with Kazaa. Alternatively, an unexpected change may prompt the user to run anti-malware tools. Thus our goal is similar to other intrusion detection and data integrity tools—we alert a user a suspicious change which requires further investigation.

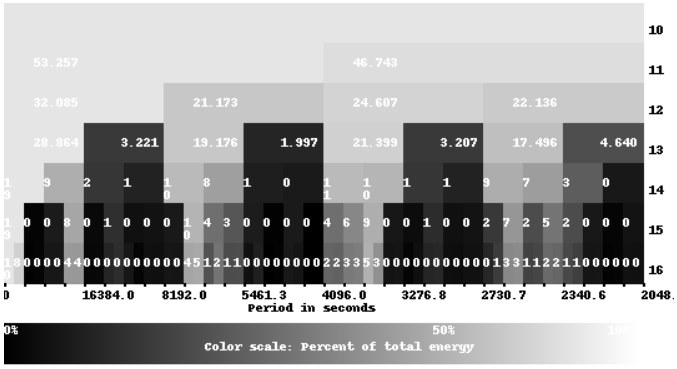
1) Detecting a keylogging application: Many keyloggers report on user activity at specified intervals, to inform their masters what they have learned and that they are still operational—we confirmed supervisor-configured reporting intervals in both SpyBuddy and Keyboard Guardian. A sudden addition of a new periodic behavior, could indicate an addition of just such a keylogger.

To demonstrate we can detect a keylogger, we install Keyboard Guardian on a dedicated Windows computer. We monitor all TCP flows from this computer for a three day period while using it for occasional e-mail and web browsing. To highlight our ability to localize events in time (Section III-E2), we analyze a single 3-day period of observation at once; in operation we would add windowed, continuous observation (details in technical report [3]).

On the second day of the experiment, we install Keyboard Guardian at 4pm, and set the keylogger to email reports every



(a) Traffic without keylogger (before installation).



(b) Traffic with keylogger (after installation).

Fig. 3. Visualization before and after installation of a keylogger.

three hours. Throughout the experiment, we continue regular usage of the test machine, which results in an overall SNR of 0.1 after the keylogger install.

Our system correctly identifies the presence of new periodic communication between 12pm and 9pm on the second day of our experiment, correctly bracketing the 4pm installation time of the keylogger (Section III-E2). The system is close to identifying the actual reporting period of our keylogger install, and identifies the period of reporting as 6 hours instead of the actual 3-hour period (around 21,600s, 46μHz and not 10,800s, 92μHz). As discussed in Section III-E1, we occasionally identify half the base frequency as the actual frequency for an identified periodic event because we choose the lowest frequency in a set which passes our detection threshold.

To gain insight into how our multi-resolution analysis helps identify the presence or absence of Keyboard Guardian's reporting, Figure 3 shows a visualization of traffic periodicity before and after the keylogger install. The figure shows clear changes between traffic before (Figure 3(a)) and after (Figure 3(b)) the installation: afterwords we see new, strong periodic traffic around our reporting period (10,800s, 92μHz), at double the frequency (around 21,600s) and at harmonics (around 5400s, 3600s, and 2700s).

Our system detects the change shown in Figure 3 *automatically* by comparing strong energy against a frequency-specific

threshold. Surprisingly, the base frequency is not very strong, less than 1%. While the base frequency may not be very distinguishable visually, our system lists it as being past the detection threshold—highlighting the importance of automatic detection (Section III-E1). This example shows our ability to detect low-rate regular traffic from malicious software. Most importantly, we can identify when *changes* in such periodic communication occur, giving the user context for identified periodic communication.

2) Positive Examples of Self-Surveillance: Our approach to self-surveillance applies to detecting positive uses of polling as well as negative uses. For example, the security policy of all operating systems and many applications includes automatic polling for updates. Just as network administrators wish to detect bad behavior, they may also wish to detect the absence of good behavior. In our technical report [3], we demonstrate we can detect a sudden absence in OS update checks for a Fedora Linux box polling for updates via `yum-updatesd`.

B. Detecting Malware and Peer-to-Peer Applications

In the previous section, we looked at how our system can be used for self-surveillance. Next we briefly discuss how our methods can be used by network administrators to identify hosts which may be running specific applications, such as malware or peer-to-peer file sharing.

Many applications such as peer-to-peer file sharing, RSS aggregators, and adware make use of regular and periodic connections, and the period of their communication is often within a known range. In high-speed links, line-rate deep-packet inspection can be difficult or expensive. Our approach can assist application detection by *triaging* line-rate traffic, isolating certain flows or hosts for deep packet inspection. While we cannot confirm the presence of an application because periodic traffic may be due to other causes, triage can reduce the cost of deep-packet inspection.

We demonstrate our system's ability to triage in our technical report [3]. In one test of 500 random hosts, we look for hosts running RSS readers. Identifying periodic traffic eliminates 68% from consideration with only a 4% false-negative rate; the same approach could triage for malicious periodic traffic such as adware or keyloggers.

V. Robustness of the Approach

Because some targets of our work are adversarial, we must assume they will understand our work and actively attempt to circumvent it. Our detection is sensitive to noise [3]. An attacker can hide from our current system by lowering the SNR to 5% or less. Fortunately, lowering traffic rates generally decreases the effectiveness of negative traffic, and we are therefore forcing the attacker to take a hit. Alternatively, an application can evade our detection scheme by adding jitter to its periodic behavior. By increasing jitter, the energy of the signal is diffused. Through experimentation, we have found that when observing for a duration of up to sixty times the length of the period, a jitter of more than 15% is relatively effective at hiding a periodic example. However, we can counter this defense by employing longer observation periods.

VI. Conclusions

In this paper we have shown that low-rate periodicity is common to several broad classes, both good (OS updates), bad (keyloggers and malware), and ugly (adware), and that these applications are widely deployed on public networks. We have explored a wavelet-based approach to identify such periodic behavior, and begun to explore the sensitivity and robustness of this approach. Promising applications of such analysis are self-surveillance, as a user watches his or her own traffic to detect unexpected changes, and pre-filtering, where a network operator deploys the scheme to reduce the number of hosts to carefully examine for possible malware infection.

References

[1] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *Proc. of ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, Oct 2002. ACM.

[2] G. Bartlett, J. Heidemann, and C. Papadopoulos. Inherent behaviors for on-line detection of peer-to-peer file sharing. In *Proc. of 10th IEEE Global Internet*, pages 55–60, Anchorage, Alaska, USA, May 2007. IEEE. An extended version is ISI-TR-2006-627.

[3] G. Bartlett, J. Heidemann, and C. Papadopoulos. Using low-rate flow periodicities in anomaly detection. Technical Report ISI-TR-661, USC/Information Sciences Institute, Jul 2009.

[4] F. Constantinou and P. Mavrommatis. Identifying known and unknown peer-to-peer traffic. In *IEEE International Symposium on Network Computing and Applications (NCA)*, pages 93–102, Jul 2006.

[5] 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. of ACM SIGCOMM Conference*, pages 301–313, Aug 1999.

[6] Y. Gu, A. McCallum, and D. Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *Proc. of ACM Internet Measurement Conf.*, pages 345–350, Oct 2005.

[7] X. He, C. Papadopoulos, J. Heidemann, and A. Hussain. Spectral characteristics of saturated links. Technical Report USC-CSD-TR-827, University of Southern California Comp. Sci. Dept., Jun 2004.

[8] X. He, C. Papadopoulos, J. Heidemann, U. Mitra, and U. Riaz. Remote detection of bottleneck links using spectral and statistical methods. *Computer Networks*, 53(3):279–298, Feb 2009.

[9] A. N. Hussain. *Measurement and Spectral Analysis of Denial of Service Attacks*. PhD thesis, U. of Southern California, Comp. Sci. Dept.

[10] T. Karagiannis, A. Broido, M. Faloutsos, and kc claffy. Transport layer identification of P2P traffic. In *Proc. of ACM SIGCOMM Workshop on Internet Measurement (IMC)*, pages 121–134, Oct 2004.

[11] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel traffic classification in the dark. In *Proceedings of the ACM SIGCOMM Conference*, pages 229–240, Philadelphia, PA, USA, Aug 2005.

[12] A. Magnaghi, T. Hamada, and T. Katsuyama. A Wavelet-Based Framework for Proactive Detection of Network Misconfigurations. In *Proceedings of ACM workshop on Network Troubleshooting*, Aug 2004.

[13] U. Mitra, A. Ortega, J. Heidemann, and C. Papadopoulos. Detecting and identifying malware: A new signal processing goal. *IEEE Signal Processing Magazine*, 23(5):107–111, Sep 2006.

[14] G. Nychis, V. Sekar, D. Andersen, H. Kim, and H. Zhang. An empirical evaluation of entropy-based traffic anomaly detection. In *Proc. of 8th ACM Internet Measurement Conf.*, pages 151–156, Oct 2008.

[15] C. Partridge, D. Cousins, A. W. Jackson, R. Krishnan, T. Saxena, and W. T. Strayer. Using signal processing to analyze wireless data traffic. In *Proc. of 1st ACM Workshop on Wireless Security*, pages 67–76, 2002.

[16] D. Taubman and M. W. Marcellin. *JPEG2000: image compression fundamentals, standards, and practice*. Kluwer Academic Publishers, Boston, MA USA, 2002.