On Detecting Abrupt Changes in Network Entropy Time Series

Philipp Winter, Harald Lampesberger,
Markus Zeilinger, and Eckehard Hermann

Upper Austria University of Applied Sciences
Department of Secure Information Systems
4232 Hagenberg / Softwarepark 11, Austria
{philipp.winter, harald.lampesberger, markus.zeilinger,
eckehard.hermann}@fh-hagenberg.at

Abstract. In recent years, much research focused on entropy as a metric describing the "chaos" inherent to network traffic. In particular, network entropy time series turned out to be a scalable technique to detect unexpected behavior in network traffic.
In this paper, we propose an algorithm capable of detecting abrupt changes in network entropy time series. Abrupt changes indicate that the underlying frequency distribution of network traffic has changed significantly. Empirical evidence suggests that abrupt changes are often caused by malicious activity such as (D)DoS, network scans and worm activity, just to name a few.
Our experiments indicate that the proposed algorithm is able to reliably identify significant changes in network entropy time series. We believe that our approach helps operators of large-scale computer networks in identifying anomalies which are not visible in flow statistics.

Keywords: entropy, anomaly detection, time series analysis, network flows

1 Introduction

Large-scale computer networks (i.e. gigabit and above) pose unique challenges to intrusion and anomaly detection; mostly due to the tremendous amounts of data. Network entropy time series were proposed to reduce high-dimensional network traffic to a single metric describing the dispersion or "chaos" inherent to network traffic [11, 18]. Our research builds upon these entropy time series.

The main contribution of this paper is a detection algorithm based on information theory and statistics. The algorithm is capable of detecting abrupt changes in entropy time series built out of network flows. We define an abrupt change as an unexpectedly high difference between two measurement intervals m_{t-1} and m_t whereas the term "unexpectedly high" depends on a configurable threshold. There is the underlying assumption that abrupt changes are caused by malicious and abnormal network events. This assumption is based on empirical evidence as shown in previous research [18, 13, 11]. After all, our proposed algorithm achieves satisfying results and allows practical and real-time anomaly detection in large-scale computer networks.

The paper is structured as follows: Section 2 provides a short overview of similar research. Section 3 introduces the basic concepts upon which our research rests. The actual algorithm is proposed in Section 4. Section 5 contains the evaluation while Section 6 provides a conclusion as well as future work.

2 Related Work

In [12], Lee and Xiang propose the use of information theoretic measures to conduct anomaly detection. Their proposed measures include entropy, relative entropy and conditional entropy, just to name a few. Lakhina et al. made use of entropy to sum up the feature distribution of network flows [11]. By using unsupervised learning, they show that anomalies can be clustered to "anomaly classes". Wagner and Plattner make use of the Kolmogorov complexity in order to detect worms in flow data [18]. Their work mostly focuses on implementation aspects and scalability and does not propose any specific analysis techniques. In [17], Tellenbach et al. go further by exploring generalized entropy metrics for the purpose of network anomaly detection. Similar work was done by Gu et al. who made use of maximum entropy estimation to reach the same goal. Nychis et al. conducted a comprehensive evaluation of entropy-based anomaly detection metrics [13]. In [15], Sommer and Paxson point out why anomaly detection systems are hardly used in operational networks. Barford et al. made use of signal analysis techniques in order to detect anomalies [1]. In [7], Feinstein et al. explored statistical approaches for the detection of DDoS attacks. In [3], Brutlag proposed the use of a statistical algorithm to detect abnormal behavior in time series. Finally, Sperotto et al. provide a comprehensive overview about flow-based intrusion detection in general [16].

3 Preliminaries

3.1 Network Flows

Our proposed anomaly detection system (ADS) analyzes network flows rather than entire network packets. In their original definition, network flows (in short: flows) provide unidirectional meta information about network packets which share the following characteristics: Source and destination IP address and ports and IP protocol number. It is important to note that all network activity on OSI layer 3 and above results in flows; this includes not only TCP connections but also stateless protocols such as UDP and ICMP.

We decided in favor of network flows since they are highly lightweight (a TCP connection which might have transferred several gigabytes accounts for only a few dozen bytes in a flow) and alleviate the analysis of large-scale computer networks. Also, they raise less privacy concerns since no payload is present. Finally, network flows are widely available on network devices, e.g. in the form of Cisco NetFlow [4] or the upcoming standardized IPFIX [5].

3.2 Entropy Time Series

The collected flows are analyzed by calculating the entropy over a sliding window. This technique is not new and has been topic of extensive research before [12, 13, 18, 11]. We make use of the Shannon entropy which is defined as $H(X) = -\sum_{i=1}^{n} p(x_i) \cdot log_2(p(x_i))$.

The variables $x_1, ..., x_n$ represent the possible realizations of the random variable X and $p(x_i)$ the probabilities of the respective realizations.

In order to make the result of the entropy analysis easier to interpret, we normalize it to the interval $[0,\ 1]$ by using the normalized entropy $H_0(X) = \frac{H}{log_2(n)}$ whereas n represents the amount of different observations, i.e. x_i values.

We calculate the entropy over five flow attributes which can all be found directly in the flow record:

- Source and destination IP
- Source and destination port
- Packets per flow

Aside from the fact that previous research already achieved promising results with these attributes [13, 18], we believe that they complement each other in a natural way.

After all, we are dealing with a total of five entropy time series – one for every flow attribute. The entropy time series are calculated by using an overlapping sliding window comprising all flows observed within the last 5 minutes. The overlapping delta is 4 minutes. So after all, every minute five entropy values are determined and added to their respective time series. Figure 1 depicts such a time series spanning a total of 24 hours. The data stems from the uplink of our university.

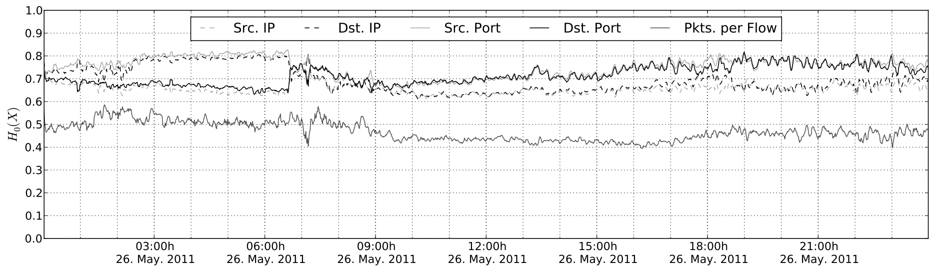

Fig. 1: An entropy time series of our universities uplink spanning 24 hours.

3.3 What can we Detect?

As mentioned earlier, packet payload is inevitably lost in network flows. For this reason, all attacks which manifest solely in packet payload (e.g. remote exploits) are invisible on the network layer. Our system focuses on detecting attacks on the network layer. Some of these attacks, such as DDoS, are quite obvious, others, such as botnets, often highly camouflaged. A comprehensive survey about current research in this domain and about the potentiality of network flows in general is done by Sperotto et al. [16].

We claim that our approach is basically capable of detecting the following malicious activity:

– (D)DoS attacks
– Large-scale scanning activity
– Worm outbreaks

Botnet detection is beyond the scope of this contribution. However, our algorithm might be of help if the activity of a certain botnet leads to abrupt changes in entropy time series.

4 The Anomaly Detection Algorithm

The most straightforward approach in order to conduct anomaly detection, a static threshold, is not sufficient due to the high variability of the time series. Hence, we need an algorithm which is 1) robust against noise and 2) sensitive enough so that attacks are still visible.

4.1 Detecting Abrupt Changes

The basic idea for detecting abrupt changes is to continuously conduct short-term predictions and determine the difference between the prediction and the actual measurement. The higher the difference, the more unexpected and hence abrupt the change is.

Many time series prediction algorithms have been proposed in the past. There are algorithms which account for trends, seasonal components or none of these. We chose to use simple exponential smoothing (SES). The SES algorithm is used to smooth time series as well as to conduct short-term predictions. The algorithm is very elementary and accounts neither for seasonal, nor for trend components.

We chose SES because it turned out to be very robust against the noise inherent to our data set. Furthermore, there is no need for a more sophisticated prediction algorithm since our measurement interval of 1 minute is so fine-grained (over a seasonal cycle of 24 hours we have 1,440 measurements) that seasonal and trend components are not of relevance.

For a time series consisting of the observations $x_1, ..., x_n$, the SES is defined as shown in Equation 1.

$$\hat{x}_t = \begin{cases} x_0 & \text{if } t \leq 1 \\ \alpha \cdot x_{t-1} + (1 - \alpha) \cdot \hat{x}_{t-1} & \text{otherwise} \end{cases} \tag{1}$$

The series $\hat{x}_1, ..., \hat{x}_n$ represents the predictions whereas $\hat{x}_i$ is the predicted value for the (later) observed value x_i. The parameter α, where $0 \leq \alpha \leq 1$, is referred to as the smoothing factor. The closer α is to 0, the smoother the predictions are. If $\alpha = 1$, no smoothing is performed and the "smoothed" time series equals the observed time series.

For each of the five time series, we now determine the prediction error which is the difference between a predicted and an observed value. The error function is defined as $err(x_i, \hat{x}_i) = |\hat{x}_i - x_i|$. However, the determined prediction errors are not equal in their significance because the underlying time series feature different levels of dispersion as can be seen in Figure 2. For this reason, we normalize the prediction errors with respect to the dispersion of the respective time series. We measure the dispersion by calculating the sample standard deviation over a sliding window spanning the last 24 hours. We chose 24 hours so that the sliding window captures an entire seasonal cycle. An alternative would be a sliding window of 7 days which covers an entire week including possible deviations on the weekend.

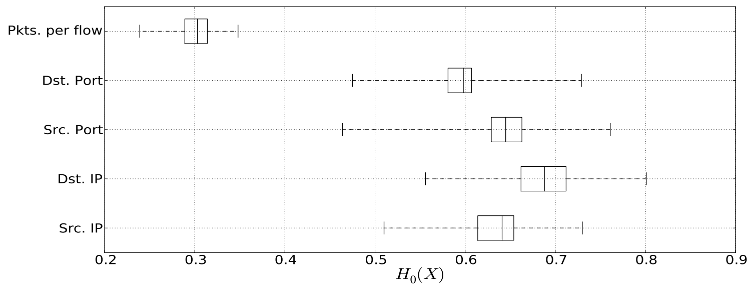

Fig. 2: Box plot depicting the distribution of the five entropy time series over our main flow trace. The whiskers represent the maximum and minimum values, respectively.

We normalize the prediction error of every time series by multiplying it with a weight factor. For each of the five time series and its respective sample standard deviation s_i, this weight factor is defined as shown in Equation 2.

$$\omega_i = \frac{1}{s_i} \cdot max(s_1, ..., s_5) \tag{2}$$

Accordingly, the time series exhibiting the highest sample standard deviation is assigned a weight factor of 1.

To make the proposed ADS for network operators easier to configure, we aggregate all five prediction errors to a single anomaly score referred to as δ.

$$\delta = \sum_{i=1}^{5} err(\hat{x}_i, x_i) \cdot \omega_i \tag{3}$$

A single threshold can now be configured. As soon as the current aggregated anomaly score exceeds this threshold, an alert is triggered.

Algorithm 1 shows the entire algorithm summed up in pseudo code.

Algorithm 1 Abrupt change detection algorithm

```
 1: Time series $\mathcal{T} = \{\{x, \hat{x}, s\}_1, ..., \{x, \hat{x}, s\}_5\}$
 2: while True do
 3:     for all $\{x, \hat{x}, s\}_i \in \mathcal{T}$ do
 4:         Normalized entropy $x_t := H_0$(flow attributes for $i$ within $\Delta t$)
 5:         Predicted entropy $\hat{x}_{t+1} := \alpha \cdot x_t + (1 - \alpha) \cdot \hat{x}_t$
 6:         Prediction error $e_i := |\hat{x}_t - x_t|$
 7:         Standard deviation $s_t := s$(flow attributes for $i$ within last 24 hours)
 8:     end for
 9:     for all $\{x, \hat{x}, s\}_i \in \mathcal{T}$ do
10:         Calculate weight factors $\omega_i := \frac{1}{s} \cdot max(\{\forall\{x, \hat{x}, s\} \in \mathcal{T} \mid s_t\})$
11:     end for
12:     Calculate anomaly score $\delta := \sum_{i=1}^{n} \omega_i \cdot e_i$
13:     if $\delta \geq$ predefind threshold then
14:         Trigger alert
15:     end if
16:     Sleep $\Delta t$
17: end while
```

The first line defines our five time series. Each time series consists of a set of three variables: x is the actual entropy time series over a specific flow attribute, $\hat{x}$ represents the entropy predictions and s stands for the sample standard deviation over the last 24 hours. The endless while-loop spanning the lines 2 – 17 triggers an analysis every Δt minutes. Then, for each time series, the current entropy, the predicted entropy, the prediction error and the standard deviation is computed (line 3 – 8). Afterwards, the respective weight factors, one for each of the five time series, are determined (line 9 – 11). Finally, the overall anomaly score is computed (line 12) by summing up all the weighted prediction errors. If the anomaly score exceeds the threshold defined by the network operator (line 13 – 15), an alert is triggered.

4.2 Choosing Parameters

Several parameters must be set for our algorithm. Each combination of parameters is associated with a trade-off: High detection rates at the cost of many false alarms or vice versa. The following enumeration lists the configurable parameters and points out strategies to set them.

Sliding window size for entropy analysis: Small windows are very sensitive: On the one hand, this will increase the detection rate. On the other hand, the false alarm rate will increase too. Large sliding windows are comparably insensitive which leads to the contrary effect.

Our sliding window spans over 5 minutes which seems to be an acceptable trade-off.

Sliding window overlapping size for entropy analysis: An overlapping sliding window results in more fine-grained time series. We want our system to react as quickly as possible to abrupt changes. For this reason, we chose to overlap the sliding window for 4 minutes. So every minute a measurement "looking back" over the last 5 minutes is added to our time series.

Sliding window size for standard deviation: We decided to set the sliding window size for calculating the standard deviation to 24 hours. That way, the sliding window covers an entire seasonal cycle. Basically, we recommend multiples of 24 hours. The next natural choice would be 7 days which covers an entire week including possible deviations on the weekend.

Smoothing parameter α: In order to use the SES algorithm, one has to come up with the smoothing factor α. As pointed out in more detail in [3], the formula $1 - (1 - \alpha)^n$ informs about how much weight is given to the most recent n observations. One can rearrange this formula to an equation which provides a natural approach to choose alpha: $\alpha = 1 - \sqrt[n]{(1 - \omega)}$. In this form, one can obtain α by specifying how much weight $0 < \omega < 1$ should be given to the most recent n observations.
A small α (i.e. a smoother time series) is less sensitive to high amounts of noise. We set α to 0.05 which means that the last 60 observations (i.e. the last hour in the time series) are assigned a weight of 95%.
However, as our experiments suggested, the choice of α should depend on the respective network. So a sound approach is to first have a look at the entropy time series and then decide which α makes sense in practice.

5 Evaluation

The following sections cover the evaluation of our proposed algorithm. In short, due to the lack of ground truth, we injected handmade anomalies into our flow trace in order to evaluate our algorithm.

5.1 The Data Set

Our primary data set stems from the server network of an ISP. It contains 5 days of unsampled unidirectional network flows: from 25th October 2010 until 30th October 2010. The set holds a total of $\sim$260 million flows which means that on average, $\sim$36,000 flows were received every minute. The network was more active during day than during night times, though. The data set was anonymized [6] but preserves a one-to-one mapping of anonymized to real IP addresses. So the traffic distributions are not affected by the anonymization.

Figure 3 contains an entropy analysis over one day of our flow trace. The overall anomaly score is shown as black time series at the bottom. The higher the anomaly score, the more abrupt are the aggregated changes in the five time series. One can see that there are a handful of events with a particularly high anomaly score, i.e. 0.4 or higher.

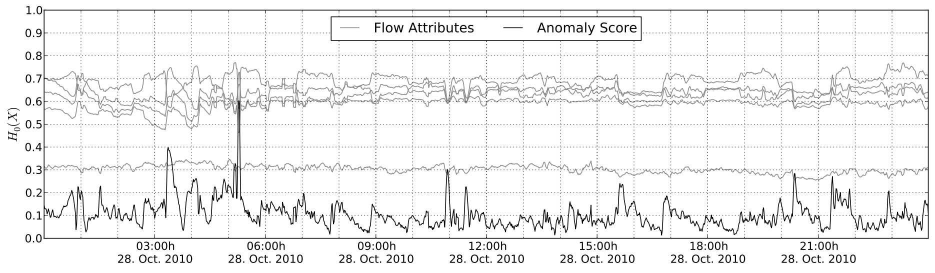

Fig. 3: Entropy time series over one day including anomaly score.

5.2 Injection of Synthetic Anomalies

A sound evaluation of an ADS requires data sets for which ground truth is available or can be established. Since we did not have any ground truth per se, we modeled and injected synthetic anomalies into our original data set. For this purpose, we used a modified version of the tool FLAME [2]. FLAME facilitates the injection of hand-crafted anomalies into flow traces. Anomalies can be described by Python scripts which serve as "flow generators". We implemented two such flow generators:

HTTP DDoS attack: The generated flows represent a synthetic middle-scale DDoS attack launched over HTTP. The DDoS attack lasted for 11 minutes and a total of 500 distributed hosts formed up as attackers. The victim of the attack was a single webserver. The amount of flows generated by the attackers represents a normal distribution since not all attackers have the same bandwidth and can attack at the same scale. As a result from the attack, the webserver was under very high load and could respond only to a small fraction of HTTP requests. The DDoS attack resulted in ∼220,000 flows which were superposed over the original data set.

Horizontal network scan: The purpose of this generator was to yield flows which represent a large-scale horizontal network scan. The attacker used a single IP address for scanning and scanned an entire /16-network which consists of 65,534 valid IP addresses. The purpose of the scan was to find open TCP ports 21, i.e. running FTP daemons. The attacker did not use any obfuscation methods such as inter-scan-delays. The scan resulted in ∼67,000 flows. Note that worm outbreaks are very similar to network scans when looked at on the network layer: Infected computers scan for other vulnerable hosts [8, 19].

We injected the two described anomalies into our original data set which resulted in a new data set containing our handmade anomalies. The anomalies were inserted at 13:00 (DDoS) and at 14:00 (Scan) on October 28th. Both dates were randomly determined.

5.3 Analysis Results

Figure 4 shows the entropy analysis for the entire day of October 28th. The upper diagram covers the original data set whereas the lower one holds our injected anomalies. Again, both diagrams contain time series for all of our five flow attributes. The black time series holding the anomaly score is the most interesting one. In both diagrams, we highlighted particularly high anomaly scores and labeled them from A to D. Anomaly A and B represent "natural" anomalies which were already present in the original data set.

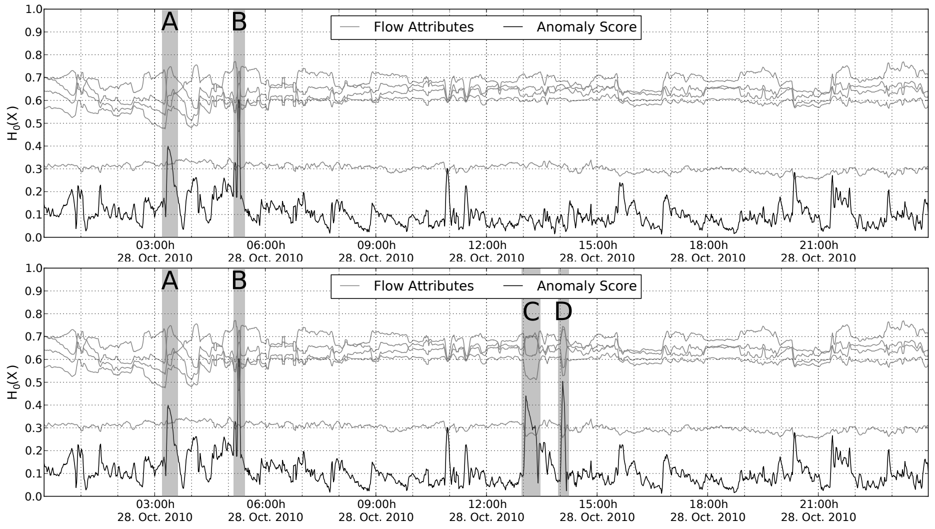

Fig. 4: Entropy time series over the original (upper) and the modified (lower) flow trace. The marked spots represent anomalies.

It turns out that anomaly A was caused predominantly by two hosts, entitled H_1 and H_2. Apparently, H_1 initiated thousands of HTTP connections over a longer period of time to H_2. After this communication stopped, the port time series converged and the entropy of the IP distributions increased significantly.

Anomaly B was again caused by HTTP communication. A set of multiple distributed hosts sent TCP segments to a single host H_3. The source port of all distributed hosts is 80 and the destination port of H_3 a random high port. This phenomenon only lasted for 5 minutes. During this time period, around 30,000 flows were sent to H_3. We presume that the anomaly was caused by a spoofed DoS attack or a misbehaving network device.

Anomaly C and D represent our handmade DDoS attack (C) as well as the network scan (D).

We also conducted experiments with synthetic anomalies of smaller scale. A network scan covering only a /20 network (i.e. 4,094 addresses) is almost invisible in our data set and disappears in noise. So depending on the network link, attacks have to reach a certain minimum scale in order to be detectable.

Eventually, if the scale of an attack gets large enough, anomalies will show up in widely used flow statistics such as "bytes per minute". However, this is not the case with the four anomalies we analyzed. Figure 5 illustrates three popular flow statistics: bytes, packets and flows per minute. The diagram was built out of the same flow trace as Figure 4. The anomalies (again, labeled from A to D) disappear in noise.

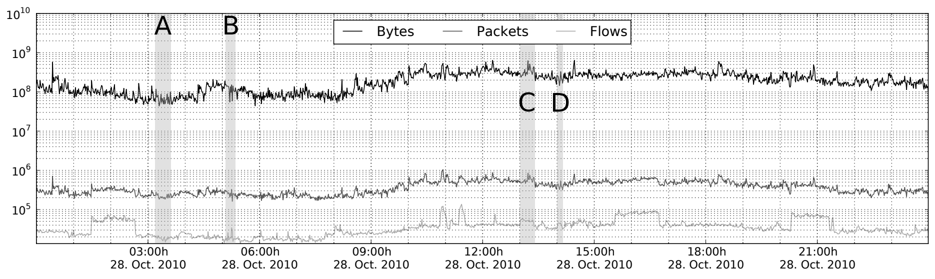

Fig. 5: Time series of three popular flow statistics; the natural and injected anomalies disappear in noise.

5.4 Evasion and Counteractive Measures

For an attacker, the obvious way to evade our detection algorithm is to launch an attack in a slow but continuously increasing way to "stay under the radar" of our algorithm. E.g., a large-scale DDoS-attack against a web server could be started by sending only a dozen HTTP requests per second. Then, the scale can be increased more and more up to millions of requests per second. The algorithm will not detect the attack if the observed changes between the single measurement intervals are not significant enough.

Our algorithm is capable of detecting such stealth attacks too if applied to a larger time scale. What might be invisible when looking at a time scale of minutes can become obvious on a time scale of hours or days.

5.5 Scalability

Finally, we want to verify our claim that the proposed approach is suitable for analyzing large-scale networks. We implemented a prototype in the form of a patch for the NetFlow analysis tool nfdump [9] (for entropy calculation) and as plugin for the corresponding frontend NfSen [10] (for visualization).

It turned out that our implementation is able to analyze up to 940,000 flows per second.[1] On average, our ISP flow trace contains 600 flows per second (with peaks of up to 2000 flows/s) whereas the uplink of our university produces around 70 flows/s (with peaks of up to 250 flows/s). Accordingly, our implementation is not even close to its limits.

6 Conclusion

In this paper, we proposed an algorithm for detecting abrupt changes in network entropy time series. Attacks such as DDoS, worms and scans often lead to such abrupt changes. The main idea of the algorithm is to continuously conduct short-term predictions and determine the difference between the predictions and the actual observed entropy value. The higher the difference, the more abrupt the change is.

Our evaluation suggests that the algorithm performs well and is robust against background noise. However, it is important to note that the attacks have to reach a certain scale in order to be detectable. Small-scale DDoS attacks might be invisible on a high-speed network link. We believe that our proposed algorithm is a valuable tool for network operators. It is straightforward to configure, fast to deploy and does not need training data.

There is much room for future work. First of all, we noticed that the root cause identification of anomalies is often a nontrivial task. The cause (e.g. a SSH brute force attack occurred) can be quite subtle and difficult to find. It would be of great help if network operators were assisted by automated tools [14]. In addition, future research could cover the adaption and testing of the algorithm on a larger time scale in order to detect slowly but continuously emerging attacks which would otherwise evade our system.

Acknowledgements

We would like to thank Peter Haag for providing us with flow traces. Also, we want to thank the anonymous reviewers for providing us with helpful comments which improved the quality of this paper.

References

1. Barford, P., Kline, J., Plonka, D., Ron, A.: A Signal Analysis of Network Traffic Anomalies. In: Proc. of the 2nd ACM SIGCOMM Workshop on Internet measurement. pp. 71 – 82. IMW '02, ACM, New York, NY, USA (2002)
2. Brauckhoff, D., Wagner, A., May, M.: FLAME: A Flow-Level Anomaly Modeling Engine. In: Proc. of the conference on Cyber security experimentation and test. pp. 1 – 6. USENIX Association, Berkeley, CA, USA (2008)

[1] The measurement was done on a 2.66GHz Intel®Xeon CPU using the ISP flow trace.

3. Brutlag, J.D.: Aberrant Behavior Detection in Time Series for Network Monitoring. In: Proc. of the 14th USENIX conference on System administration. pp. 139 – 146. USENIX Association, Berkeley, CA, USA (2000)
4. Cisco Systems: `http://www.cisco.com/web/go/netflow`
5. Claise, B.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard) (Jan 2008), `http://www.ietf.org/rfc/rfc5101.txt`
6. Fan, J., Xu, J., Ammar, M.H., Moon, S.B.: Prefix-Preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme. Computer Networks 46(2), 253–272 (2004)
7. Feinstein, L., Schnackenberg, D., Balupari, R., Kindred, D.: Statistical Approaches to DDoS Attack Detection and Response. DARPA Information Survivability Conference and Exposition 1, 303 – 314 (2003)
8. Fitzgibbon, N., Wood, M.: Conficker.C – A Technical Analysis. Tech. rep., Sophos Inc. (2009)
9. Haag, P.: NFDUMP, `http://nfdump.sourceforge.net`
10. Haag, P.: NfSen, `http://nfsen.sourceforge.net`
11. Lakhina, A., Crovella, M., Diot, C.: Mining Anomalies Using Traffic Feature Distributions. In: Proc. of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications. pp. 217 – 228. SIGCOMM '05, ACM, New York, NY, USA (2005)
12. Lee, W., Xiang, D.: Information-Theoretic Measures for Anomaly Detection. In: Proc. of the 2001 IEEE Symposium on Security and Privacy. pp. 130 – 143. IEEE Computer Society, Washington, DC, USA (2001)
13. Nychis, G., Sekar, V., Andersen, D.G., Kim, H., Zhang, H.: An Empirical Evaluation of Entropy-based Traffic Anomaly Detection. In: Proc. of the 8th ACM SIGCOMM conference on Internet measurement. pp. 151 – 156. IMC '08, ACM, New York, NY, USA (2008)
14. Silveira, F., Diot, C.: URCA: Pulling out Anomalies by their Root Causes. In: Proc. of the 29th Conference on Computer Communications. pp. 722–730. INFOCOM'10, IEEE Press, Piscataway, NJ, USA (2010)
15. Sommer, R., Paxson, V.: Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. IEEE Symposium on Security and Privacy 0, 305 – 316 (2010)
16. Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., Stiller, B.: An Overview of IP Flow-Based Intrusion Detection. IEEE Communications Surveys Tutorials 12(3), 343 – 356 (2010)
17. Tellenbach, B., Burkhart, M., Sornette, D., Maillart, T.: Beyond Shannon: Characterizing Internet Traffic with Generalized Entropy Metrics. In: Proc. of the 10th International Conference on Passive and Active Network Measurement. pp. 239 – 248. PAM '09, Springer-Verlag, Berlin, Heidelberg (2009)
18. Wagner, A., Plattner, B.: Entropy Based Worm and Anomaly Detection in Fast IP Networks. In: Proc. of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise. pp. 172 – 177. IEEE Computer Society, Washington, DC, USA (2005)
19. Weaver, N., Paxson, V., Staniford, S., Cunningham, R.: A Taxonomy of Computer Worms. In: Proc. of the 2003 ACM workshop on Rapid malcode. pp. 11–18. WORM '03, ACM, New York, NY, USA (2003)