An adaptive anomaly detector for worm detection

John Mark Agosta[†] Carlos Diuk-Wasser[‡] Jaideep Chandrashekar[†] Carl Livadas[†]
[†]Intel Research, Santa Clara, CA
{john.m.agosta,jaideep.chandrashekar,carlx.livadas}@intel.com
[‡]Dept. of Computer Science, Rutgers University, New Brunswick NJ
cdiuk@cs.rutgers.edu

Abstract

We present an adaptive end-host anomaly detector where a supervised classifier trained as a traffic predictor is used to control a time-varying detection threshold. Training and testing it on real traffic traces collected from a number of end-hosts, we show our detector dominates an existing fixed threshold detector. This comparison is robust to the choice of off-the-shelf classifier employed, and to a variety of performance criteria: the predictor's error rate, the reduction in the "threshold gap" and the ability to detect the simulated threat of incremental worm traffic added to the traces.

This detector is intended as a part of a distributed worm detection system that infers system-wide threats from end-host detections, thereby avoiding the sensing and resource limitations of conventional centralized systems. The distributed system places a constraint on this end-host detector to appear consistent over time and machine variability.

1 Introduction

Anomaly detection systems have emerged as the last line of defense in dealing with self-propagating Internet malware such as worms, viruses, and so on. They address threats not covered by the existing arsenal of virus scanners and firewalls that can only deal with known attacks and vulnerabilities. By detecting abnormal behavior they offer promise of detecting "day-zero" attacks.

In the abstract, anomaly detection resembles density estimation of a "normal" state of operation. The density is partitioned into denser parts, to represent normal behavior, and the less dense parts are considered abnormal, typically separated from normal by setting a threshold. In practice, an anomaly detector functions by setting a threshold on a signal, and flagging instances when the observed signal exceeds the threshold. In this paper, we are interested in a specific type of anomaly, i.e., a worm infection, and we assume that the worm, upon infecting a host increases the outgoing connection rate of the end-host. Thus, in our case, the anomaly detector thresholds network traffic levels, specifically, the rate of outgoing connections.

Threshold setting works for fast spreading worms like Code Red.[1] However, setting thresholds for slow worms that have connection rates that are much closer to the normal traffic levels of a system forces a compromise: A low threshold causes too many false positives, while a threshold that is too generous allows more false negatives (i.e., anomalous activity that is not detected). Even if we could fix a threshold in some reasonable way, the variability in traffic exhibited at end hosts helps a worm: When the out-going normal traffic is low relative to the threshold, it opens up a "gap"; the worm can send out traffic and remain undetected.

In this paper, we address this by describing an *adaptive* anomaly detector. The algorithm we present computes a dynamic threshold value that adjusts to track the traffic. Compared to a fixed threshold approach, the adaptive approach effectively decrease the worm's "headroom" by minimizing the *gap* it can exploit. Predictions of normal network behavior can be derived from a range of sensored values (both system activity and traffic statistics) available at individual machines. Our algorithm has two components: first, the distribution of current outgoing traffic is predicted based on measurements at previous times; secondly, this prediction is used to adjust the threshold and hence reduce the gap. Casting this in terms of densities, the predicted density is conditioned in real time on past feature time-series. Note that the detector built on top of this predictor is still an anomaly detector since there is no training data for the abnormal state.

We present the work in this paper within a specific larger context: we recently demonstrated *Distributed Detection and Inference* [4], where weak anomaly detectors on end-hosts are allowed to collaborate and exchange information. Importantly, we showed that such an aggregate detection system outperforms standalone detectors. The

[1] The *Code Red* worm, after infecting a host, initiates upwards of 2000 new connections every minute; this is an order of magnitude larger than what might be considered "normal".

individual detectors used the same fixed threshold on the number of outgoing connections from the end-host. Given the variability among different end-hosts, the uniform fixed threshold is not ideal; the work in this paper is a means to address this shortcoming, i.e., to design thresholds that are adapted to individual end-hosts and vary over time. There are many different ways of optimizing the detector performance; by placing our work in the context of the aggregate detection framework, we inherit a specific constraint—that the detectors be optimized to bound the false postive (FP) rate. This is because the FP rate of the local detector is the single operational characteristic affecting the aggregate detector. If its FP rate drifts upward, the aggregate detector will be mis-led, and the system FP rate will increase. Therefore it is necessary the adaptive threshold algorithm FP rate be constrained by bounding it from above. While the results in the paper are somewhat tied to a specific framework, we point out general ideas that are of independent interest and provide a great deal of insight into designing individual anomaly detectors.

2 Background

Recently, there's been an increasing amount of interest in using statistical and machine learning techniques to classify network traffic [2, 5, 8, 10–12] and to detect network traffic anomalies [3, 6, 9].

Roughan *et al.* [12] use traffic classification to identify the class of service (CoS) of traffic streams and, thus, enable the on-the-fly provision of distinct levels of quality of service (QoS). The authors attempt to classify traffic streams into four major traffic classes: interactive, bulk data transfer, streaming, and transactional. Moreover, a multitude of traffic statistics can be used to classify flows and these statistics may pertain to either packets, flows, connections, intra-flow, intra-connection, or multi-flow characteristics. Roughan *et al.* investigate the effectiveness of using average packet size, RMS packet size, and average flow duration to discriminate among flows. Given these characteristics, simple classification schemes produced very accurate traffic flow classification. In a similar approach, Moore and Zuev [11] apply variants of the naïve Bayes classification scheme to classify flows into 10 distinct application groups. They also search through the various traffic characteristics to identify those that are most effective at discriminating among the various traffic flow classes.

In the realm of anomaly detection, Hellerstein *et al.* [6] attempt to model the behavior of a production web server and to predict threshold violations that are indicative of abnormal behavior. Lakhina *et al.* [9] use principal component analysis to model origin-destination traffic data and split it into normal and abnormal components. They subsequently detected traffic anomalies by identifying periods during which the magnitude of the abnormal traffic component exceeded particular magnitude measures. More recently, Burgess [3] has developed an anomaly detection scheme that combines two techniques; first, the use of time-series modeling and analysis to evaluate the statistical significance of anomalies and, secondly, the qualitative identification of noteworthy events. Co-stimulation is subsequently used to look into anomalies that are both statistically significant and noteworthy.

In comparison, we use a classifier to predict the number of connections initiated within fixed-length intervals. Such predictions are subsequently used to compute thresholds for issuing alerts. We are not aware of any work in the literature that has taken this approach.

3 Data Collection, Training and Thresholding

We instrumented a number of end-hosts (both laptops and desktops) to log both network traffic and machine level activity information. While the network traffic logging was instrumented using the *Windump* utility [14], the machine level logging used a homegrown application. The applications run in the background without any user interaction. We do not expect the instrumentation to change the host's behavior in anyway.

The pair of log files were periodically uploaded to a central server. During this process, the logging was turned off. At the central server, the traffic traces were post-processed using the open source *bro* tool to obtain *connection records*. These are a succinct summary of the "sessions" that involved the particular end host. In the case of a TCP connection, a connection record has well defined semantics; that is, it is a high level description of a TCP session between two end hosts. In the case of a UDP connection, where the protocol does not have rigid semantics, the connection record simply summarizes a "train of packets" between the two end points. Finally, the resulting connection records were synchronized with the much more terse machine activity records and then inserted into a database for easy lookup. The results in this paper are based on the connection and machine activity record traces collected from 9 different hosts (instrumented as a pilot prior to a larger data collection effort).

3.1 Classifier Training

The connection and machine activity records of each host were further post-processed to obtain combined records of the behavior for each host in successive 50-sec intervals (the specific window size was not found to be important; we merely inherit the value from an existing fixed threshold detector that we compare against [1]). Table 1 lists the features comprising these 50-sec records.

Table 1: Classification Features to predict c_t, the current connection count

Features	Name	Description
Number of Network Connections	c_{t-k}	Total for $1 \le k \le 10$
	c_{tcp}	TCP_{t-1}
	c_{udp}	UDP_{t-1}
	c_{icmp}	$ICMP_{t-1}$
Host	DOW	Day of the week
	$Weekday?$	Weekday or weekend?
	TOD	Hour of the day (0..23)
	$H(dstIP)$	IP destination Entropy
	$H(dstPort)$	Destination port Entropy
	$Load_{min}$	Minimum CPU load
	$Load_{max}$	Maximum CPU load
	$Load_{avg}$	Average CPU load
	App_1	Most used application
	App_2	2^{nd} most used app.
	App_{total}	# of applications used

Using the data sets of these features, we trained classifiers to predict the number of connections, c_t, initiated by a host during 50-sec intervals. This was done as follows. First, we split the data sets for each host into training and testing sets of equal size. Secondly, we used the features in Table 1 to train naïve Bayes, Bayes network and decision tree (J48) classifiers to classify each record in the training set into classes representing the number of connections initiated by the respective host in the current interval—the predicted *class distribution*. We used 6 different connection count classes, corresponding to bins of either 0, 1, 2, 3–5, 6–10, or > 10 connections. Finally, we tested the accuracy of the resulting classifiers (either naïve Bayes, Bayes network, or J48) against the testing set—we define classifier accuracy as the frequency with which the true connection count c_t falls in the predicted connection count bin. The size of the training set and the accuracy, expressed as the error rate of each classifier type for each of the hosts are shown in Table 2.

By using the connection counts in the previous $k = 10$ time intervals as part of our feature set, we introduce a time-series flavor to classification-based prediction. In effect, our classifiers predict the number of connections initiated, taking into account the connection count history. All the training and testing of the classifiers was done using the $Weka$ [13] machine learning toolbox.

3.2 Classification-Based Thresholding

Our adaptive thresholding scheme uses the class probability distributions obtained using our classifiers as follows. Figure 2 depicts an idealized continuous probability distribution of the predicted number of connections. The idea behind our thresholding scheme is to set the threshold such that the cumulative probability distribution above the threshold amounts to the desired FP rate. Our approach has two advantages. First, setting the threshold to achieve the same FP rate, we ensure that all end-host detectors appear homogeneous to the aggregate detectors they report to. Secondly, it results in a threshold that is tied to the predicted number of connections—the threshold is larger than the most likely predicted number of connections, separated from it by the so-called "gap", as shown in Figure 1. This gap bounds from below the time-averaged probability of not detecting incremental anomalous traffic; to be precise:

$$gap(\hat{C}_t) = \frac{1}{T} E\left[\int_0^T P(c_t \le \hat{C}_t \le \hat{H}_t)\, dt\right],$$

where $\hat{C}_t$ is the predicted class distribution, c_t is the true value, $\hat{H}_t$ is the predicted threshold, and T is the duration of the test or train data. The probability term is set to zero at times when $c_t \ge \hat{H}_t$.

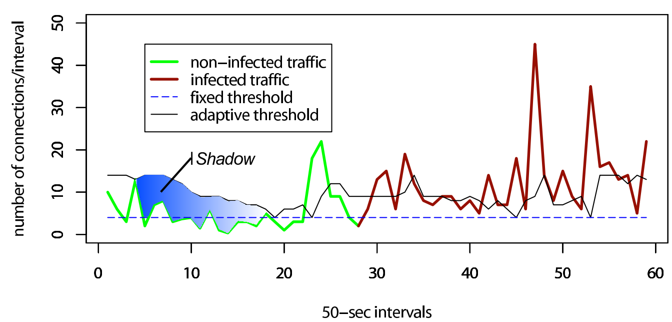

Figure 1: An adaptive threshold follows the traffic level trend, improving accuracy. Since it is not a perfect predictor, gap still appears as a "shadow" in which worm traffic can hide.

A predictor should minimize this gap in addition to minimizing its error rate. A tighter prediction distribution will result in a threshold that tracks normal traffic closely, reducing the gap. A predictor that minimizes the chance of normal traffic falling in the gap maximizes the average *true positive* (TP) rate of the detector.

Our classifier output, however, is an assignment of probabilities over classes corresponding to discrete connection count bins. Thus, our adaptive thresholding scheme computes the desired threshold by interpolating within the bin that spans the desired threshold.

Table 2: Predictor and Detector Accuracy compared to Training Set Size

Host	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Training Set Size	2779	5414	12175	2390	1831	2374	4378	282	5596
Naïve Bayes									
TestSet Accuracy (%) *	34.44	82.63	74.42	28.22	52.45	30.71	67.37	42.15	84.52
(Adaptive Gap)- (Fixed Gap) **	0.04	-0.63	-0.51	0.04	-0.07	0.03	-0.09	0.13	-0.77
Δ AUC Value *	0.0325	0.2898	0.2554	0.0010	0.1226	0.0222	0.1135	-0.0620	0.2788
Bayes Network									
Test Set Accuracy (%) *	54.81	86.15	74.96	53.76	56.54	51.94	73.21	58.74	82.77
Δ AUC Value **	0.0738	0.2968	0.2601	0.0761	0.1401	0.0594	0.1619	0.0157	0.2873
J48									
Test Set Accuracy (%) **	62.15	87.66	80.39	58.01	57.18	61.37	80.57	61.88	87.97
Δ AUC Value **	0.1041	0.3071	0.2830	0.0842	0.1413	0.0776	0.1548	0.0228	0.2932

(Spearman rank correlation with Training Set Size: * significant at 5% level, ** at 1% level)

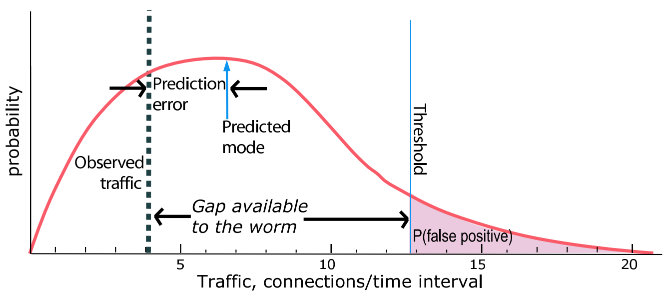

Figure 2: Various aspects of the predicted distribution of the network traffic. To reduce the network resources available to the worm, the gap between the true normal traffic level and the threshold based on a average false positive (FP) rate must be reduced.

4 Results

We evaluated the performance of our adaptive thresholding scheme using three metrics: the accuracy of the classifier used in our adaptive thresholding scheme, the reduction in the gap due to the adaptive threshold, and the improvement in detecting simulated worm traffic compared to a fixed thresholding scheme. Since we do not have a precise characterization of day-zero threats, none of these ways is definitive, however the consistency among them suggests that the algorithm is robust.

Classifier Accuracy The performance of our adaptive thresholding scheme is tied to the probability that the worm traffic falls beneath the threshold. As Figure 2 shows, the "gap" between the traffic and the threshold also depends on the width of the class distribution. Clearly a predictor that correctly places all its predicted mass in the correct bin would also minimize this variance. However, just minimizing the accuracy of the classifier may not be sufficient to improve detection performance. Nevertheless, we observe that a threshold controlled by a classifier that optimizes accuracy significantly outperforms a fixed threshold detector. The accuracy of the classifiers used in our adaptive thresholding experiments for each of the hosts are shown in Table 2.

Worm Detection Performance We compare the detection performance of our adaptive thresholding scheme against that of a fixed thresholding scheme using a worm model that adds a constant but incremental amount of traffic in each time period. Our metric for comparison are the ROC curves obtained for each thresholding scheme; that is, plots of the FP against the TP rate of detection. Figure 3 presents prototypical examples of such performance comparisons. Note the difference in horizontal scales; only the left-most portion of the graph is shown for the better curves.

We speculate that best ROC curve performance occurs when the adaptive threshold can take advantage of lulls in the traffic to lower the threshold and intersect the worm traffic, as shown in the first ROC plot. However, as shown in the second plot, there are cases when the worm traffic is quite predictable, and a fixed threshold (here, equal to 4) set below the worm traffic level picks up all the worm traffic. The worst performance occurs when the class predictions are poor, causing the threshold to wander. This explains the times when the fixed threshold out-performs the adaptive.

Reducing the "Gap" We summarize our comparison of the fixed and our adaptive thresholding ROC curves by considering the Area Under the ROC Curve (AUC) [7] values for the 27 runs (3 worm traffic rates by 9 hosts) of each classifier. The AUC is a performance measure that considers both the prediction probability and the classifier accuracy to strictly order ROC curves. Table 2 shows the difference in AUC values averaged over the 3 worm traffic rates by which each host traffic trace was

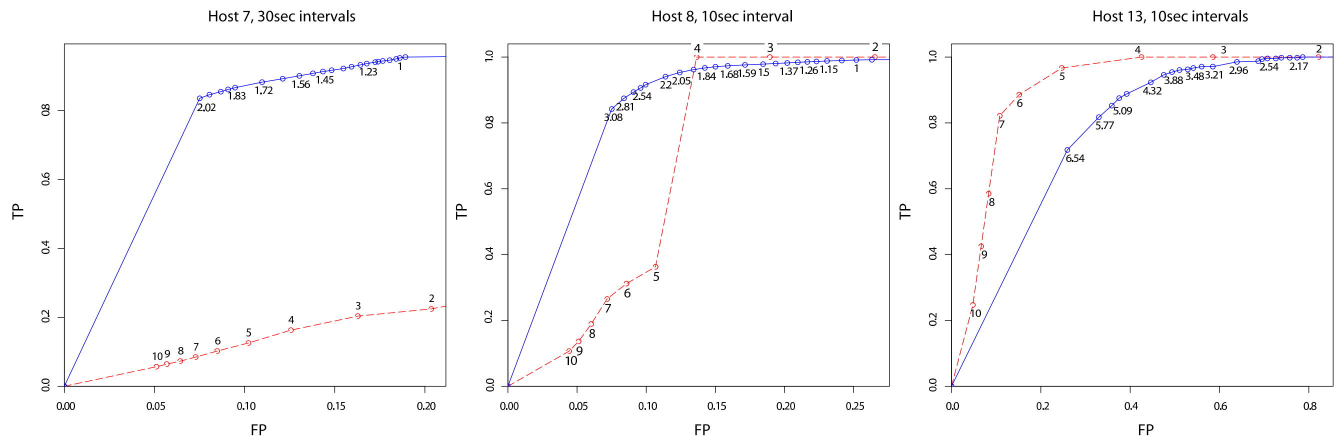

Figure 3: The three ROC plots shown here are selected from the 27 naïve Bayes cases. As illustrated by the left-most plot, adaptive threshold detectors typically show better performance. The exceptions are hosts with limited training data, as the other plots show. The middle plot shows a characteristic "knee" in the fixed threshold detector curve when the threshold equals 4 connections per time interval. The solid lines are the adaptive threshold curves; the dashed lines, fixed.

tested. Figure 4 presents the median, 50th percentile, and extrema of the AUC values for the 27 runs using each of the three classifier types. In all but a few runs, our adaptive thresholding scheme outperforms the fixed thresholding scheme.

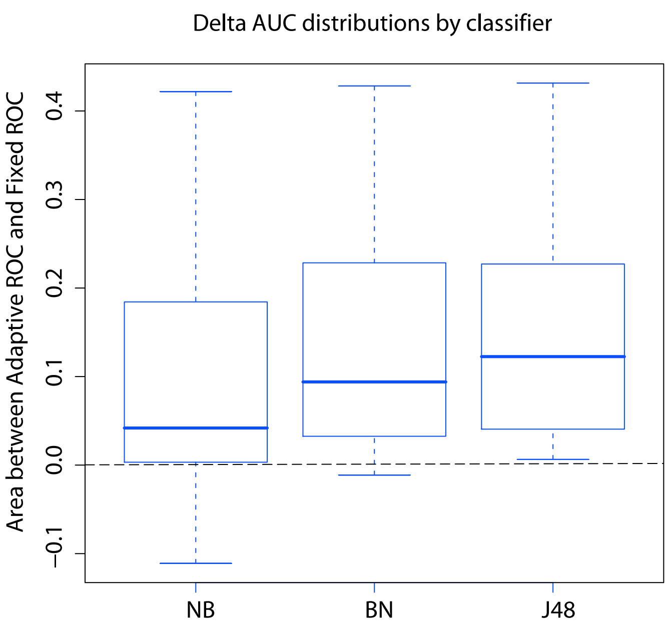

Figure 4: Barplots depicting the median, 50th percentile, and extrema of the differences between the adaptive and fixed AUC values when using each of the naïve Bayes, Bayes network, and J48 classifiers to set the alert thresholds. In almost all cases the difference between adaptive and fixed AUC values show that the adaptive threshold is strongly favored.

There is a strong relationship between the Δ-AUC values and classifier accuracy. This is supported by the Spearman's rank correlation coefficient p-values of 0.002, 0.005 and 0.016 for the naïve Bayes, Bayes network, and J48 classifiers, respectively, and the Δ-AUC values for the respective classifiers.

One may ask, since we have both positive and negative worm-traffic training examples by which we've evaluated performance, why not just frame the adaptive detector design as a supervised learning problem? The answer is that our goal is to design a robust anomaly detector, not to optimize it against a conjectured worm threat (and not a very clever one at that). Hence, we consider a more general evaluation measure, the expected gap between normal traffic and the threshold. Since FP rates are by design held constant, a comparison of the expected fixed threshold gap and the expected adaptive threshold gap offers a measure of comparison between thresholds based on fewer assumptions. We've made a crude estimate of this value by numerically integrating the empirical marginal distribution of the traffic between the threshold and the observed traffic levels when the traffic level is below threshold. Table 2 presents this difference in the case of the naïve Bayes classifier, and shows strong agreement with the other measures.

5 Conclusions

This paper has shown that, compared to a fixed threshold detector, an adaptive-threshold worm traffic anomaly detector can reduce the worm's opportunity to generate outgoing traffic and yet remain undetected. Various performance measures of the improvement indicate that the result is robust to the choice (and to some extent the quality) of an off-the-shelf classifier and to the exact scoring method applied.

We recognize that this work is preliminary. This adaptive detector still leaves some gaps that a worm can exploit. There are several directions for improvement we are pursuing, to both better define a threat model that gives better semantics for abnormal behavior, and, based on this, to optimize the threshold algorithm. Here are three vulnerabilities of the current approach:

1. *Worm traffic that exploits the prediction "shadow":* A worm that is aware of the predictor's behavior can exploit its errors. A time-series based predictor will tend to lag the actual traffic, and take a few time-steps to respond to the change in traffic level, as shown in Figure 1. Thus a resourceful worm could transmit, albeit to a more limited degree, in the "shadows" that follow bursts of normal traffic.

2. *Performance variation with training set size:* The results in Table 2 show a significant relation between longer training set traces and classifier performance. This apparent learning curve effect might actually be due to a reduction of the adaptive threshold close to zero during traffic lulls. We speculate that the longer length traces are largely due to the stretches of idle time they include. If further study shows this to be the case it may help to "share" training data among hosts.

3. *Manipulation of the prediction by adversary:* If the worm can game the inputs to the prediction function, the worm can nudge up the threshold and create a larger gap for itself. Observed network traffic would then be, in part, the result of the worm influencing the predictor. We have addressed this by including several variables in the predictor function that are derived from the host internal state, such as user activity, that are not as subject to worm manipulation. Conceivably the predictor could be composed of two classifiers that vote on the traffic level, one driven by network traffic levels, the other by machine state features.

If this detector were to be used as a component in a distributed detection system, it is not obvious that the adaptive end-host detectors performance gains would make for proportional improvements in system-wide detection. Static thresholds may not be as bad as they sound: The lulls in traffic at one host would not be coïncident with those at other hosts, so a worm would always be facing some tight-gapped detectors. Averaging over detectors presumably removes some of the worm's advantage. We plan to incorporate the adaptive detection in system-level simulations to explore this question.

Acknowledgments

We gratefully acknowledge Intel Research's DDI team, including Abe Bachrach, Denver Dash, Branislav Kveton, Alex Newman, Eve Schooler, Micah Sherr, Tomas Singliar and Nina Taft, who contributed to the theory and simulation results of the larger project of which this is a part. This project is internally funded by Intel Research.

References

[1] Ravi Sahita Gayathri Nagabhushan Priya Rajagopal David Durham, U. S. An os independent heuristics-based worm-containment system, 2005. Intel Tech.

[2] Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A., and Salamatian, K. Traffic classification on the fly. *Computer Communication Review 36*, 2 (2006), 23–26.

[3] Burgess, M. Probabilistic anomaly detection in distributed computer networks. *Sci. Comput. Program. 60*, 1 (2006), 1–26.

[4] Dash, D., Kveton, B., Agosta, J. M., Schooler, E. M., Chandrashekar, J., Bachrach, A., and Newman, A. When gossip is good: Distributed probabilistic inference for detection of slow network intrusions. In *AAAI* (2006), AAAI Press.

[5] Erman, J., Arlitt, M., and Mahanti, A. Traffic classification using clustering algorithms. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data* (New York, NY, USA, 2006), ACM Press, pp. 281–286.

[6] Hellerstein, J. L., Zhang, F., and Shahabuddin, P. A statistical approach to predictive detection. *Computer Networks 35*, 1 (2001), 77–95.

[7] Huang, J., and Ling, C. X. Using auc and accuracy in evaluating learning algorigthms. *IEEE Transactions on Knowledge and Data Engineering 17*, 3 (2006), 299–310.

[8] Karagiannis, T., Papagiannaki, K., and Faloutsos, M. Blinc: multilevel traffic classification in the dark. In *SIGCOMM* (2005), R. Guérin, R. Govindan, and G. Minshall, Eds., ACM, pp. 229–240.

[9] Lakhina, A., Crovella, M., and Diot, C. Characterization of network-wide anomalies in traffic flows. In *Internet Measurement Conference* (2004), A. Lombardo and J. F. Kurose, Eds., ACM, pp. 201–206.

[10] McGregor, A., Hall, M., Lorier, P., and Brunskill, J. Flow clustering using machine learning techniques. In *PAM* (2004), C. Barakat and I. Pratt, Eds., vol. 3015 of *Lecture Notes in Computer Science*, Springer, pp. 205–214.

[11] Moore, A. W., and Zuev, D. Internet traffic classification using bayesian analysis techniques. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (New York, NY, USA, 2005), ACM Press, pp. 50–60.

[12] Roughan, M., Sen, S., Spatscheck, O., and Duffield, N. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2004), ACM Press, pp. 135–148.

[13] Weka 3: Data mining software in java, 2006. `http://www.cs.waikato.ac.nz/ml/weka/`.

[14] Windump: tcpdump for windows, 2006. `http://www.winpcap.org/windump/`.