Clear and Present Data: Opaque Traffic and its Security Implications for the Future

Andrew M. White*, Srinivas Krishnan*, Michael Bailey†, Fabian Monrose*, Phillip Porras‡

*University of North Carolina at Chapel Hill {amw,krishnan,fabian}@cs.unc.edu

†University of Michigan mibailey@eecs.umich.edu

‡SRI International porras@csl.sri.com

***Abstract*—Opaque traffic, i.e., traffic that is compressed or encrypted, incurs particularly high overhead for deep packet inspection engines and often yields little or no useful information. Our experiments indicate that an astonishing 89% of payload-carrying TCP packets — and 86% of bytes transmitted — are opaque, forcing us to consider the challenges this class of traffic presents for network security, both in the short-term and, as the proportion of opaque traffic continues to rise, for the future. We provide a first step toward addressing some of these challenges by introducing new techniques for accurate real-time winnowing, or filtering, of such traffic based on the intuition that the distribution of byte values found in opaque traffic will differ greatly from that found in transparent traffic. Evaluation on traffic from two campuses reveals that our techniques are able to identify opaque data with 95% accuracy, on average, while examining less than 16 bytes of payload data. We implemented our most promising technique as a preprocessor for the Snort IDS and compared the performance to a stock Snort instance by running both instances live, on identical traffic streams, using a Data Acquisition and Generation (DAG) card deployed within a campus network. Winnowing enabled Snort to handle a peak load of 1.2Gbps, with zero percent packet loss, and process almost one hundred billion packets over 24 hours — a 147% increase over the number processed by the stock Snort instance. This increase in capacity resulted in 33,000 additional alerts which would otherwise have been missed.**

I. Introduction

The successful monitoring of security policies via Intrusion Detection Systems (IDS) critically depends on scalable and accurate techniques for inspecting traffic. Deep packet inspection (DPI) is a common method for performing such inspection, especially when security policies require determinations based on information not accurately reflected by network ports, protocols, or hosts. In fact, the market for DPI appliances alone reached almost $1 billion in 2009, and continues to grow.[1] Since DPI must deal with huge volumes and significant heterogeneity of traffic, DPI designers often trade off accuracy of detection with resource demands [34]. DPI systems cannot generally derive useful information from *opaque* (i.e., encrypted or compressed) packets; thus, we propose improving the performance versus quality curve through the quick and accurate *winnowing*, i.e., filtering, of opaque traffic, and evaluate a number of techniques for doing so. Such techniques can improve the performance of DPI engines by quickly and accurately separating out low-value packets from the data stream they inspect; this is particularly important in high-performance environments, where the sheer volume of traffic can be staggering. At our campus, encrypted traffic alone appears to make up an average of 13% of the total traffic, based solely on port and protocol assumptions. We believe that the fraction of encrypted traffic on real networks will only continue to rise, as more web services follow in the footsteps of Google and Facebook in offering encrypted connections and the movement toward ubiquitous end-to-end encryption [2] gains steam. In particular, private corporate networks, where secure communications are often required, undoubtedly see significantly higher proportions of encrypted traffic. The class of opaque traffic encompasses not only encrypted connections, but also compressed entities, which, in the modern era of streaming video, comprise an even more significant fraction of network traffic: some sources indicate that streaming video accounts for 35% of peak network usage, a proportion which is only increasing.[2]

In fact, our experiments revealed a surprising preponderance of opaque traffic: in a 24-hour weekday period, nearly 90% of TCP packets traversing a major university network (with peak loads of 1.2Gbps) were found to be opaque. This truly staggering figure suggests a broader issue for the security community moving forward: as more and more payloads become opaque to DPI engines, how can we detect and prevent obscured threats? While content-based signatures will continue to be relevant as a detection mechanism for direct attacks (e.g., those exploiting buffer overflows in networking routines) against networked systems, bypassing current DPI engines can be as simple as encrypting the relevant exploit code, particularly for indirect attack vectors (e.g., exploits embedded in documents). Thus we, as a community, face both an immediate need to separate the wheat from the chaff—to winnow low-value, i.e., opaque, packets to enable our existing detection methods to operate in high-speed environments—and a long-term need to develop methods for coping with attacks embedded in opaque, and particularly encrypted, traffic. This work represents a first step toward solving both problems: our techniques enable the fast and accurate identification of opaque packets, either as chaff to

[1] Gartner's *Magic Quadrant for NIPS (Dec 2010).*

[2] Sandvine's *Global Internet Phenomena Report (Fall 2011).*

be discarded, or as the inputs to specialized detection engines targeting opaque traffic.

Unfortunately, the identification of opaque network traffic is very challenging. While signatures can identify many known opaque protocols (e.g., SSL/TLS, SSH), some protocols (e.g., Bittorrent's Message Stream Encryption) are specifically designed to avoid signature matching. In addition, signature-based approaches for identifying new opaque protocols require constructing and deploying new signatures for each new protocol. More importantly, existing techniques for identifying opaque data often require examination of a large number of bytes, which can be computationally and memory intensive on high-speed networks [5, 12]. Similarly, inspecting application-layer content-types to determine opacity requires resource-intensive flow reassembly. To compound matters, such detectors cannot rely on HTTP content-types as they are often inaccurate (see §III).

We take a first step toward addressing these challenges by providing novel methods for quickly and accurately determining whether a packet is opaque. Our techniques are both *port-* and *protocol-agnostic*, allowing for the detection of opaque traffic regardless of the port(s) or protocol(s) involved. We concentrate on efficient techniques which can operate on limited portions of packet payload, such as small-sample fixed-size and sequential hypothesis tests, in order to minimize overhead and allow scarce computational resources to be reserved for high-value analyses.

In the present work, we necessarily evaluate our techniques in isolation, i.e., without attempting to fully integrate our approach with a particular system or methodology, in order to minimize the number of factors potentially biasing our results. That said, we envision the identification of opaque traffic not as the relatively standalone system portrayed herein — there is no "silver bullet" for network intrusion detection — but rather as an efficient new tool in the network analyst's toolbox. In particular, forensic recording systems, such as Time Machine [23], often avoid archiving full connections in order to reserve limited storage space for the higher-value packets at the beginning of a connection. An admitted weakness of this design decision is that attacks in the discarded portions of these connections go unnoticed. Our techniques mitigate this concern to an extent by providing a means for such a system to detect potentially high-value plaintext packets in lengthy connections. Similarly, our tests are simple and can be implemented on an FPGA, providing a benefit even to systems which make use of a hardware component, such as those following *shunting* approach proposed by Gonzalez et al. [15].

As a final note, opaque traffic identification has value that extends beyond filtering to policy monitoring and enforcement. For instance, operators may wish to monitor and enforce policies within their network (e.g., flagging encrypted packets tunneled over unencrypted connections, an odd practice used by botmasters to hide command-and-control messages in otherwise mundane HTTP traffic [6, 38]). Similarly, one might wish to flag unencrypted traffic where encrypted traffic is expected, such as might occur if an SSH connection is subverted [27], or on institutional networks where end-to-end encryption is required. Identifying opaque traffic may also help to discover applications tunneled over other protocols (e.g., video traffic tunneled over HTTP), or to provide sanity checking for strongly typed networking [26].

Our contributions include: 1) the concept of opaque traffic as a distinguishable class of network traffic; 2) the development, evaluation and comparison of multiple techniques for detecting such traffic (Section §II, Appendix §A); 3) an operational analysis of modern network traffic with respect to opacity (Section §III-A); and 4) an evaluation, at scale, of the potential of our techniques for reducing the load on modern intrusion detection systems (Section §III-B).

II. Approach

An important requirement of ours is to minimize the amount of a packet's payload that we must inspect, as doing otherwise is both computationally and memory intensive on high-speed networks [5, 12]. These overheads severely restrict the numbers of samples available to us for any of the tests we explore. Therefore, for the remaining discussion, we propose *detectors* based on small-sample fixed-size hypothesis tests and sequential hypothesis testing. The latter allows us to make decisions quickly by examining only as many samples as needed to support a given hypothesis.

Our detectors are based on determining whether the bytes examined are drawn from a uniform distribution or some other, unknown, distribution. Our first instinct, when faced with the problem of measuring the uniformity of a set of samples, was to use entropy-based measures. However, as we show later, *accurate* entropy testing requires significantly more samples than is practical in our setting, and is less efficient than more direct methods based on the samples themselves rather than on a derived statistic such as entropy.

Both encrypted and compressed traffic will exhibit high entropy, i.e., the distribution of bytes will be close to uniform. In this paper, we consider these two cases as belonging to the same equivalence class of "*opaque*" objects as far as DPI engines are concerned. That is, regardless of whether the packets that belong to a session are compressed versus encrypted, they will be forced to go through the slow path wherein the engine must analyze all packets of these sessions, but will still fail to derive any useful information in the vast majority of cases. Hence, from the perspective of DPI engines, there is little value in attempting to analyze these packets. As Cascarano et al. [5] observed in their experimental evaluations, these slow paths incurred CPU overheads that can be *several orders of magnitude higher* than the average case for transparent traffic.

In our search for the best performing test for our goals, we examine several fixed sample-size hypothesis tests. There

are a number of standard techniques for testing uniformity; however, many of these are designed for testing the uniformity of the outputs of a pseudo-random number generator, and thus assume access to millions of samples (e.g., [25]). These tests are unsuitable in our context because our sample size is extremely limited. We instead evaluate the appropriate fixed-size tests (the likelihood ratio test and the discrete Kolmogorov-Smirnov test) and two variants of the sequential probability ratio test. We assess the effectiveness of each test in two scenarios, differentiated by the domain of the underlying distribution. In the first scenario, we directly test the distribution of byte values in the packet; for the second, we instead test the distribution of entropy values over n-byte "blocks" in the packet.

For pedagogical reasons, we leave descriptions of the less successful tests, along with the details of the parameter exploration experiment itself, to §A2, and discuss only the more effective tests here. In summary, the closely-related likelihood ratio and sequential probability ratio tests, operating directly on the byte values instead of on derived entropy values, consistently outperform the other tests. The poor performance of the entropy tests is related to the birthday problem, in that the (byte-)entropy of any n bytes is the same unless there is a collision (e.g., there are two bytes which share the same value). According to the birthday paradox, the *a priori* probability of a collision in 8 bytes, even when choosing only from only the 95 printable ASCII characters, is only about 26%. This means that the entropy of 8 ASCII bytes is indistinguishable from that of 8 unique bytes 74% of the time. Thus entropy-based tests require substantially more than the number of samples available in our context to be effective. However, our parameter exploration experiment (see §A2) reveals that the the more successful tests require examining only 16 bytes of payload to be effective.

Since our goal is to discard opaque traffic (albeit encrypted or compressed), we let our null hypothesis be that the packet is opaque and our general alternative be that the packet is transparent. Specifically, the null hypothesis is that the values of the bytes in the packet are approximately uniformly distributed (e.g., the packet is compressed or encrypted); the alternative is that the distribution is non-uniform (e.g., ASCII text).

When the need arises for a probability density function for the alternative hypothesis, we use a simple distribution based on the intuition that plaintext packets will have a higher frequency of bytes whose values are less than 128 (as are the ASCII values). We parameterize this distribution by setting δ to the cumulative density of those values. For example, at $\delta = 0.75$ the alternative hypothesis is that 75% of the bytes in the packet have values less than 128.

Likelihood Ratio Test: A well-known theorem of hypothesis testing is the *Neyman-Pearson* lemma, which states that the *most powerful* test, i.e., that with the lowest expected *false positive rate* for a given *false negative rate*, of one simple hypothesis against another is the *likelihood ratio test* [41]. For a single sample, the likelihood ratio statistic is simply the ratio of the likelihood of the sample under the alternative hypothesis to the likelihood of the sample under the null hypothesis. For multiple samples, these likelihoods are replaced with the corresponding joint likelihoods.

Suppose we wish to test the simple null hypothesis $\mathcal{H}_0 : \theta = \theta_0$ against the simple alternative $\mathcal{H}_1 : \theta = \theta_1$ given a random sample $\bar{X}$. Then the Neyman-Pearson lemma [42, Theorem 10.1] states that the most powerful test of $\mathcal{H}_0$ against $\mathcal{H}_1$ is that where one rejects $\mathcal{H}_0$ if $\Lambda(\bar{X}) \geq q$ and accepts $\mathcal{H}_0$ if $\Lambda(\bar{X}) < q$, where q is determined by the desired level of statistical significance and $\Lambda(\bar{X})$ is the ratio of the likelihood of $\bar{X}$ under the alternative to the likelihood of $\bar{X}$ under the null hypothesis.

Sequential Probability Ratio Test: Sequential analysis is often used when there is a desire, such as in our context, to limit the number of samples examined. In fact, Wald and Wolfowitz have shown that, among all tests of two simple hypotheses with fixed error probabilities, the sequential probability ratio test (SPRT) minimizes the expected number of samples required for a decision under either hypothesis [43]. Sequential testing works by examining samples one-by-one, rather than all at once, and evaluating a decision function at each sample. This allows the test to stop examining samples as soon as it has found enough "evidence" for one hypothesis or the other.

Specifically, let $\alpha = P(\mathsf{accept}\ \mathcal{H}_1 | \mathcal{H}_0)$ be the probability of a *false negative*, that is, an erroneous prediction that the v_i are not uniformly distributed; similarly, define $\beta = P(\mathsf{accept}\ \mathcal{H}_0 | \mathcal{H}_1)$ as the probability of a *false positive*. In order to perform the test, we iterate through some sequence of samples $X_1, X_2, ...$; according to Wald's theory of sequential hypothesis testing [42], we choose at each iteration m one of three actions: $\mathsf{accept}\ \mathcal{H}_0$, $\mathsf{accept}\ \mathcal{H}_1$, or $\mathsf{continue}$, as follows:

$$\begin{array}{ll} \mathsf{accept}\ \mathcal{H}_0\ \mathsf{if} & \Lambda_m(X_1, X_2, ..., X_m) \leq g_0(m) \\ \mathsf{accept}\ \mathcal{H}_1\ \mathsf{if} & \Lambda_m(X_1, X_2, ..., X_m) \geq g_1(m) \\ \mathsf{continue} & \mathsf{otherwise.} \end{array}$$

Setting $g_0(m) = \frac{\beta}{1-\alpha}$ and $g_1(m) = \frac{1-\beta}{\alpha}$ gives the desired probabilities of false positives and false negatives.

A known drawback of this test is that it may not terminate within an acceptable number of samples. We alleviate this concern by exploring two variants, which we refer to as the *truncated* and *restricted* SPRTs. For the truncated SPRT, we specify a maximum number of samples which the test is allowed to examine; if this limit is reached without making a decision, then the fixed-size likelihood ratio test is applied to the samples examined so far. The restricted SPRT, on the other hand, works by sloping the decision boundaries such that they intercept the axis after the given number of samples. For the restricted case, we follow the approach

suggested by Bussgang and Marcus [4].

Based on the results of the parameter exploration experiment (see §A2), we use the truncated sequential method ($\alpha = 0.005, \beta = 0.005, \delta = 0.85$) for the remainder of the experiments in this work.

III. Evaluation

In order to ascertain the effectiveness of our techniques in different scenarios, we perform a number of experiments in both offline and online settings. First we show, in a controlled, offline experiment, that our techniques are able to accurately identify the opacity of different file types. We then verify the accuracy of our techniques under real-world network conditions by evaluating our detectors on traffic logs and traces collected at two major universities, an analysis which produced a number of interesting anomalies which we investigated with the help of a network operator. Finally, we show the utility of winnowing by comparing two otherwise identical Snort IDS deployments under identical traffic loads.

A. Offline Analysis

File Type Identification: To gain an understanding of how well our techniques were able to label the opacity of various common data types, we collected a set of files with known ground truth, including compressed archives and streams, encrypted files, executable binaries, and text files. We also attempted to cover different sub-types; e.g., we included five different text file encodings. The details of this set, which we believe to be a reasonable cross-section of common file types, are presented in Table I.

We gathered a base set of files from multiple sources, then applied standard compression and encryption algorithms to these files to create the remainder of the dataset. For executables, we used ELF binaries from the `bin` directories of a basic Ubuntu Server 10.04.3 installation (that we used for testing Bro and Snort). The files in each directory, including a number of Perl scripts, were then individually compressed using `tar` (which in turn uses `gzip` and `bzip2`) to provide `gzip` and `bz2` files and encrypted using `openssl` to provide examples of the RC4 and AES ciphers. The text files are the contents of the `man` path directory of the same Ubuntu installation, uncompressed and processed by `groff` into different encodings (ASCII, UTF-8, -16, and -32, and latin1); the PDF files are the archive of proceedings from a major computer security conference. Finally, the images were JPEGs scraped from public websites, including a departmental website and `nasa.gov`.

To simulate a network environment, we transmitted each object over an individual TCP connection on the loopback device of an Ubuntu virtual machine. We used `nc` to both send and receive, and `tcpdump` to collect the resulting packets. We report the proportion of opaque and transparent packets observed for each type of file in Table I.

Our test labeled more than 95% of compressed, encrypted and image file packets as opaque, as expected. Similarly, even with multiple different encodings, our techniques labeled more than 99.99% of text file packets correctly as transparent. Our test is less consistent on the executables. Inspection of the binaries revealed a large number of null bytes, suggesting that a more targeted alternative hypothesis, perhaps counting only the set of printable ASCII bytes rather than simply those with value less than 128, may improve our results. However, this must be contrasted with the straightforwardness and efficiency of checking whether the value of a byte is greater than 128 (which amounts to simply checking the high-order bit).

Type	Objects	Size (MB)	True Positive Rate
compressed	1410	40	97.8
encrypted	1410	91	98.9
text	5802	176	100.0
images	205	12	94.4
executable	498	43	34.5
pdf	1006	75	13.5

Table I
File Type Analysis

Content Type Matching: To assess the accuracy and performance of the techniques in §II, we instrumented the Bro IDS (version 1.6-dev-1491) to log packet-level information (with only coarse-grained payload information), providing a form of ground truth. Due to privacy restraints, we were unable to record full packet traces at scale; therefore, the experiments in this section are performed on logs generated by our instrumented version of Bro. For these experiments, we collected two logs (`log1` and `log2`) from two large university campuses.[3] Both logs were collected over several hours during a weekday. For simplicity, we only consider IPv4 TCP traffic. We labeled each packet by protocol using Bro's dynamic protocol detection [13], and restricted our analysis to two encrypted protocols (SSL and SSH) and two unencrypted protocols (HTTP and SMTP).

For each packet, we log the source and destination ports, the HTTP message type, the HTTP message encoding, and the payload length; we also store coarse-grained statistics in the form of a binary value for each byte of the payload (indicating whether the byte's value is less than 128), and the byte-value frequencies for each n-byte block of the payload. The latter are needed to calculate sample entropy at the block level and for the frequency-based tests (i.e., χ^2 and discrete K-S; see §A). In all cases, we only log information for the first 256 bytes of the payload.

We labeled each packet as 'opaque' or 'transparent' according to the expected data type for that packet: SSL and SSH packets are labeled opaque and SMTP packets are labeled transparent. For HTTP, we labeled packets based

[3]Our efforts to maintain data privacy and ensure IRB compliance for this collection effort are detailed in §VII.

on the HTTP Content-Type and Content-Encoding header fields (as given by the sender) for the corresponding HTTP message (see §B for details of the labeling). This allows us to further restrict our attention to only those content-types for which opacity should be predictable and consistent. For instance, the HTTP 1.1 specification states that "HTTP entities with no specified Content-Type should be identified by inspection of the contents" [14]; we remove any such packets from our analysis because we have no way of determining ground truth. However, as we discovered during the course of this work, the HTTP content-type headers are often inaccurate and misleading (details are given in the following sections).

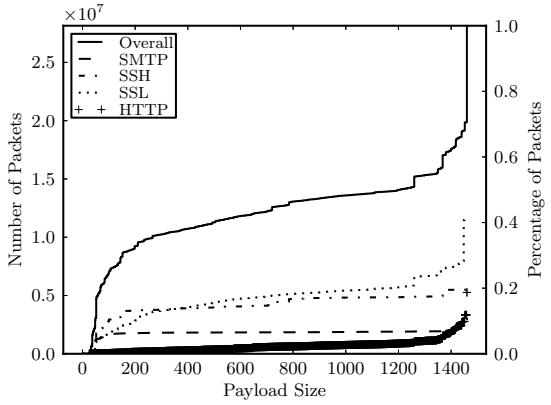
Unfortunately, Bro suffers from performance problems (see [12]) on high-speed networks, especially when port-based analysis is *disabled* (as is necessary to force Bro to determine protocols solely by inspection). Therefore, we discard any HTTP packets which belong to flows in which Bro experienced losses. We do so because the dropped packet(s) could have contained essential labeling information (e.g., a message header) necessary for determining the content-type and encoding.

After filtering the logs down to only those packets which met the criteria outlined above, over 39 million packets (across $\approx$ 3.8 million bi-directional flows) remained from `log1` and over 24 million (across $\approx$ 2.3 million bi-directional flows) remained from `log2`. The traffic makeup for both logs is shown in Figure 1.

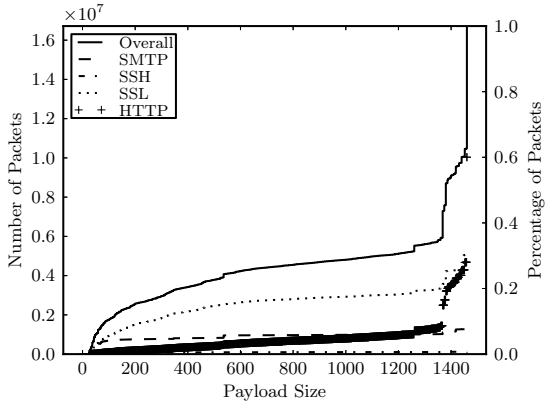
The content-type distribution (Figure 2) for the top content types in both logs reveals some interesting contrasts between the two. In particular, video types are prevalent in `log2` while `log1` consists mainly of web-browsing traffic (over 80% of `log1` HTTP packets have content type 'image/jpeg' or 'text/html', compared to 40% in `log2`). The content-encoding field is only specified for a small proportion of packets in our logs, and the only significant encoding is 'gzip,' at 4.0% in `log1` and 6.5% in `log2`. All other encodings combined account for less than a tenth of a percent of packets in each log.

We performed a large-scale analysis on both `log1` and `log2`; the overall results are given in Table II. Examining only 16 bytes per packet, offset 8 bytes from the start of each packet, we achieve a match rate, i.e., the percentage of examples for which our techniques produced the same label as expected from the content type, of 95.1% on `log1` and 96.0% on `log2`. We refer to 'match' rates here, rather than false-positive or false-negative rates, due to the large quantity of mislabeled content-types we encountered.

In the case of encrypted traffic (i.e., TLS/SSL) we accurately classified approximately 95% of the traffic. However, the mismatches are particularly interesting. Figure 3 shows the distribution of packet IDs, where a packet's ID is its position in the bi-directional flow (i.e., a packet with ID zero is the first packet sent by the originator of the connection).



(a) `log1`



(b) `log2`

Figure 1. CDFs of payload size for the protocols examined in the two campus network logs

	`log1`		`log2`	
Protocol	Match Rate	Examples	Match Rate	Examples
SSH	94%	7.3m	97%	157k
SSL/TLS	96%	16.4m	94%	5.5m
SMTP	86%	3.35m	80%	1.4m
HTTP	91%	9.7m	85%	13.8m
Total	94%	36.7m	96%	20.8m

Table II
EXPERIMENTAL RESULTS (`log1` AND `log2`)

Notice that 94.8% of the mismatches for SSL/TLS occur within the first 5 packets, and 95% within the first 6 packets for SSH. These packets are all in the connection set-up phase and hence are not, in fact, encrypted. Moreover, these connection set-up packets, particularly any SSL certificates (typically around ID 2), may be of interest to DPI engines even when the encrypted payload is not.

A closer analysis of our overall results reveals that 50% of the transparent-as-opaque mismatches for `log1`, and 15% for `log2`, are from SMTP traffic, which we surmise includes opaque attachments for which we do not have accurate

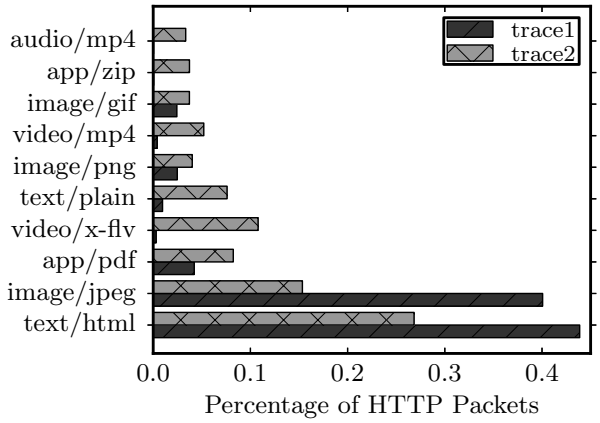
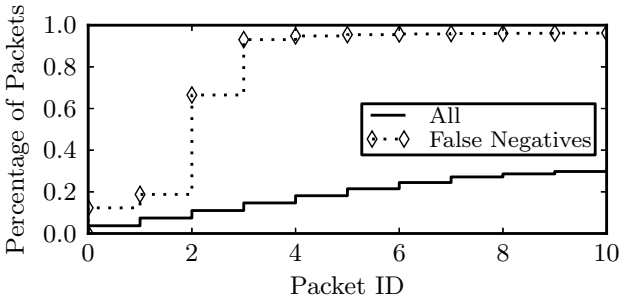
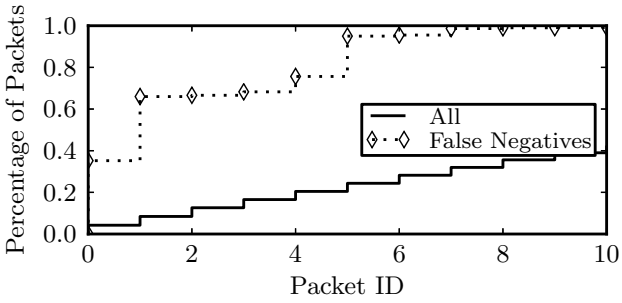


Figure 2. HTTP Content-Type Distribution



(a) SSL/TLS



(b) SSH

Figure 3. CDFs of Packet IDs

labeling information. Of the HTTP transparent-as-opaque mismatches (Figure 4), many are PDF and Flash (both of which can serve as a container for many other object types), but a surprising proportion are of type `text/plain`.

We investigated the `text/plain` mismatches from `log1` further and concluded, based on the observed byte-value distributions, that if these packets are in fact plaintext, the method used for encoding is non-standard. Figure 5(a) depicts the byte-value distribution for the mismatches and for the `text/plain` packets where the encoding was specified explicitly as one of UTF-8, UTF-16, US-ASCII

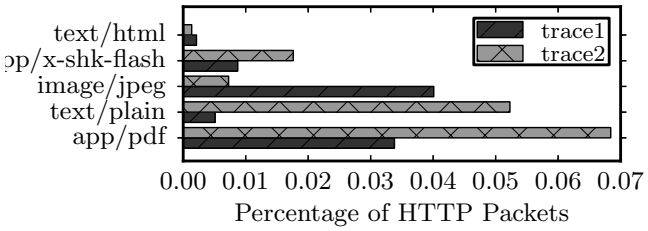


Figure 4. HTTP Mismatched Content-Types

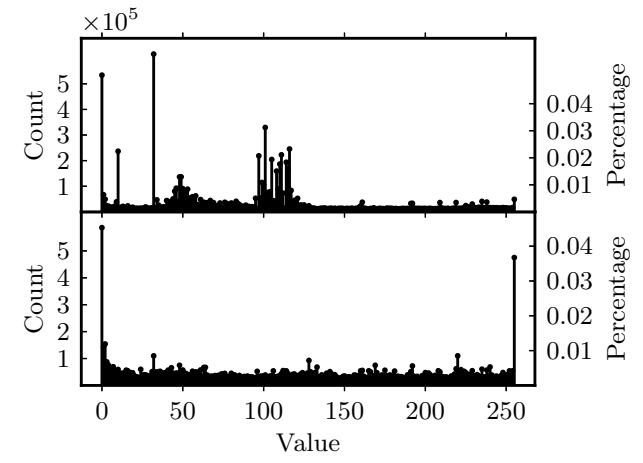
or ISO-8859-1. As expected, the latter distribution has significant mass around the ASCII characters 'A'-'z' (values $65 - 122$ in Figure 5) while the former does not. A similar finding holds for the opaque-as-transparent mismatches in the `image/jpeg` case: the distribution (bottom, Figure 5(b)) has striking similarities to that observed for the case of known plaintext encodings (top, Figure 5(a)), and moreover, is quite different from that of the opaque JPEGs (top, Figure 5(b)). Unfortunately, without access to actual payloads for these two traces, we cannot say what was the underlying cause for the peculiar distribution in the content flagged by our methods, but we believe this underscores the utility of our techniques — i.e., we *successfully identified anomalous instances in both cases*. We revisit this issue in the next section.

Finally, we examined the number of iterations needed by the truncated test to make a decision regarding each packet. With the maximum sample size set at 16 bytes, 45% of the packets can be classified in 12 bytes or less.

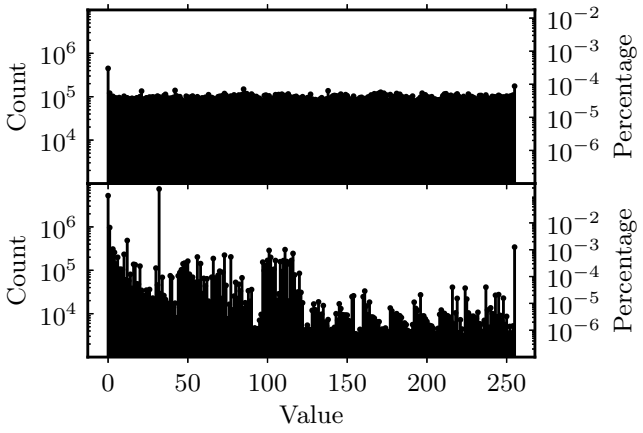
Operator Analysis: To gain deeper insights into the issue of Content-Type mismatches, we performed another experiment in which a resident network operator was able to manually inspect payload data from a third dataset, for which we were able to collect *full* payloads under the supervision of our local network operator. This trace covers four weekday afternoon hours of traffic from a university computer science department, and consists of 27 million packets. To enable this inspection, we instrumented Bro (version 2.0) to save both HTTP entities and the payloads of TCP connections to disk. We then ran the instrumented Bro against the port 80 (HTTP) and port 443 (SSL) traffic in the trace.

We determined the opacity of each packet, saving those entities and connection payloads wherein the first five packets were mismatches. For the HTTP entities, we define a mismatch as having an opacity different from that implied by the Content-Type or Content-Encoding. For the streams on port 443, we consider transparent packets to be mismatches. We also captured relevant metadata, such as the URI, Host, and MIME-type of the resulting file (determined using Bro's `libmagic` interface).

We discovered a number of mismatched HTTP entities in the trace, of both the transparent-as-opaque and opaque-



(a) `text/plain` (top: known encodings (transparent), bottom: opaque)



(b) `image/jpeg` (log-scaled; top: opaque, bottom: transparent)

Figure 5. Byte-value Distributions for Anomalies

as-transparent varieties. In the former case, many of these mismatches were HTTP entities labeled with Content-Type 'text/plain' and no stated encoding; we determined from the metadata that most of these were either images or compressed bundles. These compressed bundles included what appear to be updates to Symantec antivirus, extensions to the Plex media center, and an installer for the DivX video codec (a `.cab` file). The images were identified by MIME-type as JPEGs. One interesting mismatch declared a Content-Type of 'text/javascript', with no encoding, while the MIME-type reported was 'application/octet-stream' and the filename `.gz`.

Of the opaque-as-transparent mismatches, a number of entities were labeled as JPEG and GIF images by Content- and MIME- type but were flagged as transparent by our techniques. Many of these contain significant text, which we speculate may be metadata. We also found a large number of streams on port 443 with predominantly transparent packets. According to our network operator, these streams comprised: cleartext Yahoo Voice connection information (including *account names*), *names and email addresses* from a social networking chat site, and what appeared to be *IM conversations*. As an aside, in looking at flagged mismatches while testing, one of the authors discovered his AIM contact list transmitted in the clear over port 443 by Adium!

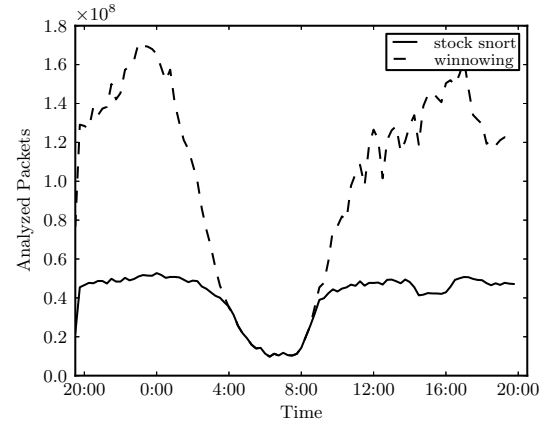
B. Online Analysis

To further demonstrate the utility of winnowing opaque traffic in a real-world environment, we implemented our techniques as a preprocessor to the Snort IDS. Much of Snort's functionality, such as stream reassembly and HTTP inspection, is implemented as preprocessors which are executed after network and transport layer packet decoding but before engaging the rule-matching engine. We positioned our winnowing preprocessor to intercept packets before reaching the stream reassembly module.[4] This allows us to drop opaque packets early enough to avoid incurring the overhead (over 30% of Snort's run-time in some experiments) of stream reassembly and long before reaching the pattern matching engine.

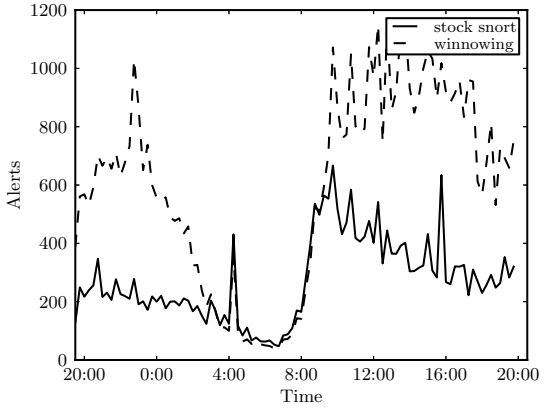
The ruleset used was provided by a major university, which uses Snort primarily for post-facto forensics, and contains 1606 rules. As a sanity check, and to provide results on a public dataset which could therefore be easily reproduced, we evaluated both stock Snort and Snort with our winnowing preprocessor on the (admittedly controversial) DARPA/MITLL 1999 intrusion detection dataset. Both configurations produced exactly the same set of alerts.

For our online experiments, we made use of an Endace 9.2X2 Data Acquisition and Generation (DAG) card to run two Snort (version 2.9.1.2) instances in parallel, one with stock Snort and the other with winnowing enabled, on live traffic. The DAG uses a packet processor to capture and timestamp packets at line rate, and employs an on-board direct memory access (DMA) engine to zero copy packets into the host's memory. The DMA engine can also be programmed to duplicate traffic over multiple memory buffers (called "streams"), which enables us to simultaneously run multiple experiments, thereby comparing different approaches on the same traffic. We use a 10Gbps DAG connected to a 2.53 Ghz Intel Xeon 6 core host with 16 GB of memory. As in our earlier experiments, we only examined traffic on ports 22, 25, 80 and 443; this filtering was performed on the DAG and therefore CPU resources are used exclusively for payload inspection. Beyond the winnowing preprocessor, there were no differences between the two configurations: both used the same ruleset with the default Snort options (including inspection of gzipped HTTP entities and the bypassing of the detection algorithms by SSL application data packets in established streams), and each was allocated 2GB of memory on the DAG.

[4]Our preprocessor only drops packets with TCP payloads and therefore does not interfere with connection tracking at the transport layer.



(a) Packets



(b) Alerts

Figure 6. Number of alerts and number of packets processed by Snort with winnowing and Snort without (in 15-minute windows).

Our primary experiment lasted for 24 hours and encompassed more than 7.6 terabytes of traffic; the load reached 1.2Gbps at peak. Figure 6(a) shows the number of packets which each instance of Snort was able to process in 15-minute intervals. Even at peak load, our winnowing Snort instance is able to handle the full volume with zero percent packet loss. In contrast, the stock Snort instance dropped nearly 60% of the 98.9 billion packets observed during the 24-hour window. In fact, the winnowing Snort instance processed 147% more packets, resulting in over 33,000 *additional alerts* (see Figure 6(b)). We note that as the drops suffered by the unmodified Snort are lost when Snort is unable to copy packets out of the network buffer fast enough to avoid old packets being overwritten by the kernel (or the DAG, in this case), and therefore Snort has no control over which packets are dropped [31].

In a second experiment, we partitioned one of the two identical traffic streams at the DAG into four disjoint streams, each of which was passed to a stock Snort instance running on a dedicated core. However, due to traffic bursts, the stock Snort instances still dropped significant numbers of packets despite our attempts to optimize its handling of the traffic. While this underscores the need for techniques like ours, we remind the reader that the advantages of winnowing traffic are not limited to situations of overload: the significant throughput increase allows systems to evaluate more rules than stock configurations under the same traffic conditions.

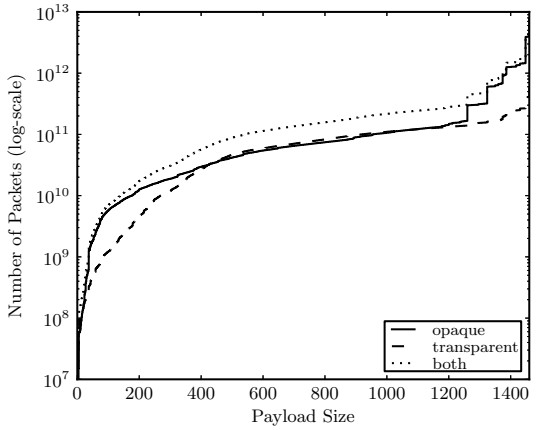


Figure 7. CDF of payload size for both opaque and transparent traffic.

We stress that, apart from the winnowing preprocessor, the Snort instances in each experiment were *identically configured* and saw *exactly the same traffic*. The substantial improvement in capacity arises due to the surprisingly high proportion of opaque traffic, which, as discussed in §II, incurs high CPU overheads (confirmed in our experiments by an observed 30% increase in average per-packet processing time compared to transparent packets). In total, an astonishing 89% of the payload-carrying TCP packets observed in our primary experiment were classified as opaque, representing 86% of the bytes transmitted. We hypothesize that this is due to the prevalence of streaming video services such as Hulu, Netflix, and YouTube operating on port 80. Differences in the packet size distributions (see Figure 7) indicate that MTU-sized packets are far more often opaque than not, which suggests that large, compressed streams (e.g., the aforementioned streaming video), are prevalent. These streams may also account for the bursty nature of traffic observed in the multiple core experiment.

Operational Impact: Since winnowing opaque packets fundamentally changes the mix of traffic which reaches the rules-matching engine in an IDS, its influence could lead to deeper insights about network activities. To evaluate this further, we compared the alerts generated by both instances of Snort. In total, the stock Snort instance generated 25,081 alerts, while the Snort instance augmented with winnowing produced 58,297 alerts—a 118% increase. Of these, the three most prevalent in both cases remained the same; two

overflow attack detections and a file-type identification alert (for portable executable binaries). We found the differences between the distributions of rules triggered by the stock Snort and by Snort with winnowing to be particularly interesting. One PowerPoint administrator privilege escalation attack rule was triggered 2000% more times by the winnowing Snort instance, an almost 10-fold increase over what one might expect just from the increased traffic volume. We also found a 760% increase in alerts for PDFs with embedded Javascript. While ascertaining whether winnowing induces any false positives is impossible due to the losses sustained by the stock Snort instance, we argue that intelligently dropping packets, as we do, is no more likely to induce false positives than dropping random packets due to overload. The only class of rules which produced fewer alerts when winnowing opaque traffic were for HTTP connections to blacklisted domains; these are the only clear false negatives. Since HTTP headers are in plaintext, and hence not dropped by Winnow, we suspect that the missed alerts are due to Snort failing to recover from missing (opaque) packets in a stream. Since parsing of HTTP headers is possible even in the presence of missing payload packets, we believe such alerts would be triggered if the IDS was better equipped to recover from such midstream losses. Alternatively, a more mature implementation of the winnowing preprocessor could inform Snort of the dropped packets.

Nevertheless, the instantiation of our prototype on a major campus network has already provided valuable information to its network operators, clearly showing that their network security exposure differs from what they originally thought.

IV. Limitations

While our focus on minimizing payload inspection suggests a trivial method for an attacker to bypass DPI engines using our technique (by padding the beginning of the packet with ASCII bytes), in current systems an attacker need do nothing more than "encrypt" their exploit code (e.g., a simple XOR may suffice) to bypass content-based signature matching engines. Furthermore, our techniques comprise a method for flagging the likely presence of such encrypted objects based on nothing more than the ciphertext itself.

At first blush, it may appear that binary protocols present an insurmountable problem for our methodology. However, we point out that the majority of traffic (i.e., 80% on our network) is HTTP, a text protocol. Furthermore, the high-volume binary protocols (SSL, RTMP, and Bittorrent) transport primarily, if not exclusively, opaque data. We believe that a different alternative hypothesis, such as that mentioned earlier in the context of identifying file types, could provide improved accuracy for binary protocols.

Additionally, while it may seem that the flow level, as opposed to the packet-level, is the natural vantage point for identifying opaque traffic, our work concentrates on packet-level analysis. We argue that the presence of tunneled traffic, and container formats such as PDF, mandates identification of opaque traffic on a per-packet basis. In the specific case of encrypted connections, many protocols (e.g., SSH and SSL) begin with unencrypted, connection set-up packets; some protocols (e.g., STARTTLS) are even designed specifically to enable upgrading a connection from unencrypted to encrypted mid-stream. Furthermore, packet-level techniques can be used in situations where flow state is not kept, such as on DAG cards. Finally, by performing packet-level analysis, we can winnow opaque packets before they reach the stream reassembly engine, significantly reducing overhead. That said, there are benefits to incorporating flow-level analysis. One interesting direction might be to limit packet-level misclassifications by utilizing information from prior and subsequent packets.

We remind the reader that our techniques are not intended to be used in isolation, but rather as components in larger systems. Therefore, the limitations of our techniques can be mitigated by the strengths of the remainder of the system architecture, such as by coordinating flow-level analysis with packet-level opacity checking. This would provide a layered defense against, e.g., attacks embedded in lengthy streams to evade systems using *selective packet discarding* [31], which is discussed in the next section, or approaches similar to the afore-mentioned Time Machine (see Section I).

V. Related Work

The related problem of forensic file and data type identification has been extensively explored over the past decade. Many of these efforts have focused on analyses of *byte frequency distributions*, i.e., the frequency of each byte value in the data of interest. For the most part, these approaches work by creating signatures for known file types (i.e., HTML or PDF) based on byte frequency distributions, and then comparing unknown files to pre-existing signatures to determine the file type (see, e.g., Ahmed et al. [1]). Veenman [40] and Hall [17] examined entropy and compressibility measurements as indicators of file type, while others have explored techniques for directly modeling the structure of files [16, 18, 21, 32]. The closest work to the problem at hand is that of Conti et al. [8], who consider distinguishing between random, encrypted and compressed files using k-nearest-neighbor classification. Shannon entropy and the χ^2 statistic, among other measures, are evaluated as distance metrics over sliding windows of 1KB or more. Similar ideas were explored by Shamir and Someren [33] for identifying cryptographic keys on disk. However, as our empirical analyses showed, the forensics scenario is fundamentally distinct from the setting we explore in this work: for real-time analysis on the network, the amount of data available is constrained and computational resources are scarce.

Approaches based on byte-frequency [44, 46] and entropy [22, 35, 36, 45] have also been applied in the context of malware identification and analysis, both on disk and in

network traffic. These approaches are not well suited for our task, as they require large sample sizes for their statistical tests to be valid and/or impose high computational costs.

Most germane to our work is the scheme proposed by Olivain and Goubault-Larrecq, which uses hypothesis testing of the sample entropy statistic to determine whether packets are encrypted [27]. Similarly, Dorfinger proposed an approach for detecting encrypted traffic using entropy estimation, intended as a pre-filtering step for a Skype flow detection engine [3, 9–11, 39]. Their encryption detection scheme shares much in common with that of Olivain and Goubault-Larrecq, and is based on calculating the sample entropy for the packet payload and comparing that entropy value to the expected entropy value for a sequence of uniformly randomly distributed bytes of the same length. However, their entropy estimation approach does not scale well to situations where the number of samples is small [28–30]. For our entropy-based tests, we addressed this issue head on by calculating the exact probability distribution function (see §A)for the (byte-)entropy of n bytes, for small n. Malhotra compared a number of standard statistical tests, including the χ^2 and Kolmogorov-Smirnov tests, for identifying encrypted traffic [24]. As with Olivain and Goubault-Larrecq, their approaches required at least 1 KB of data per trial. In any case, our byte-value tests outperformed all of these approaches.

From a systems perspective, similar notions have been suggested in the context of improving the ability of NIDS to weather inevitable traffic spikes by the *selective discarding* [31] of packets when the system is under load. Similar to Time Machine, Papadogiannakis et al. propose discarding packets from lengthy flows, reasoning that these packets are less likely to trigger alerts. We believe our approach is more general, both in that we enable new policy decisions, as discussed earlier, and our techniques can operate on flows of any length. That said, our techniques are certainly ripe to be employed in a load-dependent fashion, but we focus on the more difficult load-independent setting in order to explore the limits of our techniques.

Hardware-based approaches for reducing the load on NIDS have also been proposed. That most closely related to our work is the notion of *shunting* [15], in which packets are matched in hardware against a dynamic list of rules which the IDS/IPS updates periodically. For each packet, the FPGA-based *shunt* decides, based on these lists, whether to drop, pass, or forward the packet to the IDS/IPS. Again, we see our techniques for the identification of opaque traffic as complementary to the shunting approach. Since the likelihood-ratio test can be, in the extreme, simplified to counting high-value bits, it can be easily implemented on an FPGA, allowing the NIDS to specify opacity-based rules for packet forwarding decisions.

Lastly, the application of sequential hypothesis testing to computer security problems is not new. For instance, Jung et al. [20] applied sequential hypothesis testing to portscan detection; Jaber and Barakat [19] proposed an iterative approach that used the size and direction of each packet to determine the most likely application generating the observed traffic; and Thatte et al. [37] proposed a distributed denial-of-service detector based on a bivariate SPRT.

VI. Summary

In this paper, we propose the notion of quick and accurate winnowing of opaque traffic as a mechanism for reducing the demands upon DPI engines, and introduce a number of statistical techniques for performing such winnowing. Our techniques are compared against those that might be considered common wisdom, and through extensive evaluation, we show that our statistical approaches perform best. Our results demonstrate that we are able to identify opaque data with 95% accuracy (Section §III-A). By implementing our approach with the Snort IDS, we demonstrate that winnowing vastly improves the rate at which an IDS can process packets without adversely impacting accuracy. Our experiments show that winnowing enables Snort to process 147% more packets and generate 135% more alerts over a 24-hour period featuring a peak traffic volume of 1.2Gbps.

It is our hope that the ability to classify traffic in the ways proposed in this paper will provide significant benefits by enabling second-line (e.g., DPI) analysis engines to target each class with specialized approaches, rather than attempting to apply heavy-weight analyses to a general mix of traffic. That said, our real-world evaluations offer a cautionary tale; our experiments indicate that the vast majority—89% of payload-carrying TCP packets on our network—of modern network traffic is opaque. This finding was certainly a surprising result to us, and we believe it may have far reaching consequences in its own right. At the least, it calls into question the long-term viability of DPI techniques, and warrants revived interest in finding new ways for traffic monitoring and policy enforcement. We hope that this paper stimulates discussions along those lines.

VII. Acknowledgments

We thank Kevin Snow, Teryl Taylor, Vinod Yegneswaran and the anonymous reviewers for their valuable suggestions on ways to improve this paper. We also express our gratitude to our campus networking gurus (Bill Hays, Murray Anderegg, and Jim Gogan) for their tremendous efforts in helping deploy the infrastructure for this study, to Alex Everett for providing access to university rulesets, and to Jake Czyz for his assistance with data collection.

The researchers and their respective Technology Service Offices have longstanding memorandums of understanding (MoU) in place to collect anonymized network traffic. The MoU covers specific uses and types of networking data, as well as conditions for securing and accessing such data. To be compliant with our pre-existing Institutional Review

Board (IRB) policies, all computations on payloads were performed in memory. For this specific collection effort, the Institutional Review Board (IRB) concluded that, as we collect only coarse-grained statistics regarding packet payloads, the activities in our application "do not meet the regulatory definition of human subjects research under the Common Rule" and are therefore not regulated.

This work was supported in part by the Department of Homeland Security (DHS) under contract number D08PC75388, the U.S. Army Research Office (ARO) under Cyber-TA Grant no. W911NF-06-1-0316, and the National Science Foundation (NSF) award no. 0831245. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DHS, NSF, or ARO.

REFERENCES

[1] I. Ahmed, K.-s. Lhee, H. Shin, and M. Hong. Fast file-type identification. In *Proceedings of the Symposium on Applied Computing*, pages 1601–1602, 2010.

[2] A. Bittau, M. Hamburg, M. Handley, D. Mazières, and D. Boneh. The case for ubiquitous transport-level encryption. In *USENIX Security Symposium*, 2010.

[3] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing Skype traffic: when randomness plays with you. *Comp. Commun. Review*, pages 37–48, 2007.

[4] J. J. Bussgang and M. B. Marcus. Truncated sequential hypothesis tests. *IEEE Transactions on Information Theory*, 13(3), July 1967.

[5] N. Cascarano, A. Este, F. Gringoli, F. Risso, and L. Salgarelli. An experimental evaluation of the computational cost of a DPI traffic classifier. In *Global Telecommunications Conference*, pages 1–8, 2009.

[6] K. Chiang and L. Lloyd. A case study of the Rustock rootkit and spam bot. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, 2007.

[7] W. J. Conover. A Kolmogorov goodness-of-fit test for discontinuous distributions. *Journal of the American Statistical Association*, 67:591–596, 1972.

[8] G. Conti, S. Bratus, A. Shubina, B. Sangster, R. Ragsdale, M. Supan, A. Lichtenberg, and R. Perez-Alemany. Automated mapping of large binary objects using primitive fragment type classification. *Digital Investigation*, 7(Supplement 1):S3 – S12, 2010.

[9] P. Dorfinger. Real-time detection of encrypted traffic based on entropy estimation. Master's thesis, Salzburg University of Applied Sciences, 2010.

[10] P. Dorfinger, G. Panholzer, B. Trammell, and T. Pepe. Entropy-based traffic filtering to support real-time Skype detection. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, pages 747–751, 2010.

[11] P. Dorfinger, G. Panholzer, and W. John. Entropy estimation for real-time encrypted traffic identification. In *Traffic Monitoring and Analysis*, volume 6613 of *Lecture Notes in Computer Science*. 2011.

[12] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational experiences with high-volume network intrusion detection. In *Proceedings of the 11th ACM conference on Computer and Communications Security*, 2004.

[13] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *Proceedings of the 15th USENIX Security Symposium*, 2006.

[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, 1999.

[15] J. M. Gonzalez, V. Paxson, and N. Weaver. Shunting: a hardware/software architecture for flexible, high-performance network intrusion prevention. In *Proceedings of the 14th ACM conference on Computer and Communications Security*, 2007.

[16] J. Haggerty and M. Taylor. Forsigs: Forensic signature analysis of the hard drive for multimedia file fingerprints. In *Proceedings of IFIP International Information Security Conference*, 2006.

[17] G. A. Hall. Sliding window measurement for file type identification. ManTech Security and Mission Assurance, 2006.

[18] R. M. Harris. Using artificial neural networks for forensic file type identification. Master's thesis, Purdue University, 2007.

[19] M. Jaber and C. Barakat. Enhancing application identification by means of sequential testing. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, 2009.

[20] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy*, pages 211 – 225, may 2004.

[21] B. Li, Q. Wang, and J. Luo. Forensic analysis of document fragment based on SVM. In *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Dec. 2006.

[22] R. Lyda and J. Hamrock. Using entropy analysis to find encrypted and packed malware. *IEEE Security and Privacy*, 5:40–45, Mar. 2007.

[23] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider. Enriching network security analysis with time travel. In *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 183–194, 2008.

[24] P. Malhotra. Detection of encrypted streams for egress monitoring. Master's thesis, Iowa State University,

2007.

[25] U. Maurer. A universal statistical test for random bit generators. *Journal of Cryptology*, 5:89–105, 1992.

[26] C. Muthukrishnan, V. Paxson, M. Allman, and A. Akella. Using strongly typed networking to architect for tussle. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[27] J. Olivain and J. Goubault-Larrecq. Detecting subverted cryptographic protocols by entropy checking. Research Report LSV-06-13, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2006.

[28] L. Paninski. Estimation of entropy and mutual information. *Neural Computation*, 15(6):1191–1253, 2003.

[29] L. Paninski. Estimating entropy on m bins given fewer than m samples. *IEEE Transactions on Information Theory*, 50(9):2200–2203, 2004.

[30] L. Paninski and M. Yajima. Undersmoothed kernel entropy estimators. *IEEE Transactions on Information Theory*, 54(9):4384–4388, 2008.

[31] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos. Improving the accuracy of network intrusion detection systems under load using selective packet discarding. In *Proceedings of the Third European Workshop on System Security*, EUROSEC, 2010.

[32] V. Roussev and S. L. Garfinkel. File fragment classification - the case for specialized approaches. *IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 3–14, 2009.

[33] A. Shamir and N. v. Someren. Playing "hide and seek" with stored keys. In *Proceedings of the Third International Conference on Financial Cryptography*, pages 118–124, 1999.

[34] R. Sommer. *Viable Network Intrusion Detection in High-Performance Environments*. Doktorarbeit, Technische Universität München, Munich, Germany, 2005.

[35] S. J. Stolfo, K. Wang, and W. jen Li. Fileprint analysis for malware detection. Technical report, Columbia University, 2005.

[36] S. M. Tabish, M. Z. Shafiq, and M. Farooq. Malware detection using statistical analysis of byte-level file content. In *KDD Workshop on CyberSecurity and Intelligence Informatics*, pages 23–31, 2009.

[37] G. Thatte, U. Mitra, and J. Heidemann. Parametric Methods for Anomaly Detection in Aggregate Traffic. *ACM Transactions on Networking*, 2011.

[38] K. Thomas and D. Nicol. The koobface botnet and the rise of social malware. In *Malicious and Unwanted Software (MALWARE)*, pages 63 –70, oct. 2010.

[39] B. Trammell, E. Boschi, G. Procissi, C. Callegari, P. Dorfinger, and D. Schatzmann. Identifying Skype traffic in a large-scale flow data repository. In *Traffic Monitoring and Analysis*, volume 6613 of *Lecture Notes in Computer Science*, pages 72–85. 2011.

[40] C. J. Veenman. Statistical disk cluster classification for file carving. In *Proceedings of the Third International Symposium on Information Assurance and Security*, pages 393–398, 2007.

[41] D. D. Wackerly, W. M. III, and R. L. Scheaffer. *Mathematical Statistics with Applications*. Duxbury Press, Pacific Grove, CA, sixth edition, 2002.

[42] A. Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, June 1945.

[43] A. Wald and J. Wolfowitz. Optimum character of the sequential probability ratio test. *Annals of Mathematical Statistics*, 19(3):326–339, 1948.

[44] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, pages 203–222, 2004.

[45] M. Weber, M. Schmid, D. Geyer, and M. Schatz. Peat - a toolkit for detecting and analyzing malicious software. In *Proceedings of the 18th Annual Computer Security Applications Conference*, pages 423–431, 2002.

[46] L. Zhang and G. B. White. An approach to detect executable content for anomaly based network intrusion detection. In *IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, 2007.

APPENDIX

This appendix presents various methods for opaque traffic identification, an extensive parameter space exploration experiment to compare these methods and provide optimal values for their parameters, the asymptotic efficiency of each method, and a description of the HTTP content-type labeling rules we used to determine ground truth in our experiments.

A. Comparison of Methods

1) Preliminaries: For all tests, let $\mathcal{H}_0$ be the hypothesis that the v_i are approximately uniformly distributed (e.g., the packet is compressed or encrypted), and let $\mathcal{H}_1$ be the alternative hypothesis, i.e., that the v_i are distributed according to some distribution that is not uniform (e.g., corresponding to a transparent packet).

In what follows, we examine approaches that can be broadly classified under two different models, which we refer to as operating on the *entropy* or *byte-value* domains. For the entropy domain, the basic unit is the (byte-) entropy of a *block* of bytes, where the number of bytes n in a block is parameterized. In other words, if X is a random variable in the entropy domain, then the support of X is the set of all possible values for the byte-entropy of n bytes. If X is a random variable in the byte-value domain, then the support of X is the set of integers 0 through 255. That is, for the byte-value domain, the basic unit is simply the byte.

Before delving into details of the various tests we explore, we first present necessary notation, summarized in Table III. To represent the packet payload, let $\mathbf{v} = \{v_1, v_2, ..., v_N\}$ be

Symbol	Meaning
n	size (in bytes) of a *block*
k	size of domain (e.g., 256 for bytes)
N	(maximum) number of payload bytes
M	(maximum) number of samples
v_i	observation (byte)
w_i	observation (block)
$\mathbf{v}$	sequence of observations (bytes)
$\mathbf{w}$	sequence of observations (blocks)
α	expected false negative rate
β	expected false positive rate
δ	alternative hypothesis weight
T	offset (in bytes)

Table III
NOTATION

Param.	Domain	
	Entropy	Bytevalue$*$
α	$0.001, 0.005, 0.01, 0.05$	(*no addl.*)
$\beta\dagger$	$0.001, 0.005, 0.01, 0.05$	(*no addl.*)
$\delta\ddagger$	$0.65, 0.75, 0.85$	(*no addl.*)
N	8, 16, 24, 32, 48, 64, 80, 128	4, 12, 20, 28, 32, 40
T	0, 8, 16, 24	4, 12, 20
$\dagger$ Applicable only for sequential tests.		
$\ddagger$ Not applicable for χ^2 or discrete K-S tests.		
$*$ In addition to those for the entropy tests.		

Table IV
PARAMETER SPACE EXPLORED BY DOMAIN

a sequence of observations (e.g., payload bytes), such that $v_i \in \{0, 1, ..., k-1\}\ \forall\ i \in \{1, ..., N\}$. That is, for a sequence $\mathbf{v}$ of bytes, $k = 256$. In the entropy domain, we operate on a sequence $\mathbf{w}$ of *blocks* of bytes $\mathbf{w} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_{N/n}\}$.

Notice that while the density function for the byte-value domain is well-known and easy to compute, the entropy domain is more complicated. For many of the tests we examine, we need to evaluate the *probability mass function (PMF)* over possible values for the sample entropy of n bytes for each hypothesis. Simple threshold checking of the sample entropy fails to leverage the distribution of sample entropy values, missing important information about the relative likelihood of the values observed.

Due to space limitations, we omit the details of the derivation of this PMF: the essential idea is that the sample entropy of n bytes is the same regardless of the arrangement of those bytes. We can then enumerate the possible ways of arranging n bytes, for small n, to calculate the corresponding PMF. We implemented such an enumeration using NVIDIA's CUDA GPU parallel programming toolkit, yielding a PMF for $n = 8$.

Discrete Kolmogorov-Smirnov Test: Another appropriate test for determining whether a sample is drawn from a particular distribution is the Kolmogorov-Smirnov (K-S) test. The K-S test is often used because of its generality and lack of restricting assumptions. The test is based on quantifying the difference between the cumulative distribution functions of the distributions of interest. However, we note that the traditional Kolmogorov-Smirnov test is only valid for continuous distributions, and Conover has already shown that the p-values for the continuous K-S test applied to discrete or discontinuous distributions can differ substantially from those of the discrete analog. Therefore, we apply the discrete version as described by Conover [7].

Lastly, we note that Pearson's χ^2 test also seems appropriate in this setting. However, Pearson's χ^2 test assumes that samples are independent, the sample size is large, and the expected cell counts are non-zero — but, the latter two assumptions do not necessarily hold in our setting as the goal is to examine as few bytes as possible. Nevertheless, we include such analyses for the sake of comparison to prior work [24], but omit the derivation since the test is in common usage and our application is no different.

2) Parameter Space Exploration: We now present the results of a parameter space exploration experiment examining all of the hypothesis tests with varying parameter values. We present *ROC (receiver operating characteristic)* plots that simultaneously examine the true positive rate and false positive rate as a parameter (e.g., a threshold) varies. A set of ROC plots, one for each classifier, can then be used to compare classifiers across a range of parameter values, allowing one to judge the relative performance of classifiers.

There are several knobs we can turn. For the sequential tests we explore the desired maximum false positive rate α, the desired maximum false negative rate β, and the maximum number of payload bytes N. The parameter values we examined for fixed tests included the desired significance level α and number of payload bytes N. In all experiments, the maximum number of samples for the sequential tests equals the sample size for the fixed tests.

We consider different alternative hypotheses by changing the relative weight, δ, of the lower 128 byte values versus the higher 128 byte values. The δ parameter takes values in $(0.5, 1.0]$, and indicates the expected percentage of byte values less than 128. By changing the value, we alter our model of transparent traffic. In addition to measuring uniformity, this class of alternative hypotheses has the advantages of being extremely efficient to implement (as determining whether a byte value is less than 128 is a simple bit mask operation) and of accounting for the prevalence of ASCII characters in network traffic.

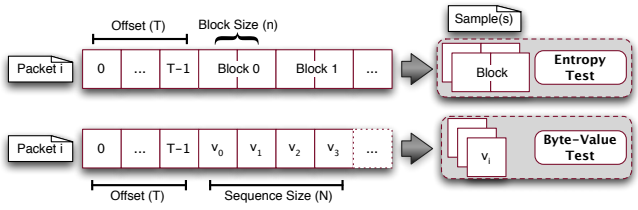


Figure 8. Illustration of length (N), offset (T), and block size (n)

Finally, we explore starting our analysis at different points within the packet, represented by the offset value T (in

bytes). As shown pictorially in Figure 8, T indicates how many bytes into the payload, i.e., past the end of the TCP header, our analysis begins. The specific values explored for each parameter are given in Table IV. Our dataset for the exploration experiment consists of the first 100,000 packets from `trace1`, and we consider only *encrypted* data as positive examples.

Figure 9 shows a single point for each unique set of parameter values (over 10,000) in our experiments. Superior classifiers should evidence a high true positive rate and a low false positive rate, represented on a ROC plot by a predominance of points in the upper-left corner of the plot. Since the byte-value tests evidence the points closest to the upper-left corner, the plots indicate that the sequential tests and the likelihood-ratio test in the byte-value domain are able to more accurately classify packets than the other tests. We can also compare our techniques based on their theoretical computational efficiency (see §A3); again, the byte-value sequential tests are the clear winners.

We also determine which specific values for the various parameters provide the optimal performance. In order to do so, we make use of the so-called F-score, which is a weighted harmonic mean of the *precision* and *recall* metrics.[5] Precision is defined as the ratio of the number of packets correctly classified as opaque, i.e., *true positives*, to the total number of packets classified as opaque. Recall is defined as the ratio of the number of true positives to the number of packets which are opaque according to our ground truth. Said another way, precision can be seen as measuring how often packets labeled as opaque are actually opaque (but says nothing about how often opaque packets are correctly identified), while recall indicates how often the method correctly identifies opaque packets (but says nothing about how often transparent packets are mislabeled).

We now examine each parameter in isolation by varying the parameter of interest while fixing the other parameters at default values ($\delta = 0.85$, $\alpha = \beta = 0.005$, $N = 32$, and $T = 8$). As might be expected, the number of bytes examined has a substantial effect on the performance of the detectors (Figure 10(a)); this effect drops off over time, suggesting that less than 32 bytes are needed in the general case to make a decision and less than 16 bytes are needed by the truncated test in the byte-value domain. The byte-value tests are the clear winners; all three sequential methods and the likelihood-ratio method performed equally well, each attaining precision over 90%. The entropy tests performed unexpectedly poorly, in few cases obtaining scores close to those of the other tests. In addition, we found that the restricted entropy test did not behave as we expected with regard to increasing the number of samples involved; an investigation revealed no underlying patterns in labels of the misclassified examples. We suspect that the particular decision boundaries (§II) used are simply poorly suited for the tests between entropy distributions.

With regard to changes in offset (Figure 10(b)), we see no clear improvement in classification for offset values larger than 8 bytes. Changes to the desired false positive and negative rates similarly had limit impact on the performance of the detectors, and so we omit those figures for brevity. Performance when varying the δ parameter, which controls the alternative hypothesis, peaks at $\delta = 0.85$.

Changes to the desired false positive rate (Figure 11(a)), similarly, have little impact on the performance of the detectors (the case for the false negative rate β is the same; we omit the figure for brevity). In either case, the parameters have little effect on the entropy tests (due to the small number of samples for the entropy tests, e.g., 4 samples at $N = 32$ bytes); we therefore omit the entropy figures due to space constraints.

We also examine changes to the δ parameter, which controls the alternative hypothesis. Notice that in Figure 11(b), the sequential tests appear to fail most often when the byte value distribution under $\mathcal{H}_1$ is assumed to be close to uniform; this is due to a large number of trials failing to make a decision, or forcibly making the wrong decision, when faced with too few samples to effectively choose between two similar distributions.

Summary of Findings: Our analysis shows that we can narrow the field to the likelihood-ratio and sequential tests in the byte-value domain based on accuracy rates alone. Perhaps surprisingly, the entropy-based methods do not perform as accurately as those in the byte-value domain; we believe that this indicates that accurate entropy tests require more samples than are available in our context. In addition, examining more than 16 bytes provided little benefit in terms of accuracy. Interestingly, the offset parameter had little effect in our tests. In summary, we found the truncated sequential test to one of the most accuracy and efficient tests, and therefore made use of this technique for the experiments in the body of the text.

3) Theoretical Efficiency: We can also compare our techniques based on their theoretical computational efficiency (Table V), in terms of the number of online floating-point operations required for each case. In the following, any operations which can be precomputed are omitted. In the byte-value domain, the sequential tests each require only a single multiplication and comparison per iteration, so the number of standard floating-point operations is no more than $2M$ for M samples (in the byte-value domain, samples are bytes, so $M = N$); the likelihood ratio test is the same, but with equality in the relation. Working in the entropy domain introduces the overhead both of binning the samples ($M = N/n$ operations) and of computing the sample entropy ($3k$ standard operations plus k logarithms). Pearson's χ^2 test requires M operations to bin the samples

[5]More precisely, we use the F_1-score, where recall and precision are evenly weighted.

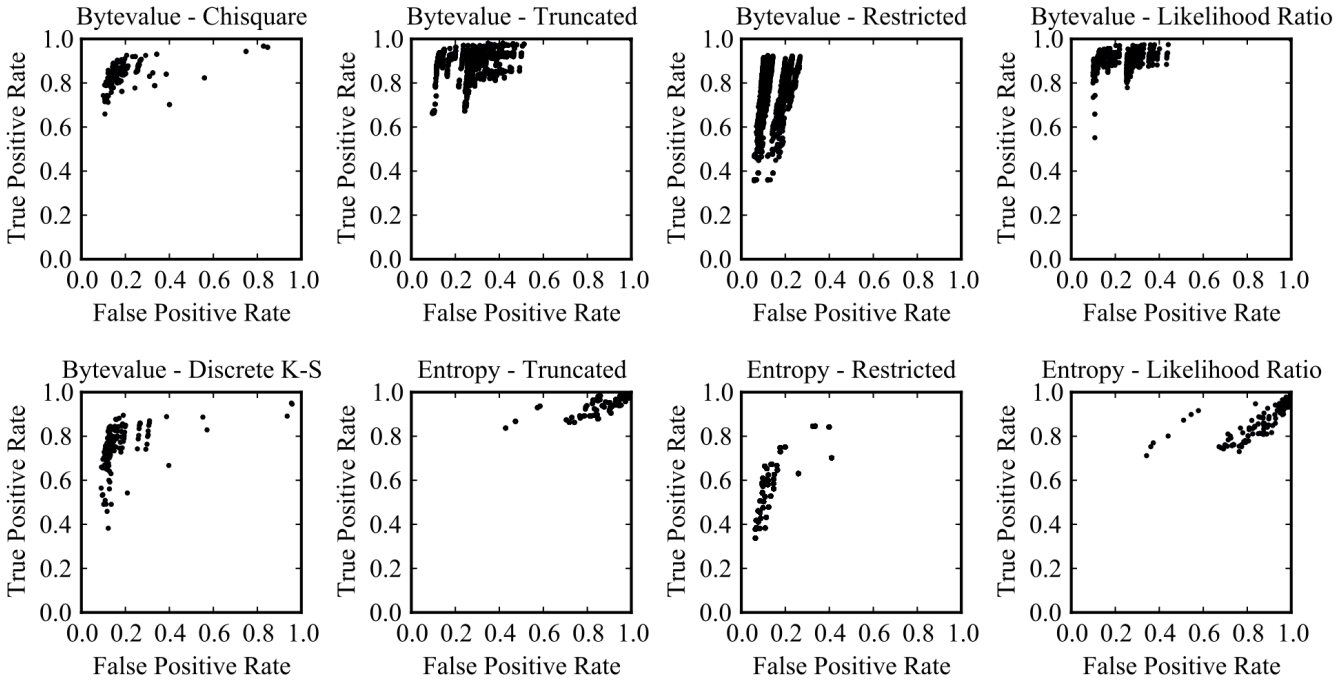


Figure 9. ROC plots for the techniques examined in this work.

and $3k$ operations, where k is the size of the domain (e.g., $k = 256$ for the byte-value domain), to sum the squared difference between the observed count and expected count over all bins. Finally, the discrete K-S test is $O(M^2)$. Obviously, from a performance standpoint, neither the K-S test nor the entropy tests are a good choice.

Test Type	Floating-Point Operations
Sequential	$\leq 2M$
Likelihood Ratio	$2M$
Pearson's χ^2	$M + 3k$
Discrete K-S	$O(M^2)$

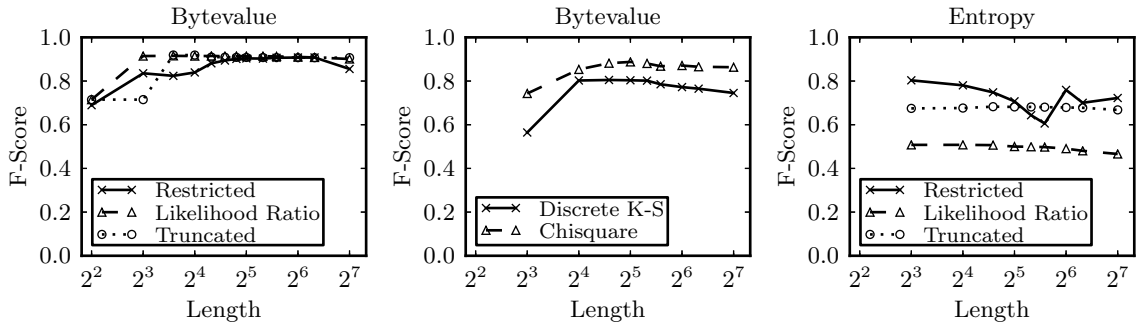
Table V
THEORETICAL EFFICIENCY

B. HTTP Labeling Rules

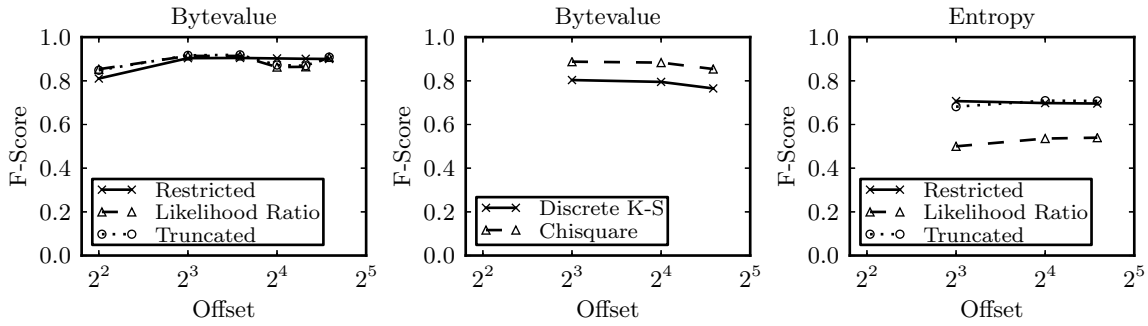
Base Type	Sub-type	Action/Label
image, video, audio	*	opaque
text	*	transparent
application	pdf, xml, flash	transparent
application	gzip, zip	opaque
application	*	omit
*	*	omit

Table VI
HTTP CONTENT-TYPE FILTERING RULES

To determine ground truth for HTTP packets, we first examine the content-encoding: if the strings 'gzip' or 'deflate' appear in the content-encoding, the packet is labeled opaque; otherwise, the label is determined by content type. HTTP content-type fields contain a base type, such as 'text' or 'video', a sub-type, such as 'html' or 'mp4', and optional parameters, such as 'charset=us-ascii'. For our filtering, if the base type is 'image', 'video', or 'audio', the traffic is labeled opaque; if the base type is 'text', the traffic is labeled transparent. For the 'application' base type, we also consider the sub-type: 'zip' and 'x-gzip' are considered opaque, while 'xml', 'pdf', and 'x-shockwave-flash' are considered transparent. While some of these sub-types (e.g., PDF) can be containers for many other formats, we choose to be conservative and simply classify them as transparent since we have no clear cut way of drawing the line between semi-structured and more opaque formats. All other sub-types are dropped, since hand-labeling of all potential content-types is infeasible and error-prone. Our HTTP filtering and labeling rules are summarized in Table VI (given in order of application).

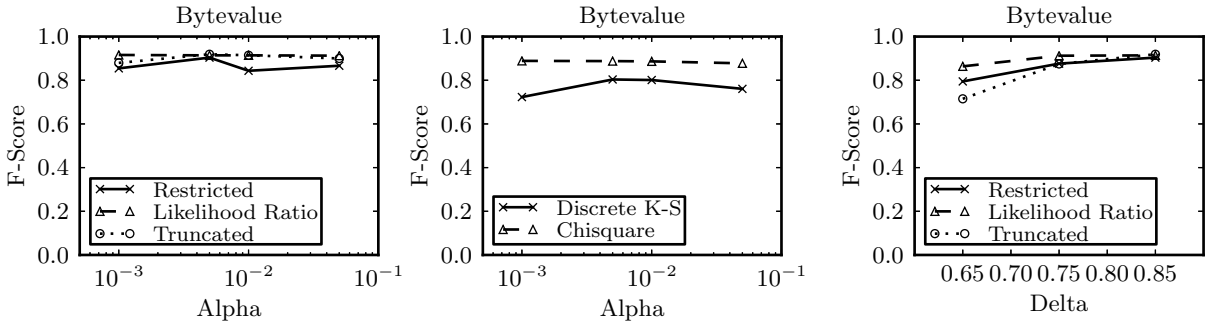


(a) F_1-Score for sample size



(b) F_1-Score for offset

Figure 10. Effects on accuracy when varying sample size and byte offset



(a) F_1-Score for significance level

(b) F_1-Score for alternative hypothesis

Figure 11. Effects on accuracy when varying significance level and alternative hypothesis