Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices

Hyunchul Kim
CAIDA and Seoul National University
hkim@mmlab.snu.ac.kr

kc claffy
CAIDA, UC San Diego
kc@caida.org

Marina Fomenkov
CAIDA, UC San Diego
marina@caida.org

Dhiman Barman
UC Riverside
dhiman@cs.ucr.edu

Michalis Faloutsos
UC Riverside
michalis@cs.ucr.edu

KiYoung Lee
UC San Diego
kiylee@bioeng.ucsd.edu

ABSTRACT

Recent research on Internet traffic classification algorithms has yield a flurry of proposed approaches for distinguishing types of traffic, but no systematic comparison of the various algorithms. This fragmented approach to traffic classification research leaves the operational community with no basis for consensus on what approach to use when, and how to interpret results. In this work we critically revisit traffic classification by conducting a thorough evaluation of three classification approaches, based on transport layer ports, host behavior, and flow features. A strength of our work is the broad range of data against which we test the three classification approaches: seven traces with payload collected in Japan, Korea, and the US. The diverse geographic locations, link characteristics and application traffic mix in these data allowed us to evaluate the approaches under a wide variety of conditions. We analyze the advantages and limitations of each approach, evaluate methods to overcome the limitations, and extract insights and recommendations for both the study and practical application of traffic classification. We make our software, classifiers, and data available for researchers interested in validating or extending this work.

1. INTRODUCTION

Political, economic, and legal struggles over appropriate use and pricing of the Internet have brought the issue of traffic classification to mainstream media. Three of the most important and acrimonious tussles are: (a) the file sharing tussle, between the file sharing community and intellectual property representatives RIAA (Recording Industry Association of America) and MPAA (Motion Picture Association of America); (b) the battle between malicious hackers, e.g. worm creators, and security management companies; and (c) the network neutrality debate, between ISPs and content/service providers. In all cases the algorithmic playing field is traffic classification: stopping or deprioritizing traffic of a certain type, versus obfuscating a traffic profile to avoid being thus classified. Traffic classification is also relevant to the more mundane but no less important task of optimizing current network operations and planning improvements in future network architectures, which means the increasing incentives to prevent accurate classification of one's own traffic presents an obstacle to understanding, designing, operating, financing, and regulating the Internet.

In the early Internet, traffic classification relied on the use of transport layer port numbers, typically registered with IANA to represent a well-known application. More recently, increasingly popular applications such as those that support peer-to-peer (P2P) file sharing, hide their identity by assigning ports dynamically and/or using well-known ports of other applications, rendering port-based classification less reliable [24, 30, 37]. A more reliable approach adopted by commercial tools [2, 3] inspects packet payloads for specific string patterns of known applications [12, 22, 25, 30, 37]. While this approach is more accurate, it is resource-intensive, expensive, scales poorly to high bandwidths, does not work on encrypted traffic, and causes tremendous privacy and legal concerns. Two proposed traffic classification approaches that avoid payload inspection are: (1) host-behavior-based, which takes advantage of information regarding "social interaction" of hosts [23, 24, 25], and (2) flow features-based, which classifies based on flow duration, number and size of packets per flow, and inter-packet arrival time [29, 31, 36, 9, 16, 17, 7, 14, 42, 20, 39, 27].

Despite many proposed algorithms for traffic classification, there are still no definitive answers to pragmatic questions: *What is the best available traffic classification approach? Under what link characteristics and traffic conditions does it perform well, and why? What are the fundamental contributions and limitations of each approach?*

Rigorous comparison of algorithms remains a challenge for three reasons [19]. First, there is few publicly available trace data to use as a benchmark, so every approach is evaluated using different traces, typically locally collected, often without payload (ground truth). Second, different techniques track different features, tune different parameters and even define flows and applications differently. Third, authors usually do not make their tools or data available with their results, so reproducing results is essentially impossible.

To begin to calibrate the research community's efforts, we conducted a comprehensive evaluation of three traffic classification approaches: port-based, host-behavior-based, and flow-features-based. We tested each technique on a broad range of data sets: seven payload traces collected at two backbone and two edge links located in Japan, Korea, and the US. Diverse geographic locations, link characteristics, and application traffic mix in these data allowed us to test the approaches under a wide variety of conditions.

We highlight the main contributions from our study:

a. We evaluate the performance of CoralReef (ports-based), BLINC (host-behavior-based), and seven commonly used machine learning algorithms (flow-features-based). We analyze the advantages and limitations of each approach, evaluate methods to overcome the limitations, and extract insights and recommendations for both the study and practical application of traffic classification.

b. We found the Support Vector Machine (SVM) algorithm, which we explain in section 3.3.3, achieved the highest accuracy on every trace and application, with $>$98.0% accuracy on average when trained with more than only 5,000 flows (2.5% of the size of the testing sets).

c. We found a set of single-directional key flow features that were consistent within an application across our traces; ports, protocol, TCP header flags, and packet size. A limitation of previous attempts based on flow features [29, 31, 36, 9, 16, 7, 14, 42, 20, 39, 27] is that they used bidirectional TCP connection statistics, which do not work for UDP traffic, or for backbone links, which only see both directions of traffic under (atypical) symmetric routing conditions. We also found that ports remain one of the most important discriminators, particularly when used in combination with other flow features such as packet size information, TCP header flags and protocol.

d. We empirically found that the accuracy of host-behavior-based methods such as BLINC strongly depends on topological location. The best place to use BLINC is the border link of a single-homed edge network, so BLINC can capture full (bidirectional) behavioral information of the internal hosts. BLINC is not recommended for backbone links, where (i) only a small portion of behavioral information is collected for each host and (ii) we often miss one direction of traffic due to asymmetric routing.

e. To mitigate the limitation of BLINC on backbone traffic classification, we extended BLINC to identify some application traffic (Web, P2P, ...) even when both directions of flows are not observed. This process significantly improved the accuracy on backbone traces by as much as 45%.

f. We propose a robust traffic classifier which achieves $>$94.2% accuracy on every trace we examined, using the SVM algorithm, our suggested key flow features, and an unbiased (or less biased) training set extracted from multiple traces from Japan, Korea, and the US. This is the first study to show the feasibility of such a robust traffic classifier.

g. We make our developed code, dataset labeled with ground truth, and classifiers available for researchers interested in validating or extending the work.

The remainder of this paper is organized as follows. After reviewing related work in section 2, we describe our data and methodology in section 3. Section 4 presents the results of classification on the data from real networks. We propose and evaluate a robust classifier which works well on both available and unseen data in section 5. We discuss lessons learned in section 6. Section 7 concludes the paper.

2. RELATED WORK

Port-based approach. Although port-based traffic classification is the fastest and simple method, several studies have shown that it performs poorly, e.g., less than 70% accuracy in classifying flows [15, 30]. We acknowledge the coarseness of assessing performance over an entire trace rather than for the applications actually using well-known ports [19]. This performance metric essentially indicates the amount of traffic in the trace using well-known ports, which can vary widely, and does not classify traffic that is mis-using well-known ports assigned to a different application. We will evaluate the performance of each algorithm on a per-application basis [19], and explore why a given algorithm works well or poorly for certain applications. We used CoralReef [1] for the port-based approach.

Payload-based approach. Payload-based classification algorithms inspect the packet contents to identify the application. Once a set of unique payload signatures is available for an application, this approach produces extremely accurate classification. After early work showed the value of payload signatures in traffic classification [12, 30, 37], others have proposed automated ways to identify such signatures [22, 28], while they evaluated the automated schemes only on conventional applications such as FTP, SMTP, HTTP, HTTPS, SSH, DNS, and NTP, not on newer applications such as P2P, Games, and Streaming. We use the payload-based classifier developed in earlier efforts [25, 16, 41] to establish ground truth for our traces.

Host-behavior-based approach. The host-behavior-based approach was developed to capture social interaction observable even with encrypted payload [24, 25, 23]. For example, BLINC [25] captures the profile of a host, in terms of the destinations and ports it communicates with, identifies applications the host is engaged in by comparing the captured profile with (built-in to BLINC) host behavior signatures of application servers, and then classifies traffic flows. Recently Iliofotou, *et al.* proposed Traffic Dispersion Graphs that can potentially be used to classify applications using the network-wide interactions of hosts [23].

Table 1: Characteristics of analyzed traces

Set	Date	Day	Start	Duration	Link type	Src.IP	Dst.IP	Packets	Bytes	Avg. Util	Avg. Flows (/5 min.)	Payload
PAIX-I	2004-02-25	Wed	11:00	2h	backbone	410 K	7465 K	250 M	91 G	104 Mbps	1055 K	16 Bytes
PAIX-II	2004-04-21	Wed	19:59	2h 2m	backbone	2275 K	17748 K	1529 M	891 G	997 Mbps	4651 K	16 Bytes
WIDE	2006-03-03	Fri	22:45	55m	backbone	263 K	794 K	32 M	14 G	35 Mbps	312 K	40 Bytes
Keio-I	2006-08-06	Tue	19:43	30m	edge	73 K	310 K	27 M	16 G	75 Mbps	158 K	40 Bytes
Keio-II	2006-08-10	Thu	01:18	30m	edge	54 K	110 K	25 M	16 G	75 Mbps	92 K	40 Bytes
KAIST-I	2006-09-10	Sun	02:52	48h 12m	edge	148 K	227 K	711 M	506 G	24 Mbps	19 K	40 Bytes
KAIST-II	2006-09-14	Thu	16:37	21h 16m	edge	86 K	101 K	357 M	259 G	28 Mbps	21 K	40 Bytes

Flow features-based approach. Substantial attention has been invested in data mining techniques and machine learning algorithms using flow features for traffic classification [29, 31, 36, 9, 16, 17, 7, 14, 42, 20, 39, 27]. Nguyen *et al.* surveys, categorizes and qualitatively reviews these studies in terms of their choice of machine learning strategies and primary contributions to the traffic classification literature [33]. Their survey is complementary to our work, where we pursue quantitative, measurement-based, performance evaluation of the seven machine learning algorithms using multiple datsets collected from Japan, Korea, and the US.

Machine learning algorithms are generally categorized into *supervised learning* and *unsupervised learning* or clustering. Supervised learning requires training data to be labeled in advance and produces a model that fits the training data. The advantage of these algorithms is that they can be tuned to detect subtle differences and they clearly label the flows upon termination, unlike the unsupervised ones. Unsupervised learning essentially clusters flows with similar characteristics together [16, 26]. The advantage is that it does not require training, and new applications can be classified by examining known applications in the same cluster. Erman *et al.* [16] compared the performance of unsupervised machine learning algorithms in traffic classification. Since our main focus is on evaluating the predictive power of a built/trained traffic classifier rather than on detecting new applications or flow clustering, we first focus on supervised machine learning algorithms in this paper, leaving a performance comparison study of both supervised and unsupervised machine learning algorithms as future work.

3. COMPARISON METHODOLOGY

This section describes our comparison methodology, including performance metrics, dataset, comparison benchmark, and experimental setup for machine learning algorithms. We use the definition of a flow based on its 5-tuple (source IP address, destination IP address, protocol, source port, destination port) with a timeout of 64 seconds [13].

3.1 Performance metrics

To measure the performance of CoralReef, BLINC, and machine learning algorithms, we use four metrics: *overall accuracy, precision, recall*, and *F-Measure.* [1]

- *Overall accuracy* is the ratio of the sum of all True Positives to the sum of all the True Positives and False Positives for all classes.[2] We apply this metric to measure the accuracy of a classifier on the whole trace set. The latter three metrics are to evaluate the quality of classification results for each application class.
- *Precision* of an algorithm is the ratio of True Positives over the sum of True Positives and False Positives or the percentage of flows that are properly attributed to a given application by this algorithm.
- *Recall* is the ratio of True Positives over the sum of True Positives and False Negatives or the percentage of flows in an application class that are correctly identified.
- Finally, *F-Measure*, a widely-used metric in information retrieval and classification [40], considers both precision and recall in a single metric by taking their harmonic mean: $2 \times precision \times recall/(precision + recall)$. We use this metric to compare and rank the per-application performance of machine learning algorithms included in the WEKA.

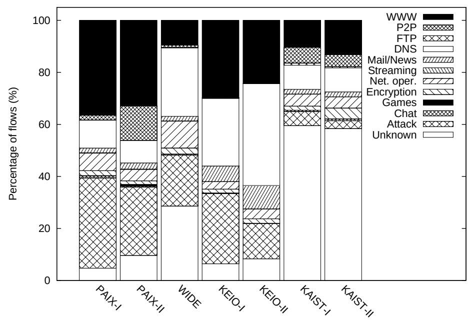
3.2 Data set and comparison benchmark

Our datasets consisted of seven anonymized payload traces collected at two backbone and two edge links located in the U.S., Japan, and Korea (Table 1). The PAIX backbone traces were taken on a bidirectional OC48 trunk of an US Commercial Tier 1 backbone link connecting San Jose and Seattle. The WIDE trace was captured at a 100 Mbps Ethernet US-Japan Trans-Pacific backbone link that carries commodity traffic for WIDE member organizations. The Keio traces were collected on a 1 Gb/s Ethernet link in Keio University Shonan-Fujisawa campus. The KAIST traces were captured at one of four external links connecting a 1 Gb/s KAIST campus network and a national research network in Korea.

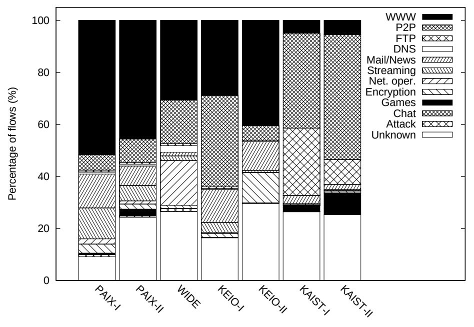
To establish a reference point in evaluating the algorithms, we used the payload-based classifier developed in [25], which we augmented with more payload signatures from [37, 16, 41] and manual payload inspection. Our resulting classifier includes payload signatures of various popular applications, summarized in Table 2. The payload classification procedure examines the payload contents of each packet against our array of signature strings, and in case of a match, classifies the corresponding flow with an application-specific tag. Previously classified flows are not re-examined again unless they have been classified as HTTP, in which case re-

[1] Given space limitations, we do not study computational time in depth, which we leave for future work, but discuss it briefly wherever possible. Computational complexity and its trade-off with resources (e.g., memory) is important in real-time deployments and hardware implementations.

[2] True Positives is the number of correctly classifi ed fbws, False Positives is the number of fbws falsely ascribed to a given application, and False Negatives is the number of fbws from a given application that are falsely labeled as another application.



(a)Percentage of flows



(b)Percentage of bytes

Figure 1: Application breakdown. Note that some of the filler-patterns are repeated.

examination may allow identification of non-web traffic relayed over HTTP (e.g., Streaming, P2P, etc.) [25].

After the payload-based classification process, we identify scanning activities using scan detection heuristics in [5]. Flows that could not be classified during the signature matching and scanning detection processes are categorized as unknown, which represents 4.7%-9.6% of flows in the PAIX and Keio traces, 28.6% in the WIDE trace, and around 60% in the two KAIST traces. Approximately 90% of those unknown flows in the KAIST traces were from/to three PlanetLab [21] machines. Our experience and Karagiannis *et al.*'s study [24] with payload classification suggest that the first 16 bytes of payload suffice for signature-based classification for most legacy and P2P applications except Gnutella[3] particularly on the PAIX and Keio traces where unknown flows represent less than 5%-10%. We exclude attack, unknown, and SSH/SSL encrypted (which represents 1.9%-4.5% of flows across our traces) flows from our analysis.

Figure 1 shows payload classification results for our traces. The traces vary widely in application mix, motivating our per-application analysis. Scanning (Attack) traffic contributes 14%-35% of flows in the WIDE, Keio, and PAIX traces.

Table 2: Application categories.

Category	Application/protocol
web	http, https
p2p	FastTrack, eDonkey, BitTorrent, Ares
	Gnutella, WinMX, OpenNap, MP2P
	SoulSeek, Direct Connect, GoBoogy
	Soribada, PeerEnabler
ftp	ftp
dns	dns
mail/news	smtp, pop, imap, identd, nntp
streaming	mms(wmp), real, quicktime, shoutcast
	vbrick streaming, logitech Video IM
network operation	netbios, smb, snmp, ntp, spamassassin
	GoToMyPc
encryption	ssh, ssl
games	Quake, HalfLife, Age of Empires, Battle fi eld Vietnam
chat	AIM, IRC, MSN Messenger, Yahoo messenger
attack	address scans, port scans
unknown	-

[3]Gnutella (and its variants) uses variable length padding; Erman *et al.*'s measurements indicate that 400 payload bytes of each packet is required to identify 90% of the Gnutella fbws using payload signatures [20].

3.3 Machine learning experiments

In this paper, we address three main challenges of the flow features-based traffic classification using supervised machine learning algorithms: (i) finding a set of key flow features that capture fundamental characteristics of different types of applications [31, 7, 39], (ii) finding the most accurate algorithm(s) with acceptable computational cost [39], and (iii) obtaining representative datasets with ground truth for various applications, i.e., datasets that contain correct and complete instances of application flows, in terms of their fundamental flow features [20].

3.3.1 Flow features

We use unidirectional flow features of TCP and UDP traffic to build a classifier that handles both TCP and UDP as well as backbone and edge traffic. We use 37 unidirectional flow features most of which were inspired from the 248 bidirectional features used in [31, 7] and the 22 bidirectional features in [38, 39]. The 37 features are: protocol, source and destination ports, the number of packets, transferred bytes, the number of packets without Layer 4 (TCP/UDP) payload, start time, end time, duration, average packet throughput and byte throughput, max/min/average/standard deviation of packet sizes and inter-arrival times, number of TCP packets with FIN, SYN, RSTS, PUSH, ACK, URG (Urgent), CWE (Congestion Window Reduced), and ECE (Explicit Congestion Notification Echo) flags set (all zero for UDP packets), and the size of the first ten packets.

3.3.2 Feature Selection

Feature selection, as a preprocessing step to machine learning, is the process of choosing a subset of original features that will optimize for higher learning accuracy with lower computational complexity. The process removes irrelevant [34] and redundant [6] features, i.e., those that can be excluded from the feature set without loss of classification accuracy, thus improving algorithm performance.

We use the Correlation-based Filter (CFS), which is computationally practical and outperforms the other filter method (Consistency based Filter) in terms of classification accuracy and efficiency [38, 39]. The Correlation-based Filter examines the relevance [10] of each feature, i.e., those highly correlated to specific class but with minimal correlation to each other [39]. We use a Best First search to generate candidate sets of features from the feature space, since it provides higher classification accuracy (percent of correctly classified instances) than Greedy search [38, 39].

3.3.3 Supervised machine learning algorithms

We use the WEKA machine learning software suite [4], often used in traffic classification efforts [29, 18, 16, 32, 31, 39], to evaluate the seven most commonly used supervised machine learning algorithms. We reveal (i) which algorithm(s) performs best for traffic classification, (ii) the effects of training set size on the classification performance of learning algorithms, i.e., how many training instances each algorithm requires to achieve a certain level of, say, >90% or 95% overall accuracy and per-application performance (F-Measure), and (iii) whether the results are consistent or not across different traces. To this end, we conduct seven experiments for the comparison of the algorithms on each trace, varying the size of the sampled training set while using the same, fixed size testing set. To separate training and testing sets, 50% of each trace is chosen randomly to form a pool of training flows, and the remaining 50% is used for a pool of testing ones. As these sets contain more than millions or hundreds of thousands of flows, we randomly sample 100; 500; 1,000; 5,000; 10,000; 50,000; and 100,000 training flows from the former training pool, while we randomly sample 200,000 flows as a testing set from the latter testing pool.[4] We briefly describe algorithms we evaluted:

Naive Bayes [31, 39] is the simplest probabilistic classifier based on Bayes' theorem, which analyzes the relationship between each feature and the application class for each instance to derive a conditional probability for the relationships between the feature values and the class.

Naive Bayes Kernel Estimation [31, 39] is a generalization of Naive Bayes which models features using multiple Gaussian distributions, known to be more accurate than a single Gaussian distibution for traffic classification.

Bayesian Network [39, 38] is a directed acyclic graph model that represents a set of features (or classes) as its nodes, and their probabilistic relationship as edges. If the conditional independence assumption is not valid, Bayesian Network learning may outperform Naive Bayes.

C4.5 Decision Tree [38] constructs a model based on a tree structure, in which each internal node represents a test on features, each branch represents an outcome of the test, and each leaf node represents a class label. In order to use a decision tree for classification, a given tuple (whose class we want to predict) corresponding to flow features, walks through the decision tree from the root to a leaf. The label of the leaf node is the classification result.

***k*-Nearest Neighbors (*k*-NN)** [36] computes Euclidean distances from each test instance to the *k* nearest neighbors in the *n*-dimensional feature space. The classifier assigns the majority class label among the *k* nearest neighbors to the test tuple. We use *k* = 1, by which we obtained the highest overall accuracy among the experiments where we tested with *k* = 1, 3, 5, 7, 9, 11, 13, 15, 17, and 19.

[4]The smallest trace of ours contains approximately 420,000 fbws labeled with payload classifi cation results.

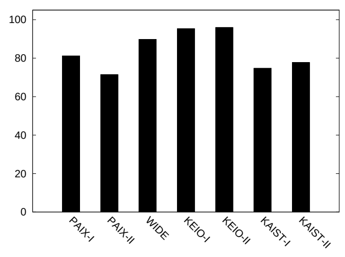


Figure 2: Overall accuracy of CoralReef

Neural Networks [7, 38] is a highly interconnected network of units, neurons, whose output is a combination of the multiple weighted inputs from others neurons. We use the most common and simple Neural Network classifier called the Multilayer Perceptron, which consists of a single input layer of neurons (features), a single output layer of neurons (classes), and one or more hidden layers between them. Following [38, 4], we set learning rate (weight change according to network error) to 0.3, the momentum (proportion of weight change from the last training step used in the next step) to 0.2 and we ran the training for 500 epochs (an epoch is the number of times training data is shown to the network).

Support Vector Machines (SVM) [38, 8, 27] The basic principle of SVM is to construct the optimal separating hyperplane, which maximizes the distance between the closest sample data points in the (reduced) convex hulls for each class, in an *n*-dimensional feature space [8]. Intuitively, we would expect that this boundary to generalize better than other possible boundaries between classes. We use the Sequential Minimal Optimization (SMO) [35], a faster algorithm for training SVM that uses pairwise classification to break a multi-class problem into a set of 2-dimensional subproblems, eliminating the need for numerical optimization. The two most important parameters in SVM are the complexity parameter *C* and the polynomial exponent *p* [27, 38]. Li *et al.* [27] showed that varying the complexity parameter *C* influenced the overall accuracy of their SVM traffic classifier by only a little (around 1% at most). We use 1 for both parameters as in [38, 4].

4. RESULTS

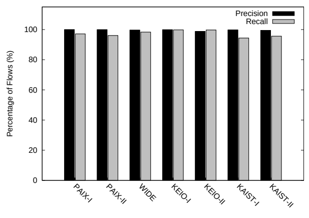
We evaluate the performance of nine algorithms for Internet traffic classification: CoralReef, BLINC, and the seven machine learning algorithms described.

4.1 CoralReef

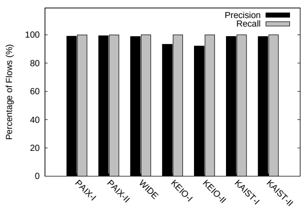
To evaluate port-based classification, we compare the performance of CoralReef's port classification rules [1] with our payload-based classifier, which we use as ground truth.

4.1.1 Overall accuracy

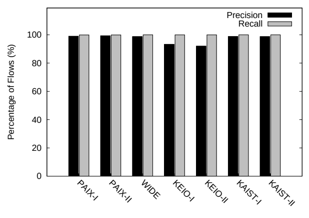
The overall accuracy of any port-based classification reflects how much traffic in the examined traces obeys the



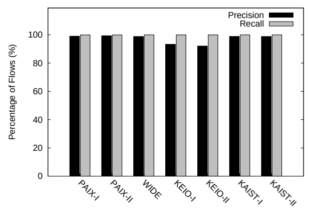
(a) WWW



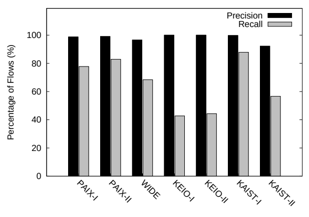
(b) DNS



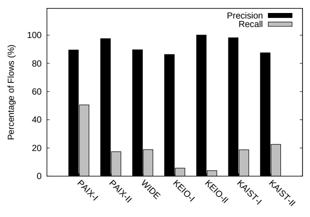
(c) Mail



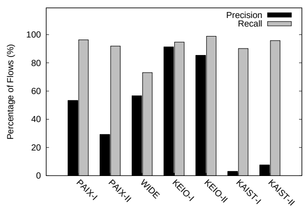
(d) Chat



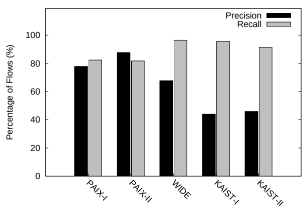
(e) FTP



(f) P2P



(g) Streaming



(h) Game

Figure 3: Per-application precision and recall of CoralReef

default ports usage. Figure 2 shows that the overall accuracy of CoralReef on the traces ranges from 71.4% to 95.9%. Comparing Figure 2 with Figure 1 (a), we find that the overall accuracy of CoralReef is highly dependent on the traffic mix, e.g., inversely proportional to the fraction of P2P flows in a given trace. The PAIX-II and KAIST traces with the highest fraction of P2P flows (4.0%-13.2%) have the lowest overall accuracy with CoralReef classification. In contrast, the WIDE and Keio traces on which CoralReef achieves the highest overall accuracy contain the smallest portion of P2P flows (less than 1%) among all examined traces. These observations motivate our detailed study of per-application performance of CoralReef, which we summarize next.

4.1.2 Per-application performance

Figure 3 shows the per-application precision and recall of CoralReef on eight major applications: WWW, DNS, Mail, Chat, FTP, P2P, Streaming, and Games, which comprise most (86.6%-95.7%) of the traffic flows whose ground truth we know. As shown in Figure 3, we find that each application consistently shares one of three sets of distinct characteristics across all traces – (i) high precision and high recall (WWW, DNS, Mail, and Chat); (ii) high precision but lower recall (P2P and FTP); and (iii) lower precision but high recall (Streaming and Game). The high precision of a port-based classifier such as CoralReef on an application implies that its default ports are seldom used by other applications whereas high recall implies that the corresponding application mostly uses its default ports.

Despite the common perception that ports are no longer (or generally less) reliable and useful, port-based application still identifies legacy applications and protocols quite accurately, and often these constitute the majority of traffic on a link. For WWW, DNS, Mail, News, SNMP, NTP, Chat, and SSH flows, CoralReef achieves high precision and recall on our traces (both $> 90\%$). Flows belonging to DNS, Mail, SNMP, News, and NTP are classified with more than 98.9% precision and recall on all examined traces.

Nonetheless, it is important to recognize that port-based classification fails to yield accurate classification results in the following two cases: (i) when an application uses ephemeral non-default ports, e.g., P2P and passive FTP data transfer degrade the recall of CoralReef. In our data set, 49.4%-96.1% of P2P flows use ephemeral ports. (ii) when the default ports of an application coincide with port masquerading P2P applications, e.g., Streaming and Game ports were often used by P2P applications, which degrades the precision of CoralReef. 12.0%-75.0% of flows on the default ports of Streaming and Game applications turned out to be P2P traffic, according to payload inspection. Contrary to recent claims of P2P applications masquerading on WWW ports to evade detection and blocking, we found little evidence of such masquerading in our traces: only 0.1%-0.5% of the flows on WWW ports were deemed P2P (We are not aware of any firewalling or filtering on the monitored links that might motivate such masquerading, so we cannot claim it is so rare on more heavily firewalled parts of the Internet).

Finding 1 *Port-based approach still accurately identifies most legacy applications for the dataset at our hand (though the two backbone traces were collected in 2004); its weakness is in identifying applications that use ephemeral ports or traffic masquerading behind a port typically used for another application. Although we did not apply our analysis to attack flows or those for which we did not have any ground truth, this finding suggests that ports still possess significant discriminative power in classifying certain types of traffic.*

4.2 BLINC

For each trace we perform about 25 trials to configure BLINC's 28 threshold parameters for the best performance in precision and recall (precision takes precedence in tradeoffs, since recall errors can be mitigated by other methods [25]). Parameter values that optimize the precision may differ on different links, so separate (per-trace) tuning prevents degradation of overall accuracy by 10%-20%. Our experience also

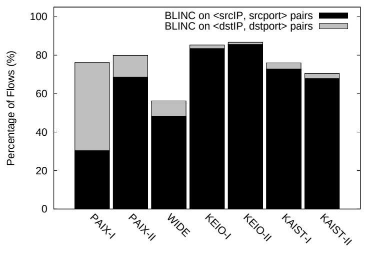


Figure 4: Overall accuracy of BLINC

suggests that one should tune the BLINC parameters related to P2P applications first since almost every BLINC module relies on them.

4.2.1 Overall accuracy

The original BLINC implementation generates graphlets of source ⟨IP, port⟩ pairs that represent communication behavior, and then investigates whether each source graphlet follows a typical server pattern, e.g., WWW, DNS, SMTP. Once BLINC finds a source ⟨IP, port⟩ pair behaves like a specific type of application server, it classifies all ⟨IP, port⟩ pairs that have talked to this server as the same application clients. Thus, if a non-bidirectional backbone trace contains client flows but misses response flows from the corresponding servers, BLINC can not classify those client flows (classifies them in its 'unknown' class). To address this critical limitation in classifying non-bidirectional backbone traffic, we extend the BLINC implementation to generate node profiles of not only source ⟨IP, port⟩ pairs but also of destination ⟨IP, ports⟩ pairs, because we find that server ports of some applications like Web can be identified by applying the same graphlet matching algorithm on destination ⟨IP, port⟩ pairs of client flows in the opposite direction.

Figure 4 shows the overall accuracy of the modified code, Reverse BLINC, on our traces. Reverse BLINC on destination ⟨IP, port⟩ pairs improved the overall accuracy on the PAIX and WIDE backbone traces by as much as 45%, since in those traces one of the two directions of traffic is often missing due to asymmetric routing. Most of the flows that Reverse BLINC identified were of WWW and P2P clients. However, our (admittedly research-grade) code extension almost doubles execution time.

4.2.2 Per-application performance

Figure 5 shows BLINC's per-application precision and recall. Once tuned, BLINC classifies WWW, DNS, Mail, Chat, FTP, and Streaming flows with greater than 90% precision. However, recall for these applications is weaker than precision, since all classification is threshold-based: the number of application flows from a given source must exceed a certain threshold in order to trigger classification. If there are too few flows from this source, its traffic remains unclassified. DNS, Mail and Chat have lower recall in backbone traces than in edge traces, because even Reverse BLINC could not capture those application flows when server flows were missing from backbone traces. Recall for FTP, Streaming, and Game is always lower than 25.8% across all traces, since host behavior signatures of BLINC for these applications do not cover the following cases: (i) when a Streaming or FTP server concurrently provides any other application services; (ii) when a Game client sends any TCP flows or talks to only a few destination hosts.

With proper tuning, BLINC reliably identifies P2P flows, particularly when we first apply port-based classification to filter out DNS server-to-server (indeed essentially P2P) flows which BLINC often misclassifies as P2P. When we filter out DNS flows first and then apply BLINC to the remaining flows, BLINC achieves >85% precision for P2P application flows. However, recall of P2P traffic measured in bytes is significantly (20.5%-61.9%) less than measured in flows. This difference in recall is due to the fact that some P2P applications usually assign different ephemeral ports for every single data transfer. If such transfers are large, then they account for a large number of bytes, but the number of flows remains below our classification triggering threshold, so this traffic remains unclassified.

Finding 2 *We emprically found that, since BLINC (i) classifies traffic based on the observed behavior of server hosts and (ii) adopts a threshold-based triggering mechanism, it depends on whether the traffic containing enough behavioral information about each host. Thus, the best place to use BLINC is the border link of a single-homed edge network where it can observe as much behavioral information of internal hosts as possible. For the same reason, BLINC is not appropriate for backbone links, where (a) only a small portion of behavioral information is collectible for each logged host and (b) we often miss one direction of traffic due to asymmetric routing.*

4.2.3 Computational performance

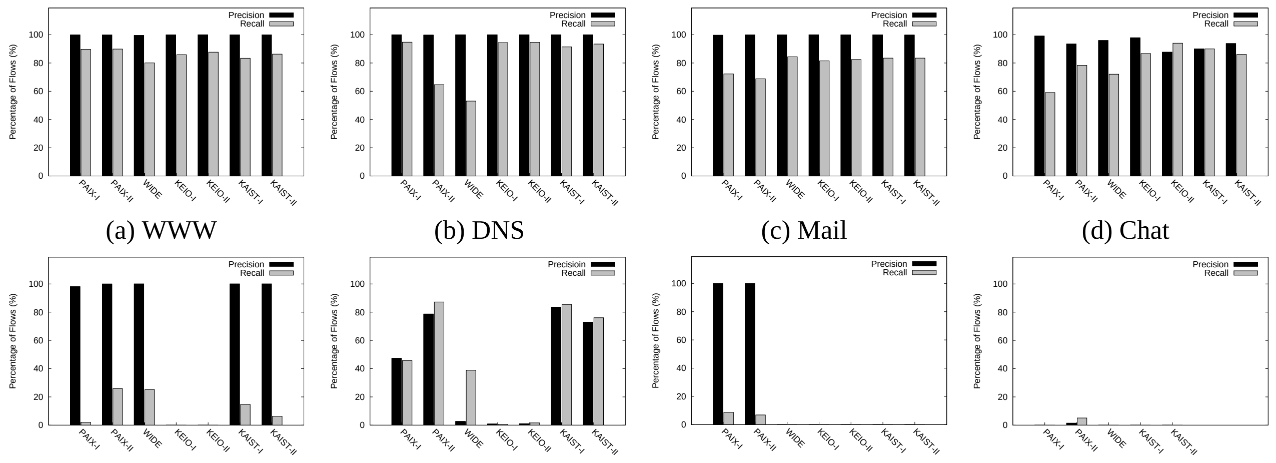
When running BLINC, the running time and memory usage depend on the number of flows that need processing in a time interval. The BLINC code (in C++) ran on the Keio, KAIST, WIDE, and PAIX-I traces in real-time using less than 2 GB of main memory. These traces contain less than one million flows per five minute interval on average. However, it took 16 hours to process the 2 hours of PAIX-II trace containing 4.7 million flows per interval on average, consuming around 9-10 GB of memory. We used a PC server with two 2.4 GHz Zeon CPUs and 4 GB of memory to run BLINC on the Keio, KAIST, and WIDE traces. For the PAIX backbone traces, we used SDSC's SUN Fire 15000 system with 228 GB of memory and SDSC's 72 UltraSPARC3 900 MHz CPUs (we used only one CPU, since the BLINC code is not implemented to support parallel processing with multiple CPUs).

4.3 Supervised Machine Learning Algorithms

We next evaluate the classification performance of the seven most well-known supervised machine learning algorithms.

4.3.1 Key flow features

We first find key flow features for accurate traffic classification using the CFS algorithm with Best First search.



(a) WWW (b) DNS (c) Mail (d) Chat

(e) FTP (f) P2P (g) Streaming (h) Game

Figure 5: Per-application precision and recall of BLINC

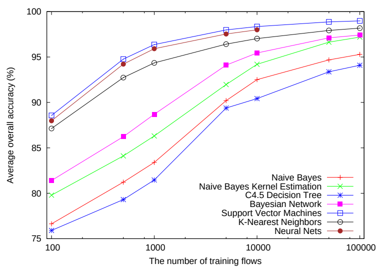


Figure 6: Average overall accuracy of machine learning algorithms by training set size

For every trace, the CFS selected four categories of features: protocol, ports, TCP flags and packet size information, reducing the number of features required from 37 to 6-10.[5] Features such as packet inter-arrival times, which vary greatly by link, are not chosen as a key discriminator in any trace. The results shed light on the feasibility of a truly robust traffic classifier, which we address in the next section.

According to our analysis, using the selected feature subset degrades overall accuracy by only 0.1-1.4% compared to using all 37 features, while dramatically reducing the required training time, which increases the model (classifier) building speed by 3-10X. The feature selection process thus provides an excellent trade-off between feature space reduction and loss of accuracy, confirming findings in [39]. Henceforth we will use the selected key features to evaluate the performance of the learning algorithms.

4.3.2 Overall accuracy

Figure 6 shows the overall accuracy of the seven machine learning algorithms as the training set size varies (from 100 to 100,000). Figure 6 does not show the results of the Neural Network method for larger training set sizes, since the algorithm was prohibitively slow in building a classifier with more than ten thousand training instances (Figure 7(a)).

For every trace, with any size training set, we always obtained consistent results. In our experimental setup, the SVM achieves the highest overall accuracy, followed by Neural Network (although it is quite slow to train) and *k*-NN. The highest performing method, SVM, achieves more than 98.0% average accuracy on all traces with 5,000 training flows, which amounts only 2.5% of the size of the testing sets. The SVM classifier appears to need little training – around 5K-10K training instances sufficed in our study – which makes it promising for practical Internet traffic classification since training data is scarce [20]. The Neural Net method achieves similar accuracy but is 10-1000X slower than SVM in training and testing, when evaluated on the same dataset.

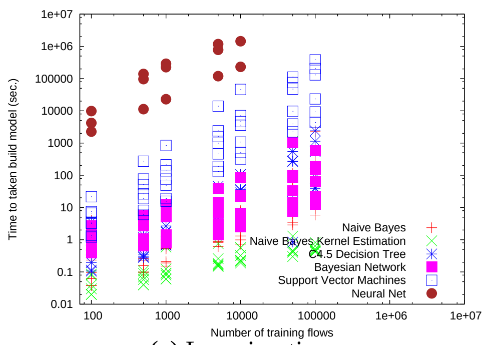
Bayesian Network, Naive Bayes Kernel Estimation, Naive Bayes, and C4.5 Decision Tree follow the top three algorithms, requiring many more (around ten to several hundred times) training instances than the top three methods do to achieve the same level of overall accuracy.

4.3.3 Computational performance

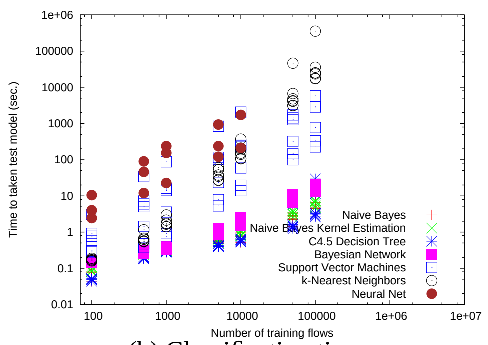
Figure 7(a) and 7(b) show the learning time and classification time of the seven algorithms with increasing training set size.[6] Naive Bayes, Naive Bayes Kernel Estimation, Bayesian Networks, and C4.5 Decision Trees are the four fastest algorithms in learning as well as classification followed by *k*-NN, SVM, and Neural Network. Since *k*-NN does not really involve any training, Figure 7(a) does not include plots for the algorithm. The fastest classification algorithm is C4.5 Decision Tree. While *k*-NN learns and classifies quickly with a smaller training set, its classification

[5]We leave a detailed study on feature ranking across different applications/traces as future work.

[6]Note that we have evaluated the performance of concrete implementations in the Java-based (slow) WEKA software suite on our test platform, not the theoretical complexity of the algorithms because (i) traffi c classifi cation efforts [29, 18, 16, 32, 31, 39] have often used WEKA, and (ii) this approach yields tangible performance numbers for and comparisons [39]. Optimized implementations would likely yield faster learning and classifi cation speeds for all algorithms.



(a) Learning time



(b) Classification time

Figure 7: Computational performance of machine learning algorithms by training set size

time curve shows the steepest increase as the training set size grows, eventually becoming slower than SVM when trained with more than ten thousand instances. While it takes longer to build an SVM classifier, its classification time is 10-100 times shorter than its learning time, making it more practical than the *k*-NN and Neural Network. The Neural Network is the slowest, particularly in learning. We run WEKA on two different SDSC platforms: a SUN Fire 15000 system with 228 GB memory and seventy two 900 MHz UltraSPARC3 CPUs, and an IBM DataStar system with 256 GB memory and thirty two 1.7 GHz Power4+ CPUs (used only one CPU, since the WEKA code is not implemented to support parallel processing with multiple CPUs).[7]

4.3.4 Per-application performance

Using the F-measure metric described in Section 3, Figure 8 shows per-application performance of the seven machine learning algorithms by training set size. The SVM performs the best in terms of the per-application F-measure as well, showing over 95% F-measure for any application with more than a few thousand training flows.[8] Figure 8 clearly shows that the per-application F-measure of the SVM significantly drops as the training set size decreases to fewer than 1000. *k*-NN achieves lower F-measures than those of SVM particularly on P2P, FTP, Streaming, and Chat. The Neural Network also underperforms on our traces, though we have only limited results for per-application F-measure due to its extremely slow training.

For all the algorithms in every trace, Web and DNS are the easiest to classify. A few hundred training flows are enough to identify them with more than 88%-95% F-measure. In contrast, P2P and FTP applications require the most training, not surprising since each application category itself contains multiple applications and/or communication patterns, e.g., data channel and control channel of FTP, etc.

Finding 3 *Protocol, ports, packet size, and TCP flags are key flow features in accurate classification of unidirectional traffic flows. The SVM using these key features performed the best for every application studied and on every backbone and edge trace examined, requiring the fewest training flows (at most around a few thousand) for each application compared to other algorithms.*

4.4 Comparative Analysis

Across all the traces, the SVM classifier consistently outperformed all other methods we evaluated. Table 3 compares CoralReef, BLINC, and supervised machine learning algorithms from various perspectives for practical Internet traffic classification summarizing our results.

5. ROBUST TRAFFIC CLASSIFICATION

A major goal of traffic classification research is to produce a robust classifier which performs well on both available and unseen datasets. Progress toward this noble goal is limited by the lack of measurement data from a sufficient variety of links. We explore the feasibility of building a robust classifier based on what we have learned so far.

5.1 Support Vector Machine Classifier

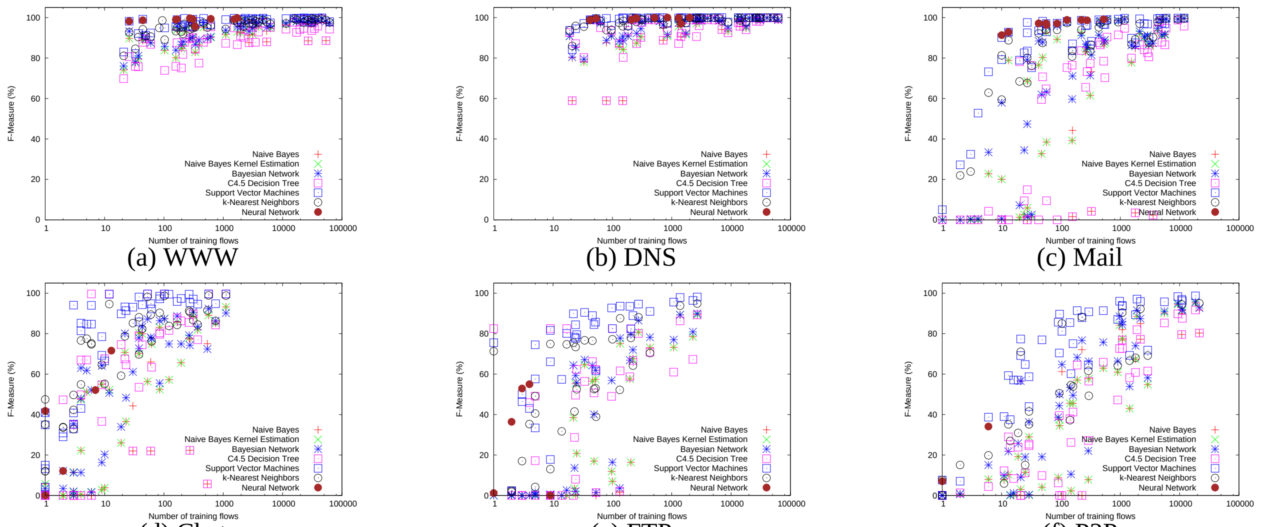
To build a robust classifier, we must consider three factors. First, a set of discriminating features; we use protocols, ports, TCP header flags, and packet size information, recommended by the Correlation-based Filter (CFS) algorithm in Section 4.3.1. Second, an effective classification algorithm; we choose the SVM, which consistently outperformed all others we tested. Third, a correct and complete training set; To minimize the effects of data bias, we compose a training set from three traces collected at different links, the PAIX-II, Keio-I, and KAIST-I. We randomly sample 1,000 flows for each application from each of the three traces (in total, 3K flows for each application) and then merge all of the sampled flows into a single training set.

To test the robustness of the SVM classifier, we use all flows in ten payload traces as testing sets without sampling – seven traces that we have used so far (Table 1) plus three new unseens, namely KAIST-III,WIDE-II,and POSTECH. [9]

[7]Although all the experiments in this paper were done on supercomputers, please note that we only used one 900 MHz or 1.7 GHz CPU. We are testing the same algorithms on our new desktop PCs with Intel 2.83 GHz Quad-Core CPUs, 8-16 GB RAM, and have thus far not had performance problems.

[8]F-measure plots for Streaming and Games are not shown due to space limitations.

[9]All the new traces were captured during the *"A Day in the Life of*



(a) WWW (b) DNS (c) Mail

(d) Chat (e) FTP (f) P2P

Figure 8: Per-application F-measure of machine learning algorithms by training set size

Table 3: Comparison of Traffic Classification Tools. Adaptability refers to whether the approach is adaptive to changing or different traffic characteristics. Detectability refers to detection ability of new or anomalous applications.

Tools	**CoralReef**	**BLINC**	**Flow features based learning**
Key features	Ports and protocols	Communication behavior patterns of hosts and ports	Ports, protocols, TCP fhgs, and packet size info.
Input data format	Packet header trace or fbw table	Flowtable, with the size of payload bytes for every fbw	Training/testing data with the key features above
Robust to asymmetric routing	Yes	Yes, only for WWW and P2P fbws with Reverse BLINC	Yes, with the key features above
Adaptability	No	Re-tuning needed	May need to retrain, depending on training data
Detectability	No	Yes	No for supervised learning, Yes for unsupervised learning
Stateful or not	Stateless	Heavily stateful	Moderately Stateful
Confi guration complexity	Very Low	Very High (28 parameters)	Depends on algorithms, usually low for traffi c classifi cation
Rules buildup	Manual port number DB update	Manual graphlets setup and parameter tuning	Automatic learning
Advantages	Fast, easy-to-use, and scalable. Good at identifying conventional applications (except Passive FTP)	Good at identifying P2P fbws. Detects new as well as encrypted applications.	Automatic learning. Highly accurate and robust to link and traffi c conditions.
Limitations	Port-masquerading applications	Tuning overheads. Strongly depends on the topological locations and traffi c mixes. Low bytes accuracy.	High computational overheads (SVM, Neural Net., and k-NN) and memory requirements (C4.5)
Challenges	Port DB update; Do we have authorative references for lots of new applications?	(Automatic) Parameter tuning. Code optimization for faster processing and better scalability.	Finding right features, fast and accurate algorithms. Obtaining representative training data for every application to build a robust traffi c classifi er.

5.2 Performance Evaluation

To demonstrate the robustness of our proposed SVM classifier we first compare its overall accuracy with those of the SVM classifiers trained with sampled flows from a single trace as in the previous section. The proposed classifier achieves more than 94.2% overall accuracy over all ten traces, while the previous single-trace-trained SVM classifiers perform poorly on at least one other trace, with 49.8%-83.4% of overall accuracy on these other traces. For example, the SVM trained with 1% of sampled flows from all flows in the KAIST-I trace achieves 49.8% of overall accuracy on the PAIX-II trace. The SVM trained with a 1% sample of flows from the Keio-I trace achieved 63.6% overall accuracy on the KAIST-I trace. Considering that those single-trace-trained SVM classifiers in the previous section achieved more than 95% average overall accuracy on the same trace from which they were trained (using a 0.5% sample; only 1,000 flows), their poor performance on a trace from a different link indicates that the training sets from one trace are not complete enough to allow accurate classification of traffic on different links. The two aspects that allowed the proposed robust SVM to classify traffic accurately on all our traces were: (i) a correct and complete training set for applications contained in a target trace, and (ii) the use of our suggested unidirectional flow features.[10]

the Internet (DITL)" simultaneous Internet data collection event on January 9-10, 2007 (http://www.caida.org/projects/ditl/).

6. DISCUSSION

On ports as key features. One of the key findings of this paper is that port numbers are still relevant to traffic classification, for the datasets we tested. Although their utility for classification may not always hold, we acknowledge that

[10]Given space limitations, we leave a detailed study on the performance of our robust SVM classifi er, e.g., per-application F-measure, as future work.

port-based method should not be ignored. In particular, port lookup can reliably identify many conventional applications accurately, especially when used with packet size information, TCP header flags and protocol. Excluding port information from the above key features in training an SVM classifier reduced the overall accuracy from 95%-99% to 56%-70%. On the other hand, conventional applications are not what have catapulted traffic classification activities into the popular press. The more interest there is in identifying traffic in order to constrain or charge users, the more incentive there will be to hinder port-classification methods. Indeed, at any time, or on any link, traffic may use unrecognized ports, or misuse recognized ports to explicitly hide traffic. Thus, the real challenge (and fear) is not in recognizing conventional applications, but in detecting applications that are trying to hide, e.g., via port masquerading or encryption.

On behavior based classification. While port information and flow-features-based approaches make classification decisions on a per-flow basis, host-behavior-based classification as implemented in BLINC aggregates flow information for an interval to derive behavioral patterns of observed hosts. The accuracy of a host-behavior-based classification strongly depends on whether the observed link is located at a topologically appropriate place to collect enough behavioral information on hosts [25]. Consequently, BLINC is effective on links that capture both directions of every flow to a host, such as the border link of a single-homed edge network. We empirically showed that host-behavior analysis is less powerful on aggregated, e.g., backbone, links, where often only a small portion of flows from/to an end-host can be observed, and where asymmetric routing prevents observation of both directions of traffic.

On byte accuracy. The other limitation of the aggregated-behavior-based approach is, even at a topologically appropriate place, these techniques will fail to classify traffic from/to entities whose flows seldom traverse the target link. As a result, they often misclassify as unknown a few large "elephant" flows from/to such entities, achieving lower byte accuracy (or recall) than flow accuracy (or recall). For example, the byte accuracy of BLINC was significantly lower (13.1%-59.3%) than its flow accuracy (56.2%-86.7%) on our traces. Karagiannis *et al.* had similar results in [25]. This weakness is a serious flaw for practical traffic classification, as elephant flows may account for the majority of bytes transferred on typical networks [11]. A complementary per-flow based classification process on remaining unclassified flows is needed to overcome this limitation. Erman *et al.* showed that a cost-sensitive sampling approach allowed machine learning algorithms to achieve high byte accuracy and flow accuracy [19]. This approach trains a classifier with more of the rare but resource-intensive cases, i.e., elephant flows. They trained their classifier with a training set that contained 50% of flows below the 95% percentile of flow size and 50% of flows above the 95% percentile of flow size. This technique substantially improved the byte accuracy for the classifier, with only a marginal reduction in flow accuracy.

On single vs. bidirectional flow features for backbone traffic classification. Accurate traffic classification is feasible only when a classifier possesses correct, complete fingerprints for target applications. Previous efforts on flow-features-based classification [29, 31, 36, 9, 16, 7, 14, 42, 20, 39] have shown that bidirectional TCP flow statistics provide such fingerprints for various applications. However, these methods are not appropriate for classifying backbone traffic where one direction of a TCP flow is unobserved due to routing asymmetries. Backbone traffic classification is challenging because only partial information about bidirectional TCP connections is available. Erman *et al.* addressed this limitation by proposing an algorithm that uses the packets of an unidirectional TCP flow to estimate the flow statistics of the unobserved direction [17], leaving UDP traffic classification as future work. In this paper we addressed this problem by using ports as key features in addition to TCP flags and packet size, based on (i) our finding that ports still identify many conventional applications and (ii) the results of the Correlation-based Feature Selection. The resulting classifiers show accuracies comparable to or higher than those of previous work, on all our backbone and edge traces, using our suggested single-direction flow features. While port information does not seem necessary when we train a learning algorithm with bi-directional flow features to classify TCP traffic, it is indispensable when using only single-direction flow features to classify both TCP and UDP traffic.

On the SVM-based robust traffic classifier. The SVM had already been shown to perform well in multiple areas of various pattern classification problems [8]. We showed that the SVM performs traffic classification accurately as well, once we train the algorithm with a representative training set using our suggested key flow features. To build a robust traffic classifier with the SVM and our suggested flow features, we need to find a representative training set which is not biased toward link characteristics and traffic conditions, for every application. Progress on this goal will require that traffic classification researchers share their tools and data, labeled with ground truth (particularly of new applications with port masquerading and encryption), collected from diverse links on the Internet.

7. CONCLUSIONS

We conducted a detailed comparison of three well-studied approaches to traffic classification: ports-based, host-behavior-based, and flow-features-based. Our study yielded several insights: (a) The effectiveness of port-based classification in identifying legacy applications is still impressive, further strengthened by the use of packet size and TCP flag information. This fact explains why research attention has shifted to detecting and identifying new applications that use port-masquerading and encryption, i.e., traffic deliberately trying to evade traffic classification. Unfortunately, increasing attention to classifying traffic for purposes not necessarily approved by originator of the traffic is likely to increase this category of traffic, inducing an arms race between those trying to classify traffic, and those trying to avoid having their traffic classified. (b) Each approach has its own strengths and weaknesses, and careful combinations can provide synergy. When an approach has a fundamental weakness in classifying particular types of traffic, integrating aspects of other techniques can help. For example, host-behavior-based methods such as BLINC can be augmented with per-flow based classification process to increase byte accuracy. (c) The SVM consistently achieved the highest accuracy. We also developed a robust SVM which achieved 94.2%-97.8% accuracy on both available and unseen traces.

Scientifically grounded traffic classification research requires that researchers share tools and algorithms, and baseline data sets from a wide range of Internet links to reproduce results. In pursuit of this goal, we make our code, dataset labeled with ground truth, and classifiers available for researchers interested in validating or extending our work.

8. ACKNOWLEDGMENTS

This research was supported in part by MKE/IITA, Korea, for the IT Scholarship program, the ITRC support program (grant number IITA-2008-C1090-0803-0004), NSF grant CONMI CRI-0551542, and SDSC and NSF for SDSC's TeraGrid and compute resources. Special thanks to Thomas Karagiannis, Kenjiro Cho, Kilnam Chon, Sue Moon, James Won-Ki Hong, Nevil Brownlee, and Peter Shin for sharing their codes and datasets, and excellent feedback.

9. REFERENCES

[1] *CoralReef*. `http://www.caida.org/tools/measurement/coralreef/`.

[2] Ellacoya. `http://www.ellacoya.com`.

[3] Packeteer. `http://www.packeteer.com`.

[4] *WEKA: Data Mining Software in Java*. `http://www.cs.waikato.ac.nz/ml/weka/`.

[5] M. Allman, V. Paxson, and J. Terrell. A brief history of scanning. In *ACM IMC*, April 2004.

[6] A. Appice, M. Ceci, S. Rawles, and P. Flach. Redundant feature elimination for multi-class problems. In *ICML*, July 2004.

[7] T. Auld, A. W. Moore, and S. F. Gull. Bayesian neural networks for internet traffi c classifi cation. *IEEE Transactions on Neural Networks*, 18(1):223–239, January 2007.

[8] K. Bennett and C. Campbell. Support vector machines: Hype or hallelujah? *ACM SIGKDD Explorations*, 2(2):1–13, 2000.

[9] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identifi cation. In *ACM CoNEXT*, December 2006.

[10] A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artifi cial Intelligence*, 97(1-2):245–271, 1997.

[11] K. chan Lan and J. Heidemann. A measurement study of correlation of Internet fbw characteristics. *Computer Networks*, 50(1):46–62, January 2006.

[12] T. Choi, C. Kim, S. Yoon, J. Park, B. Lee, H. Kim, and H. Chung. Content-aware internet application traffi c measurement and analysis. In *IEEE/IFIP NOMS*, April 2004.

[13] K. Claffy, H.-W. Braun, and G. C. Polyzos. A parameterizable methodology for internet traffi c fbw profi ling. *IEEE JSAC Special Issue on the Global Internet*, 1995.

[14] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffi c classifi cation through simple statistical fi ngerprinting. *ACM SIGCOMM CCR*, 37(1):7–16, January 2007.

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

[16] J. Erman, M. Arlitt, and A. Mahanti. Traffi c Classifi caton Using Clustering Algorithms. In *ACM SIGCOMM MineNet Workshop*, September 2006.

[17] J. Erman, M. Arlitt, A. Mahanti, and C. Williamson. Identifying and Discriminating Between Web and Peer-to-Peer Traffi c in the Network Core. In *WWW*, May 2007.

[18] J. Erman, A. Mahanti, and M. Arlitt. Internet Traffi c Identifi cation using Machine Learning. In *IEEE Globecom*, November 2006.

[19] J. Erman, A. Mahanti, and M. Arlitt. Byte Me: A Case for Byte Accuracy in Traffi c Classifi cation. In *ACM SIGMETRICS MineNet Workshop*, June 2007.

[20] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. Offline/Realtime Traffi c Classifi cation Using Semi-Supervised Learning. In *IFIP Performance*, October 2007.

[21] M. E. Fiuczynski. Planetlab: overview, history, and future directions. *ACM SIGOPS Operating Systems Review*, 40(1):6–10, January 2006.

[22] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. Acas: Automataed construction of applicatin signatures. In *SIGCOMM MineNet Workshop*, August 2005.

[23] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffi c dispersion graphs. In *ACM IMC*, October 2007.

[24] T. Karagiannis, A. Broido, M. Faloutsos, and kc claffy. Transport layer identifi cation of p2p traffi c. In *ACM IMC*, October 2004.

[25] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. Blinc: Multilevel traffi c classifi cation in the dark. In *ACM SIGCOMM*, August 2005.

[26] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffi c feature distributions. In *ACM SIGCOMM*, August 2005.

[27] Z. Li, R. Yuan, and X. Guan. Accurate Classifi cation of the Internet Traffi c Based on the SVM Method. In *ICC*, June 2007.

[28] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected means of protocol inference. In *ACM IMC*, 2006.

[29] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *PAM*, April 2004.

[30] A. Moore and K. Papagiannaki. Toward the accurate identifi cation of network applications. In *PAM*, April 2005.

[31] A. Moore and D. Zuev. Internet traffi c classifi cation using bayesian analysis techniques. In *ACM SIGMETRICS*, June 2005.

[32] T. T. Nguyen and G. Armitage. Training on multiple sub-fbws to optimise the use of machine learning classifi ers in real-world ip networks. In *IEEE LCN*, November 2006.

[33] T. T. Nguyen and G. Armitage. A survey of techniques for internet traffi c classifi cation using machine learning. *IEEE Communications Surveys and Tutorials*, to appear, 2008.

[34] V. Pervouchine and G. Leedham. Extraction and analysis of forensic document examiner features used for writer identifi cation. *Pattern Recognition*, 40(3):1004–1013, March 2007.

[35] J. C. Plat. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, April 1998.

[36] M. Roughan, S. Sen, O. Spatscheck, and N. Duffi eld. Class-of-service mapping for qos: a statistical signature-based approach to ip traffi c classifi cation. In *ACM IMC*, October 2004.

[37] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identifi cation of p2p traffi c using application signatures. In *WWW*, May 2004.

[38] N. Williams, S. Zander, and G. Armitage. Evaluating machine learning algorithms for automated network application identifi cation. Technical Report 060401B, CAIA, Swinburne Univ., April 2006.

[39] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of fi ve machine learning algorithms for practical ip traffi c fbw classifi cation. *ACM SIGCOMM CCR*, 36(5):7–15, October 2006.

[40] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd ed.* Morgan Kaufmann, 2005.

[41] Y. J. Won, B.-C. Park, H.-T. Ju, M.-S. Kim, and J. W. Hong. A hybrid approach for accurate application traffi c idenfi ciation. In *IEEE/IFIP E2EMON*, April 2006.

[42] S. Zander, T. Nguyen, and G. Armitage. Automated traffi c classifi cation and application identifi cation using machine learning. In *IEEE LCN*, November 2005.