The Pennsylvania State University
The Graduate School

A FLOW CLASSIFIER WITH TAMPER-RESISTANT FEATURES AND AN EVALUATION OF ITS PORTABILITY TO NEW DOMAINS

A Thesis in
Electrical Engineering
by
Guixi Zou

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2011

Abstract

Flow classification techniques can be applied to network and end-host security, *e.g.*, a flow whose destination port number indicates one application type, but whose features reflect another, is an anomaly that may indicate malicious activity. Also, network planners may wish to digest [7],[31] the types and quantities of packet-flows they handle in order to decide how to expand their network to better accommodate them. Moreover, any legal terms-of-use policies may need to be enforced by administrators of private-enterprise networks and ISPs.

Flow classification by application type is motivated by on-line anomaly detection, off-line network planning, and on-line enforcement of terms-of-use policies by public ISPs or by administrators of private-enterprise networks. Both signature matching and a variety of feature-based pattern recognition methods have been applied to address this problem. In this thesis, we propose a TCP flow classifier that employs neither packet header information that is protocol-specific (including port numbers) nor packet-payload information. Techniques based on the former are readily evadable, while detailed yet scalable inspection of packet payloads is difficult to achieve, may violate privacy laws, and is defeated by data encryption.

In this thesis, our classifier is tested on two contemporary publicly available datasets recorded in similar networking contexts. We consider the often encountered scenario where ground-truth labels, necessary for supervised classifier training, are unavailable for a domain where flow classification needs to be applied. In this case, one must "port over" a classifier trained on one domain to make decisions on another. We address issues in reconciling differences in class definitions between the two domains. We also demonstrate by our results that domain differences in the class-conditional feature distributions, which will exist in practice, can lead to substantial losses in classification accuracy on the new domain. Finally, we also propose and evaluate a hypothesis testing approach to detect port spoofing by exploiting confusion.

Table of Contents

List of Figures **vi**

List of Tables **vii**

Acknowledgments **viii**

Chapter 1
Introduction **1**
1.1 Introduction 1
1.2 Problem statement 1
1.3 Challenge 2
1.4 Contribution of the thesis 4

Chapter 2
Related Work **5**
2.1 Related work 5
2.2 Classification based on Bayesian approach 6
2.3 Classification based on non-Bayesian approach 7
2.4 Limitations of classification based on port numbers 7

Chapter 3
Experiment Design, Results Analysis and Discussion **8**
3.1 Development of the proposed
flow classifier 8
3.1.1 Packet trace data 8
3.1.2 Flow definition 9
3.1.3 flow-level features 11
3.1.4 C4.5 decision tree 13
3.1.5 Consensus class definitions 14
3.1.6 Ground truth labels 19
3.2 Classifier performance evaluation 22
3.2.1 Experiments on the Cambridge traces and SiteB dataset 22
3.2.2 Porting experiments on the Berkeley trace 24
3.2.2.1 Error source analysis based on the confusion matrix 25

3.2.2.2 Differences in feature distributions per class 25
3.2.2.3 Minimal impact of consensus class definitions 26
3.2.3 Port spoofing detection on Cambridge 27

Chapter 4
Conclusion and Future Work **30**
4.1 Conclusions 30
4.2 Future work 30

List of Figures

3.1 Histogram of feature 180 (fragmented on left) 11
3.2 Histogram of feature 95 and feature 96 12
3.3 Consensus class definitions 18
3.4 Berkeley TCP-subset application histogram 20
3.5 Histogram of feature 83, class-independent (Berk. on left, Camb. in middle and Cons. on right) 21
3.6 Histogram of feature 95, class independent (Berk. on left, Camb. in middle and Cons. on right) 21
3.7 Histogram of feature 83, class-dependent (Berk. on left, Camb. in middle and Cons. on right) 21
3.8 Histogram of feature 83 for "www" class (Camb. on left) 26
3.9 Histogram of feature 95 for "www" class (Camb. on left) 26

List of Tables

3.1 Composition of Cambridge Day Traces and SiteB by Flows 9
3.2 Cambridge Flow feature definitions and descriptions 13
3.3 Cambridge flow class definitions and example applications 15
3.4 Berkeley Flow Class definitions and sample applications 16
3.5 Berkeley dataset ground-truth confusion matrix before class splitting operations 17
3.6 Six consensus flow-class pairs 18
3.7 Berkeley dataset ground-truth confusion matrix after class split operations to achieve consensus types. (A: Augmented Berkeley Port Mapper and C: Cambridge Port Mapper) 19
3.8 Cambridge flow classifier's classification accuracy (from [13]) 22
3.9 Port-only classifier's performance on Cambridge dataset 23
3.10 Port-excluded, 10-feature classifier's performance on Cambridge data set 23
3.11 C4.5 accuracy on the Cambridge dataset without port numbers and TCP protocol-dependent features, based on five retained Cambridge features 23
3.12 Cambridge-trained classifier vs. Berkeley-trained classifier performance on the Berkeley dataset with consensus labels 25
3.13 Partial confusion Matrix of the Cambridge-trained classifier used to predict consensus labels on the Berkeley dataset (C/B: Consensus/Berkeley class and C5+3: Cambridge trained "5+3") 25
3.14 Cambridge-trained classifier vs. Berkeley-trained classifier performance on the Berkeley dataset with Cambridge port-mapper labels 27
3.15 Confusion matrix on the Cambridge training set (P: Port Mapper and F: Feature-based classifier) 28
3.16 Confusion matrix on the Cambridge test set before port spoofing(P: Port Mapper and F: Feature-based classifier) 28
3.17 Confusion matrix on the test set of Cambridge data after port spoofing (P: Port-based mapper and F: Feature-based classifier) 28
3.18 Spoofing detection results: TD (true detection) or MD (missed detection, P:Port-based mapper and F: Feature-based classifier) 29

Acknowledgments

First, I would like to express my deep gratitude to my advisor Dr. David Miller for the continuous support he provided me throughout the course of my research work. I would also like to thank Dr. George Kesidis for his valuable guidance to my work. Also I want to thank Dr. Kenneth Jenkins, who agreed to advise me despite his heavy schedule.

Second, I would like to thank my friends and colleagues during my study and stay in the room115, special thanks given to Yanxin Zhang, Xiaonan Zang, Zhufang Lin,Mingwei Liu, and Arnab Roy, for making the lab a fun yet thought provoking environment. I will miss the discussions we have had for the research as well as for the life. I must also thank all my friends at Penn State, especially my roommates and also friends on the basketball court. I must also thank Dr. Dongzhe Yang and Dr. Veena Mellarkod for the help and guidance during my stay at the MathWorks.

Last, but by no means the least, I would like to thank my family: my dearest wife Xiaoqian Yu, for simply being the best and for being a constant source of motivation, my mom, for her love and encouragement and my dad for his constant support and advice. They keep me up in this difficult world.

Chapter 1

Introduction

1.1 Introduction

The principal problem considered in the following is on-line classification of observed traffic flows into application types. Here, a flow is defined to be a sequence of packets with common five-tuples (source IP address, destination IP address, source port number, destination port number, and protocol) that are proximal in time. [1]

Flow classification techniques can be applied to network and end-host security, *e.g.*, a flow whose "server" port number indicates one application type, but whose features reflect another, is an anomaly that may indicate malicious activity. Also, network planners may wish to digest [7, 31] the types and quantities of packet-flows they handle in order to decide how to expand their network to better accommodate them. Moreover, any legal terms-of-use policies may need to be enforced by administrators of private-enterprise networks and ISPs.

1.2 Problem statement

There are several key elements of the flow classification problem that are addressed herein. First, the classifiers we develop are based on features that are (computationally and memory-wise) feasible to measure in practice and which are robust, *i.e.*, features that are not *readily* susceptible to obfuscation or tampering. In particular, we will not employ features based on packet payload information acquired on-line by so-called deep packet inspection (DPI) technology. Inspecting payload information may violate privacy laws and is generally difficult to perform carefully at scale. Moreover, simple encryption methods (*e.g.*, a randomized substitution cypher) can render payload information useless for classification purposes. Also, unlike [14], the classifier we develop will not consider fields that can be readily tampered with such as port numbers or protocol-

[1]More precise definition for the flow will be provided in chapter 3, where we will discuss the impacts of the fragmented flow (ones we can not observe the complete three-way handshake), and the uncertainty in determining the client and server to our investigation.

specific information in the payload header, *e.g.*, the *push-pkts-serv* feature used in the classifier of [14] which is simply a count of packets with the push bit set in the TCP option field of the layer-4 header.

Regarding the use of port numbers, we note that the flow and congestion control mechanisms of TCP can be moved to the application layer and then the application can be run over arbitrary port numbers and UDP. This can be done for legitimate quality-of-service reasons *e.g.*, for reliable UDP media streaming or interactive real-time applications. In the case of BitTorrent, the UDP-based *utorrent* client [29]:

- opts out of the communal congestion control mechanism of TCP,
- is immune to session termination by third parties via forged TCP RST (reset) or FIN packet transmission, and
- is not connection oriented in layer 4 and can easily mount additional measures to complicate flow classification.

A second problem with using port numbers as features is that, increasingly, applications are engaging in port-number "spoofing", *e.g.*, employing standard port numbers corresponding to nominally permitted application types in order to obfuscate their activity and pass through firewalls that block certain port number ranges (which nominally correspond to unauthorized or simply unwanted application types). In the following, we will demonstrate the deleterious effects of port-number spoofing on the performance of classifiers that rely on ports as features. We will also develop methodology for detecting port-number spoofing.

A second, and the most important, focus of the current study concerns issues that relate to the *porting* of a classifier from one domain (the source domain where it was trained, in a supervised fashion) to operate on a second (target) domain, where it is used to classify received flows. This porting is often necessary in practice because supervised training of a classifier specifically for the second domain may be precluded – in particular, the expense, labor, and time associated with determining ground-truth class labels (necessary if supervised training is to be performed on the new domain) may be prohibitive. In this study, as domains we employed two publicly available packet traces, one recorded at the University of Cambridge, UK [2], and the other at U.C. Berkeley's Lawrence Berkeley National Laboratory (LBNL or now just LBL) [12]. The main goal was to evaluate the accuracy of classifiers trained in one domain (Cambridge) and then operationally applied on another (Berkeley).

1.3 Challenge

In performing this study, we encountered several problems endemic to working with publicly available (anonymized) network data from multiple domains, with the aim of building classifiers that can be used on all such, and with the need to assess classifier accuracy on each. First, network traces captured at different sites may not involve the same applications or may not define classes in the same way. For example, one site may contain BitTorrent traffic while another may not. Moreover, one site might define a “Database” class, whereas another might group “Database” traffic within a larger class dubbed “Miscellaneous”. Somewhat less problematic, even when classes defined at two different sites correspond to similar traffic, the class nomenclature used at the two sites will in general differ, *e.g.*, “mail” and “email”.

Accordingly, it is problematic to simply port a classifier from a *source* site, which was trained to discriminate its defined classes, over to operate on a different *target* site, where there is a different set of defined classes of interest. In the sequel we encounter this problem in working with the Cambridge and Berkeley traces. To address it, we will define a set of “consensus” classes for the two sites that aims to best reconcile the distinct sets of defined classes used by the two sites.

Second, it is crucial, not only for supervised classifier training but also for classifier accuracy evaluation, to have ground-truth class labels for each flow. However, classifier porting is needed in the first place because there are no labeled flows for directly training a classifier for the target site. Thus, both in practice and in this study, the absence of ground truth labels for the target site, even for evaluating classifier porting accuracy, is a problem that must be overcome. In this thesis, since we study classifier porting, even if class labels are not assumed to be available for supervised classifier training at the target site, we still need labels for (test) flows from this domain in order to evaluate the classification accuracy of the classifier ported from the source site. Indeed, the Cambridge trace provided ground-truth label information for all flows, which was obtained via DPI (*e.g.*, the *Wireshark* tool [32] can be used for this purpose). Unfortunately, no flow-class labels or packet payloads were provided for the Berkeley trace; thus, some procedure for establishing ground-truth was needed for this trace in order to evaluate accuracy of a classifier ported to this domain. To obtain the necessary ground truth, we applied a port-to-application mapping approach that will be detailed in the sequel. For example, absent port spoofing (which is assumed in performing this mapping for the Berkeley traces), flows with destination port numbers 25, 53, and 80 are reliably bound to the applications “email”, “dns”, and “web”, respectively.

Once we resolved the issues of heterogeneity in class definitions and missing ground truth discussed above, we were ready to evaluate the accuracy of a classifier trained on one domain (Cambridge) but now operating on another (Berkeley). It is not surprising (as will be seen by our results) that accuracy may degrade when operating in a different domain. One possible reason is that different traces may predominantly capture different traffic types. For example, one dataset may come from a traffic monitor deployed close to a mail server, with another coming from a gateway router; email traffic will not predominate the latter dataset, while the former may not have as much intra-enterprise network management traffic. In addition to differences in class priors, there may be (even subtle) differences between class-conditional feature distributions measured at two different sites, which can degrade accuracy when a classifier trained at the source site is ported to the target site.

1.4 Contribution of the thesis

In summary, the contributions of this thesis are as follows:

- a study of classifier porting from one site to another;
- network flow classification based on robust features; and
- a port-spoofing detection methodology and its evaluation.

Chapter 2

Related Work

2.1 Related work

Intrusion detection systems (IDSs) that rely on packet inspection, *e.g.*, Snort [27] and Bro [11], use deterministic and simple statistical signatures to determine known threats and highly suspicious behavior, respectively. Intrusion detection based on packet payload information has also been extensively studied, *e.g.*, detection based on "prevalent content" [7, 20, 30]. Some of the decisions made by network-based IDSs clearly amount to packet-flow classification decisions. [7] describes a new method of traffic characterization that automatically groups traffic into minimal clusters of conspicuous consumption. Rather than providing a static analysis specialized to capture flows, applications, or network-to-network traffic matrices, their approach dynamically produces hybrid traffic definitions that match the underlying usage. Furthermore, this approach can automatically classify new traffic patterns, which could be proved very useful in certain situation, such as network worms or peer-to-peer applications, without knowing the structure of such traffic a priori. [20] describes the design space of worm containment systems using three key parameters: reaction time, containment strategy and deployment scenario. Using a combination of analytic modeling and simulation, they describe how each of these design factors impacts the dynamics of a worm epidemic and, conversely, the minimum engineering requirements necessary to contain the spread of a given worm. [30] describes a new method which integrates header-based multidimensional flow clustering as front-end processing, with content sifting (signature extraction) performed, separately, solely on each cluster in the (small) subset of identified suspicious clusters. It also provides one way to salt public traffic with worm traffic based on both the length and frequency information.

Network flow classification based on a flow's statistical features was proposed in [26]. [26] describes the requirements for Quality of Service (QoS) guarantees to traffic from different applications and associated challenges, and outlines a solution framework for measurement based classification of traffic for QoS based on statistical application signatures. In their approach the signatures were chosen in such a way as to make them insensitive to the particular application layer protocol, but rather so that they determine the way in which an application is used. Using, *e.g.*, average packet size and flow duration, they applied two supervised machine learning approaches: K-nearest neighbor (KNN) and Linear Discriminant Analysis. As typically done for packet traces recorded prior to 2005, they obtained ground-truth application labels from the standard IANA port-application mapping list [9].

2.2 Classification based on Bayesian approach

A naive Bayes classifier combined with kernel estimation and a correlation-based feature selection strategy was used in [19] to solve offline TCP flow classification. In [19], they illustrated the high level of accuracy achievable with the Naive Bayes estimator. Their results indicated that with the simplest Naive Bayes estimator they were able to achieve about 65% accuracy on per-flow classification and with two powerful refinements they could improve this value to better than 95%, which was a vast improvement over traditional techniques that achieved 50.70%. While their technique used training data, with categories derived from packet-content, all of their training and testing was done using header-derived discriminators. Using twelve different features (given in Section 3.1.3 below), they achieved a classification accuracy of 96%. One difficulty with this classifier is that it is based on both port features and TCP-dependent features, both of which can be easily tampered with to trick the classifier.

Following this investigation, [13] reduced the complexity of the approach in [19] to one suitable for on-line deployment by limiting the "observation window" of inspected packets per flow. [13] illustrated the classification of UDP traffic. They used the same classification approach for both stateful flows (TCP) and stateless flows based upon UDP. Importantly, they demonstrated high levels of accuracy: greater than 92% for the worst circumstance regardless of the application. We note that [13] did consider classifier portability both temporally and to another (spatial) domain. However, the classifiers evaluated in [13] used port numbers as features, which we already noted can be easily tampered with. We will demonstrate fragility of such classifiers in the presence of port spoofing in the sequel. Another limitation of [13] is that it did not consider the case where different class definitions are used at the source and target sites. This is a genuine problem, as we note that different definitions were in fact used in defining the Cambridge and Berkeley classes. Subsequently [15] proposed a methodology and detailed design of a technical framework that significantly boosted the efficiency in compiling the application traffic ground truth. Also in [15], the Cambridge researchers developed a ground-truth derivation tool [3]. [3] is backed by the L7-filter [6], *i.e.*, it requires layer-7 information from the packet payloads. So it has limited usage when packet payload information is not available because of laws protecting privacy, which is the case for available, publicly disseminated packet-trace data.

2.3 Classification based on non-Bayesian approach

Recently, [5] correlated packet-level alarms with a feature vector derived from corresponding flow-level statistics not involving payload information. Their flow-level classifier exploited ground-truth labels derived with the help of packet inspection by intrusion detection devices. Their experimental results showed little impairment of classifier performance in deployment over periods of several weeks.

[8] proposed an approach to TCP flow classification based on a flow representation using the statistical properties of an application protocol. The features they used included payload-size statistics of packets composing the flows. bf [8] described an approach to traffic classification based on Support Vector Machine(SVM). The accuracy of the proposed classifier was then evaluated over three sets of traces, coming from different topological points in the Internet. Although the results were relatively preliminary, they confirmed that SVM-based classifiers could be very effective at discriminating trace generated by different applications, even with reduced training set sizes.

All these previous research results demonstrate the effectiveness of classification based on extracted flow statistics; however, all these methods with the exception of [8], have a common weakness in that they used highly protocol-dependent features for classification.

2.4 Limitations of classification based on port numbers

[1] demonstrated that the simple port number scheme can achieve 70% accuracy in the context of packets with payload on the Internet backbone. However, according to [28], P2P(peer-to-peer) utilized non-standard and random port numbers for communication, which rendered the traditional port-based classification scheme less reliable; furthermore, P2P traffic is on the rise, which again could lead to further performance degradation.

[21, 16] focused on P2P traffic classification. They utilized three different approach: port number, application-layer signature and transport-layer longitude over a two-year period. Their results showed that the classical port number based approach is not accurate, which is in accordance with [28]; on the contrary, the application-layer signature achieved higher accuracy. In this thesis, the classification is based on the C4.5 decision tree, where we build a model to correlate the class/application labels and their features' distributions to avoid the pitfall of the traditional simple port number scheme.

Chapter 3

Experiment Design, Results Analysis and Discussion

3.1 Development of the proposed flow classifier

In this section, we describe the Cambridge and Berkeley packet trace data, the flow features we extracted, the flow-class definitions at the two sites and how we reconciled them, the way ground-truth flow-class labels were derived for the Berkeley site, and the classification methods we employed.

3.1.1 Packet trace data

According to the description in [14], the Cambridge trace consists of two consecutive weekdays of Internet traffic taken weekly over an eight month interval. The trace was collected using a high speed packet monitoring apparatus [18] installed at the gateway between a research campus and the Internet. The campus is a research facility with about 1,000 employees and is connected to the Internet via a full-duplex Gigabit Ethernet link. The datasets Day1, Day2 and Day3, consisting of TCP traffic only, were chosen from a collection detailed in [17]. In [13], the authors also evaluated performance of their Cambridge-trained classifier on a network trace from a different spatial domain which they dubbed SiteB. Every flow in the datasets was hand-classified, using a content (including payload) based mechanism, into one of 13 application classes. A number of TCP flows in the datasets were left unconsidered by Cambridge researchers, *e.g.*, those whose start-of-flow packets were not observed (typically, long duration flows which were initialized prior to the start of the day's observation window). For example, the resulting Day1 trace contains 42 million packets in 377 thousand TCP flows and the Day2 trace contains 35 million packets in 175 thousand TCP flows, giving a moderately complex mix of applications in each dataset as shown

Table 3.1. Composition of Cambridge Day Traces and SiteB by Flows

Class	Day1(%)	Day2(%)	SiteB(%)
www	84.558	80.198	85.557
mail	8.682	9.384	4.377
bulk	3.800	6.146	0.223
attack	0.787	0.562	1.614
p2p	0.589	1.572	7.188
data -base	0.862	1.483	0.000
multi -media	0.137	0.002	0.004
service	0.555	0.633	0.187
intera -ctive	0.027	0.021	0.128
games	0.002	0.000	0.060
voip	0.000	0.000	0.420
chat	0.000	0.000	0.204
grid	0.000	0.000	0.037

in Table 3.1 (taken from Table 4 in [13]). See [13] for the corresponding information about Day3 and SiteB.

According to [23], the Berkeley dataset, contemporary with the Cambridge dataset, represents internal enterprise traffic recorded at a medium-sized site. The Berkeley researchers collected packet traces that spanned more than 100 hours of activity from a total of several thousand internal hosts. This data was publicly released in anonymized form and contains a wide range of flow types. All together, 11GB of packet header traces from October 2004 through January 2005 are available for analysis. It is worth noting that the measurement system at LBNL was only able to simultaneously record the traffic out of two of the (more than twenty) router ports monitored. Thus, the packet traces are from a successive sampling of subnets. Their protocol (TCP/UDP/ICMP) distribution is shown in Table 3 of [23] for 5 different days. The raw trace used in the following is a combination of those recorded on Oct. 4, 2004, router port 19, and on Dec. 15, 2004, router port 8[1], consisting of the first 10^6 packets of each, for a total of 6701 TCP flows (again, no UDP or ICMP flows were considered by us). For security and privacy reasons, the Berkeley researchers anonymized their IP addresses. They used a prefix-preserving scheme to remap the external and internal IP addresses. Additionally, the subnet and host portions of the internal addresses were further transformed in a one-to-one fashion. However, the port number in the TCP and UDP headers are intact, which can provide information about application type at the time this trace was recorded.

3.1.2 Flow definition

In this thesis, we investigated both fragmented and unfragmented traffic flow classification. We

[1]From [12], files lbl-internal.20041004-1458.port019.dump.anon and lbl-internal.20041215-0510.port008.dump.anon.

define unfragmented flows as the ones for which we can observe the complete three-way handshake[2]; similarly, we define the fragmented flows as the ones for which we observe partial or no part of the three-way handshake packets. The advantage of the unfragmented flows is that it provides the information of client and server relationship with certainty, which proves to be of great value in classifying a traffic flow according to features' statistical distributions. We will analyze the impact of fragmented flows in our investigation later.

In our thesis, we determine a flow in the following way: for all the packets sniffed, if we meet one with five-tuples [3], this packet marks the start of a flow. We would then check its bit fields in the TCP header. If only its 'SYN' bit set to 1, we then continue to check the following packets to see whether they are 'SYN+ACK' and 'ACK' packets; if so, we will have a complete three-way handshake[4], and thus a unfragmented flow; if not, namely, if we could not observe the 'SYN+ACK', or 'ACK', we have a fragmented flow. If this five-tuples appears before, we then check whether it has timed out [5] by comparing its arrival time with the last packet having the same five-tuple; if no timeout, we will update our flow statistics; otherwise, we mark this packet as the start of a new flow.

We demonstrated the impact of the fragmented and unfragmented flow characteristics by drawing their feature distributions. Taking feature 180 in figure 3.1 as example, it is defined as the variance of packet length from server to client. As we could see from the figure, these two distributions have different ranges, as well as different means and modes; furthermore, the class/label (shown by different colors) distributions are different across the range. These discrepancies could be explained in the following way: since the client mainly sends request, or acknowledgement packets to the server, these packet lengths tend to be small and roughly the same; thus their variance tends to be small; on the contrary, the server needs to send large amount of packets, for transferring data, image or sound and acknowledgement, to client, hence packets from server would have different size, ranging from MTU (Maximum Transmission Unit) to zero (for acknowledgement); therefore this feature shows a higher variance from server to client.

We adopted four rules in determining the client and server relationship.

- If we observe the complete three-way handshake, we have all the information on client and server, and there is no ambiguity in determining the client and server relationship.
- If we do not observe the complete three-way handshake, but we do observe the 'SYN' packet at the beginning of the connection, we can use this 'SYN' packet to estimate the client and

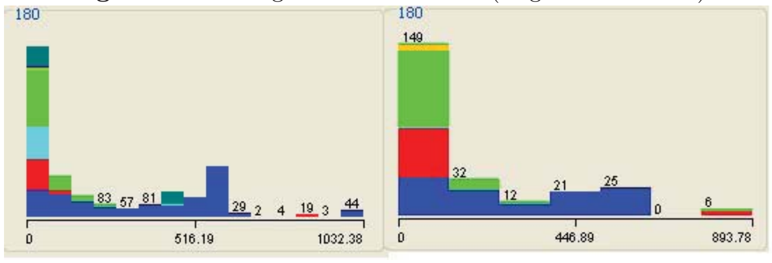
[2]The three-way handshake occurs when TCP (Transport Control Protocol) tries to establish a connection between two hosts, which includes the exchange of the 'SYN', 'SYN+ACK' and 'ACK' packets consecutively between the client and server. A 'SYN' packet is one packet with only the 'SYN' bit set to 1 in TCP header, similarly, a 'ACK' packet is one with only the 'ACK' bit set to 1 and 'SYN+ACK' packet is one with both 'SYN' and 'ACK' set to 1.

[3]A five-tuple is defined as a five-tuple (server IP address, client IP address, server port number, client port number and transport layer protocol) which does not appear in our records. We implemented a hash table based dictionary to record all the five-tuples appeared in the trace, and we would checked the dictionary to see its existence. We will update our dictionary by inserting this five-tuples.

[4]We did allow the duplicated 'SYN' and 'SYN+ACK' packets in our three-way handshake process considering the possible packet loss and duplication in establishing the connection.

[5]Timeout value is application-specific and can be checked from Cisco router configuration manual.

Figure 3.1. Histogram of feature 180 (fragmented on left)



server relationship, considering the 'SYN' packet is mostly used to initiate the conversation by client. This rule may introduce errors since a server may send a 'SYN' packet to client to resume the connections in case the connection suspends.

- If we do not observe the complete three-way handshake, nor the 'SYN' packet, but we observe one of the port numbers is mapped to a popular service assigned by IANA, then it is highly likely that the host with this service running is a server.
- If all the rules above fail to determine the client and server relationship, we will determine it by the port number of two hosts, as IANA assigns lower-end port numbers to system use, and high-end port for individual use. If we could observe a flow associated with one port number within the range of well known ports and the other corresponding to private/dynamic ports, or vice-versa, we consider the host with the smaller number to be server [6] and the other one to be the client.

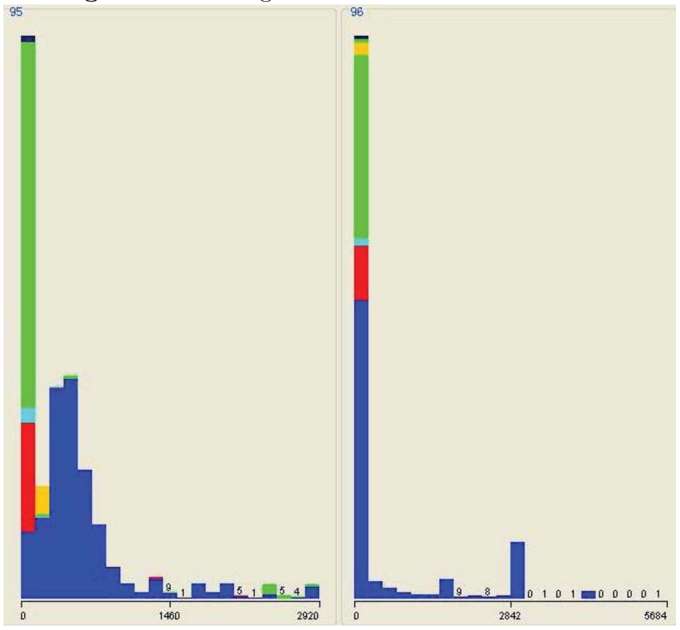
After carefully examining the four rules in determining server and client relationship, we can spot one more potential problem, or ambiguity, in the traffic flow definition. When all the rules fail to determine the server and client, we do not know for sure which host is client and which host is server. In this case, we assume the first packet always comes from the client. This assumption solves the problem, but introduces errors into our feature calculation. To demonstrate the importance and also the difference in the feature distribution between the two opposite directions, we again draw feature distributions. In 3.2, feature 95 is defined as the total number of bytes sent in initial window from client to server and feature 96 is basically the same except that it is defined in the server to client direction. As we can see, big differences exist between these two features. Not only the modes, means and ranges, but also the class-dependent distributions vary between these two. Luckily in our dataset, less than 5% falls in this last category; hence , this will not impact our results much.

3.1.3 flow-level features

In [13], for the Cambridge trace, twelve flow-level features were selected from among 248 features made available, as detailed in [17]. Note that neither anonymized packet headers nor netflow data

[6]According to IANA, the ports 0 through 1023 are well known ports, ports from 1024 to 49151 are registered ports and all other ones above until 69535 are considered as dynamic and/or private ports.

Figure 3.2. Histogram of feature 95 and feature 96



were publicly disseminated – only the 248 features per TCP flow. More specifically, according to [13], the authors used a correlation-based filtering mechanism on each of their ten Day1 subperiods. They found that the feature subsets selected by their algorithm had fairly good stability, and they manually picked twelve features which appeared in at least one third of their features subsets.

The definitions of the twelve features selected by this approach are given in Table 3.2. Note that both the server port number and client port number were chosen by the correlation-based feature extraction approach. The fragility of a classifier using ports as features in the presence of port spoofing will be shown in the sequel. Moreover, it will be shown that good accuracy can be achieved without using the ports. Among the twelve features from [13], we identified the following five features as least dependent on the TCP protocol: avg-seg-size-serv, IP-bytes-med-clnt, act-data-pkt-clnt, data-bytes-var-serv and min-seg-size-clnt. We also identified the following three "derived" features as promising ones that are also not strongly dependent on TCP:

- homo-size-clnt: the ratio of the maximum to the minimum packet size from client to server,
- homo-size-serv: same as homo-size-clnt but in the server to client direction, and
- real throughput (goodput): defined as the total nonduplicate payload length (excluding the header length) divided by transmission time.

The first feature is a standard statistic from the packet-size time-series which, *e.g.*, we expect to be close to 1 for large web downloads predominantly consisting of 1500 Byte payloads. According to [4] for the last feature, transmission time is defined as flow duration minus idle time, where idle times less than 15 seconds are quantized to zero.

Again, simple manipulation of the TCP or IP header options will not significantly affect these features' statistical characteristics. It is possible to obfuscate these features, but not without more sophisticated and resource-intensive tampering, which may also impact the perceived service quality of the affected application. Indeed, these features will not be significantly affected even if the applications are instead run over UDP. Therefore, a classifier built based on these features should tend to be more robust than one based in part on highly TCP-dependent features. The comparative performance of flow classifiers based only on these more "tamper-resistant" features is discussed in the sequel.

Table 3.2. Cambridge Flow feature definitions and descriptions

Abbreviation	Description
serv-port	server port
clnt-port	client port
push-pkts-serv	count of all packets with push bit set in TCP header (server to client)
init-win-bytes-clnt	the total number of bytes sent in initial window (client to server)
init-win-bytes-serv	the total number of bytes sent in initial window (server to client)
avg-seg-size-serv	average segment size: data bytes divided by number of packets
IP-bytes-med-clnt	median of total bytes in IP packet
act-data-pkt-clnt	count of packets with at least one byte of TCP data payload (client to server)
data-bytes-var-serv	variance of total bytes in packets (server to client)
min-seg-size-clnt	minimum segment size observed (client to server)
RTT-samples-clnt	total number of RTT samples found (client to server), see [17]
push-pkts-clnt	count of all packets with push bit set in TCP header

3.1.4 C4.5 decision tree

In this section, we briefly introduce the C4.5 decision tree classifier [24, 25, 10] used in the following experiments and by the Cambridge group in their study.

Binary decision trees perform classification via a nested sequence of one-dimensional tests, each applied to one of the features, which determine whether one branches left or right in traversing to the next layer of the tree, where another test is performed. In this way, for each test (or training) sample one traverses the tree from the root down to a leaf node, where the sample is assigned to one of the classes.

Given a labeled training set, C4.5 designs the tree in a greedy fashion. At each step of the algorithm, one selects, from the set of current leaf nodes, the node and the binary attribute split for that node, such that the *normalized information gain* (NIG) is greatest. Normalized information gain, measured for each attribute split, for every leaf node, is defined as:

$$\mathsf{NIG}(C|X) \quad = \quad \frac{H(C) - H(C|X)}{H(C)}.$$

Here, C is the discrete class random variable, X is the binary variable obtained by a particular split for a given feature, and H is Shannon's entropy evaluated with respect to the training samples classified to the selected node. In this way, one sequentially grows a tree starting from a single (root) node, until the class purity at each of the leaves is sufficiently high. C4.5 also implements a subsequent tree pruning step to simplify the tree's decision structure.

3.1.5 Consensus class definitions

In this section, we develop a procedure to reconcile heterogeneity in the class definitions at two sites, thereby allowing a classifier trained at the source site (Cambridge) to be deployed at the target site (Berkeley). Before developing our approach, it is important to emphasize why the definition of "consensus" classes is needed in our particular study, and more generally whenever a classifier is ported from one domain to another. Even if the Berkeley trace *had* come with ground-truth class labels, the Cambridge and Berkeley class definitions are *not the same.* Even the number of defined classes is not the same for the two domains. Moreover, even though one might be able to match some Berkeley classes with Cambridge classes on the basis of similarity of names and/or claimed applications or protocols that fall under the defined class, such matching may be quite misleading. In particular, in the sequel, we will apply both a Cambridge-based strictly port-based classifier (*i.e.*, a port mapper) and a Berkeley IANA port mapper to the flows in the Berkeley trace, with the former assigning Cambridge-defined class labels and the latter assigning Berkeley-defined class labels. Based on these respective port mapping approaches, we will see that Berkeley's "email" class contains many flows that Cambridge's port-mapper declares to be "p2p", not "mail". More generally, if the number of classes defined on the test domain is smaller than the number of classes defined on the training domain, one must decide whether the test domain classes map one-to-one to a *subset* of the training domain classes, or whether some test domain classes are unions of multiple training domain classes. Even *if* the number of classes is the same, class splitting and merging operations, applied both to the training domain classes and the test domain classes, may be needed in order to define (consensus) class definitions that are *as consistent as possible* with both the training and test class definitions.

The procedure we will follow in defining consensus classes is as follows:

1. Make initial one-to-one assignments between class definitions at the two sites (source and target) based both on similarities in nomenclature and similarities in the applications and protocols that fall under respective class definitions at the two sites.

2. Use a source-dataset port-based mapper to assign each target-dataset flow to one of the source-dataset's classes. Likewise, use a target-dataset's port-based mapper to assign each target-dataset flow to one of the target-dataset's classes. Then based on the initial one-to-one mapped classes, build a *confusion matrix* and measure the *consistency* as the number of off-diagonal entries in the confusion matrix divided by the total number of target-dataset flows[7].

3. Perform class splitting and merging operations, as few as possible, as needed, for both source and target datasets, and map the resulting classes at each of the respective sites to a common set of *consensus* classes. These operations should aim to improve the consistency measure achieved by the initial class assignments.

Table 3.3. Cambridge flow class definitions and example applications

Class	**Applications**
www	http https
mail	imap,pop2/3,smtp
bulk	ftp
attack	portscan, worms, viruses, email attack
p2p	napster, kazaa, eMule, gnutella, edonkey
database	mysql.dbase, Oracle, SQLNet
multimedia	windows, mediaplayer, realmedia
service	X11, dns, ident, ldap, ntp
interactive	ssh, telnet, klogin, rlogin
games	Microsoft DirectPlay
admin	
chat	
voip	

Now, how appropriate is it to apply the Cambridge classifier to the Berkeley trace, to ultimately predict these consensus classes? This depends on the level of the consistency measure

[7]More specifically, consistency $c := N_c/N_t$, where N_c is the total number of consensus flows between the two mapper decisions, and N_t is the total number of measured flows. So, $0 \leq c \leq 1$. Smaller c means larger mapper discrepancy. If $c = 1$, then there is perfect consensus between the class definitions at the two sites.

Table 3.4. Berkeley Flow Class definitions and sample applications

Categories	**Protocols**
backup	dantz, veritas, “connected-backup”
bulk	ftp, hpss
email	smtp, imap, imap/s, pop, pop/S, ldap
interactive	ssh, telnet, rlogin, X11
name	dns, netbios-ns, SrvLoc
net-file	nfs, ncp
net-mgnt	dhcp, ident, ntp, snmp, nav-Ping, SAP, NetInfo-Local
streaming	rtsp,IP Video, RealStream
web	http, https
windows	cifs/smb, dce/rpc, netbios-SSN, netbios-DGM
misc	steltor, MetaSysy, lpd, ipp, Oracle-SQL, MS-SQL
other-tcp	
other-udp	

between the Cambridge and Berkeley class definitions achieved by these consensus classes – if the consistency is 100%, then there is perfect agreement between the consensus class definitions and the Cambridge class definitions, as measured on the Berkeley flows, and it is *perfectly* suitable to test the Cambridge classifier to predict consensus classes on the Berkeley domain. For consistency less than 100%, some of the error rate measured in applying the ported classifier to the new (test) domain is attributable to this (imperfect) agreement between the training class and consensus class definitions.

To quantify this, let $\mathsf{P}[\text{error}]$ be the test set error rate of the ported classifier and $\mathsf{P}[\text{error}|\text{Con}]$ be the *conditional* test set error rate, where we condition on the event that a flow gets assigned to the same consensus class by both the source-site and target-site port mappers. Likewise, $\mathsf{P}[\text{error}|\overline{\text{Con}}]$ is the error rate given that a flow does *not* get assigned to a common consensus class by the two port mappers. Further, note that $\mathsf{P}[\text{Con}] = c$ and $\mathsf{P}[\overline{\text{Con}}] = 1 - c$. $\mathsf{P}[\text{error}]$ is the error rate that we will *measure* for the ported classifier on the target dataset. A more reasonable estimate of the ported classifier's generalization accuracy is a quantity that we may not directly measure, $\mathsf{P}[\text{error}|\text{Con}]$ – this captures the classifier's generalization accuracy over the set of flows for which the class definitions at the two sites are in agreement. However, note that

$$\begin{aligned}\mathsf{P}[\text{error}] &= c \cdot \mathsf{P}[\text{error}|\text{Con}] + (1 - \text{c}) \cdot \mathsf{P}[\text{error}|\overline{\text{Con}}] && (3.1)\\ &\leq c \cdot \mathsf{P}[\text{error}|\text{Con}] + (1 - \text{c}), && (3.2)\end{aligned}$$

with the inequality following from $\mathsf{P}[\text{error}|\overline{\text{Con}}] \leq 1$. Moreover, in fact we would expect $\mathsf{P}[\text{error}|\overline{\text{Con}}] \leq 1$ to be *very close* to 1. Assuming equality in the inequality above, we find that:

$$\mathsf{P}[\text{error}|\text{Con}] = \frac{1}{c}(\mathsf{P}[\text{error}] - (1-c)). \tag{3.3}$$

Note that when $c = 1$, this error rate simply reduces to $\mathsf{P}[\text{error}]$. Moreover, when $c = 1$, generalization accuracy is evaluated over the *full* test domain. When $c < 1$, $\mathsf{P}[\text{error}|\text{Con}]$ does give an accurate generalization measure for the subset of the test domain on which it is reasonable to evaluate the ported classifier's accuracy – the flows for which the two sites map to a common consensus class. However, if c is not close to 1, then this subset does not include some flows that frequently occur on the test domain. Thus, while $\mathsf{P}[\text{error}|\text{Con}]$ is in fact always an accurate measure of the performance of the ported classifier (which we will use in our experimental results, based on (3.3)), it is still desireable to have c as close to 1 as possible. One cannot expect *perfect* consistency in practice, but we will next see that the consistency we do achieve in our current study is reasonable.

Specifically, the Cambridge researchers defined thirteen different application types, as detailed in Table 3.3. On the other hand, the Berkeley group used their own set of thirteen flow class definitions, given in Table 3.4. Note how certain application types (ports) get mapped to (ostensibly) different class types. In our study, to obtain Cambridge class labels for each Berkeley flow, we apply the (strictly) port-based Cambridge-trained C4.5 classifier. Also, to obtain Berkeley class labels for the Berkeley flows, we applied Berkeley's IANA port-mapper, *cf.*, Section 3.1.6.

Table 3.5. Berkeley dataset ground-truth confusion matrix before class splitting operations

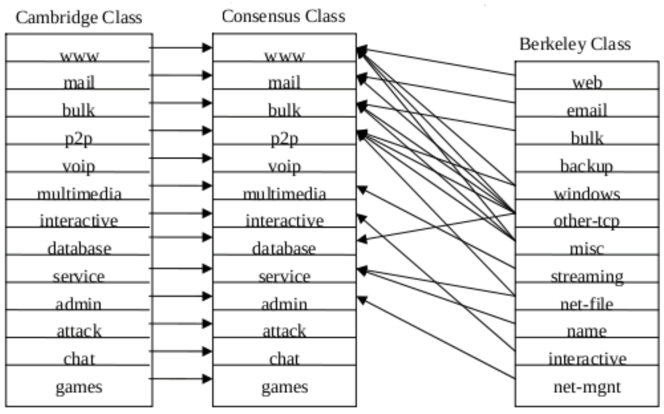
Berkeley Port Mapper \ Cambridge Port Mapper	www	mail	bulk	p2p	db	srvc	intact
web	2851	0	3	11	0	0	0
email	5	97	33	168	1	0	0
bulk	0	0	22	0	0	0	0
backup	0	0	0	0	0	0	0
windows	0	304	1	329	0	0	0
other-tcp	131	7	590	466	42	1	0
misc	84	0	733	283	0	0	0
streaming	88	0	0	30	0	0	0
net-file	0	0	0	278	0	50	0
name	15	1	0	10	5	0	0
interactive	0	0	7	10	0	0	64
net-management	0	0	0	0	1	0	0

For Berkeley flow class definitions on the Berkeley dataset, the confusion matrix of Table

Table 3.6. Six consensus flow-class pairs

Camb.	www	mail	bulk	mmed.	serv.	intera.
Berk.	web	email	bulk	stream.	name	intera.

Figure 3.3. Consensus class definitions



3.5 (given at the end of the article) was obtained using IANA as the row port mapper and Cambridge's C4.5 as the column port mapper, *i.e.*, the $(i, j)^{th}$ entry is the number of flows mapped to class i according to Berkeley's IANA and to class j according to Cambridge's C4.5. We initially measured consistency based on the six consensus pairs of Table 3.6. These six pairings are justified by the fact that the elements in each pair use some or all of the same protocols as seen by Tables 3.3 and 3.4. The measured consistency was 45.3%, which is quite poor.

To improve on this, we defined consensus classes as follows. About one fifth of the Berkeley flows were classified as "other-tcp" according to the Berkeley port-application mapper, which is simply due to the inability of the port-application mapper to assign these ports to Berkeley-defined classes. Since the Cambridge C4.5 port-based mapper achieved 99.42% precision on the Cambridge trace (*cf.*, Section 3.2), we used this same (Cambridge C4.5) port mapper to determine the Berkeley class when Berkeley's IANA port mapper failed to make a decision, *i.e.*, when it assigns to "other-tcp". That is, we partitioned Berkeley's "other-tcp" class into "www", "bulk", "p2p", "database" and "mail", based on the C4.5 port mapper's decisions on "other-tcp" flows. In the same way, we partitioned Berkeley's "misc", "net-file", and "windows" classes. The complete consensus class definition mappings are shown in Figure 3.3. Note that the Cambridge classes map 1-to-1 to consensus classes, *i.e.*, no splitting or merging of Cambridge classes was needed, in mapping to consensus classes. The resulting confusion matrix (again, on the entire Berkeley dataset) is shown in Table 3.7 (given at the end of the article). Based on the consensus class definitions, consistency improved to (a much more reasonable) 94%. Though not perfect, as per our previous discussion on $\mathsf{P}[\text{error}|\text{Con}]$, 94% consistency is sufficiently high for a flow-classifier trained on the Cambridge dataset (*not* using ports as features) to be suitable for porting to (and accuracy evaluation on) the Berkeley domain.

Table 3.7. Berkeley dataset ground-truth confusion matrix after class split operations to achieve consensus types. (A: Augmented Berkeley Port Mapper and C: Cambridge Port Mapper)

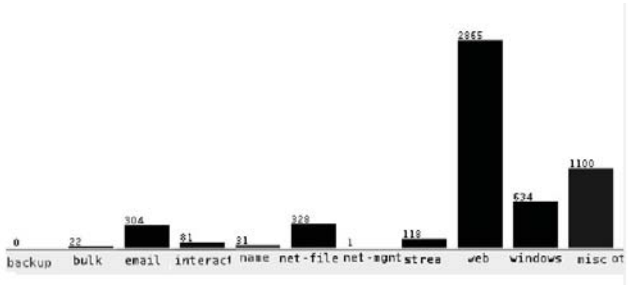
A \ C	www	mail	bulk	p2p	interactive	database	service
www	3066	0	3	11	0	0	0
mail	5	408	33	168	0	1	0
bulk	0	0	1346	0	0	0	0
p2p	0	0	0	1336	0	0	0
multimedia	88	0	0	30	0	0	0
interactive	0	0	7	10	64	0	0
database	0	0	0	0	0	42	0
service	15	1	0	10	0	5	51
admin	0	0	0	1	0	0	0

3.1.6 Ground truth labels

Generally, in a statistical classification setting, it is necessary to ascertain ground-truth class labels for training and test data. Obtaining highly accurate ground truth typically requires expensive, laborious, and time-consuming human expert labeling – the associated cost of ground-truthing grows linearly with the number of data samples. Moreover, even with such effort, mislabeling still occurs in practice (consider, for example, the frequency of medical misdiagnosis). Thus somewhat ironically, in practice, especially when working with *large* datasets, it may not be possible to guarantee that the ground-truth labels are pristinely accurate. While a best effort should be made to label samples as accurately as possible, at a minimum one should and must guarantee that the procedure used to ground-truth label training samples from the domain is the same as the procedure used to label test samples from the domain. When this is done, test set performance for the classifier trained using the domain's training set accurately characterizes generalization ability for the given classification domain. For our Berkeley-trained feature-based classifier, this is exactly the procedure that we followed – thus, test-set accuracy for this classifier accurately measures the classifier's generalization accuracy (which we will compare with accuracy of the classifier ported from the Cambridge domain).

As noted earlier, the class labels for the Cambridge trace came from a manual operation involving deep packet inspection. Neither packet payload nor packet headers were publicly disclosed. The 248 features and the class label per flow were disclosed. For the Berkeley trace, the anonymized raw packet header trace without payloads is publicly available; however, neither the packet payload nor class labels for each flow were provided. Since payloads were not given, it is not possible to use ground-truthing tools such as GTVS [3, 15], which work off of payload information. Fortunately, for this packet trace data, port numbers were unaffected by the anonymization process, with the exception involving the traffic associated with one particular port used for the internal security monitoring application. Our approach to obtain the ground-truth consensus

Figure 3.4. Berkeley TCP-subset application histogram



class labels for Berkeley flows, consistent with our procedure for defining consensus classes, was as follows. First, for the Berkeley packet trace data:

1. We extracted the destination/server and source/client port numbers for each Berkeley flow, and then applied the IANA port-application mapper to determine its class label.
2. If the decision by the above step was "other-tcp", then the Cambridge C4.5 port-based mapper was applied, and this decision was considered as the ground-truth consensus label.
3. For each flow with a Berkeley-type class label, we replaced the label by the corresponding consensus class label, as per Figure 3.3.

Since both steps 1) and 2) label flows based on port values, our approach effectively amounts to a port-to-application mapping rule for ground-truthing the Berkeley trace.

Confidence in the derived ground-truth labels for the Berkeley dataset is further enhanced by the Cambridge port-based classifier's 99.5% precision for the contemporary Cambridge trace recorded in a similar campus network context, *cf.*, Table 3.8.

The histogram of different application types for the TCP subset of the Berkeley flows used in our experiments is given in Figure 3.4 (note that Figure 1b in [23] includes non-TCP flows).

At this point, let us dispel any concern the reader may have that our process of deriving ground-truth labels for the Berkeley dataset introduces bias when we subsequently evaluate the accuracy of porting the Cambridge-trained *feature-based* classifier to the Berkeley domain. In particular, one may be concerned that in step 2) above, we used a Cambridge port-based mapper to label Berkeley flows that the IANA port-mapper essentially leaves unassigned. However, no such bias was introduced because the *feature-based* flow-classifiers evaluated in the following, whether trained on Berkeley or Cambridge datasets, do not employ any port information. That is, *no* information used to ground-truth label the Berkeley flows is subsequently used by the classifiers (to be evaluated) that seek to accurately predict the labels for these flows.

To better see the mapping and feature distribution under Berkeley (Berk.), Cambridge (Camb.) and consensus (Cons.) class definition, we will draw the feature distribution under three different class definitions. Figure 3.5 and figure 3.6 show the histogram of the minimum segment size in the first five packets, excluding the three-way handshake, if any, and the total

Figure 3.5. Histogram of feature 83, class-independent (Berk. on left, Camb. in middle and Cons. on right)

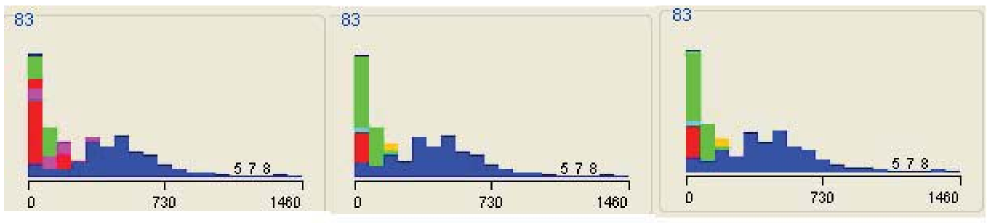
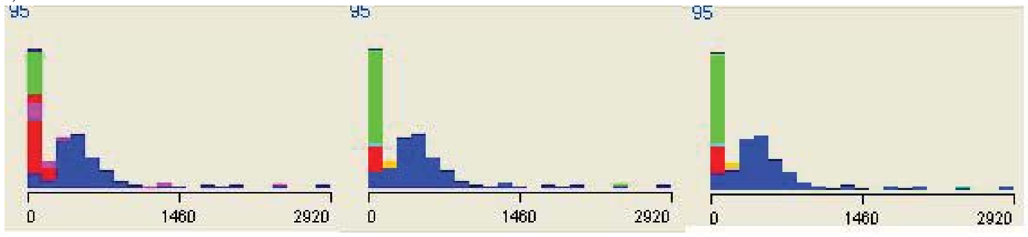


Figure 3.6. Histogram of feature 95, class independent (Berk. on left, Camb. in middle and Cons. on right)



number of bytes sent in the initial window from client to server respectively. As we can see, the feature distribution is considerably different between the class definition of Berkeley and that of Cambridge, but it's roughly the same under Cambridge and consensus class definition. The discrepancy of feature distribution between the Berkeley and Cambridge can be explained in the following way: recall we derived the ground-truth label on Berkeley through a two-stage phase, and there are still ”other-tcp”, which is in fact alias for “unknown traffic flow”, and these outlier flows mix with the normal flows, and influence the feature distribution. As for the similarity between the Cambridge and consensus class definition, we could see figure 3.3. Since the direct one-to-one mapping exists between the Cambridge and consensus class, it makes perfect sense for feature distributions to be the same under these two class definitions.

Figure 3.7. Histogram of feature 83, class-dependent (Berk. on left, Camb. in middle and Cons. on right)

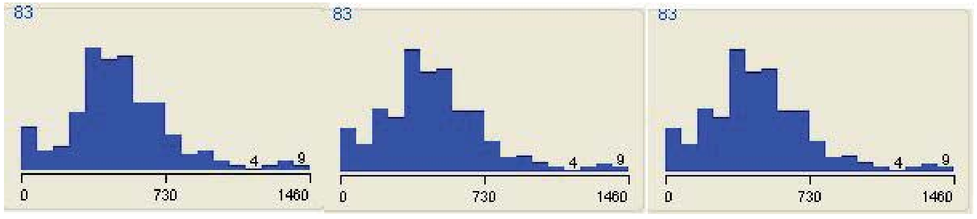


Table 3.8. Cambridge flow classifier's classification accuracy (from [13])

Classifier / Perf.	Naive Bayes kernel est.	C4.5 Decision tree	AdaBoost C4.5
Overall accuracy	92.38% ±0.35%	99.834% ±0.052%	99.816% ±0.057%

3.2 Classifier performance evaluation

For feature-based classification, we also used the C4.5 decision-tree framework, which is implemented in the open-source tool Weka [22]. Our other programs for feature generation, test set classifier evaluation and spoofing detection were implemented by us in Java. All of our experiments were run on an Intel Core Duo machine with four gigabytes of random-access memory.

We conducted a variety of experiments, as next detailed.

3.2.1 Experiments on the Cambridge traces and SiteB dataset

In [13], twelve features were used based on the first five packets from each flow, with the resulting classification accuracies, for several different classifiers, shown in Table 3.8. Among the features extracted, server port numbers and client port numbers *were* used to build the C4.5 decision tree; again, we note that [13] did consider their classifier's portability both temporally and to another (spatial) domain, which the authors referred to as SiteB.

Our first experiments investigated how discriminating these port features are working alone (and whether the remaining ten features used in [13] are contributing significantly to the classification accuracy when the two port features are also used). Accordingly, we built C4.5 decision trees treating only the two port numbers as features. For building a model on one day's traffic and testing on another day's, we used the *entire* day's trace for training (test). For building a model on one day's traffic and testing on the same day, we used two-fold cross validation. The total flows are 324,277 on Day1, 175,662 on Day2, 260,074 on Day3, and 248,362 flows at SiteB. The results for this port-only classifier, shown in Table 3.9, are very close to the accuracy achieved by the C4.5 tree from [13] based on all twelve features; this supports the use of this classifier to aid in the ground-truth labeling of the Berkeley trace as discussed in Section 3.1.6.

We also evaluated this same C4.5 decision tree classifier except based only on the remaining ten features, excluding the server and client port numbers. The performance is shown in Table 3.10 where we can see that the classifier based on these ten features still gives good accuracy (though not as accurate as when the ports are used). However, this accuracy may be more realistic in practice, given that use of port features will make the classifier non-robust to port spoofing (to be seen shortly). Note that the good accuracy in training on a Cambridge Day and testing on SiteB in Table 3.9 further validates and strengthens the results from [13], where it was also shown that porting to the SiteB domain gave good results (but, again, the classifiers in [13] used port numbers as features).

Table 3.9. Port-only classifier's performance on Cambridge dataset

Test \ Training	**Day1**	**Day2**	**Day3**	**SiteB**
Day1	99.70%	99.03%	99.05%	91.18%
Day2	98.80%	99.36%	90.73%	92.93%
Day3	97.87%	97.40%	99.89%	91.46%
SiteB	97.08%	95.44%	94.91%	99.42%

Table 3.10. Port-excluded, 10-feature classifier's performance on Cambridge data set

Test \ Training	**Day1**	**Day2**	**Day3**	**SiteB**
Day1	99.02%	94.27%	84.49%	89.44%
Day2	94.46%	98.59%	88.17%	88.79%
Day3	91.47%	88.82%	99.29%	90.04%
SiteB	88.28%	89.23%	93.90%	98.90%

Next, we investigated the removal both of the port information and other Cambridge flow features that are easily obfuscated and highly dependent on the TCP protocol, *e.g.*, the feature involving the TCP push bit. The performance of a decision-tree classifier based on the five retained (tamper-resistant) Cambridge features (as described in Section 3.1.3 above) is given in Table 3.11. Note that the accuracy of this classifier is quite comparable to that of the 10-feature classifier in Table 3.10. We also evaluated a "5 + 3" decision-tree classifier, which used the five retained Cambridge features and the three additional features described in Section 3.1.3. Evaluated on Day3 data, the precision was 99.74%, which was slightly better than the five retained Cambridge features' performance (99.45%), and a little worse than the Cambridge twelve-feature classifier performance (99.89%).

Finally, we evaluated the Cambridge twelve feature classifier's performance degradation in the presence of port spoofing. An experiment was set up with three different cases: (i) only the server port number spoofed, (ii) only the client port number spoofed, and (iii) both the server and client port numbers spoofed. We used a randomized but "range-preserving" spoofing strategy: if the port was in the well-known range (< 1024), then we replaced it with another port chosen at random from {1,2,...,1023}; similarly, for ports in the registered (1024-49151) and dynamic/private (49152-65535) ranges. The resulting accuracy of Cambridge's 12-feature classifier (which includes ports as features) was 31.8465%, 98.82% and 31.8642% for spoofing the

Table 3.11. C4.5 accuracy on the Cambridge dataset without port numbers and TCP protocol-dependent features, based on five retained Cambridge features

Test \ Training	**Day1**	**Day2**	**Day3**	**SiteB**
Day1	98.82%	92.32%	85.53%	87.62%
Day2	89.96%	98.55%	88.06%	88.04%
Day3	93.10%	92.44%	99.45%	89.66%
SiteB	90.42%	90.33%	93.08%	98.40%

server port, client port and both ports, respectively. As we can see, server-port spoofing has a dramatic, deleterious effect on the accuracy of a classifier that relies on ports as features. Server-port spoofing has a much greater effect than client-port spoofing, as the former is obviously much more predictive of application type (flow class). As an alternative type of spoofing, we also considered the case where ports are all simply mapped to port 80 (*i.e.*, the very common www class). The resulting accuracy under the three cases for this type of spoofing was 84.0544%, 99.813% and 84.0544%, respectively, *i.e.*, much higher than under randomized spoofing, simply because so much of the Cambridge traffic is in fact www traffic.

3.2.2 Porting experiments on the Berkeley trace

Calibration (*i.e.*, supervised training) of enterprise network intrusion detection systems comes at a very significant cost, as we discussed earlier. In this section, we consider the following basic question: Does calibration in one networking context give value/accuracy (and how much?) when the classifier is deployed in a somewhat different one? In Section 3.2.1, we evaluated porting accuracy to SiteB. Here, we evaluate accuracy in porting a Cambridge-designed classifier to the Berkeley domain. We compared the $5+3$ feature classifier trained on the Cambridge Day3 trace against a classifier using the same features but whose decision tree was instead trained on the Berkeley trace using the consensus ground-truth flow-labels, derived as discussed in Section 3.1.6. In our experiment, the $5+3$ classifier trained on the Cambridge dataset played the role of the ported classifier, while the $5+3$ classifier trained on the Berkeley dataset was, thus, custom-calibrated for the Berkeley dataset with accuracy evaluated using a two-fold cross-validation approach.

As reported in Table 3.12 (together with the 5 feature classifier's performance), the ported classifier's accuracy, $1 - \mathsf{P}[\text{error}]$, was only 50.25% and the conditional accuracy $1 - \mathsf{P}[\text{error}|\text{Con}]$ was 53.46%, while the custom-calibrated classifier's accuracy was 89.21%. Note that the conditional accuracy is higher than the unconditional accuracy, as expected. Evaluating $1-\mathsf{P}[\text{error}|\text{Con}]$ accounts for the lack of perfect consistency achieved by our use of consensus classes. Thus, the difference between the accuracy rates in columns two and three is only attributable to the following factors:

1. The (Cambridge) training set and (Berkeley) test set may have different class-conditional feature distributions.

2. There are significant differences in the class prior distributions as indicated by Table 3.1 and Figure 3.4.

3. Possibly, *higher* accuracy for the Berkeley-trained classifier on the subset of flows with inconsistent labeling at sites A and B. *At most*, this may account for an absolute accuracy difference of $1-c$, *i.e.*, 6% in this case. Since the difference in accuracy is much larger than this, the first two factors must be the primary sources of the performance difference.

Table 3.12. Cambridge-trained classifier vs. Berkeley-trained classifier performance on the Berkeley dataset with consensus labels

Classifier \ Train. Set	Camb.	Berk.	Camb.
	$1 - \mathsf{P}[e]$	$1 - \mathsf{P}[e]$	$1 - \mathsf{P}[e \mid \text{Con}]$
"5" feat. based	50.98%	85.48%	54.36%
"5+3" feat. based	50.25%	89.21%	53.46%

3.2.2.1 Error source analysis based on the confusion matrix

The results for the ported "5+3" classifier are shown in greater detail in the partial confusion matrix of Table 3.13 (given at the end of the article). The column indicates the decision counts for each consensus class, made by the Cambridge "5+3" feature-based classifier. The row indicates three ground-truth consensus classes *and*, to give more detail to the sources of error, the originating Berkeley class for the flow, *i.e.*, for consensus classes "www", "mail" and "bulk", we are indicating the *component* Berkeley classes consistent with Figure 3.3. As seen from the table, for a particular consensus flow class, there are very different error contributions from the Berkeley component classes that comprise this flow class. Note in particular that most of the "bulk/other-tcp" flows are assigned to the "mail" class by the Cambridge classifier. Recalling that the "other-tcp" Berkeley class consisted of the flows that could not be labeled by the IANA port mapper, these errors are perhaps more "forgiveable" than other error sources. We also note, however, the high error rates for the Berkeley "email" and "windows" classes, and also a significant number of errors for Berkeley's "web" traffic.

Table 3.13. Partial confusion Matrix of the Cambridge-trained classifier used to predict consensus labels on the Berkeley dataset (C/B: Consensus/Berkeley class and C5+3: Cambridge trained "5+3")

C/B \ C5+3	www	mail	bulk	p2p	voip	mmed	intact	db	service	attack
www/web	2046	179	214	106	52	0	1	0	238	29
www/other-tcp	45	49	20	2	1	0	0	0	14	0
www/misc	2	1	1	2	0	0	0	0	78	0
mail/email	93	106	20	34	0	0	1	0	39	11
mail/windows	123	82	5	29	3	0	1	0	8	53
mail/other-tcp	2	2	0	0	0	0	0	0	3	0
bulk/bulk	0	0	0	0	0	0	0	0	22	0
bulk/windows	0	0	0	1	0	0	0	0	0	0
bulk/other-tcp	40	482	8	24	0	0	0	0	35	1
bulk/misc	4	0	729	0	0	0	0	0	0	0

3.2.2.2 Differences in feature distributions per class

In order to investigate the first hypothesis on performance degradation for the ported classifier, we extracted class-conditional feature histograms from the Cambridge and Berkeley datasets, shown in Figures 3.8 and 3.9 for (the largest) consensus class "www" of Cambridge and Berkeley

Figure 3.8. Histogram of feature 83 for "www" class (Camb. on left)

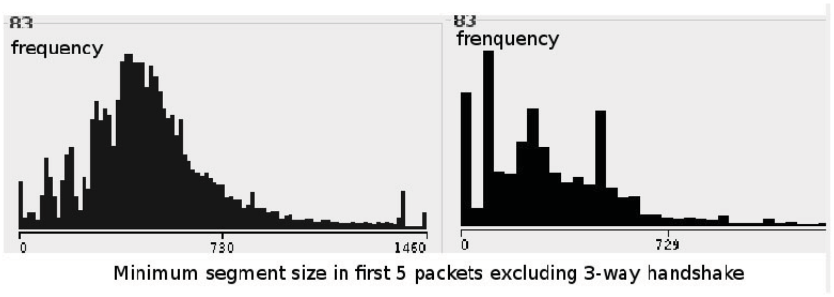
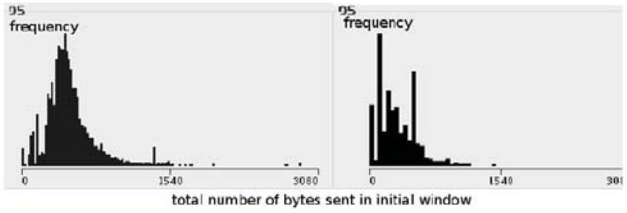


Figure 3.9. Histogram of feature 95 for "www" class (Camb. on left)



(the left graphs are based on Cambridge traffic, while right graphs are based on Berkeley traffic). In [17, 13], features were calculated based on whole lifetime observation of the flows. However, according to [14], the Cambridge dataset's features are based on the first five packets, excluding the 3-way handshake; we followed this same protocol in extracting features from the Berkeley dataset. Thus, feature 83 is the minimum segment-size among the first five packets (three-way handshake packets excluded) from client to server. Also, feature 95 is defined as the total number of bytes sent in the initial window, *i.e.*, the number of bytes sent in initial flight of data before receiving the first ACK packet from the other endpoint. For features 83 and 95 of the www class, we can see that the Cambridge and Berkeley histograms have similar range and form, and such similarities do provide a foundation for porting the Cambridge classifier to the Berkeley dataset, but even these histograms, which are in fact among the most *similar* between the two datasets, are significantly different, *e.g.*, in the histogram's mode. This suggests that differences in class-conditional histograms, along with differences in class priors (recall that www traffic dominates the Cambridge trace, while there is more class diversity in the Berkeley trace), are significant sources of performance degradation for the ported classifier.

3.2.2.3 Minimal impact of consensus class definitions

Finally, we evaluated performance of the Cambridge 5 feature-based and 5+3 feature-based classifiers on the Berkeley dataset, but in this case we used the Cambridge C4.5 port-mapper to directly derive ground-truth labels for the Berkeley flows, rather than using the consensus flow labels. Thus, in this experiment, the Cambridge classifier is used to predict Cambridge class labels for the Berkeley flows, rather than consensus class labels. Accordingly, a lack of perfect "consistency" is not a source of error for this experiment – the only error sources are differences in class priors and in class-conditional feature distributions, between the training (Cambridge) and test (Berkeley) domains. The results, shown in 3.14, are *very* similar to those of Table 3.12. This further confirms that differences in class priors and class-conditional feature distributions are the major sources of performance degradation not only in this experiment, but also in the above (primary) experiment where the classifier is being ported to predict the consensus labels.

Table 3.14. Cambridge-trained classifier vs. Berkeley-trained classifier performance on the Berkeley dataset with Cambridge port-mapper labels

Training Set / Classifier	Cambridge	Berkeley
"5" features based	52.66%	86.81%
"5+3" features based	49.68%	88.27%

3.2.3 Port spoofing detection on Cambridge

There are different types of port-spoofing detection strategies. Some are feasible on-line while the complexity of others require off-line implementation. An on-line approach may simply employ the measured port-purity of a cluster to detect port-spoofing of a flow classified to that cluster, *e.g.*, the newly classified flow may have a port number rarely seen in that cluster. Moreover, the *level* of detection may vary, with some methods detecting only flow subsets that *contain* flows with port spoofing without identifying the individual culprit flows, while others give more detailed detection of the individual flows. Here we propose an off-line method, based on the one proportion z-test (see, *e.g.*, [**?**]), that gives the former capability, narrowing port-spoofing to a subset of flows all classified in the same way by a pair of classifiers.

To implement the method, we first separately applied the Cambridge C4.5 port-based mapper and C4.5 "tamper-resistant" 5-feature flow classifier to the Cambridge *training* set. Treating the port-mapper result as ground truth, we obtain a training-set confusion matrix. This confusion matrix is used to formulate the null hypothesis that there is no port-number spoofing for the set of flows that correspond to each (i, j) confusion-matrix entry. Then, based on the classification results on the Cambridge test set from these same two classifiers, we obtain a second confusion matrix (here again, the output of a port-only decision tree gives a surrogate for the ground-truth class, since in practice this class will not be known on the test set).

Based on a specified statistical significance threshold, one can then evaluate a z-statistic to determine whether to accept or reject the hypothesis that the test set is not significantly different from the training set in the (i, j)-th entry of the confusion matrix; *i.e.*, whether the set of port numbers corresponding to flows meeting this definition (ground truth class i, but classified to class j by a feature-based decision tree) do not contain any spoofed flows.

More specifically, let p_0 and $\hat{p}$ be the fraction of type i flows (according to port-mapper) misclassified as j by the feature-based classifier in the training and test sets, respectively. For a

Table 3.15. Confusion matrix on the Cambridge training set (P: Port Mapper and F: Feature-based classifier)

P \ F	www	mail	bulk	chat	p2p	db	mmed	voip	service	intact
www	109166	58	29	0	54	0	0	0	3	0
mail	60	1833	76	0	9	3	0	4	0	4
bulk	54	52	2568	0	1	0	0	0	0	0
attack	12	2	0	0	0	0	0	0	4	0
chat	7	4	0	16	0	0	0	0	6	0
p2p	106	3	0	0	11034	0	0	0	0	0
db	4	4	0	0	0	4583	0	0	0	0
mmed	10	0	0	0	0	0	0	0	0	0
voip	9	6	0	3	1	0	0	27	0	1
service	22	2	0	0	0	0	0	0	11	0
intact	123	29	0	0	0	0	0	0	0	9

Table 3.16. Confusion matrix on the Cambridge test set before port spoofing(P: Port Mapper and F: Feature-based classifier)

P \ F	www	mail	bulk	chat	p2p	db	voip	service	intact
www	109143	66	29	0	71	1	0	0	0
mail	71	1849	52	0	11	0	2	0	4
bulk	55	72	2549	0	0	0	0	0	0
attack	15	2	0	0	0	0	0	0	0
chat	4	6	0	21	1	0	0	0	1
p2p	109	1	0	0	11034	0	0	0	0
db	6	5	0	0	0	4579	0	0	0
mmed	8	0	0	0	1	0	0	0	0
voip	12	4	0	2	0	1	27	0	0
service	27	2	0	0	0	0	0	6	0
intact	119	35	0	0	0	0	0	0	8

test sample-size N, we calculate the test z-statistic as

$$z = \frac{(\hat{p} - p_0)}{\sqrt{\frac{p_0(1-p_0)}{N}}}.$$

Finally, we compare $\Phi(z)$, the normal error function (erf) of z, with a chosen threshold (from 2% to 5%) to decide whether port-spoofing activity is detected in the test set, for the flows that are both port-mapped to i and classified to j.

Table 3.17. Confusion matrix on the test set of Cambridge data after port spoofing (P: Port-based mapper and F: Feature-based classifier)

P \ F	www	mail	bulk	chat	p2p	db	voip	service	intact
www	109569	2042	2630	23	11118	4581	29	6	13

Table 3.18. Spoofing detection results: TD (true detection) or MD (missed detection, P:Port-based mapper and F: Feature-based classifier)

P \ F	www	mail	bulk	chat	p2p	db	voip	service	intact
www	TD	TD	TD	MD	TD	TD	MD	MD	MD

We conducted a simulated port spoofing detection experiment on the Cambridge dataset in the following way. We obfuscated the packets' server port numbers by setting them *all* to port 80. Recall from the end of Section 3.2.1 that this approach resulted in higher classification accuracy compared to the randomized spoofing approach; so here we expect detection will be more difficult under port-80 spoofing (than under randomized spoofing). In order to detect the spoofing activity, we calculated the confusion matrix of the training set, shown in Table 3.15, of the test set before spoofing, shown in Table 3.16, and of the test set with spoofing, shown in Table 3.17. Then, according to the z-statistic definition we obtained the detection results of Tables 3.18 (all tables given at the end of the article) for a 5% threshold.

From Table 3.18, we see missed detections when the classes involved had fewer than 10^2 total flows (for a total sample size on the order of 10^5). But, for example, the detection algorithm found significant statistical difference in the percentage of misclassified "www/p2p" flows between the training set and spoofed test set, rejected the hypothesis that there was no spoofing activity between this pair, and thus detected "p2p" flows impersonating "www" flows. Likewise, it correctly detected that "mail", "bulk", and "db" flows impersonated "www" flows.

Chapter 4

Conclusion and Future Work

4.1 Conclusions

In this thesis, we developed and evaluated TCP flow classifiers that are not based on payload information, port numbers, or highly TCP protocol-dependent features. A primary motivation for our study was that these features are often legally unavailable, infeasible to process online in detail, or are easily obfuscated by the end-user, *e.g.*, via payload encryption and port spoofing. To address the latter problem, we proposed and experimentally evaluated a hypothesis testing approach. Finally and most importantly, we addressed fundamental issues associated with porting a classifier from one domain to another. We then investigated the performance of our "tamper-resistant" flow classifier, trained in one networking context/domain (Cambridge) and then deployed in a different one (Berkeley). It was found that classification accuracy was highly degraded in the new domain, with the sources of this performance loss identified. While supervised classifier training specific to the new domain results in much better classification accuracy, in many practical settings it may be too laborious or expensive to obtain the ground-truth labels needed for such training.

4.2 Future work

We demonstrated the efficiency of classifier porting from one domain to another, and showed how to reach a consensus class definition through class merging and splitting based on the confusion matrix. In the future, we can further improve this classifier porting scheme. More specifically, we will investigate the porting with potentially different class-conditional feature distributions. Even more specifically, the new approach presumes that the class-conditional feature distribution are multimodal, and that there is underlying unknown affine transformation that relates features given the class and mixture component of origin. We believe this new porting approach will be superior to the traditional classifier porting without considering the class-conditional feature distribution difference.

Bibliography

[1] A.W.Moore and K. Papagiannaki. Toward the accurate identification of network applications. In *Passive and Active Measurements Workshop*, 2005.

[2] Cambridge Brasil Group. Characterizing network-based applications. `http://www.cl.cam.ac.uk/research/srg/netos/brasil/data/index.html`.

[3] Cambridge Brasil Group. Ground truth verification system GTVS. `http://www.cl.cam.ac.uk/research/srg/netos/brasil/gtvs/index.html`.

[4] Cisco System Inc. CSS administration guide (software version 6.10) - configuring flow and port mapping parameters. `http://www.cisco.com/en/US/docs/app_ntwk_services/data_center_app_services/css11000series/v6.10/configuration/administration/guide/Flow.html#wp1009071`.

[5] N. Duffield, P. Haffner, E. Krishnamurthy, and H. Ringberg. Rule-based anomaly detection on IP flows, 2009.

[6] E. Sommer and M. Strait. Application layer packet classifier for linux. `http://l7-filter.clearfoundation.com/`.

[7] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proc. ACM SIGCOMM, Karlsruhe, Germany*, 2003.

[8] A. Este, F. Gringoli, and L. Salgarelli. Support vector machines for TCP traffic classification. *Computer Networks*, 53(14):2476–2490, 2009.

[9] IANA. Internet assigned numbers authority. `http://www.iana.org/assignments/port-numbers`.

[10] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31(3):249–268, 2007.

[11] Lawrence Berkeley National Laboratory. Bro Intrusion Detection System-Bro Overview. `http://www.bro-ids.org/`.

[12] Lawrence Berkeley National Laboratory and ICSI. LBNL/ICSI Enterprise Tracing Project. `http://www.icir.org/enterprise-tracing/`.

[13] W. Li, M. Canini, A. Moore, and R. Bolla. Efficient application identification and the temporal and spatial stability of classification schema. *Comput. Netw.*, 53(6), 2009.

[14] W. Li and A. Moore. A machine learning approach for efficient traffic classification. In *Proc. IEEE MASCOTS*, 2007.

[15] A. M. M. Canini, W. Li and R. Bolla. GTVS: Boosting the collection of application traffic ground truth. In Proc. First International Workshop on Traffic Monitoring and Analysis (TMA), Aachen, Germany*, year = 2009*.

[16] A. Madhukar and C.L.Willianson. A longitudinal study of p2p traffic classification. In *MASCOTS*, 2006.

[17] A. Moore, M. Crogan, D. Zuev, and M. Crogan. Discriminators for use in flow-based classification. Technical Report RR-05-13, Dept. of Computer Science, Queen Mary, University of London, 2005.

[18] A. Moore, J. Hall, C. Kreibich, E. Harris, and I. Pratt. Architecture of a network monitor. In *Proc. Passive and Active Measurement Workshop*, 2003.

[19] A. Moore and D. Zuev. Internet traffic classification using Bayesian analysis techniques. In *Proc. ACM SIGMETRICS, Banff, Alberta, Canada*, 2005.

[20] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proc. IEEE INFOCOM*, 2003.

[21] A. I. C. of Research Excellence Networks and W. Communications. Peer-to-peer traffic classification. In *Intelligent Software Systems*, 2006.

[22] T. U. of Waikato. Weka 3 - data mining with open source machine learning software in java. `http://www.cs.waikato.ac.nz/ml/weka/`.

[23] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *Proc. ACM IMC, Berkeley, CA*, 2005.

[24] J. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[25] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, Jan. 1993.

[26] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *Proc. ACM IMC, Taormina, Italy*, 2004.

[27] Sourcefire Inc. Snort. `http://www.snort.org/`.

[28] N. B. T. Karagiannis, A.Broido and K.Claffy. Is p2p dying or just hiding? In *IEEE Globecom*, 2004.

[29] utorrent. http://utorrent.en.softonic.com/.

[30] J. Wang, I. Hamadeh, G. Kesidis, and D. Miller. Polymorphic worm detection and defense: system design, experimental methodology, and data resources. In *Proc. ACM LSAD Workshop, Pisa, Italy*, 2006.

[31] J. Wang, D. Miller, and G. Kesidis. Efficient mining of the multidimensional traffic cluster hierarchy for digesting, visualization, and modeling. *IEEE JSAC special issue on High-Speed Network Security*, 2006.

[32] Wireshark. Wireshark.go deep. `http://www.wireshark.org/`.