Controlling False Alarm/Discovery Rates in Online Internet Traffic Classification

Daniel Nechay

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

November 2009

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Masters of Engineering.

Abstract

Classifying Internet traffic flows online into applications or broader classes without inspecting the packet payloads or without relying on port numbers has become a necessity for network operators. The operators can use this information to monitor their networks and provide per-class quality of service. There has been a great deal of research done on Internet traffic classification recently and numerous techniques have been proposed. While the current techniques can obtain a high accuracy classifying Internet traffic, providing performance guarantees for particular classes of interest has never been addressed. In this thesis, we provide two novel types of online Internet traffic classifiers that can provide performance guarantees on the false alarm and false discovery rates, respectively. These guarantees can be for an entire class (class-wise) or between two classes (pair-wise). Controlling false alarm rates is well-suited for application prioritization (i.e. prioritizing time-sensitive applications like VoIP over HTTP) whereas controlling false discovery rates is better suited for blocking or rate-limiting a targeted class of traffic (i.e. Peer-to-Peer). The classifier that provides false alarm rate guarantees is based on a Neyman-Pearson classification framework while the classifier that provides false discovery rate guarantees is based on the Learning to Satisfy (LSAT) framework. Both of these classifiers are implemented using a machine learning technique, namely, the 2-nu Support Vector Machine (SVM). Moreover, all previous work done with these two statistical methodologies focused on binary classification only; we extend these statistical methodologies to a multi-class setting. In addition to the regular application classification problem, we also present preliminary work on a binary LSAT classifier that can detect, after the reception of only a handful of packets, whether a flow will be large, as defined by a network operator. This large flow detector can act as a preprocessor for regular application classifiers. By allowing only large flows to pass to the classifier, this allows the classifier to focus on the more resource-intensive flows. We validated our Internet traffic classifiers by testing our approaches using data provided by an ISP.

Abrégé

Identifier l'application (ou autre classe plus générale) qui génère un flux de trafic Internet, sans compter sur le numéro du port ou inspecter la charge des paquets, est devenu une nécessité pour les opérateurs de réseau. Les opérateurs peuvent utiliser cette information pour surveiller leurs réseaux et fournir une qualité de service propre à chaque classe. Il y a eu beaucoup de travaux de recherche portant sur la classification du trafic Internet effectué récemment et de nombreuses techniques ont été proposées. Bien que les techniques actuelles puissent obtenir une grande précision pour classer le trafic Internet, offrir des garanties de performance pour des catégories particulières est un problème encore inexploré.

Dans ce mémoire, nous proposons deux nouvelles techniques de classement du trafic Internet en ligne capables de fournir des garanties de performance sur le taux de fausses alarmes et le taux de fausses découvertes, respectivement. Ces garanties peuvent être sur une classe entière (class-wise) ou entre deux classes (pair-wise). Contrôler le taux de fausses alertes est bien adapté à prioriser les applications (VoIP via HTTP), tandis que le contrôle des taux de fausses découvertes est mieux adapté pour bloquer ou limiter le débit de certaines classes (Peer-to-Peer). Le classificateur qui fournit les garanties de taux de fausses alarmes est basé sur le cadre de classification Neyman-Pearson, tandis que le classificateur qui fournit les garanties sur le taux de fausses découvertes est basé sur le cadre Learning to Satisfy (LSAT). Ces deux classificateurs sont mis en oeuvre en utilisant une technique d'apprentissage automatique nommé 2-nu SVM (Support Vector Machine). De plus, tous les travaux antérieurs réalisés avec ces deux méthodes statistiques mettaient l'accent sur la classification binaire uniquement, nous les adaptons à un environnement supportant plusieurs classes.

En plus de l'application au problème de classification, nous présentons aussi des travaux préliminaires sur un classificateur binaire LSAT qui permet de détecter, à l'aide de seulement quelques paquets, si un flux de trafic sera important, selon l'opérateur de réseau. Ce détecteur de flux de trafic importants peut agir comme un préprocesseur pour l'application de classification. En ne permettant que d'importants flux de se rendre au processus de classification, le classificateur peut se concentrer sur les flux qui utilisent plus de ressources.

Nous validons notre classificateur de trafic Internet par des essais en utilisant des données fournies par un FAI.

Acknowledgments

First and foremost, I would like to thank my supervisor, Mark Coates, for all his knowledge, insight and patience for the duration of my thesis. I also would like to thank Yvan Pointurier for his expertise in helping me set up my experiments. I would like to thank OmniGlobe Networks for giving me the inspiration for my thesis and for providing me with data to use in my experiments. I would also like to thank Bradford Stimpson, Frederic Thouin and Laura Leong for taking the time in reviewing my thesis and providing valuable feedback. Finally, I would like to thank my family for their love and support.

Contents

List of Figures

List of Tables

List of Acronyms

ISP	Internet Service Provider
SVM	Support Vector Machine
LSAT	Learning to Satisfy
QoS	Quality of Service
FAR	False Alarm Rate
FDR	False Discovery Rate

Chapter 1

Introduction

As the number of applications on the Internet is increasing, Internet traffic classifiers are becoming an important tool for Internet Service Providers (ISPs) to help monitor their networks. Internet traffic classification involves the automatic association of a user-defined class to a traffic flow. Online Internet traffic classification can be an especially powerful tool for ISPs as they can, in real time, use the classification results to provide different levels of service to various applications (a form of Quality of Service; QoS). To provide a better overall level of service, ISPs can decide, based on their classification results, to prioritize time-sensitive flows (VoIP, video-conferencing, etc.) and to throttle bandwidth-intensive applications like Peer-to-Peer file sharing, which are known to consume a large portion of the network's resources [1] and therefore degrade the overall performance for everyone on the network. The classification results can also be used for network security and network provisioning.

Classifying Internet traffic is not a trivial task. In the Internet's earlier days one could have compared the port numbers of traffic flows to the port numbers assigned by the Internet Assigned Numbers Authority (IANA) [2] to determine what type of application it is (an example is the common use of port 80 for HTTP traffic). The problem with classifying Internet traffic exclusively with port numbers is that Internet developers are not bound to follow the port number mapping by IANA and can choose whichever port they wish for their application. This can lead to security risks as malicious or unwanted traffic could try to pass through trusted ports. Therefore, additional measures need to be taken in order to accurately classify Internet traffic. In this thesis, we propose two online Internet

traffic classifiers with the added constraints that certain performance guarantees must be met for classes of interest. We also propose a large flow detector that predicts whether an Internet traffic flow will become a "large" flow or not (with "large" being defined by the user) based on just the first few packets of a flow. We implement this "large" flow detector as a pre-processor for the Internet traffic classifiers.

1.1 Motivation

In this thesis, we propose two online Internet traffic classifiers. What makes these two classifiers different from existing classifiers that have been proposed is that we set hard performance guarantees for the classifier to adhere to. While current Internet traffic classifiers that have been proposed can classify traffic with little error, none have provided hard performance guarantees [3]. The two performance guarantees that we focus on are the False Alarm Rate (FAR) and the False Discovery Rate (FDR). Our classifier that controls the FAR is based on a multi-class generalization of Neyman-Pearson (NP) classification [4–8] while our classifier that controls the FDR is based on the Learning to Satisfy framework [9]. These performance guarantees can be applied to a single class against all the other classes or between two specific classes.

The false alarm rate for class i (FAR_i) refers to the expected fraction of the flows that do not belong to traffic class i that are incorrectly classified as belonging to i. This concept can be extended to a pairwise false alarm rate FAR_{ij} for classes i and j, which specifies the expected proportion of the flows belonging to class j that are incorrectly labeled as class i by the classifier. The false alarm rate differs from the false discovery rate, FDR_i, which is the expected fraction of incorrectly classified flows among all traffic flows classified as class i. This can also be extended to a pairwise false discovery rate, FDR_{ij}, which is the expected fraction of all flows classified as class i which do in fact belong to class j.

The reason we propose using a classifier with performance guarantees is that while the overall accuracy of the classifier is important, sometimes it is more critical to focus on a particular application. This is while still maximizing the overall accuracy for the classifier. The following example illustrates this difference. Consider a network that has 1100 flows which are broken down as follows: 900 HTTP flow, 100 VoIP flows and 100 P2P flows. The network administrator wishes to prioritize all VoIP traffic and limit (or block) the P2P traffic. In this network, HTTP is the dominant application so the overall accuracy of

an Internet traffic classifier will be biased towards how many flows of HTTP the classifier classifies correctly (classifying all flows as HTTP still gives an overall accuracy of over 80%). While the overall accuracy is still high in this case, for the two classes of interest to the administrator, the accuracy is 0%. That is why having a classifier with performance guarantees (either class-wise or pairwise) is more beneficial for a situation like this.

The classifier with FAR constraints is better suited for dealing with prioritizing traffic (in this case, VoIP) as the FAR constraint limits the number of non-prioritized traffic being classified as the prioritized traffic. This allows the administrator to save on network resources as he does not have to allocate additional resources to non-prioritized traffic. If the administrator sets the FAR_{VoIP} at 5%, there would be at most 50 non-VoIP flows classified as VoIP. Also, by setting the $FAR_{HTTP,VoIP}$ and $FAR_{P2P,VoIP}$ at 5%, this ensures that at most 10 VoIP flows are misclassified. Table 1.1 shows some examples of possible FARs for this network and their impact on the network.

Table 1.1 Example FARs for a network with 900 HTTP flows, 100 VoIP flows and 100 P2P flows

FAR_i or FAR_{ij}	Maximum threshold for FAR (%)	Maximum allowable number of flows misclassified as class i
FAR_{HTTP}	5	10 ($= (100 + 100) \times 0.05$)
FAR_{VoIP}	5	50
FAR_{P2P}	10	100
$FAR_{HTTP,VoIP}$	5	5
$FAR_{P2P,VoIP}$	5	5
$FAR_{P2P,HTTP}$	10	90

The classifier with FDR constraints is better suited for dealing with limiting/blocking traffic (in this case, P2P) because if a drastic action like blocking the traffic is taken, the goal should be to minimize the number of flows that are incorrectly blocked. In our example, setting the FDR_{P2P} at 1%, means that for every 99 flows that are correctly identified as P2P, there would be a maximum of 1 flow misclassified as P2P.

We also propose using a "large" flow detector as a pre-processor as this allows the classifiers to focus on the more important flows. The reason is that on high-speed routers, the routers are dealing with a high volume of traffic that could possibly overload the classifiers. Therefore by having a "large" flow detector act as a pre-processor for the classifiers, we allow the classifier to focus on the flows that are likely to consume most of

the network resources. Addressing these flows will have the greatest impact on network performance. For example, if the user wishes to prioritize traffic, he/she needs to focus on flows that would stay on the network for a while after the classification is made rather than flows that would end a short time after the classification is made.

1.2 Thesis Problem Statement

Since this thesis proposes two new online Internet traffic classifiers and a large flow detector, there are three problem statements that we address. For the classifier with FAR constraints, the goal is to minimize the overall misclassification rate while still meeting all the FAR constraints set. Similarly, the goal for the classifier with FDR constraints is to minimize the overall misclassification rate while still meeting all the FDR constraints set. Finally, for the large flow detector, after the user defines what a large flow is, the classifier maximizes the number of large flows classified correctly while minimizing the number of small flows that are classified as large. This is done so the classifier does not have to waste time classifying these small flows.

1.3 Thesis Contribution and Organization

This thesis is divided into six chapters. In **Chapter 2**, we review the existing techniques used to for Internet traffic classification. We separate the techniques proposed into three main categories (port-based classification, deep packet inspection and shallow packet inspection) and elaborate on the methods and the results obtained in the papers.

In **Chapter 3**, we provide background for the algorithms that we use. Since all of our algorithms are based on Support Vector Machines (SVMs) we first provide an introduction on what a SVM is and how it is formulated. The Internet traffic classifier with FAR constraints is implemented using a 2ν-SVM so we show how a SVM is transformed into a 2ν-SVM. The Internet traffic classifier with FDR constraints and the large flow detector are implemented using the Learning to Satisfy (LSAT) framework. We first show how the LSAT framework is formulated. This algorithm is implemented using the 2ν-SVM and we show how 2ν-SVM can be extended to implement LSAT.

In **Chapter 4**, we give a detailed description of the two Internet traffic classifiers and the large flow detector that we propose. We first formalize the three problems we are

trying to solve — Internet traffic classifier with FAR constraints, Internet classifier with FDR constraints and a large flow detector. Then we discuss how the algorithms we selected (2ν-SVM and LSAT) can be used to solve our problems. This includes for the two Internet traffic classification algorithms showing how we transformed the binary classifiers of 2ν-SVM and LSAT to handle our multi-class setting.

In **Chapter 5**, we tested our proposed algorithms on Internet traces provided by a Canadian ISP and provide the results. We examine the steps taken to process the trace so the data can be used by our algorithms. Also discussed is how all the parameters were selected when running our experiments. Finally we show interesting results for the Internet traffic classifiers from the data trace.

In **Chapter 6**, we summarize our work and discuss in more detail the results presented in the previous chapters and conclude with proposed future work.

1.4 Published Work

Our proposed algorithms that were discussed in Chapter 4 and the accompanying results from Chapter 5 have been published and were presented at the IEEE Conference on Computer Communications (IEEE INFOCOM).

- D. Nechay, Y. Pointurier and M.J. Coates, Controlling False Alarm/Discovery Rates in Online Internet Traffic Flow Classification, in Proc. IEEE INFOCOM, Rio de Janeiro, Brazil, April 2009. [9 pages]

Chapter 2

Literature Review

In the past decade, there has been significant research addressing Internet traffic classification. One of the biggest reasons for this is that Internet Service Providers (ISPs) are more interested in exactly what kind of traffic is passing through their routers. By knowing what kind of traffic is passing through their network the ISPs are able to perform an appropriate action depending on their policy. A common example is providing Quality of Service (QoS) for certain applications. If the ISP is able to identify time-sensitive applications (i.e. VoIP, video conferencing) they can then prioritize this traffic to improve its performance. For example, latency can be improved by assigning more bandwidth to these time-sensitive applications if they can be identified. Other uses for Internet traffic classification include network provisioning and security measures.

Current Internet traffic classification techniques can be grouped into three main categories: port-based, deep-packet inspection and shallow-packet inspection. Port-based classification is the simplest approach in which classification is done simply based on which port the application is using. For example, if the application is using Transmission Control Protocol (TCP) port 80 then it is classified as HTTP. Deep-packet inspection consists of looking inside the payload of a TCP packet to find a signature to match to an application. Finally, shallow-packet inspection (such as our work) involves applying machine learning techniques (clustering, decision trees, Bayesian learning) to statistics derived from the packet headers of the TCP packet to classify traffic. The following subsections of this chapter explain these three categories in more detail. Also note that all classifiers discussed deal with classifying TCP traffic flows only unless otherwise stated. Therefore, packet clas-

sifiers [10, 11] (classifiers that attempt to classify every packet rather than every flow) are out of the scope of this literature review. We also do not discuss anomaly detectors as this is a different subset of Internet traffic classification than what we are pursuing [12–15].

2.1 Port-based Classification

Port-based classification is one of the simplest forms of Internet traffic classification. An incoming flow is classified based on the port number it is using and classification is done by finding which application is matched to this port number on the Internet Assigned Numbers Authority's (IANA) [2] known port numbers list. CoralReef [16] is a software suite that can provide port-based classification. The port number of a TCP flow can be found by looking at the header of the SYN packet, which is the first packet sent in the TCP handshake communication. Common examples include TCP port 80 for HTTP, TCP port 25 for SMTP and TCP port 443 for HTTPS. The drawback to this type of classification is that this mapping by IANA is just a suggestion and developers do not have to follow it. As a result, applications that do not wish to be identified on the network generally do not follow this mapping. These applications can (i) choose their port numbers dynamically by changing the port number every time the application runs, (ii) employ port hopping (if the port the application is using is blocked, attempt to find another an unblocked port), (iii) use a common port to camouflage their application. For example, Skype (a popular VoIP program) is an application that is known to use port 80 to try to disguise itself as HTTP traffic if the other ports it commonly uses are blocked [17].

One of the biggest reasons that port-based classification has become ineffective is the increased usage of Peer-to-Peer (P2P) applications. In 2004, Karagiannis et al. [1] showed that the volume of P2P in networks is not decreasing as previously thought [18, 19] but maintaining its presence on the network or in some cases increasing significantly. Some thought that P2P usage was decreasing but Karagiannis et al. found it was still present but it was disguising itself using the techniques mentioned above. In 2006, Madhukar et al. [20] believed that 30-70% of all traffic from their data set (from the University of Calgary network) was P2P. As a result, port-based classification on their network was useless; at least 30-70% of all traffic would be misclassified because most P2P applications nowadays use dynamic port numbering. Studies done by Sen et al. [21] and Moore et al. [22] also demonstrated that port-based classification is ineffective. Sen et al. examined

various P2P protocols and found that three popular P2P protocols (KaZaA, Gnutella and DirectConnect) use anywhere between 35-70% of non-standard ports which could not be classified by IANA's mapping. Moore et al. found that 30% of all bytes collected from their campus network could not be classified using IANA's mapping. Due to these limitations of port-based classification, more complex classifiers are necessary to accurately identify Internet applications.

2.2 Deep-Packet Inspection

Deep-Packet Inspection (DPI) searches the payload of a TCP packet to find a signature to map to an application class. Most commercial Internet traffic classifiers use some sort of DPI to classify traffic [23–25]. DPI is also popular for intrusion detection systems [12,26,27], which look for harmful or malicious traffic trying to enter the network.

In [21], Sen et al. looked to find signatures for five popular P2P applications (Gnutella, eDonkey, DirectConnect, KaZaA and BitTorrent). Each P2P application has its own specific protocol it uses to communicate and Sen et al. looked to find a signature on this protocol in the payload of a single TCP packet (most of their signatures were found in the HTTP request header). They did not look for signatures that spanned multiple packets as they chose to ignore these as a tradeoff for performance. The signatures they propose can be either fixed length or variable length. Their results showed that their approach was much more effective than port-based classification in classifying P2P.

Karagiannis et al. [28] also examined signatures for popular P2P applications but they used this just as a ground truth for the non-payload approach they developed in their research. Their non-payload approach will be discussed in the next section. Karagiannis et al. search the payload for bit strings to identify the different P2P flows. In addition to this they also keep track of the destination/source IP addresses and port numbers as they use the past history of these addresses and port numbers to aid in classifying their flows.

In [29], Haffner et al. provide a method to automate the construction of application signatures. Only the first N bytes of the flow were used as input for the classifier (for their paper N is between 64 and 256). They experimented with three different statistical machine learning techniques as the classifier: Naïve Bayes, AdaBoost and Regularized Maximum Entropy. They were able to classify over 99% of all applications correctly with their approach and showed that they could still use their signatures months later.

Ma et al. [30] also attempted to construct a signature based on the first 64 bytes of a traffic flow. In these bytes, they were interested in the statistical and structural aspects of the messages exchanged in the protocol of an application. They experimented with three classifiers: product distributions, Markov models and common substring graphs. Using data collected from their campus network they showed that all three classifiers were able to correctly classify between 90-98% of network traffic.

In [22], Moore et al. use a combination of port-based classification and signatures to classify traffic. They propose a nine-stage classifier for classifying traffic flows. The first stage classifies the flow based on the port number it uses. The classifier then verifies the classification and if it is confident with the classification, it skips the other stages and uses this classification. If it is not confident with this result the classifier proceeds to the second stage which looks at the packet headers. This process continues iteratively through the final seven stages, until the classifier is confident in its prediction. Stages 3-8 of the classifier involve trying to identify a signature inside of the flow. In each stage, the search for a signature becomes more complex from searching just the first packet to searching the entire flow for a signature. The final stage of the classifier keeps a record of the combinations of IP addresses and port numbers and the applications associated with them. The classifier then uses these records to try a find a match for classifying the flow. Classification accuracy approached 100% in the later stages of this classifier.

Choi et al. [31] also use a combination of port numbers and signatures to classify traffic. The goal of their classifier is to monitor the usage of all the users on a network (if an ISP switches its billing system to usage-based accounting). Their approach states that there are four main types of Internet flows: Type FP (Fixed Port-based), Type PI (Payload Inspection-based), Type DP (Dynamic Port-based) and Type RR (Reverse Reference-based). Type FP are applications that always use a fixed port (i.e. FTP or SMTP) and can be classified using port classification. Type PI are applications that can share a port number with other applications so payload analysis is required to determine the application. Type DP are applications that use dynamic port numbers. Payload inspection is done on flows that share similar flow characteristics to the Type DP flows and already have already been classified and this classification is used to identify the flow the Type DP flow. Type RR are generally TCP control flows. If a flow from host X to host Y has already been identified, there may be a reverse flow between host Y to host X (with the source/destination IP addresses and port numbers reversed with the previous flow).

To find a signature inside the packet, Choi et al. propose the Application Configuration Recognition Language (ACRL). The syntax of ACRL is divided into 3 levels. The top level is the actual application. The next hierarchy is port numbers that this application uses. The last level is based on signatures in the payload for this application. This last level can be very specific as which packet(s) and where inside of these packet(s) to search for the signature can be specified. They implemented their approach into WiseTrafView and tested it on campus and enterprise networks. With their approach they found that not all applications that use port 80 are HTTP.

Another classifier that uses a hybrid approach is the one proposed by Won et al. [32]. The first stage of their classifier looks for signatures inside the payload. Won et al. use the Karp-Rabin algorithm [33] for the string matching inside the payload. If the classifier cannot find a signature for the flow, the classifier then examines the behaviorial aspects of the flow to try to identify the flow. This includes looking at the IP address and port number pair and verifying if any applications have already used that IP address and port number combination. If so, these flows are probably the same application. Other behaviorial-based classifiers will be discussed in the next section (Section 2.3.4). To validate their approach they used synthetic and real traffic traces. For the synthetic traces, they collected a single application per trace. On these traces they were able to obtain 99% accuracy so they were confident that their classifier was able to distinguish between the different applications. Then they tested their classifier on a real traffic mix. The real traffic was collected from their campus network. They also apply the Flow Relationship Map (FRM) [34] on the real traffic as a comparison. They found that their approach produced more unknown flows than FRM but was able to have a higher accuracy than FRM.

While all the aforementioned methods can classify traffic confidently, there are drawbacks to DPI. The main problem is that if the flow is encrypted this approach is rendered useless as it cannot look inside the packets. P2P applications are starting to encrypt their traffic and any traffic behind a VPN connection can not be identified as well. Another drawback is that even slight changes to a protocol could require a change for the signature. This means signatures need to be monitored so that they are kept up-to-date, a time consuming but necessary task. Designers also need to consider that there may be multiple signatures inside of a payload. For example, P2P applications are known to run over HTTP so a P2P and HTTP signature could be found in the payload. Payload inspection is also computationally expensive and requires a lot of overhead. Finally, DPI is in a gray area

in regards to legal and privacy concerns. Arguments have been made that it is illegal for ISPs to look at what users are sending over the network as it is an invasion of their privacy. For these reasons, Shallow-Packet Inspection (SPI) has become a popular alternative to DPI. SPI does not require any information from the payload and derives its information from either the behavior of a flow or statistics collected from a flow's packet header. The following section elaborates on all the techniques used in SPI.

2.3 Shallow Packet Inspection

Claffy [35] was one of the first people to show that Internet applications can be separated by their statistical properties. Other work [36–39] has also shown that statistical properties of a flow can be mapped to an application class. Recent research (known as Shallow-Packet Inspection (SPI)) has focused on developing classifiers that can classify traffic flows based on the statistical properties of a flow. From information derived just from the packet headers (without inspecting the payload), Moore et al. [40] were able to find over 200 statistics related to a flow. Note that this method is not an intrusive method like DPI. Rather, all the statistics can be collected just from observing the flow on a network (a program like NetFlow [41] could be used to collect information about the flow) and does not involve looking at exactly what is being sent. More generally, we define SPI as any classifier that does not need to look at the payload to classify the traffic or that looks at the behavioural aspects of a traffic flow. When using SPI, two important decisions need to be made — which of the available statistics are relevant (generally done using a feature selection method) and what kind of classifier to use. This section discusses all the classifiers that have been used to classify Internet traffic using SPI.

2.3.1 Feature Selection

When working with a classifier, one of the most important things is to make sure relevant inputs are being sent to the classifier. If irrelevant or redundant features are being sent, this can lead to a decrease in accuracy (as the classifier can become biased), an increase in the build time of the classifier and a decrease in classification speed (as the dimensionality of the input space is increased). Feature selection algorithms can be divided into two main categories — filter methods or wrapper methods. Filter methods performs the feature selection based on the characteristics of the feature themselves and can be used with any

classifier. On the other hand, wrapper methods find the best set of features for a specific algorithm.

The WEKA toolbox [42] provides many different types of filter methods. Examples of filter methods are consistency-based feature selection [43] and correlation-based feature selection [44]. Consistency-based feature selection tries to find the optimal subset of features that performs consistently as well as the full feature set. Correlation-based feature selection looks at the inter-correlation between the features then finds the optimal subset of features. The optimal subset of features excludes the redundant features that are not relevant for classification. Williams et al. compare these two methods extensively in [45].

The alternative is to use wrapper methods which can provide a higher accuracy than filter methods since they are tailored for a specific classifier. The drawback is that over-fitting can occur. Over-fitting is when the classifier is trained to perform well on just the training set and performs poorly when new points are introduced. Also, this approach is more computationally expensive as the classifier needs to be trained and tested for each subset of features. An example of a wrapper method is the backward greedy feature selection used in [46]. This method starts by training and testing the classifier with n features. Then the algorithm looks to find the best $n-1$ features that results in the highest accuracy for the classifier. This process is repeated until the optimal subset of features is found (i.e. the subset that gives the highest accuracy).

2.3.2 Clustering

Clustering is an unsupervised method in which the clustering algorithm simply groups similar data points together without labeling them. A heuristic then needs to be applied to map the cluster to a particular class. The simplest heuristic is to map a cluster to the dominant class inside of the cluster (assuming the training set has data points which have already been mapped to classes). Clustering is a logical choice for classifying Internet traffic as it has been shown that traffic flows that share similar traffic characteristics are believed to be the same application [35–39], so a clustering algorithm can group these flows together. A clustering classifier would function by first performing clustering on a training set and then mapping all the clusters to an application class based on an appropriate heuristic. Then incoming flows would be classified by which cluster it is the closest to based on the distance metric chosen (e.g., Euclidean distance). Then the flow is labeled based on which

application class the cluster has been mapped to.

In early work on clustering, Hernandez-Campos et al. [47] found that they could classify traffic flows by observing the number of bytes being transferred in the flow. The authors construct an abstract communication model that uses a three-dimensional vector for every flow. The dimensions consist of the number of bytes sent from the client to the server in the given time period, number of bytes sent from the server to the client in the given time period and the amount of time since the last data exchange. From this vector, they derive statistics about the flow such as total bytes sent, minimum number of bytes sent in a time period and average number of bytes in a time period. Then agglomerative and divisive hierarchical clustering is performed using the flow statistics they selected. The authors demonstrated that the clusters that are formed separate the flows well with examples where web traffic (ports 80, 443, 8080 and 8443) were all clustered together based just on the flow statistics. Unfortunately, Hernandez-Campos et al. did not provide a heuristic to map clusters to an application type (clustering groups similar data together, it does not provide a label), nor do they provide any quantitative results on how well they classify traffic. Nonetheless, this early work showed that traffic flows can be separated based on flow statistics.

Erman et al. have proposed different types of clustering methods [46, 48]. In [48], Erman et al. experimented with the K-Means, Density Based Spatial Clustering of Applications with Noise (DBSCAN) and AutoClass [49] (which uses Expectation Maximization (EM)) traffic clustering algorithms. They chose these three clustering algorithm as they were able to compare three different types of clustering algorithms: a partition-based (K-Means), density-based (DBSCAN) and probabilistic model-based (AutoClass). To validate the algorithms they chose, they used a publicly available trace, the Auckland IV trace [50] (which contained only packet headers) and collected their own trace from their campus at the University of Calgary (from March 2006). To provide a base truth for these traces, port-based classification was used for the Auckland trace and payload-based classification for their campus trace. Erman et al. believed that using port-based classification was valid for the Auckland trace as it was from the early 2000s when applications that used dynamic ports were not as prevalent as they are now.

In testing their algorithms, Erman et al. noticed that HTTP was by far the most prevalent application in their traces. To ensure that they were able to test all the applications, Erman et al. chose to construct their data set had the same amount of flows for each application. This may have biased their classifier as they were not inputting a representa-

tive mix of the applications from the traces. To calculate their results, the data was split up into multiple sets and the average was taken across all the data sets. For K-Means clustering, the authors needed to specify the value K (how many clusters to use). They found that the overall accuracy increased when the number of clusters increased. After 150 clusters, though, the increase in accuracy was minimal and Erman et al. noted that when K increases, the likelihood of over-fitting the classifier to the training set increases as well. For $K = 100$, the average overall flow accuracy was 79% and 84% for the Auckland IV and campus data sets respectively.

For DBSCAN clustering, clusters are formed in areas with a high density of points (note that with this algorithm, outliers may not be placed in any cluster — these points are considered noise). This algorithm required Erman et al. to specify the radius around a given point (eps) to look for neighboring points and the minimum number of points (minPts) inside this radius for a cluster to be formed. The cluster is then formed by taking all of these points plus all the neighbors of these points. From their results, Erman et al. found that the overall accuracy was better when minPts was lower, as smaller clusters were formed thereby decreasing the probability that multiple applications were inside the cluster. For the eps parameter, Erman et al. found that the overall accuracy was increasing when eps was increased until it reached a certain threshold where increasing eps caused a drastic decrease in overall accuracy. Erman et al. found that smaller clusters (that contained different applications) were being merged to form one larger cluster when eps was getting too large. In their optimal configuration of DBSCAN, the average overall flow accuracy was 76% and 72% for the Auckland IV and campus trace, respectively. One reason for the low accuracy is that all the points that are considered noise are misclassified. While the average accuracy was low, Erman et al. found that the average precision for seven of the nine applications they measured was over 95% for the Auckland IV data sets. This showed that DBSCAN produced highly accurate clusters for their data sets.

For the AutoClass classifier, nothing had to be configured as all the parameters are automatically determined. For these data sets, AutoClass was able to obtain the highest overall flow accuracy out of the three classifiers — 92% and 89% for the Auckland IV and campus trace, respectively. The drawback to AutoClass, though, is the long training time. The other two classifiers took a matter of minutes to train while AutoClass took close to five hours to train.

Erman et al. compared their AutoClass implementation to a supervised machine learn-

ing algorithm, the Naïve Bayes classifier in [51]. Based on their results on the Auckland IV & VI traces they found that the AutoClass had a higher overall flow accuracy than the Naïve Bayes classifier (91% to 83%) for these traces. Given this result, the authors argue then that an unsupervised learning algorithm can perform just as well as a supervised learning algorithm with the added bonus that the training set does not need to be labeled. Rather, if proper clusters are formed, the user needs to look at only a couple flows in each cluster to determine which application to map the cluster to.

In [46], Erman et al. expanded their work on K-Means clustering. The first thing that they expanded on is changing the problem from a supervised problem to a semi-supervised one. The problem was originally a supervised one as all the flows inside the training set had to be labeled. This was because the heuristic Erman et al. chose to map a cluster to an application class was based on the dominant application class inside the cluster. The drawback to this is that labeling the entire data set is arduous. There are not many instances of labeled traffic traces available. Generally, some sort of payload-based classifier [26,27] is applied to label the traces but the signatures still need to be determined. Instead they experiment with two approaches in their paper. In the first approach, they first perform clustering on the data set, then after the clusters were formed, they chose a handful of random points inside each cluster and labeled them. Then the cluster is mapped to the dominant application from these labeled points. Erman et al. found that labeling as few as 2 flows per cluster, they were able to obtain 94% accuracy with $K = 400$. In the alternative approach, Erman et al. mix labeled and unlabeled flows in the data set then applied clustering. By carefully choosing the labeled flows, they will be evenly distributed between all the clusters, and all the clusters will have a proper mapping. An advantage with this approach is that if a cluster has no labeled flows inside it, this may signify that a new application has been found. Erman et al. also found that the precision can be increased simply by adding more unlabeled flows to the data set. Precision is defined by the percentage of flows correctly identified excluding all unknown flows. Since unlabeled flows are relatively inexpensive to obtain, this is a simple yet effective way to increase precision.

Erman et al. also considered the importance of byte accuracy. Byte accuracy is the proportion of bytes that are correctly classified relative to all the bytes in the network [52]. Byte accuracy is important to measure as it shows what fraction of network data is being correctly classified. Flow accuracy does not show if large flows are being classified correctly

and if these flows are misclassified this could signify that a large proportion of network data is being misclassified (even with a high flow accuracy). Using randomized or sequential sampling to construct their data sets, Erman et al. found that they were able to obtain over 90% flow accuracy for all their traces but their byte accuracy varied from 50-85% for these traces. The reason for this was that large flows were not being represented equally in the data set. To remedy this, Erman et al. used weighted bytes (or duration) sampling. For this sampling, they chose 50% of the flow above the 95th percentile (and 50% below) for the flow transfer size (or flow duration). With these approaches, flow accuracy stayed the same but they were able to increase byte accuracy.

Another task that Erman et al. undertook was to develop an online classifier. They constructed a separate classifier for every packet milestone of a flow that they specified. They chose a one hour trace from their campus on April 2006 to test. After the first packet milestone (8 packets), 78% of all flows were classified correctly but the byte accuracy was only 40%. The reason for such a low byte accuracy is that Erman et al found it is hard to distinguish the large flows after so few packets. It was only at their last milestone (16K packets) when their classifier was able to start identifying the large flows as the byte accuracy was able to reach 78% (the flow accuracy was 82%).

Erman et al. applied their K-Means clustering algorithm to classify traffic at the network core as well [53]. This poses a different challenge than their previous work as at the network core, flows are generally unidirectional due to routing asymmetries. As a result, there is only half of the information about the flow. Erman et al. propose a method to estimate the missing half of the flow given the available information. They looked to estimate the duration, number of bytes and number of packets for the missing half of the flow as most of the other flow statistics can be derived from these. They found that server-to-client flows were better for classifying traffic than client-to-server flows as the server-to-client flows generally contained more information about the flow. As a result, they obtained a flow accuracy of 95% and byte accuracy of 79% for a server-to-client flows only data set while with a client-to-server flows only data set they obtained a flow accuracy of 94% and byte accuracy of 67%. Erman et al. found that if they used their estimation technique to estimate the server-to-client statistics on the client-to-server flows only data set, the flow and byte accuracy approached the same results as the server-to-client flows only data set. This validated their estimation technique as Erman et al. were able to increase the accuracy of client-to-server flows only data set to the accuracy of the server-to-client flows

only data set which had the higher accuracy of the two.

In [54–56], Bernaille et al. applied different clustering algorithms in classifying traffic but their work focused on classifying after the first P packets of a flow. In [54], the clustering algorithms that they applied were K-Means and Gaussian Mixture Models (GMM) in Euclidean space and spectral clustering on Hidden Markov Models (HMM). The features used for their classifiers are the packet size and direction of the first P packets. They found there was enough distinction between applications using just these two features. Bernaille et al. also produce a new way to classify flows based on a cluster called cluster & port. The common approach is when a flow is assigned to a given cluster, this flow is mapped to the dominant application of the cluster. Their approach involves looking at the port number of the flow. They define a list of applications that still use standard ports like HTTP, POP3 and SMTP. Then when a flow is assigned to a cluster, if the flow is using a port from the standard list and there are flows inside the cluster already using this port number the flow is mapped to the standard application that uses that port number. If there are no other flows inside the cluster that uses the flow's port number then the flow is labeled as a masquerade flow as this flow is trying to disguise itself as a common application. Similarly, if the flow is using a non-standard port, the flow is labeled as the dominant application from the flows that do not use standard ports inside of the cluster. If there are no applications that use non-standard ports inside the cluster, the flows are labeled as a masquerade flow as this is probably a standard service using a non-standard port. In their experiments on a campus and enterprise trace they were able to identify 98% of all flows with all three of their clustering methods (with $P = 4$). This was an improvement over mapping flows to the dominant application of a flow as the flow accuracy varied from 74-95% for the clustering algorithms. GMM and HMM have the advantage that they have a high likelihood of classifying new applications (applications that were not in the training set) as unknown. Therefore if there is an increase in the unknown class, the network administrator can look through these flows to see if there is a new application that needs consideration. Their work is similar to our work as we are also trying to classify traffic based on only the first P packets of a flow. Similarly, our classifier is designed to classify previously undiscovered applications as unknown as well.

Bernaille et al. extend their work in [56] by looking at how their classifier works on encrypted flows. For this work only SSL traffic was considered. To obtain SSL traffic, they filtered HTTPS and POPS flows from a campus network. They also chose to manually

encrypt non-encrypted traffic by replaying these flows through a secure tunnel. They were able to identify over 85% of all encrypted flows using their approach. This was an expected result because if flow statistics are used as inputs, it does not matter if the traffic flow is encrypted (it is irrelevant). This is assuming that the packet header is not encrypted.

While most of the work reviewed so far has been about partition-based clustering (K-Means), there has been significant work done with probabilistic-based clustering as well. We have already mentioned one type of probabilistic clustering (AutoClass) in [48]. Probabilistic-clustering differs from partition-based clustering by the number of clusters to which a data point can belong. In partition-based clustering, a given data point can only belong to one cluster. On the other hand, probabilistic-based clustering yields the probability of a data point belonging to a given cluster, so it is possible for the data point to be associated with more than one cluster. This better models a real-world situation given that it may not be accurate to associate a given data point to one exclusive cluster based on the training set. McGregor et al. [57] used the Expectation Maximization (EM) algorithm to classify flows. The authors believed Internet traffic flows can be clustered by application as they were able to see distinct applications based on the packet size and inter-arrival time of packets. Using these statistics and other flow statistics gathered from traces from Auckland-VI and the University of Waikato, they created clusters using the EM algorithm. These clusters separated flows by separate type such as bulk transfer, multiple transaction and small transactions. While it performed well on this high-level classification, it performed poorly on the application-level as they saw HTTP flows in multiple clusters. This made sense, though, as HTTP flows have many different characteristics — for example, HTTP can be used for simple web browsing or for downloading large files. As a result, the authors are looking for ways to improve on the application-level but their approach is applicable if only high-level information about the flows is needed (i.e. if it is a bulk transfer flow, multimedia flow, etc.)

Preceding Erman et al. [48], Zander et al. also used AutoClass to classify Internet traffic flows using flow statistics. AutoClass is an unsupervised Bayesian classifier that uses EM to find the best cluster set. AutoClass was also used to determine the number of clusters needed. Using their method, the authors were able to classify over 86% of the flows on the public traces Auckland and NLANR [58]. It is worth noting that for public traces, only the packet headers are available, so to provide a ground truth, flows are classified by port number. Most of the public traces are from the early 2000s though, where dynamic

port numbers were not as commonly used so port numbers can provide a fairly accurate representation of the application being used. Zander et al. propose another metric for evaluating their clusters — a homogeneity metric. The homogeneity metric is defined as the fraction of the dominant application of a cluster to all the flows in the cluster. The goal is to have this metric as close to one as possible as this shows that quality clusters are being made (the majority of the class is only one application). For their traces the intra-class homogeneity was between 0.85 and 0.89 which showed their approach was able to separate classes effectively. They also investigated the homogeneity of Internet applications. They found that certain applications have high homogeneity (online games were an example) and can separate from the other applications easily while other applications had a lower homogeneity and could be found in multiple clusters like FTP. They attributed this to there being many diverse uses for FTP.

2.3.3 Machine Learning Algorithms

Another subset of Internet traffic classifiers uses popular machine learning techniques to classify traffic. The machine learning algorithms described here use supervised learning in which all training flows need to be labeled by their application type. Below are the different types of machine learning algorithms that have been used.

Probabilistic Machine Learning Methods

Probabilistic machine learning methods do not assign a given traffic flow to just one discrete class. Instead they assign a probability for a traffic flow to belong to a given class. Moore et al. [59] believe this is the proper approach to take for Internet traffic classification as it allows the classifier to be robust to measurement errors that arise from collecting statistics about the flows on a network. Also, a network flow could belong to two classes, as, for example, there are many applications that are tunneled over HTTP.

In [59], Moore et al. constructed a Naïve Bayes classifier to classify Internet traffic flows into broad classes (i.e. WWW, BULK, DATABASE, etc.). They collected a traffic trace from their campus network and were able to extract 248 statistics (or discriminators) about each flow. With Naïve Bayes classifiers it is important to only input flows that are relevant in helping to separate the application classes. Otherwise, the classifier can act poorly with irrelevant or redundant information. Inputting all the discriminators into the classifier,

they obtained only a 65% flow accuracy. Moore et al. used two techniques to improve the accuracy: kernel estimation for Naïve Bayes which provides a generalization of Naïve Bayes, and the Fast Correlation-Based Filter (FCBF) which determines which discriminators are relevant. With these two methods, they were able to obtain a flow accuracy of 96% on the same trace. The byte accuracy for these traces was measured and oddly enough, the byte accuracy was around 84% for both implementations. The authors attribute this to the discriminators being poor for a certain type of flow present in their trace. Moore et al. showed that their classifier has temporal stability by using this classifier to classify a flow that was taken a year later and the classifier was still able to achieve a flow accuracy of 93% with the two techniques. All the techniques for their experiments were obtained from the WEKA toolbox [42]. Erman et al. [51] based their Naïve Bayes classifier on the work done by Moore et al. here.

In [60], Auld et al. improve on the Naïve Bayes classifier by implementing a Bayesian trained Neural Network instead. Their Neural Network had 246 flow features as inputs and there were n outputs, where n was the number of classes in their data set (there was one hidden layer as well). Auld et al. looked for ways to reduce the number of inputs in order to reduce the computation time. The classes divided the applications into broad categories as in [59]. They compared their procedure to [59] and found that their classifier was able to outperform the optimized Naïve Bayes classifier. They were able to obtain a flow accuracy of 99% for a data set that was collected on a 24-hour period from their campus network. Using this trained classifier on a traffic trace gathered 8 months later, the flow accuracy was still 95%. Auld et al. observe that both classifiers compared find some features that they deem important for classifying traffic flows but use different features as well. This shows that some features may be universally used, while other features could be classifier-specific.

Jiang et al. [61] demonstrated an inexpensive way of doing online traffic classification by using only information that was gathered by Cisco NetFlow. This included deriving features based on the information gathered by NetFlow as well. By using the symmetric uncertainty measure to rank the relevance of the features gathered by NetFlow and inputting the relevant features into a Naïve Bayes classifier (with Kernel Estimation), they were able to classify applications into broad classes with a flow accuracy of 88%. Their work shows that using statistics that are inexpensive to obtain and readily available to most network operators (as most operators use NetFlow) can provide the means for an accurate network classifier.

In [45], Williams et al. compared three different Bayesian algorithms (Naïve Bayes with kernel density estimation, Naïve Bayes with discretization, Bayesian network and Naïve Bayes tree algorithms) with a deterministic machine learning algorithm (C4.5 decision tree). All of these classifiers were implemented from the WEKA toolbox. They selected 22 flow statistics for their classifiers and experimented with either the consistency-based or correlation-based feature selection techniques inside of the WEKA toolbox to reduce the number of features needed. Williams et al. were able to reduce the features required to seven and nine for the correlation-based and consistency-based feature selection algorithms, respectively, and there was only a minimal drop in flow accuracy when compared to using all the features. This minimal decrease in flow accuracy, they argue, is a good trade-off for a significant decrease in training. In comparing all the algorithms, Williams et al. found that they all performed similarly for their data sets from the NLANR trace (all classifiers had a flow accuracy of over 80%). As a result of this, the authors looked at the time taken to train the classifier and the classification speed. The Naïve Bayes tree was by far the slowest classifier to train, while the other four classifiers were comparable in their build time. C4.5 had the fastest classification speed of all the classifiers. Our classifiers were designed to be able to classify flows online so classification speed is critical for us. We will show in Chapter 4 why implementing our classifiers with a 2ν-SVM will allow for quick classifications.

Deterministic Machine Learning Algorithms

The alternative to probabilistic machine learning algorithms is deterministic machine learning algorithms. These algorithms map the input traffic flow to only one possible output. The work done by Roughan et al. [62] attempts to separate traffic flows into four broad classes (interactive, bulk data transfer, streaming and transactional) for Quality of Service (QoS) guarantees. The three classifiers that were used were k-Nearest Neighbour (k-NN), Linear Discriminant Analysis and Quadratic Discriminant Analysis. Roughan et al. collected traces from five different locations to validate their algorithms. Interestingly, when constructing their training sets they did not include any HTTP traffic as they felt it did not belong to any of their four classes. Roughan et al. found that they need only three features to separate applications into these three classes: average flow duration, average packet size and inter-arrival variability metric. The inter-arrival variability metric is the

average ratio of the standard deviation of the inter-arrival time of a flow to the mean inter-arrival time of a flow. All of the classifiers performed well with the flow accuracy ranging from 92-95%. For the k-NN the best results were for when k was small, either 3 or 5. The reason they chose to classify the flows into broad classes was because for QoS only the class of an application matters and not the type. But Roughan et al. still tested their classifiers to classify on an application level (they identified 7 applications in their traces) and they found the flow accuracy for their classifiers ranged from 88-90%.

Roughan et al. observed that an application needs to be in the training set otherwise the classifier might have difficulties classifying the flow. The error rate when they tried to classify a new application to one of their four broad classes ranged from 14-57% for their classifiers. Our approach is more robust to a new application because if our classifier does not believe the flow belongs to one of the applications we specified, the classifier marks this flow as unknown rather than try to map it to an application. As such, if we see the unknown class rapidly increasing, the goal then is to investigate these flows to see if there is a new application present on our network.

In [63], Li et al. evaluate the temporal and spatial stability of the C4.5 decision tree and 3 other types of Internet traffic classifiers — port-based classification, DPI using a l7-filter [64] and Naïve Bayes with kernel estimation. Naïve Bayes with kernel estimation was previously worked on by one of the authors, Moore, in [59]. For their work, Li et al. consider both TCP flows and UDP traffic. Li et al. collected traces from two sites (Site A and B). From Site A, Li et al. collected traces from 3 separate weekdays in 2003, 2004 and 2006 and the Site B trace was collected from a weekday in 2007. Li et al.'s classifiers are online classifiers so they experiment with the number of packets received in a flow before attempting to classify a flow (they vary between 4-10). For the rest of the experiments Li et al. settle on classifying after 5 packets of a flow have been received. To examine the temporal stability, Li et al. trained the classifier on one day from Site A and examined how the classifier performed on the other days. To evaluate the spatial stability, a classifier would be trained on one site and then evaluated on the other site. In all their experiments, the C4.5 classifier had the highest overall accuracy which shows that the C4.5 decision tree is robust to temporal and spatial changes. The C4.5 classifier also had a high overall accuracy for UDP traffic when trained and tested on one site and day, but the performance degraded for the temporal and spatial changes. The issues mostly were from the P2P and multimedia classes as these classes have changed in the past couple of years as new

application are always emerging and different sites potentially use different applications.

Support Vector Machines (SVMs)

A Support Vector Machine (SVM) constructs an optimal hyperplane that separates two classes [65]. We use a generalization of the 2ν-SVM [8] for both of our classifiers. We do not use a SVM directly to classify Internet traffic flows. Rather, we use Neyman-Pearson (NP) classification and the LSAT framework to do so. The NP classifier provides the ability to set a maximum threshold for the false alarm rate for a given application class (or classes) while the LSAT classifier provides the ability to set a maximum threshold for the false discovery rate for a given application class (or classes). Note that we chose to not include SVMs as either a probabilistic or deterministic machine learning algorithm. The reason being that although all the approaches described below use a deterministic approach, SVMs can be implemented using a probabilistic approach as well [66].

In [67, 68], the authors focus on classifying P2P traffic only. In [67], the authors accomplish this by inputting transport-level flow statistics to a two stage classifier. The first stage is a binary SVM that determines whether or not the flow is P2P or not. If the flow is determined to be P2P, a multi-class SVM is used in the second stage to determine the type of P2P application. The multi-class SVM that they use is different than ours. In their work, they extended a binary classifier into a multi-class one as proposed in [69]. Our approach is different as we use a chain of binary classifiers to solve the multi-class problem. Using a set of binary SVMs is known to perform as well as a single multi-class SVM, without the severe scalability issues incurred by multi-class SVMs [65]. Chapter 4 will elaborate more on how our classifiers were implemented and why. Another requirement specified by Yang et al. was that the classifier needed to be online. In a real-time environment based on a traffic trace from their campus network, the first stage of the classifier was able to correctly identify approximately 99% of the flows while the second stage was able to successfully identify approximately 85% of the P2P applications.

In [68], González-Castaño et al. propose a solution to identify P2P on high speed routers. They argue that even when you sample packets you can achieve a high accuracy rate for identifying P2P traffic using a SVM. They also demonstrated that by adjusting a parameter inside their SVM they can achieve higher accuracy on P2P flows over other flows by giving P2P flows a higher penalty when they are misclassified. This is the coarse-grained

approach for assessing penalties on a particular application as there are no other restrictions imposed on this classifier — the higher penalty simply tells the classifier which class to emphasize but does not guarantee any performance level for this class. Our approach takes a more fine-grained approach by setting a restriction on the maximum tolerable misclassification and forcing the classifier to adhere to it. Unlike [67,68], our classifiers can classify multiple application types and not just P2P.

In [70,71], these authors divide the Internet traffic applications into broad classes. For example, the multimedia class contained all streaming applications while the bulk class contained file-transferring applications like FTP. Then Li et al. [70] use a multi-class SVM to classify incoming flows into one of these classes while Liu et al. [71] convert binary classifiers into an online multi-class classifier but did not specify how. Li et al. [70] was able to achieve over 99% flow accuracy on the campus trace they collected (or a flow accuracy of 96% when they biased the classifier to have equal mix of all the applications) while the classifier Liu et al. [71] used was able to achieve a flow accuracy of approximately 80% on the Auckland IV trace.

In [72], Kim et al. compared the performance of a port-based classifier (CoralReef), BLINC [73] (discussed in behavioural-based classifiers subsection below), three probabilistic classifiers (Naïve Bayes, Naïve Bayes with Kernel Estimation and Bayesian Network), three deterministic algorithms (k-NN, Neural Networks and C4.5 decision trees) and a Support Vector Machine (SVM). The SVM consistently outperformed all the other classifiers they tested on their traces (from backbone and edge networks in the US, Japan and Korea in 2004). To expand on these findings, they used the SVM in designing a robust traffic classifier that can be trained with data from one network and still classify well on another. By ensuring that the SVM had a complete training set (i.e., it contained all relevant applications), the SVM was able to achieve 94% flow accuracy on unseen traces.

Another finding from the Kim et al.'s work was that ports could still be relevant in classification for applications that are known to use IANA's port number mapping. For example, an input for their SVM was the port number and removing this feature caused a drastic decrease in flow accuracy. Note though that their traces were from 2004 so the application mix on networks can be different now with more applications using non-standard ports, especially since Karagiannis et al. have stated that P2P (an application that is known to use non-standard ports) use is not declining [1].

Este et al. [74] use a collection of SVMs to classify Internet traffic. Each SVM in their

work is a ν-SVM (which will be described in greater detail in the next chapter). The ν-SVM is a one-class SVM where only one class is inputted to the SVM and the SVM determines the spatial region that this class occupies in the input space. The input to the classifier is the packet size for the first n packets of a flow. To account for who is sending the packet, the value is positive when sent by the client and negative when sent by the server. The classifier they use has 2 stages. The first stage is the c ν-SVMs. If none of the SVMs determine that an incoming flow belong to their application then the flow is classified as unknown. If exactly one SVM determines that an incoming flow is in the region of its classifier then it is classified as that application. If multiple SVMs believe that the incoming flow belong in the region of their application, then the incoming flow is passed onto the second stage. The second stage is a multi-class SVM which is similar to ours as it consists of c binary classifiers. The difference is the heuristic chosen in selecting a class (also they use a regular SVM and we use 2ν-SVM which provides performance guarantees). The SVMs are "one vs all" classifier where each SVM determines whether the incoming flow belongs to a particular application or not. For their heuristic, if multiple SVMs believe that an incoming flow belongs to their class, Este et al. select the class where the incoming flow is the furthest away from the decision boundary while our approach consists of placing the classifiers in a chain and selecting the first class that identifies a flow as its own. We consider all possible ordering to find the optimal ordering.

Este et al. experiment on three different data sets: a trace from their faculty network and two public traces from the Lawrence Berkeley's National Laboratory (LBNL) [75] and Cooperative Association for Internet Data Analysis (CAIDA) [76]. For their trace, they had access to the payload so they used a Deep-Packet Inspection technique to establish the ground truth, while for the public traces, port numbers had to be used to identify applications. For each data set, six applications were chosen for classification (varied from data set to data set). The accuracy of almost all the selected classes were over 85%. The classes where the class accuracy was below 85% were found in the public traces and they were classes associated with port 80 and 443 (ports associated with HTTP and HTTP respectively). The problem is that port 80 is also known to be used by a variety of other applications (Skype for example) so without being able to look at the payload, Este et al. are classifying multiple applications as one class which leads to a low accuracy. Este et al. found that the port 443 class had a low accuracy due to having similar characteristics to flows that use port 993 (IMAPS) making it difficult for the classifier to distinguish between

the two. Este et al. also inputted into the classifier classes that they did not train on to see if they would be classified as unknown. This worked to a varying degree, depending on the class and how closely it resembled other classes.

2.3.4 Behavioural-based Classifiers

Karagiannis et al. [73] propose a classifier that they call BLINd Classification (BLINC) that investigates the behavior of network hosts and present a heuristical approach to traffic classification in which classification is done on a social, functional and application level. Their classifier can handle TCP and UDP traffic. The social level examines the popularity of a host: how many other hosts does this host interact with. In particular, it is important to determine which of the hosts are servers, because if these hosts can be identified, all the flows from a client to this host can be identified. Also at this level, they determine that if IPs in the same subnet are using the same port, they are likely providing the same service (part of a server farm). The functional level determines whether the host is a client or server. Generally they found that if the host is using 2 or fewer ports it is a server. The application level examines the 4-tuple of the flow - source/destination IP address and port numbers to determine the application. Karagiannis et al. visualize this with graphlets. Heuristics using flow statistics are applied as well to aid in the classification. One example of a heuristic is that they found that the average packet size in a flow for gaming applications is constant, so this knowledge can be used to classify these flows. One drawback to their approach is that the classifier needs to be exposed to several flows from a host before it can decide what it is. This can lead to low byte accuracy if these large flows are sent infrequently from a host.

To assess BLINC, the authors measured the flow accuracy and the completeness of their classifier. Completeness is the fraction of flows they try to classify compared to the total number of flows in the trace; the classifier can classify a flow as unknown if it does not know what it is. They tested their classifier on two different campus traces and found the completeness ranged from 80-90% and the flow accuracy was over 95% for both traces. The performance of their algorithm is coarsely tunable. Kargiannis et al. can tune their classifier to balance between flow accuracy and the completeness of the classifier. Higher completeness compromises accuracy because the classifier will try to classify flows more aggressively. The authors argue that tuning the classifier for a higher accuracy is

more desirable as it is easier to examine flows in the unknown class rather than have a misclassified flow mixed into an application class. We go further than this by providing finer-grained constraints, both in terms of false alarm and false discovery rates.

Another approach that examines the social behavior of hosts is the work done by Iliofotou et al. [77]. The authors create traffic dispersion graphs to monitor the interactions of all the hosts. In the graph, the hosts are the nodes and edges are formed between two nodes when the hosts communicate with each other. These edges can be either directed or undirected depending on how the graphs are implemented. For TCP flows, they applied an additional filter to only add an edge if the flow is using a specific port number. For UDP flows, they only added an edge for the first packet sent in a flow. From their preliminary results they were able to use graph metrics to see what type of application is present in the graph (client-server, P2P or malware). Future work involves finding heuristics for these graphs so that they can be mapped to applications.

2.4 Our Contribution

Although numerous traffic classification methodologies have been proposed, none provide strict guarantees on performance, particularly false alarm/discovery rates (a claim validated by Nguyen et al. in [3]). Such guarantees are essential if a network operator wishes to employ traffic flow control procedures based on the classification labels. The primary contribution of this work is the proposal of two novel online classifiers that provide class-specific performance guarantees. The first classifier controls false alarm rates, and is based on a multi-class generalization of Neyman-Pearson (NP) classification [4–8]. The second controls false discovery rates, and is based on a multi-class generalization of the Learning Satisfiability framework (LSAT) [9]. The FAR/FDR constraints are the parameters that must be specified by the operator. Both classifiers are implemented as generalizations of the 2ν-Support Vector Machine (2ν-SVM) [78], which implies that classification can be performed online with a small number of elementary arithmetic operations. The 2ν-SVM was chosen over other methods because this SVM provides the opportunity to tune performance guarantees and the classifier can be run online. Deep Packet Inspection just looks for patterns inside the payload so this method is limited to what it can do with performance guarantees. Clustering only groups similar flows together so a proper heuristic would need to be developed to provide performance guarantees (which has not yet been developed in

the Internet traffic classification field for clustering). We could have chosen other machine learning techniques but we found the 2ν-SVM is the best fit for our needs due to its robustness (can be used as basis for both the NP and LSAT classifiers). Behavioural-based classifiers that were mentioned are not able to provide any strict performance guarantees. Our classifiers operate solely on statistics derived from packet headers, which are relatively easy to obtain.

Chapter 3

Background

This chapter lays the foundation for the algorithms that we use to solve the three problem statements. We provide background for the two algorithms we use to solve our three problems — Neyman-Pearson classification [4–8] and the Learning to Satisfy framework [9]. Since both algorithms use Support Vector Machines (SVM) (and more specifically the 2ν-SVM [8,78]) as their foundation, we describe what SVMs are and how they are implemented. We also detail the changes that must be made to a SVM to modify it to a 2ν-SVM which is required for our problems. Then, we describe how NP classification and the LSAT framework can be implemented with a 2ν-SVM.

3.1 Support Vector Machines (SVM)

Support Vector Machines (SVMs) [79] are a popular and robust method for classification. The goal for a SVM is to find the optimal hyperplane that separates two classes (if this hyperplane exists). The optimal hyperplane in this case is the hyperplane that maximizes its margin from the nearest point from both classes (*the max-margin principle*[1]). The margin is the distance from the closest data point of a class to the hyperplane. If there is no hyperplane that can separate the two classes, we discuss later in this section two methods to deal with this — (i) using a kernel function which maps the input space into a higher-dimensional space where the two classes are separable and (ii) introducing a slack parameter to the SVM which allows some points to be misclassified. We discuss two types

[1]The *max-margin principle* defines a hyperplane that is maximum equidistance from points in the two classes.

of SVMs with a slack parameter — C-SVM (Section 3.1.1) and 2ν-SVM (Section 3.1.2).

To construct a SVM, a training set with m points, $T = (\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1, 1\}, i = 1, \ldots, m$, is required. The first step in constructing the SVM is transforming the input features $\mathbf{x}_i$ via a kernel function which is a mapping, $\Phi : \mathbb{R}^n \to \mathcal{H}$, where $\mathcal{H}$ is a high-dimensional Hilbert space. This step is necessary as it can be difficult to separate the classes in the given input space, but when they are mapped to a higher-dimensional space the classes become easier to separate. Table 3.1 shows common examples of kernel functions. Generally, when constructing a SVM, different kernel functions are considered to find the optimal SVM. We selected the radial basis function (RBF) as our kernel for all of our evaluations as this kernel provided the best results.

Table 3.1 Four common kernel functions: linear, polynomial, radial basis function (RBF) and sigmoid. The parameters r, γ and d need to be tuned in order to find the optimal SVM.

Kernel Type	Kernel Function ($k(\mathbf{x}, \mathbf{x}')$)
linear	$\mathbf{x}^T\mathbf{x}'$
polynomial	$(\sigma\mathbf{x}^T\mathbf{x}' + r)^d, \sigma > 0$
radial basis function	$e^{-\sigma\|\|\mathbf{x}-\mathbf{x}'\|\|^2}, \sigma > 0$
sigmoid	$\tanh(\sigma\mathbf{x}^T\mathbf{x}' + r)$

After the mapping has been made, the optimal hyperplane needs to be constructed to separate the points. The hyperplane takes the form,

$$f(\mathbf{x}) = \mathbf{w}\mathbf{x} + b \tag{3.1}$$

where $\mathbf{w}$ is the normal vector to the hyperplane and b is the affine shift. We set the hyperplane to $\mathbf{w}\mathbf{x} + b = 0$. We then normalize $\mathbf{w}$ and b so that the closest point from each class is $\frac{1}{||\mathbf{w}||}$. Then we can deduce that the margin for the classifier is $\frac{2}{||\mathbf{w}||}$ where minimizing $||\mathbf{w}||$ will maximize the margin. Since there are no points from the $y = 1$ class inside the margin of the classifier, the decision boundary for the $y = 1$ class is all the points where $\mathbf{w}\mathbf{x} + b \geq 1$. Similarly, the decision boundary for the $y = -1$ class is all the points where $\mathbf{w}\mathbf{x} + b \leq -1$. As stated previously, the optimal separating hyperplane is the hyperplane that maximizes the margin between itself and the two classes so the optimization problem for support vector machines is:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \tag{3.2}$$
$$\text{s.t.} \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, \ \forall i = 1, \dots m$$

The optimization problem described in (3.2) consists of the objective function to maximize the margin and one constraint that is used to provide a boundary between the classes. These type of problems are known as constrained optimization problems and can be solved with Langrangian multipliers $\theta_i > 0$. Then the optimization problem can be re-written as:

$$L(\mathbf{w}, b, \theta) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \theta_i (y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1) \tag{3.3}$$

Solving this Lagrangian requires maximizing L with respect to the primal variables $\mathbf{w}$ and b while minimizing L with respect to the dual variables θ_i. This is the same as looking for the saddle point[2]. Note that since we are trying to maximize L with respect to θ, if $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 > 0$ then $\theta_i = 0$ so only points that are on the boundary of the margin are of interest. To solve this problem, the Karush-Kuhn-Tucker (KKT) conditions are applied which state that at the saddle point the primal variables must vanish, i.e.,

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \theta) = 0 \text{ and } \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \theta) = 0 \tag{3.4}$$

which leads to

$$\sum_{i=1}^{m} \theta_i y_i = 0 \text{ and } \mathbf{w} = \sum_{i=1}^{m} \theta_i y_i \mathbf{x}_i \tag{3.5}$$

Substituting these two variables back into (3.3) gives the dual optimization problem which is easier to solve:

$$W(\theta) = \sum_{i=1}^{m} \theta_i - \frac{1}{2} \sum_{i,j=1}^{m} \theta_i \theta_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{3.6}$$
$$\text{s.t.} \quad \theta_i \geq 0, \ \forall i = 1, \dots m \text{ and } \textstyle\sum_{i=1}^{m} \theta_i y_i = 0$$

[2] A saddle point is a point that is a stationary point ($L'(\mathbf{w}, b, \theta) = 0$) and a point of inflection (a point in L where the sign of the second derivative changes).

Therefore only a fraction of the points from the training set are relevant (the points on the margin, $\theta_i > 0$). These points are known as the *support vectors*. These support vectors are used to classify new points by creating a decision function,

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{m} y_i \theta_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b\right) \tag{3.7}$$

which determines which side of the hyperplane the point is on. Note that the sign function returns $+1$ for a positive number and -1 for a negative number.

Note that the kernel function aids to reduce the computational complexity of this decision function. The kernel function has the following property,

$$k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle = \langle \mathbf{x}, \mathbf{x}' \rangle^2 \tag{3.8}$$

(3.8) shows what is known as the kernel 'trick'. This 'trick' shows that when comparing two vectors, instead of mapping both of the vectors into their higher-dimensional space and then calculating the inner product, the inner product can be calculated directly (and squared). This significantly reduces the computational complexity and saves from mapping the features to sometimes infinite-dimensional spaces. To be a valid kernel function, the kernel function needs to be continuous, symmetric and positive semi-definite.

3.1.1 C-SVM

In practice, it is not always possible to find a hyperplane that can separate the two classes. Therefore a slack parameter, $\xi_i > 0$ is introduced at the margin which provides a tolerance for misclassification. This allows points to be inside the margin. This type of SVM is known as a soft-margin SVM. Then the new optimization problem is shown in (3.9) where $C > 0$ is the misclassification penalty. The parameter C is used to find a trade-off between minimizing the training set error and maximizing the margin. The issue with this parameter is that it is not intuitive to find the optimal value for this parameter.

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \frac{||\mathbf{w}||^2}{2} + C\sum_i \xi_i & & & \\ \text{s.t.} \quad y_i(k(\mathbf{w}, \mathbf{x}_i) + b) &\geq 1 - \xi_i & &\text{for } i = 1, \ldots, n \\ \xi_i &\geq 0 & &\text{for } i = 1, \ldots, n \end{aligned} \tag{3.9}$$

3.1.2 2ν-SVM

Scholkopf et al. [80] re-formulated a soft-margin SVM so that the parameter to tune is more intuitive — the ν-SVM. Scholkopf et al. show that this ν term provides more interesting properties about the SVM than C. The optimization problem changes from the one found in (3.9), to:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \frac{||\mathbf{w}||^2}{2} - \nu\rho + \sum_i \xi_i & \qquad (3.10) \\ \text{s.t.} \quad y_i(k(\mathbf{w},\mathbf{x}_i)+b) &\geq \rho - \xi_i \quad \text{for } i = 1,\ldots,n \\ \xi_i &\geq 0 \quad \text{for } i = 1,\ldots,n \\ \rho &\geq 0 \\ \nu &\in [0,1]. \end{aligned}$$

Another parameter is introduced in (3.10), ρ. This parameter is used to tune the margin of the classifier. Note that if $\xi_i = 0,\ \forall i$ then the margin is simply $\frac{2\rho}{||\mathbf{w}||}$. Using this implementation, the parameter ν provides a lower bound on the fraction of support vectors used and an upper bound on the margin errors. Therefore tuning ν provides a method to control the error rate as opposed to the parameter C in the original implementation, (3.9), which just provides a penalty for misclassification.

Chew et al. [78] take this one step further by giving each class its own parameter ν, (ν_+ and ν_-) to tune and call their approach 2ν-SVM. This form of cost-sensitive classification allows there to be different penalties for the separate classes. Another advantage of 2ν-SVM is that it addresses a training set bias as if there is a dominant class inside the training set, this will not bias the classifier towards this class as both classes are treated separately in this implementation. The optimization problem for the 2ν-SVM is:

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\rho} \frac{(\nu_+ + \nu_-)||\mathbf{w}||^2}{2} - 2\nu_+\nu_-\rho + \frac{\nu_-}{n_+}\sum_{i\in I^+}\xi_i + \frac{\nu_+}{n_-}\sum_{i\in I^-}\xi_i & \qquad (3.11) \\ \text{s.t.} \quad Z_i(k(\mathbf{w},\mathbf{x}_i)+b) &\geq \rho - \xi_i \quad \text{for } i = 1,\ldots,n \\ \xi_i &\geq 0 \quad \text{for } i = 1,\ldots,n \\ \rho &\geq 0. \end{aligned}$$

Two new variables are introduced in (3.11) — n_+ and n_-. These variables state the number of data points in both classes.

Davenport et al. [8] extend this work by introducing a way to control false alarm rates

with the 2ν-SVM. Minimizing the overall misclassification rate while setting a maximum allowable threshold on the false alarm rate is known as Neyman-Pearson (NP) classification [4–7]. Two advantages of NP-classification are that it is more intuitive to deal with false alarm constraints than to arbitrarily assign penalties to a class (or data point) as is done with other classification methods. The second advantage is that NP classification does not assume a priori class probabilities. This is advantageous because it does not matter if the class mix in the training set corresponds to the real class mix in the input space. This is an important fact when the training set does not necessarily reflect the entire input space.

The 2ν-SVM can be used for NP classification as follows. Let P_M and P_F be the misclassification rate and false alarm rate, respectively, and $\widehat{P_M}$ and $\widehat{P_F}$ be the estimated values of these rates from the data set. Then a grid search is performed to find the optimal values of ν_+ and ν_- in $[0,1]^2$ that minimizes $\widehat{P_M}$ while keeping $\widehat{P_F} \leq \alpha$, where α is the maximum allowable false alarm rate. Davenport et al. show that this process can be sped up by using coordinate descent instead of a grid search.

When constructing a 2ν-SVM, the optimal combination of the 2ν-SVM parameters (ν_-, ν_+ and the kernel parameter σ) need to be found. Davenport et al. [8] provide a risk function to assess what is the optimal combination of the 2ν-SVM parameters. The risk function is required as one classifier could have a lower misclassification rate while the other classifier could have a lower false alarm rate so it can be difficult to decide which is the better one to use. Davenport et al. use a risk function from [81] to assess the classifiers. The risk function for the classifier f is:

$$r(f) = \frac{1}{\alpha} \max\{P_F(f) - \alpha, 0\} + P_M \tag{3.12}$$

This risk function takes into account both the false alarm and the misclassification rates. Here, the false alarm rate is class specific so it is chosen between one of the two classes while the misclassification rate is the overall misclassification rate of the classifier. In our work, we use a chain of these binary classifiers where each classifier inside the chain is responsible for classifying a particular application class (a one-versus-all type classifier). For our work, the false alarm constraint will always be chosen for the application class that the classifier is responsible for. If the false alarm constraint is satisfied there is no penalty for that term. If the constraint is not satisfied, the relative error is measured $\frac{P_F(f)-\alpha}{\alpha}$.

Therefore, for stricter constraints, if the classifier does not satisfy the constraint set, a larger penalty is induced due to the $\frac{1}{\alpha}$ term. The reason we have a penalty for the false alarm constraint and do not just reject all classifiers that violate the false alarm constraint is because we are estimating the false alarm rate through the training set. As a result of this, we allow classifiers that exceed α by a small margin that may be attributed to the noise from estimating the false alarm rate. Another reason this risk function was chosen is because this risk function favours classifiers that minimize the P_M while satisfying the false alarm constraint which is exactly the problem statement we specified for this classifier. Note that future work can involve adding a weight to P_M if more emphasis wants to be added to the misclassification rate.

3.2 Learning Satisfiability

The binary version of the Learning to Satisfy framework, recently proposed in [9], introduces a novel learning method where the goal is to create the largest set in the input space that satisfies certain output constraints based on expected values or event probabilities. LSAT distinguishes itself from other classifiers by assessing the output behaviour on only the solution set rather than over the entire input space. This differentiates LSAT from other learning frameworks as for example, Neyman-Pearson (NP) learning assesses the output behaviour over the entire input space [8].

The LSAT framework is formulated as follows [9]. Let $\mathbf{X}$ be the random vector inside the input space $\mathcal{X}$ (with d features) and Y the corresponding output vector in $\mathcal{Y}$. Let the data $(\mathbf{X}_i, Y_i); i = 1, \ldots, N$ be independent and identically distributed (i.i.d.) according to an unknown probability measure $P \in \mathcal{P}$ (where $\mathcal{P}$ is the collection of all probability measures in this space) on the space $\mathcal{X} \times \mathcal{Y}$. Assume that there are $k+1$ constraints, each with the form $C_j(G, P) \geq 0$, where $\mathcal{G}$ is the collection of all sets that satisfy the constraint function mapping $\mathcal{C} : \mathcal{G} \times \mathcal{P} \rightarrow \mathbb{R}^{k+1}$, where $\mathcal{C}$ is the set of the constraints $\{C_j\}_{j=0,..,k}$. Therefore, for a probability measure P, the goal of LSAT is to then find the largest set $G \in \mathcal{G}$ that satisfies the constraint $C(G, P) \geq 0$. This inequality is applied element-by-element within the set on constraints. By letting $\mu(G)$ be a positive measure of choice then the LSAT optimization problem can be re-written as:

$$\max_{G \in \mathcal{G}} \mu(G) \quad \text{subject to} \quad C(G, P) \geq 0 \tag{3.13}$$

Note that there does not necessarily need to be a solution to this optimization problem. This can occur when the constraints specified are too strict. In this case, the solution would be the empty set.

An alternative way to pose this optimization problem is to find a risk function, R_0, such that minimizing R_0 (under the constraints $C_1, \ldots, C_k$) is equivalent to maximizing the volume of the set $\mu(G)$ subject to satisfying C_0 and the remaining constraints. Then the optimization problem can be re-written as:

$$\min_{G \in \mathcal{G}} R(G, P) \text{ subject to } C_j(G, P) \geq 0, j = 1, \ldots k \tag{3.14}$$

Particular care is required in identifying a risk function, as the risk function needs to have two properties. The first property is that if there is a non-empty solution to the optimization problem in (3.13) then the solution from this optimization problem should coincide with it. The second property is that if the empty set is the only possible solution, then the empty set must have a smaller risk than any set failing to satisfy C_0.

The LSAT framework can use two type of constraints — point-wise constraints and set-average (class-wise) constraints. Point-wise constraints are a function on the input variable $\mathbf{x}$, so $C(G, P) \geq 0$ becomes $C(\mathbf{x}, G, P) \geq 0$, $\forall \mathbf{x} \in G$. On the other hand, set-average constraints are a function of the entire set G and involve constraints that need to satisfy the average output of the set.

We illustrate the different types of constraints with examples from the classifier with FDR constraints and the large flow detector. The classifier with FDR constraints only needs set-average constraints as this classifier requires only false discovery rate guarantees. This can be expressed mathematically for each classifier in the chain (where each classifier is trying to create the largest set for a particular application) as $Pr(Y = 0|\widehat{Y} = 1) \leq \beta$ where β is the FDR constraint specified for the set of points with $Y = 1$ and $\widehat{Y}$ is the output of the classifier. Here, the classifier has two possible values — 1 if the input flow belongs to the set, G and 0 otherwise.

The large flow detector requires both point-wise constraints and set-average constraints. The point-wise constraint is expressed mathematically as, $E[Z|\mathbf{X} = \mathbf{x}] - U \geq 0, \quad \forall \mathbf{x} \in G$, where Z is the number of bytes in the flow (which is a random quantity, i.e. $Pr(Z = z|\mathbf{X} = x)$), and U is a threshold defining a large flow. The set-average constraint for the large flow detector is $Pr(Z \geq L|\mathbf{X} \in G) - (1 - \omega) \geq 0$, where L is a threshold specifying the maximum

size of a small flow and $1-\omega$ is a probability threshold and ω is the false discovery rate constraint. These two constraints are used in tandem as the point-wise constraint states that the expected number of bytes in a flow that is in the set is above U (i.e. a large flow). On the other hand, the set-average constraint gives the maximum allowable threshold for small flows that are allowed to be in the set G.

In practice, since the probability measure, P, is unknown, it needs to be estimated. A training set, $T = \langle \mathbf{x}_i, y_i \rangle, i = 1, \ldots, m$ is used to estimate $\widehat{P}$. Using this $\widehat{P}$, estimated versions of the constraint function $C(G, \widehat{P})$ and the risk function $R(G, \widehat{P})$ are calculated. Also $1 \gg \epsilon > 0$ is introduced as a tolerance for the constraint functions so small violations of the constraints can be tolerated. Then the optimization problem for the estimated set $\widehat{G}$ is:

$$\widehat{G} = \min_{G \in \mathcal{G}} \widehat{R}(G, \widehat{P}) \text{ subject to } \widehat{C}_j(G, \widehat{P}) \geq -\epsilon_j, j = 1, \ldots k \tag{3.15}$$

The LSAT framework is implemented by adapting the 2ν-SVM described earlier. LSAT has been previously implemented using Dyadic Decision Trees (DDT) [82] but we chose the 2ν-SVM approach as it is more robust for higher dimensional input spaces. The LSAT framework can be implemented using the 2ν-SVM by introducing a penalty (γ_i) for (i) leaving a point out of the set G that should be in the set; (ii) including a point in the set that can contribute towards the violation of any of the constraints specified. With these penalties and the other 2ν-SVM parameters, we can control the size of the set G where the tradeoff is between a larger set or a smaller set that has less chance of violating any of the constraints. (3.16) shows the new problem formulation for LSAT with the 2ν-SVM.

The following method is the general way to calculate γ_i. Every data point $(\mathbf{x}_i, y_i)$ has an artificial label $D_{i,j}$ and misclassification penalty $\gamma_{i,j}$ for every constraint j. This artificial label $D_{i,j}$ states whether $\mathbf{x}_i$ positively or negatively contributes to satisfying the given constraint j; $D_{i,j} = 1$ if $\mathbf{x}_i$ positively contributes to affecting the constraint (i.e. including this point in the set will help to satisfy the constraint j) and $D_{i,j} = 0$ if $\mathbf{x}_i$ does not (i.e. including this point in the set will hinder the ability to satisfy the constraint j). The misclassification penalty $\gamma_{i,j}$ gives the penalty for violating constraint j. However, all these labels need to be converted to one single label so the point $\mathbf{x}_i$ can be classified. Therefore we use the following mapping to determine the label D_i for every $\mathbf{x}_i$ and the misclassification penalty γ_i. If $D_{i,j} = 1, \forall j$ then $D_i = 1$ and $\gamma_i = \sum_{D_{i,j}=1} \lambda_j \gamma_{i,j}$ where λ_j provides a weight

for the constraint j. Similarly, if $D_{i,j} = 0, \forall j$ then $D_i = 0$ and $\gamma_i = \sum_{D_{i,j}=0} \lambda_j \gamma_{i,j}$.

Extra care needs to be taken if for a data point $\mathbf{x}_i$, $D_{i,j} \neq 1, \forall j$ or $D_{i,j} \neq 0, \forall j$. In this case, an auxiliary point is created. The auxiliary point is created to examine both cases for $\mathbf{x}_i$: $D_i = 1$ and $D_i = 0$. The data point $\mathbf{x}_i$, considers all the constraints that have been satisfied so $D_i = 1$ and $\gamma_i = \sum_{D_{i,j}=1} \lambda_j \gamma_{i,j}$. On the other hand, the auxiliary point $\mathbf{x}_{\tilde{i}}$ considers all the constraints that have been violated so $D_{\tilde{i}} = 0$ and $\gamma_{\tilde{i}} = \sum_{D_{\tilde{i},j}=0} \lambda_j \gamma_{\tilde{i},j}$.

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\rho} \frac{(\nu_+ + \nu_-)||\mathbf{w}||^2}{2} - 2\nu_+\nu_-\rho + \frac{\nu_-}{n_+} \sum_{i \in I^+} \xi_i \gamma_i + \frac{\nu_+}{n_-} \sum_{i,\tilde{i} \in I^-} \xi_i \gamma_i \qquad (3.16) \\ \text{s.t.} \quad D_i(k(\mathbf{w}, \mathbf{x}_i) + b) \geq \rho - \xi_i \quad \text{for } i = 1, \ldots, n \\ \xi_i \geq 0 \qquad \text{for } i = 1, \ldots, n \\ \rho \geq 0. \end{aligned}$$

Chapter 4

Methodology

In this chapter, we elaborate on the algorithms we propose to perform Internet traffic classification and large flow detection. We use multi-class versions of Neyman-Pearson (NP) classification [4–8] and the Learning to Satisfy (LSAT) [9] framework for Internet traffic classification and a binary LSAT classifier for the large flow detector. The rest of the chapter is organized as follows. We first elaborate on the three problems we are trying to solve. Then we discuss how these algorithms are adapted for Internet traffic classification, which includes transforming the classifiers to a multi-class setting. We use a chain of binary classifiers for this transformation. We also describe how the binary LSAT is suited for the large flow detector.

4.1 Problem Statements

This section describes the three problem statements that this thesis addresses. These problem statements can be categorized as belonging to the field of network operation management. For Internet traffic classification, we classify flows not packets. For this work, we focus on only TCP flows. A TCP flow has three stages: (i) establishing the connection which involves a 3-way handshake[1]. (ii) After the connection is setup, the data transfer

[1]To establish a connection, the client first sends a synchronize (SYN) packet to the server which is done by setting the SYN field in the packet header. In establishing the connection, flow properties such as maximum segment size are also specified. The server acknowledges receiving this packet by replying with a SYN-ACK packet which sets the synchronize field (SYN) in the packet header again to show that it is still in the connection phase and acknowledges (ACK) receiving the synchronize packet from the client. To complete the 3-way handshake, the client acknowledges receiving the SYN-ACK packet from the server

between the client and server begins. (iii) The final stage is terminating the flow. After the client (or server) is finished sending data it sends a packet to the server (or client) with the finished (FIN) bit set and the server (or client) acknowledges receiving this packet. The flow is terminated when the other device also completes this procedure. Flows can also be terminated because of a timeout (if no packet is sent in a given time interval). All packets belonging to a flow are characterized by having the same source IP address/TCP port number and the same destination IP address/TCP port number.

The task of the two Internet traffic classifiers is to assign a user-defined label corresponding to an application to each flow. From the i-th TCP flow, we derive a feature vector $\mathbf{x}_j$, which contains d_1 relevant flow statistics which will be the input to the Internet traffic classifiers and d_2 relevant flow statistics which while be input into the large flow detector. Let $\mathbf{x} \in \mathbb{R}^d$ be the statistics from a sample flow. Note that the statistics are derived after only k packets in the flow have been transmitted as the classification needs to be done online. The traffic classifiers then map this input to a discrete, finite output variable $\hat{Y}$, where $\hat{Y}$ is one of the $(c+1)$ application classes. Of the $c+1$ application classes, there are c known application classes (i.e. classes that we are training the classifier to classify) and the $(c+1)$th class is the "unknown" class (or a catch-all class). If the classifier does not think the flow is one of the c other application classes, the flow is then classified as "unknown". For the large flow detector, Z is the number of bytes when the flow is completed. Then, if Z is greater than a user-defined threshold, the flow will be considered large. Let z be the output of a sample flow. Note that Y and Z are random variables, that is, for any $\mathbf{X}$ the output is a probabilistic quantity. We denote Q_1 as the unknown probabilistic measure on $\mathbb{R}^{d_1} \times \mathbb{R}$ and we assume that every pair $(\mathbf{X}, Y)$ is independently, identically distributed (i.i.d.) according to Q_1. Similarly, we denote Q_2 as the unknown probabilistic measure on $\mathbb{R}^{d_2} \times \mathbb{R}$ and we assume that every pair $(\mathbf{X}, Z)$ is independently, identically distributed (i.i.d.) according to Q_2.

4.1.1 Problem Statement 1: FAR-constrained classifier

The first problem statement deals with the Internet traffic classifier with false alarm rate (FAR) constraints. The goal of this classifier is to assign a user-defined label for each flow that enters the classifier while adhering to false alarm constraints for some classes and

with an ACK packet.

minimizing the overall misclassification rate. The maximum allowable FAR constraint for class i, α_i, is determined by the user. Recall that the FAR for class i is the expected fraction of flows that do not belong to class i that are classified as being part of class i. Mathematically, this is expressed as $Pr(\widehat{Y} = i|Y \neq i)$ where $\hat{Y}$ is the output of the Internet traffic classifier and Y is the true underlying class. This is the general (classwise) definition of the FAR. The FAR can also be considered between just two classes, where there is a FAR constraint for misclassifying flows belonging to class j as class i (i.e. a pairwise FAR). The definition of the pairwise FAR is the expected fraction of flows that belong to class j which are classified as belonging to class i. Here, the maximum allowable FAR constraint is denoted by α_{ij}. Mathematically, the pairwise FAR is denoted as $Pr(\widehat{Y} = i|Y = j)$ where Y is the output of the Internet traffic classifier. Note that if the FAR constraints are too strict or there are too many constraints this may degrade the overall performance of the classifier. This is because the classifier might then classify everything as unknown to ensure that all the constraints are met (if you do not classify anything as class i you ensure that the FAR for class i is zero). The user then needs to find the right balance between FAR constraints and the overall performance. It is possible that there are no constraints set for a given class (the class may not be important) and in this case α_i (or α_{ij}) would be simply set to 100%. Then the goal for this class would be to minimize the misclassification rate. Summing up this problem statement, the goal is to find the classifier, f^* such that,

$$f^* = \arg\min_f Pr(\widehat{Y} \neq Y) \text{ where } \widehat{Y} = f(\mathbf{X}) \tag{4.1}$$

with either FAR classwise constraints,

$$Pr(\widehat{Y} = i|Y \neq i) \leq \alpha_i,\ i = 1, \ldots, c \tag{4.2}$$

or FAR pairwise constraints,

$$Pr(\widehat{Y} = i|Y = j) \leq \alpha_{ij},\ i, j = 1, \ldots c,\ i \neq j \tag{4.3}$$

4.1.2 Problem Statement 2: FDR-constrained classifier

The problem statement for the false discovery rate (FDR) constrained classifier is similar to the FAR-constrained classifier as the only difference is that this classifier minimizes the

overall misclassification rate while adhering to FDR constraints instead of FAR constraints. The maximum allowable FDR constraint, β_i for class i is set by the user. The classwise FDR for class i is the expected fraction of flows classified as class i that do not belong to class i. Mathematically, this is expressed as $Pr(Y \neq i|\widehat{Y} = i)$ where $\hat{Y}$ is the output of the Internet traffic classifier and Y is the true underlying class. A pairwise FDR can also be measured (for classes i and j) which is the expected fraction of flows that are classified as class i but belong to class j which can be denoted mathematically as $Pr(Y = j|\widehat{Y} = i)$ where Y is the output of the Internet traffic classifier. Let the user-defined maximum allowable pairwise FDR constraint be β_{ij}. Again, the same caveat is specified where the Internet traffic classifier's performance can degrade if the constraints are too strict or if there are too many constraints. Also if no constraints are required for class i, then simply set $\beta_i = 100\%$. Then the considerations for designing the optimal classifier f^* are,

$$f^* = \arg\min_f Pr(\widehat{Y} \neq Y) \text{ where } \widehat{Y} = f(\mathbf{X}) \tag{4.4}$$

with either FDR classwise constraints,

$$Pr(Y \neq i|\widehat{Y} = i) \leq \beta_i,\ i = 1, \ldots, c \tag{4.5}$$

or FDR pairwise constraints,

$$Pr(Y = j|\widehat{Y} = i) \leq \beta_{ij},\ i, j = 1, \ldots c,\ i \neq j \tag{4.6}$$

4.1.3 Problem Statement 3: Large flow detector

The large flow detector is the pre-processor for Internet traffic classifiers. By being able to identify if a flow is going to be large after only k packets of the flow have been transmitted, we can send to the Internet traffic classifiers only the large flows. This allows the classifiers to focus on flows that have a greater impact on the network (as thee flows use more network resources as they stay on the network longer and use more bandwidth) and allows the classifier to ignore the inconsequential flows. For this detector, we also provide a maximum allowable FDR such that there is a maximum threshold of small flows being classified as large, thereby ensuring the Internet traffic classifier does not waste time classifying the small flows. We denote L as the maximum number of bytes an entire flow can have to be

considered small and U is the minimum number of bytes an entire flow must have to be considered large.

The detector is posed as a constrained level set estimation problem. We denote Q_2' as the marginalization of the probability measure, Q_2 in $\mathbb{R}^d$. Then the problem is to find the largest subset G inside the feature space, which is the a subset of the feature space that maximizes the measure Q' and that all the constraints are met. For the "large" flow detector, we set two constraints. The first constraint, C_0, (a pointwise constraint) states that for a given flow $\mathbf{x}$, the expected value of Z (the total number of bytes in the flow) should be greater than U. The second constraint, C_1, (a set-average constraint) sets a maximum allowable FDR threshold, ω for the large flows. Then the constrained level set estimation problem becomes finding G^* such that,

$$G^* = \arg\max_{G \subseteq \mathbb{R}^d} Q_2'(G) \tag{4.7}$$

such that:

$$C_0 : \quad E[Z|\mathbf{X} = \mathbf{x}] \geq U \quad \forall \mathbf{x} \in G, \tag{4.8}$$

$$C_1 : \quad \mathrm{P}(Z < L|\mathbf{X} \in G) < \omega. \tag{4.9}$$

4.2 Network Operations Management Algorithms

This section describes how the 2ν-SVM and LSAT framework are applied to our network operation management problems from Section 4.1. We chose classifiers based on SVMs for our approach as once the SVM is trained it requires few operations to make a prediction on incoming data points and this is perfect for an online classifier [65]. Particular care is provided in discussing how both of the binary classifiers are adapted for our multi-class setting. Previous work with SVM-based Internet traffic classifiers show that they perform comparably to any other classifier used [3] so our approach should perform just as well with the added advantage of having performance guarantees. Also for the Internet traffic classifiers, we discuss the feature selection tool that we used so that only relevant features are sent to the classifiers.

4.2.1 Internet Traffic classification

For both of our Internet traffic classifiers, we use a chain of the respective binary classifier to provide multi-class classification, $f = (f_{s(1)}, \ldots, f_{s(c)})$ (for Neyman-Pearson classification, 2ν-SVM and for the FDR-constrained classifier, LSAT). Here $s(\cdot)$ is a permutation of the c application classes and $s(i)$ corresponds to the i-th application class in the permutation. This chain is illustrated in Figure 4.1 along with the "large" flow detector acting as the pre-processor. For many classification tasks, a chain of binary SVM classifiers has been shown to perform as well as a multiclass SVM [65, 83]. Multiclass SVMs are much more computationally demanding and do not scale well as the number of classes increases. Each binary classifier inside the chain is responsible for determining if the flow is one of the c application classes. For example, for class i, the classifier $f_{s(i)}(\mathbf{x})$ produces an output of 1 if the feature vector $\mathbf{x}$ belongs to class i and 0 otherwise. Mathematically, this is expressed as,

$$\hat{z} = \arg \min_{s(1) \leq i \leq s(c)} \{s(i) : f_{s(i)}(\mathbf{x}) = 1\} \tag{4.10}$$

Note that in our implementation the ordering in the chain is important as the first classifier that detects a match, determines the label. This is a simple approach in deciding which label to choose but there are other possible approaches as well. For example, Este et al. [74] used another approach where they chose the class that maximizes the margin from the deciding hyperplane. Since the first classifier that detects a match, determines the label, we need to consider all possible permutations to find the best classifier. If all the binary classifiers inside the chain determine the flow does not belong to the class it is classifying, then the flow is labeled as unknown (the $(c + 1)$ class). If the number of flows that are being classified as unknown is increasing, this may signify a new application class on the network. Then the user can train a new classifier for this class and add it to the chain.

For every classifier inside the chain (for both types of Internet traffic classifiers) we need to find the optimal parameters for the 2ν-SVM parameters — ν_+ and ν_- and a kernel parameter for the RBF kernel, σ (the LSAT classifiers have an additional parameter γ). Also as mentioned, the ordering of the classifiers inside the chain need to be considered. Each classifier in the chain has their own set of labels, $Y_i' = \delta_{Y=s(i)}$. We train the classifiers on a training set and perform a grid search over all the parameters. To perform a grid search, possible values for all the parameters are given, then the grid search searches all

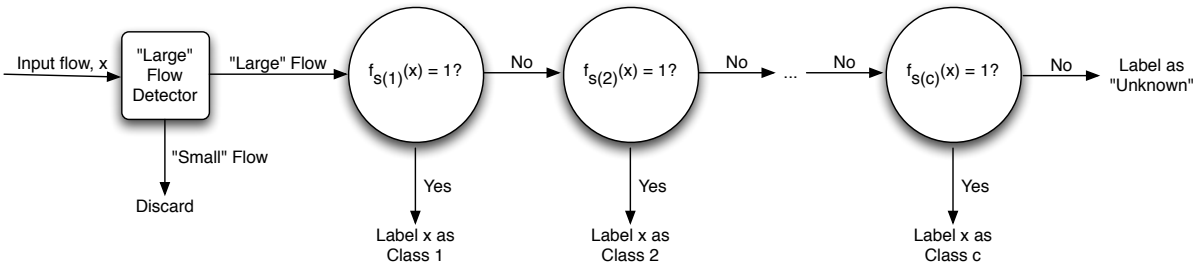

Fig. 4.1 An overview of how our Internet traffic classifiers are implemented.

possible combinations of these parameters to find the optimal combination. Table 4.1 shows the possible values we selected for ν_+, ν_-, σ and γ. We found that the resolution of the grid shown in Table 4.1 was able to effectively represent the input space of the parameters. Having a finer resolution will not cause a significant increase in the performance of the SVMs. k-fold cross-validation is applied on the training set to reduce the variability of the performance of the Internet traffic classifier. The training time is in the order of hours (on a Pentium IV 3.4 GHz with 2 GB of RAM), and is dependent on the number of features in the input space and the training set size. In order to compare different classifiers, a metric needs to be established in order to determine the better one. A risk function is used for the comparison where the classifier that minimizes the risk function is chosen.

Table 4.1 Values for ν_+, ν_-, σ and γ

Variable	Possible Values for the Variables
ν_+, ν_-	[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]
σ	$[10^{-4}10^{-2.4}10^{-.8}10^{.8}10^{2.4}10^{4}]$
γ	[0.001 0.01 0.1 1 10]

This risk functions for both the FAR-constrained and FDR-constrained classifier can be broken up into a sum of class-specific costs. Using this information, to find the chain with the lowest value, we employ a branch and bound search over the SVM parameters we have to tune. Let s be the ordering of the classes inside the chain. Then for $s(1)$, we find $f^*_{s(1)}$ that is the minimizer of the risk function $\widehat{R}(f_{s(1)})$. Then the classifier for $s(2)$, we search for $f^*_{s(2)}$ that minimizes $\widehat{R}(f_{s(1)}) + \widehat{R}(f_{s(2)})$. This process is repeated for the remaining classes and the overall value for the risk function is found and we call this κ. The parameter κ is the current minimum achieved by greedy search. We still explore the

various other branches to see if a lower risk function can be found. A lower risk function is still possible as the performance of classifiers later in the chain is dependent on what the classifiers earlier in the chain do. Therefore, it could be possible that having a higher risk function for the classifiers early in the chain can lower the overall risk of the Internet traffic classifier. Therefore we try a different branch (i.e., for the classifier $f_{s(1)}$ we choose the classifier that has the second lowest value for $R(f_{s(1)})$) and apply the same procedure as before. We terminate this branch once its risk exceeds κ. If it does not then this branch minimizes the risk function and is the new κ. This search is reasonably efficient, because evaluation of the partial costs is a very simple calculation, and the nature of the cost function means that many candidate classifier chains do not need to be evaluated because their partial costs grow very rapidly compared to the bound on the minimum cost. For example, any chain that violates one or more FAR constraints automatically has a much higher cost than a chain that can meet all constraints for the FAR-constrained classifier. For the FDR-constrained classifier, a violation of the FDR constraint eliminates the chain from consideration.

Now that the optimal classifier has been found for one ordering, we need to determine which ordering is the best ordering. When $c < 6$, it is feasible to search all possible permutations of the chain to find the optimal Internet traffic classifier. When $c \geq 6$, it begins to become computationally prohibitive to search over all the possible permutations. For this case, we use three heuristics to help reduce the number of orderings we explore. The first is to keep classes with strict FAR constraints at the beginning of the chain. The second heuristic is after an evaluation of an ordering, if we see that a class has a high partial cost, we promote this class in all future orderings. Both of these heuristics use the fact that classifiers at the beginning of the chain have more flexibility as fewer flows have already been classified. By being able to view a greater number of unlabeled flows, the classifiers at the beginning of the chain have more control over their performance. These classifiers can decide whether the unlabeled flows belong to their class or not. Classifiers near the end of the chain see fewer unlabeled flows so these classifiers might miss flows that belong to their class and this can negatively affect their performance. The final heuristic is to place classes that have a small share of the application mix early in the chain so that classes with a larger share do not "swallow" these flows into their class.

For our problems, using a chain of binary classifiers instead of selecting the classifier with the largest margin (as was done by Este et al. [74]) is better suited. If we had

chosen to do classification based on the largest margin, it is then harder to control our performance guarantees. With our approach, when a classifier is chosen, the class-specific cost associated to this classifier is fixed. It does not matter which classifiers are chosen later in the chain; these classifiers do not affect the cost of classifiers already chosen. If we had chosen the largest margin approach, controlling the performance guarantees is more difficult as changing any classifier can affect the previous classifiers that have already been chosen. This is because if a larger margin is found, labels for flows will be changed and this can change the performance guarantees of the classifiers.

With our approach, we find the optimal set of parameters for the first classifier in the chain and then move onto the next classifier. If we had chosen the largest margin approach, we need to find the optimal set of parameters for all the classifiers at once. This is a much larger space to search in than ours. For example, let t be the number of possible combinations of the possible values for the parameters for one classifier. With our approach, this search is done c times, for a total of $c \times t$ searches. For the largest margin approach, all the parameters need to be considered at the same time. Therefore we need to consider all possible combinations of all the parameters, so t^c searches are required, which is a much larger number of searches. Also even with small values of c it might not be feasible to find the optimal parameters for the largest margin approach as the value t^c, can increase rapidly making it difficult to search over the entire space. Note that from our work that it is important to search over the entire space due to the unpredictability that we found changing parameters (even slightly) has on the performance of the classifier. The largest margin approach may have a slightly higher overall performance than our approach due to a more exhaustive search but our approach is faster to train and better suited for our problems. Also an issue with using the largest margin as the method of determining which class a flow belongs to is that if not all the classifiers are using the same inputs then it is not a fair comparison to compare the classifiers' margins [65].

The rest of the chapter is organized as follows. First, we discuss the feature selection method that we used to select the relevant features as inputs to the Internet traffic classifiers. Then we discuss the risk functions chosen for the FAR-constrained classifier and the FDR-constrained classifier.

Feature Selection

In [45], Williams et al. examined different feature selection methods. We chose the Correlation-based feature selection [84] (with Best First Forward as its subset search) for our feature selection tool as it performed marginally better than their Consistency-based counterparts in [45]. Correlation-based feature selection measures the correlation between features and the class labels and eliminates the redundant features that are not correlated in mapping a class label. The correlation between two features, A and B is measured by the symmetric uncertainty,

$$U(A,B) = 2\frac{H(A) + H(B) - H(A,B)}{H(A) + H(B)} \tag{4.11}$$

where $H()$ is the entropy function. The entropy measures the uncertainty associated with a random variable. The entropy function is defined in (4.12) where $p(A)$ is the probability mass function of the random variable A.

$$H(A) = -\sum_i p(a_i)log(p(a_i)) \tag{4.12}$$

Then to measure the goodness of a set of features for a class C, the following equation is used,

$$\frac{\sum_j U(A_j, C)}{\sqrt{\sum_i \sum_j U(A_i, A_j)}} \tag{4.13}$$

and the values of this equation can range from 0 to 1. The goal is then to find the best set of features that maximizes (4.13) thereby removing all redundant features that are poor predictors of a class. To search for the optimal subset of features, the Best First Forward search is used. Best First Forward is based on a greedy search where the process starts with an empty set (when it is a forward search) and features are added one by one that best correlate to the class labels and features are added to the set until it is not worthwhile to do so ((4.13) starts to decrease). Best First Forward search allows backtracking to see if there is a better possible subset. Backtracking is done when a feature that is added decreases (4.13) so another feature is selected (the second-best feature of the features not yet selected that best correlates to the class labels). We use the software package WEKA [42] to perform feature selection. WEKA has a parameter that controls the number (N) of consecutive

features that are attempted to be added to the set without increasing (4.13) before the program terminates. The default value for N is 5 and that is what we kept it at. This is a classifier independent feature selection tool (a filter method) so the same features can be used for both the FAR-constrained and FDR-constrained classifiers.

FAR-constrained classifier

The metric the FAR-constrained classifier uses, is the risk function from (3.12). For each chain, the sum of all the risk functions of the classifiers inside the chain are taken (and the Internet traffic classifier that minimizes this risk is chosen),

$$R(f) = \sum_{s(i)} \frac{1}{\alpha_{s(i)}} \max(P_F(s(i)) - \alpha_{s(i)}, 0) + P_M(s(i)). \tag{4.14}$$

Since we cannot measure this quantity directly, it is estimated from the training set,

$$\begin{aligned} \widehat{R}(f) &= \sum_{s(i)} \widehat{R}(f_{s(i)}) \\ &= \sum_{s(i)} \frac{1}{\alpha_{s(i)}} \max(\widehat{P}_F(s(i)) - \alpha_{s(i)}, 0) + \widehat{P}_M(s(i)) \end{aligned} \tag{4.15}$$

FDR-constrained classifier

The FDR-constrained classifier follows the same procedure as the FAR-constrained classifier but with slightly different notation. Since we are using the LSAT framework, each classifier i in the chain looks to create the largest set of flows of class i while meeting the FDR constraints. Re-writing this in LSAT notation, for every class $s(i)$ in the chain, let $G_{s(i)} \subset \mathbb{R}^d$ be the set of feature vectors for which class $s(i)$ is detected, that is $f_{s(i)}(\mathbf{x}) = 1$ if and only if $\mathbf{x} \in G_{s(i)}$. Then we are looking for the optimal classifier, $f^*_{s(i)}$, that can find the largest set $G^*_{s(i)}$,

$$G^*_{s(i)} = \arg \max_{G_{s(i)} \subset \mathbb{R}^d} Q'(G_{s(i)}) \tag{4.16}$$

such that the FDR constraint for class $s(i)$ is met:

$$C_0 : \quad Pr(Y = 0|\hat{Y} = 1) < \beta_{s(i)}. \tag{4.17}$$

C_0 is equivalent to $Pr(Y = 1|\mathbf{X} \in G) - (1 - \beta_{s(i)}) > 0$, which is in the form of a set-average constraint. Then the branch and bound method is used to find the optimal classifier inside a chain and various permutations of the ordering are explored. Here, the risk function deals with all the points that are misclassified. The further a misclassified point is away from the threshold U the greater the risk will be. The risk function is formalized in (4.18) as,

$$R(f) = \sum_{s(i)} \sum_{\text{misclassified } j} |y_j - U| \tag{4.18}$$

which can be estimated through the training set as,

$$\widehat{R}(f) = \sum_{s(i)} \sum_{\text{misclassified } j} |\widehat{y}_j - U| \tag{4.19}$$

The misclassification penalty γ_i for every classifier inside the chain is separated for the points that are included in the set and the points that are excluded from the set. For all points that are inside the set ($Y = 1$), the risk of exclusion, r_i^+, is,

$$\gamma_i = r_i^+ = |y_i - U|1_{y_i > U} + \lambda_1 \tag{4.20}$$

while for the points that are excluded from the set ($Y = 0$) the risk of inclusion, r_i^-, is,

$$\gamma_i = r_i^- = |y_i - U| + \lambda_1 1_{y_i < L} \tag{4.21}$$

where $\mathbf{1}_{condition}$ is the indicator function. The risk of exclusion attempts to ensure that points that should be included in the set are included by making the risk higher for the points that are further away from the threshold U. The risk of inclusion, on the other hand, increases the risk for the points that are below the threshold L.

4.2.2 Large flow detector

The large flow detector problem is a direct application of the binary LSAT framework in [9]. For this problem, every input vector $\mathbf{X}$ has an associated output scalar value Z, which is

the number of bytes in the flow at completion. Here the user needs to specify L and U which are the lower bound threshold of a large flow and the upper bound threshold of a small flow respectively (in terms of bytes). Then the detector creates the largest set of large points possible while minimizing the number of small flows inside the set and adhering to a FDR constraint, ω — $Pr(Z < L|\mathbf{X} \in G) < \omega$.

The large flow detector has two constraints — $C_0 = \min_{\mathbf{x}\in G} E[Z|\mathbf{X} = \mathbf{x}] - U$ and $C_1 = Pr(Z > L|\mathbf{X} \in G) - \omega$. Re-writing this problem as an optimization problem, we now attempt to minimize the estimated risk function $\widehat{R}_0$,

$$\widehat{R}_0(G) = \sum_{\text{misclassified i}} |z_i - U| \tag{4.22}$$

with the temporary class label $D_{i,0} = 1_{Z_i>U}$ and the misclassification penalty $\gamma_{i,0} = |Y_i - U|$. For the second constraint, C_1, we assign a temporary class label $D_{i,1} = 1_{Z_i>L}$ and $\gamma_i = 1$. Combining these two constraints, we assign the label $D_i = 1$ if $Z_i > U$ and if $Z_i < L$ then $D_i = 0$. If $L < Z_i < U$ then a duplicate point (as described in the previous chapter) is created so this point will have a label of 0 and 1. The misclassification penalty for both classes is the same as the one found in (4.20) and (4.21).

Chapter 5

Results

In this chapter, we test our algorithms on a real-world 24-hour trace provided by a Canadian ISP. In the first section we discuss how the raw data collected from the trace was transformed and inputted into our algorithms. This involved, identifying the packets of a flow, collating the packets into a flow and finally calculating statistics about the flow. To evaluate the accuracy of our approach, we generated a ground truth which was simply a class label assigned to each flow. Finally, in Section 5.2, we present the results for our Internet traffic classifiers using a multiclass SVM as a baseline classifier for comparison. Our experiment consisted of training the classifier with one hour of trace data and evaluating the performance with the remaining 23 hours.

5.1 Data and processing

We obtained a traffic trace from OmniGlobe Networks, a Canadian ISP that specializes in providing satellite-link Internet service, corresponding to 24 consecutive hours of traffic on April 15 and 16, 2008. For this work, we only consider TCP flows. We define a flow as packets that have the same 4-tuple (source IP address, destination IP address, source TCP port, destination TCP port). For each TCP flow in the trace, we extracted a large number of features (or statistics). We also derived a notion of "ground truth" for the class and number of total bytes of a flow which will be the output of the Internet traffic classifier and the large flow detector respectively. The features and ground truth were used to train our classifiers, and to assess their performance in a validation step.

We used tcpdump [85] to capture up to the first 100 bytes for each packet including the

MAC, IP and TCP layer headers. We sliced our full trace into 24 single-hour traces, and for each hour of traffic, we identified the flows. We ignored flows which did not start or did not end within the considered hour. We verified that these flows crossing hour boundaries represented a negligible amount of traffic. Each flow was processed individually to extract the features and ground truth of the class and number of bytes.

To extract features from each flow, we use the tcptrace tool [86]. This tool can provide 142 different statistics for a flow, including the IP addresses and TCP ports for source and destination, given only the captured headers. The tool is also able to give the same set of statistics for truncated flows, that is, flows for which we only process the first k packets. This is important in our context, because we are striving to classify traffic in real-time. We exclude the 4 statistics that define a flow from the default 142 statistics returned by tcptrace and we use our feature selection tool (as discussed in Chapter 4) to find a subset of the remaining 138 statistics to train and evaluate our classifiers.

To associate an application to each flow, we use Bro, an intrusion detection tool capable of deep packet inspection [87]. The signatures we used to establish our ground truth can be found in Appendix A. Most of the signatures we used were based from the ones found in [88]. From these signatures we found four dominant applications inside our data trace. We show the complete application breakdown in terms of flows and bytes in Tables 5.1 and 5.2. In Table 5.1 we show the breakdown for all the flows in the 24 hour trace while Table 5.2 shows flows that have more than 6 packets. Note that about two thirds of the "Other" flows have fewer than six packets. Furthermore, over 80% of the "Other" class are smaller than 10 packets so these flows disappear from the network rather quickly. Typical applications that we found in the "Other" class included port scans and broken TCP connections. These type of flows are known as "background radiation" [89] as they represent fundamentally unproductive traffic on the network.

For this network, HTTP traffic is dominating the traffic mix. We also notice that the users of this network do not engage in P2P file exchange as P2P traffic is blocked by the ISP through a firewall. Figure 5.1 shows the number of flows that have more than six packets per hour in our data trace. Our data trace is from 2 p.m. on April 15 to 2 p.m. on April 16. Therefore it makes sense that hours 11 — 19 have fewer flows than the other hours as these hours correspond to midnight to 8 a.m. where most people are sleeping or not using the Internet. Figures 5.2 and 5.3 show the application breakdown per hour for flows that have more than six packets. The HTTP class has its own graph (Figure 5.2) due

to it having a greater number of flows than the other classes. Notice that in the night time hours (midnight to 8 a.m.) there are few MSN messenger, POP3 and HTTPS flows and the number of HTTP flows are drastically reduced compared to the other hours. For these hours, every misclassification has a greater impact on the FAR or FDR due to the reduced number of flows. In the next section, we evaluate our classifiers after the 6th packet for each flow has been received.

Table 5.1 Application breakdown

	Flows		Size	
Application	**Number**	**Percentage**	**GB**	**Percentage**
HTTP	341907	62.9%	4.4	75.4%
HTTPS	22934	4.2%	0.31	5.3%
MSN	3529	0.7%	0.04	0.7%
POP3	1466	0.2%	0.01	0.1%
OTHER	173719	32.0%	1.1	18.5%

Table 5.2 Application breakdown for flows > 6 packets

	Flows		Size	
Application	**Number**	**Percentage**	**GB**	**Percentage**
HTTP	315375	78.3%	4.1	74.6%
HTTPS	20736	5.2%	0.29	5.4%
MSN	3364	0.8%	0.04	0.7%
POP3	1311	0.3%	0.01	0.2%
OTHER	61870	15.4%	1.05	19.1%

5.2 Performance Evaluation

Our main goal is to assess whether we are able to provide performance guarantees on classes of interest while still achieving a high overall flow accuracy. We first reduce the input feature space so only relevant features are sent to our classifiers. To reduce the features in this work, we used the so-called "correlation-based with best first search (forward)" [90]. The previous chapter describes this algorithm while Section 5.2.1 provides more detail on how we selected the features. The resulting features for our data set are: (1) total number of

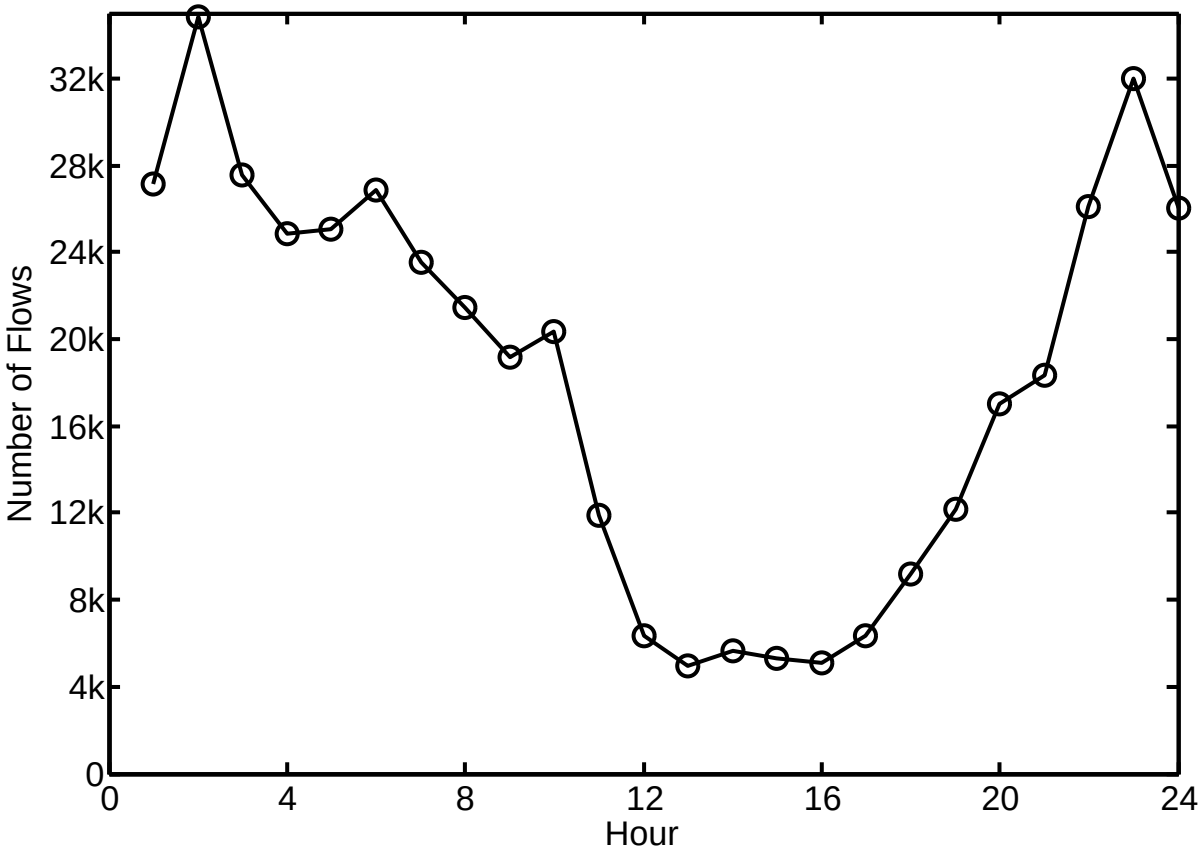

Fig. 5.1 The total number of flows that have more than six packets per hour for our 24 hour data trace.

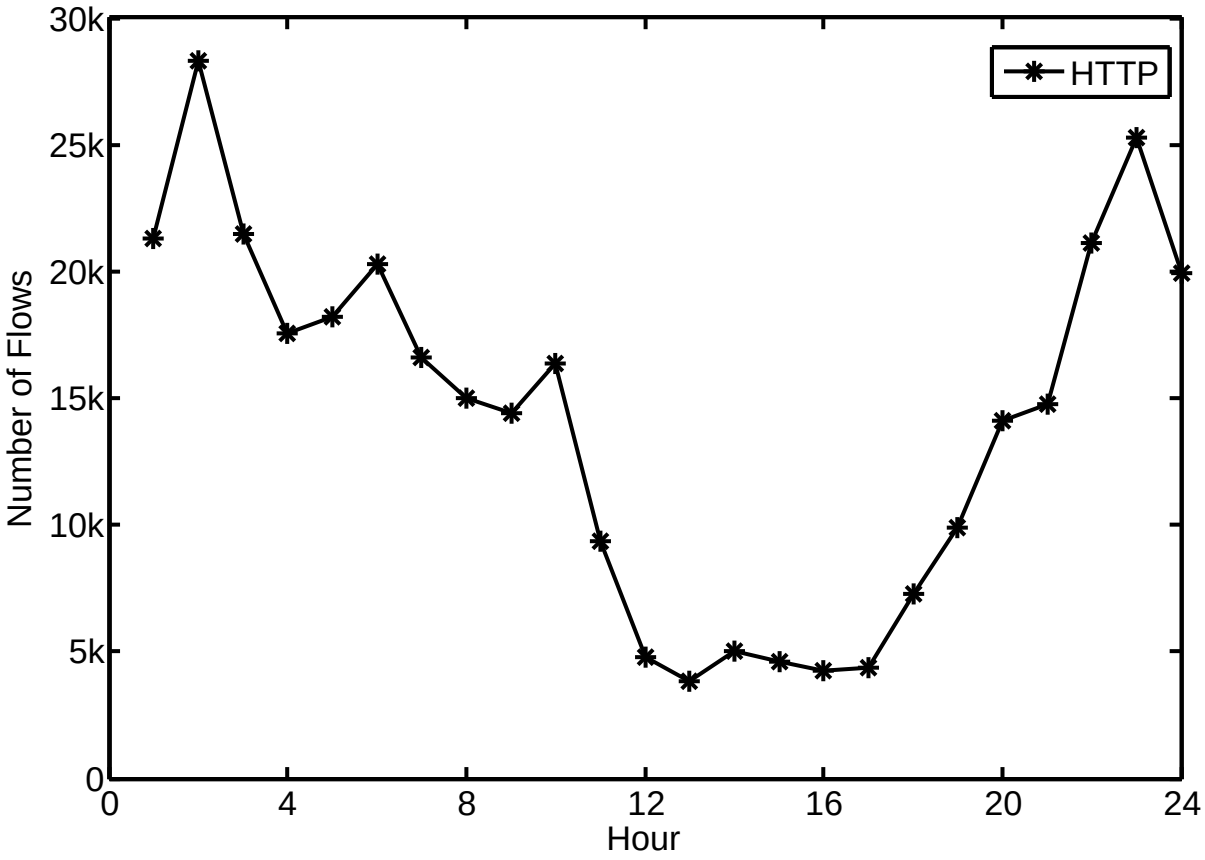

Fig. 5.2 The total number of HTTP flows that have more than six packets per hour for our 24 hour data trace.

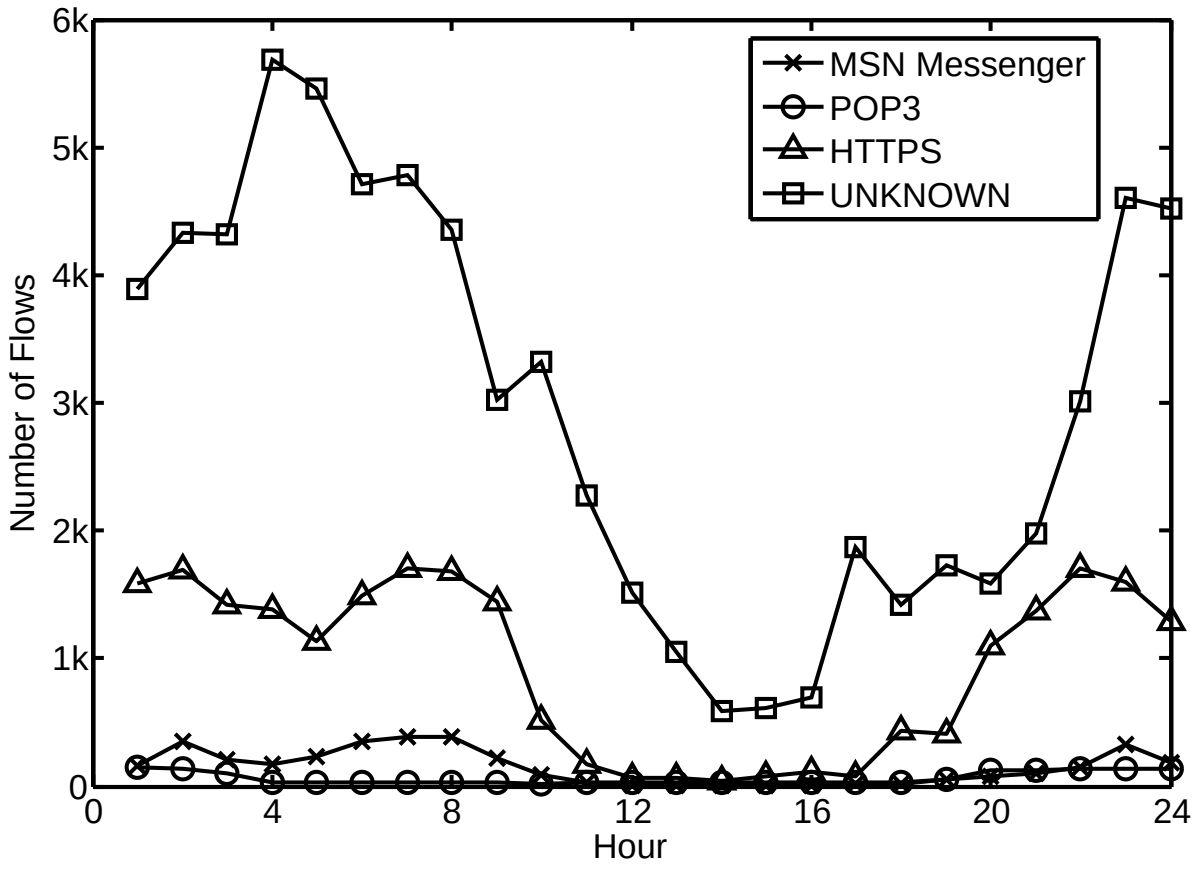

Fig. 5.3 The total number of MSN Messenger, POP3, HTTPS and unknown flows that have more than six packets per hour for our 24 hour data trace.

bytes sent from the client to the server; (2) number of packets with the FIN field set sent from the client to the server; (3) the window scaling factor used in packets from the client to the server; (4) total number of bytes truncated in the packet capture from the client to the server; and (5) total number of packets truncated in the packet capture from the server to the client (shown in Table 5.3). The fifth feature comes as a direct result of truncating our trace as we only recorded the first 100 bytes of every packet due to space limitations. We refer the reader to [86] for a full description of these features.

After the features are selected, we train our classifiers, choosing the best-performing parameters. These parameters include ν_+, ν_-, σ, λ and the order of the binary classifiers in the classifier chain. We use 1000 randomly chosen flows for the training set (chosen from hour 1) and we start classification after six packets of a flow are detected (parameter p).

For all our work, we will be focusing on classifying the four dominant applications in our data set: HTTP, HTTPS, MSN messenger and POP3. All other flows will be classified as "Other". To emulate a real-time environment, we ignored flows smaller than the milestone of p packets (since such flows are never classified).

5.2.1 Feature Selection

To select the features relevant to our traffic classification problem, we used a correlation based feature selection algorithm found in the WEKA toolbox [42] (discussed in Chapter 4.2.1). A ten-fold cross-validation technique was applied, selecting only the features identified in all ten-folds as being relevant. The ten-fold cross validation partitioned hour one of our data set into ten validation sets and the feature selection algorithm we chose was applied to each set. Table 5.3 shows the features that were selected after six packets of a flow were received. For our data set, features rendered from the client to server statistics (a2b) dominate the final feature space. Intuitively, this makes sense because in a flow the majority of client packets are simple content requests which diverge slightly from one packet to the next regardless of the content. For example, the server to client packets sent by a multimedia server may differ greatly from a file transfer server but the client to server packets are relatively similar in both cases. Therefore, application profiling using client to server statistics is more consistent hence the reasoning for why they dominate the final feature space.

Table 5.3 Features selected for the different packet milestones. Note that 'a2b' means it is a client to server statistic while 'b2a' is a server to client statistic.

Features
actual_data_bytes_a2b
FIN_pkts_sent_a2b
adv_wind_scale_a2b
truncated_data_a2b
truncated_packets_b2a

5.2.2 Classification with FAR constraints

Our first experiment explores the performance of our algorithm for traffic classification under FAR constraints. In this experiment, the flows comprising the training set were selected randomly from the first hour (1000 in total) and the remaining 23 hours were used as test data. We compare the performance of three classifiers: (i) a baseline multiclass SVM classifier, as described in [67]; (ii) our proposed FAR-constrained classifier with a single-class FAR constraint on the HTTP class ($\alpha\{HTTP\} = 0.4\%$) and (iii) our proposed FAR-constrained classifier with a pairwise FAR constraint on the HTTP and HTTPS classes ($\alpha\{HTTPS, HTTP\} = 0.05\%$). Since there were only 4 classes, our algorithms searched over all possible orderings ($4! = 24$) of the classifiers in the binary chain.

We initially evaluate performance based on the remaining flows from the first hour, for which the classifier is best matched. The baseline classifier achieves the highest overall accuracy (lowest misclassification rate) at 98.5%, but the accuracies of the FAR-constrained classifiers are only marginally lower at 97.7% and 97.6%, respectively. In terms of single-class FAR for the HTTP class, the baseline classifier has an FAR of 3.7%, whereas the single-class FAR-constrained classifier reduces this below the threshold to 0.3%. The pairwise FAR constrained classifier achieves a pairwise FAR of 0.02%, which is at the threshold of 0.02%; the baseline classifier is at 0.07%.

Figures 5.4- 5.6 displays the results of the experiment for hours 2-24. As indicated in Figure 5.4, the accuracy of the classifiers varies over the different periods, ranging from 89.3% to 99.1%, but no classifier achieves a consistently superior accuracy. Figure 5.5 shows that our proposed single-class FAR constrained classifier consistently achieves a FAR below 1.0%, whereas the FAR for the baseline multi-class SVM is much larger and reaches

values as large as 30%. Our proposed classifier does not always achieve a FAR below the specified constraint of 0.4%, which may be partially due to discrepancies between the first hour of traffic and the remaining 23 (the training data may not sufficiently represent the later flows). Figure 5.6 examines the pairwise HTTPS-HTTP FAR, and indicates that the pairwise constrained classifier achieves a pairwise FAR consistently lower than that of the baseline multi-class SVM classifier; again the pairwise FAR constraint is not always met.

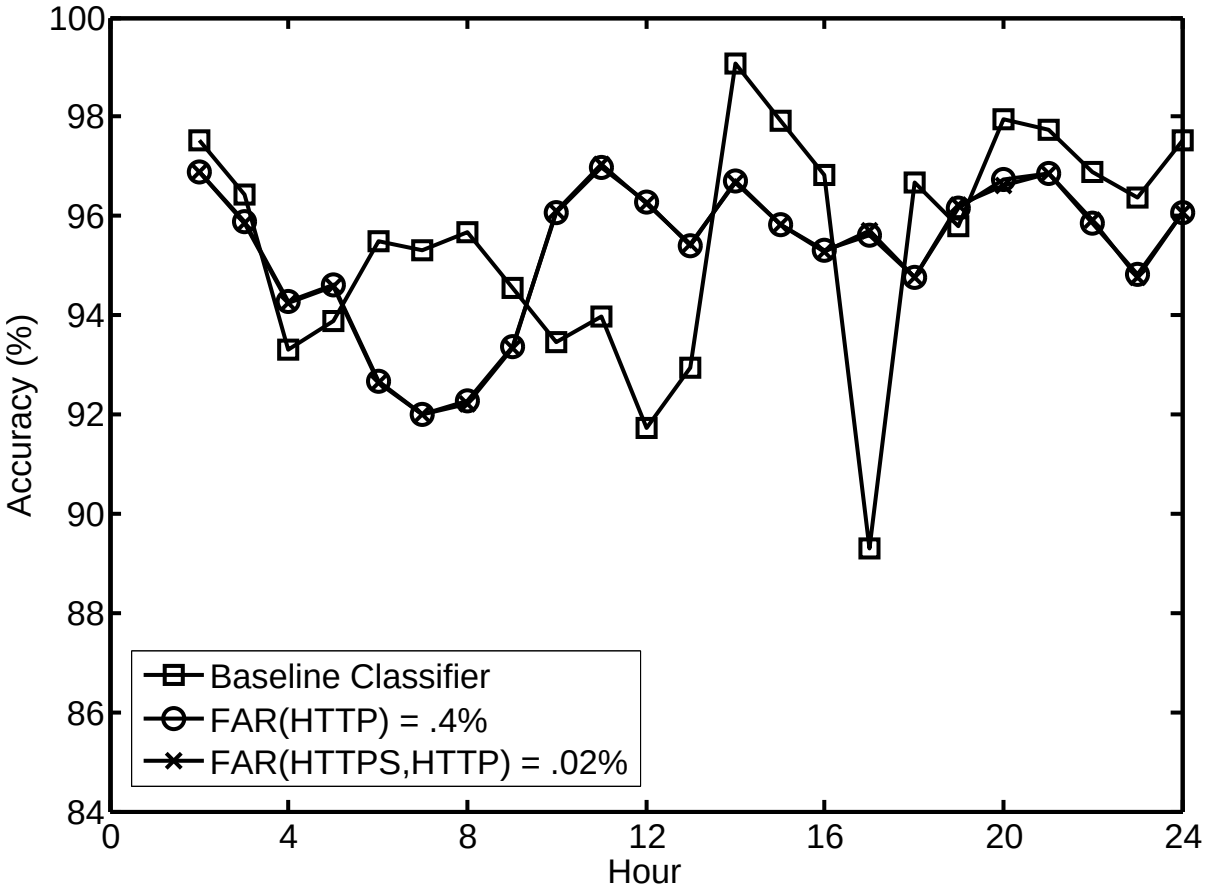

Fig. 5.4 The overall accuracy for hours 2-24 for the three classifiers discussed.

5.2.3 Classification with FDR constraints

Our second experiment examines the performance of our proposed algorithms for traffic classification under FDR constraints. We compare the performance of three classifiers:

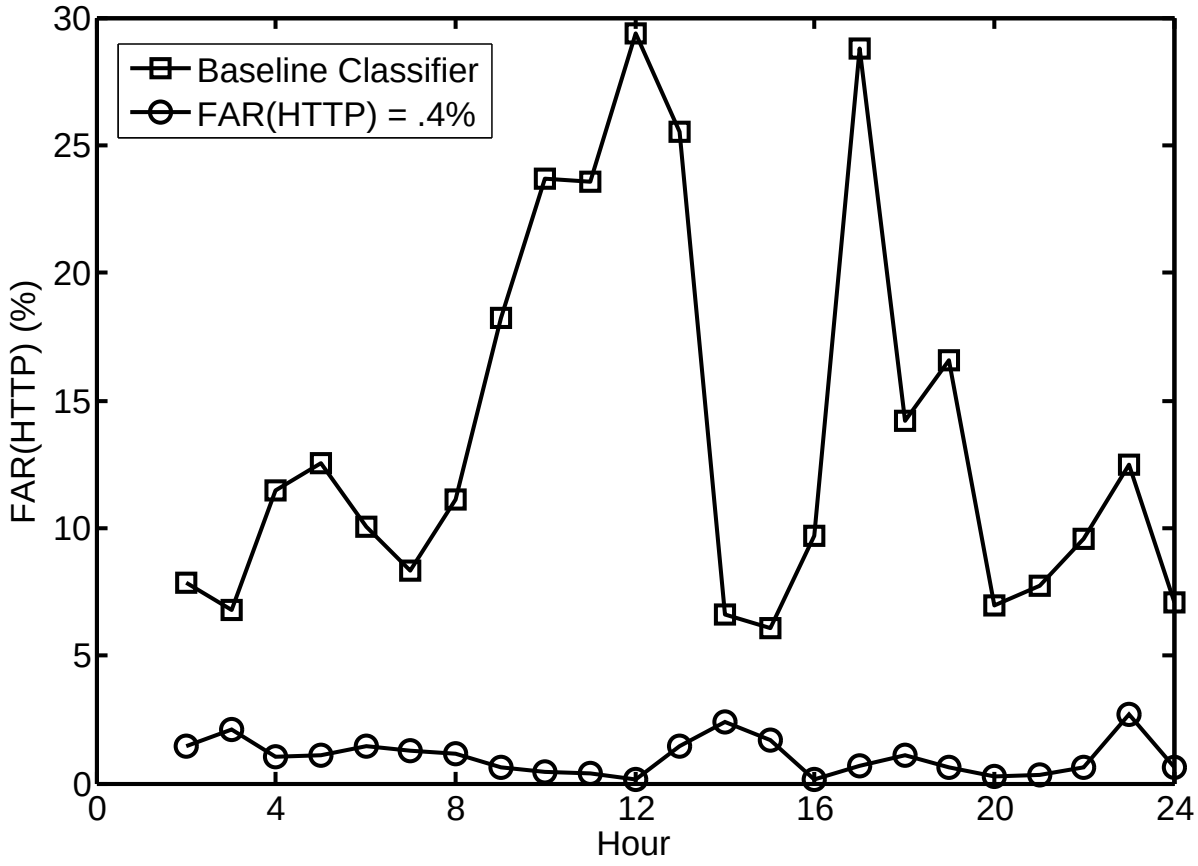

Fig. 5.5 The False Alarm Rate (FAR) of HTTP for the baseline classifier and for the NP classifier when the FAR for HTTP is set to .4%.

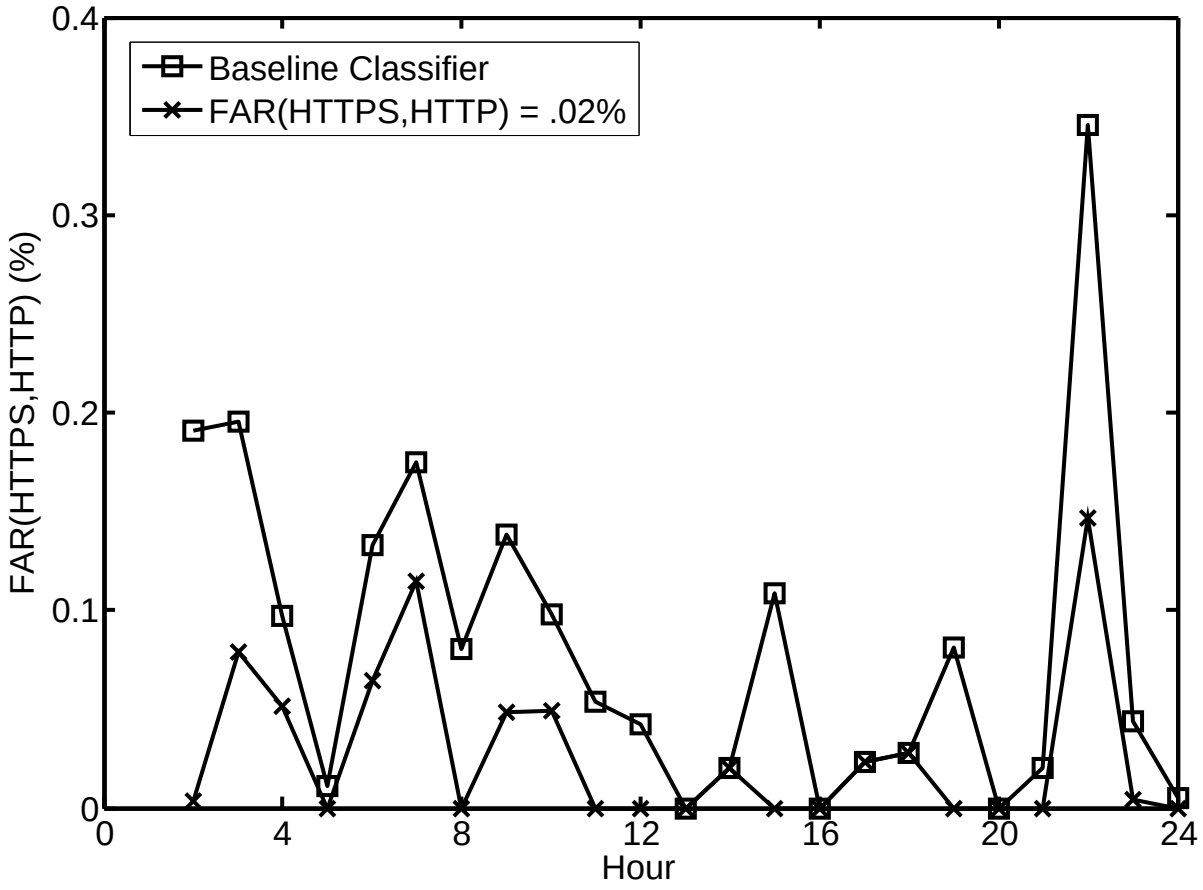

Fig. 5.6 The pairwise FAR for HTTP flows being misclassified as HTTPS for the baseline classifier and the NP classifier where the FAR{HTTPS,HTTP} is set to .02%.

(i) the baseline multiclass SVM classifier; (ii) an unconstrained binary-chain classifier (in which we set $\beta = 100\%$ for all classes) and (iii) our proposed FDR-constrained classifier with a single class FDR constraint on the HTTPS class ($\beta\{HTTPS\} = 5\%$). Training is again performed using 1000 flows randomly selected from the first hour. We conduct performance evaluation using the remaining flows in the first hour, and then examine the stability of the performance over the remaining 23 hours.

For the first hour, the baseline multiclass SVM classifier achieves the highest overall accuracy at 98.5%, but the accuracies of the other two classifiers are only marginally lower, with the unconstrained binary-chain classifier achieving 98.0% and the FDR-constrained classifier achieving 97.9%. The FDR for the remaining flows of the first hour for the unconstrained binary chain classifier is 7.0% while with our proposed classifier we were able to reduce the FDR to 4.2%.

Figures 5.7 & 5.8 shows the results of the experiment for hours 2-24. In Figure 5.7, the overall accuracies for all three classifiers are comparable, although the accuracy of the multiclass SVM drops significantly for hours 10-13 and hour 17. In Figure 5.8, the FDR achieved by our proposed constrained classifier is comparable to the baseline multiclass SVM classifier and significantly lower than that of the unconstrained binary chain classifier (which exceeds 20% for several hours and reaches 44.6% for hour 12). The FDR-constrained classifier meets the specified constraint (or marginally exceeds it) for hours 1-2 and 8-24, but the FDR for hours 3-7 all exceed 10%. This indicates that there may be a need to train several classifiers for different periods of the day. The results do indicate that a binary-chain classifier can achieve performance comparable to that of a multiclass SVM, with the advantage that it is significantly more scalable as the number of classes increases.

5.2.4 Online classification complexity for the Internet Traffic Classifiers

One other area that needs to be addressed is whether our classifiers can predict incoming flows fast enough to function in an online environment. Running a single SVM for a sample flow requires extracting the features x, and computing the quantity $h(x)$ as in (3.7). Note that the dot-product $\langle w, x\rangle_{\mathcal{H}}$ is actually computed through a kernel evaluation, i.e. $\langle w, x\rangle_{\mathcal{H}} = k(w, x)$. Evaluating (3.7) thus requires summation of the results of N_v (where N_v is the number of support vectors of the considered SVM) kernel evaluations. The full classifier may require to go through up to c SVMs, leading to a computational cost of $N_v c$

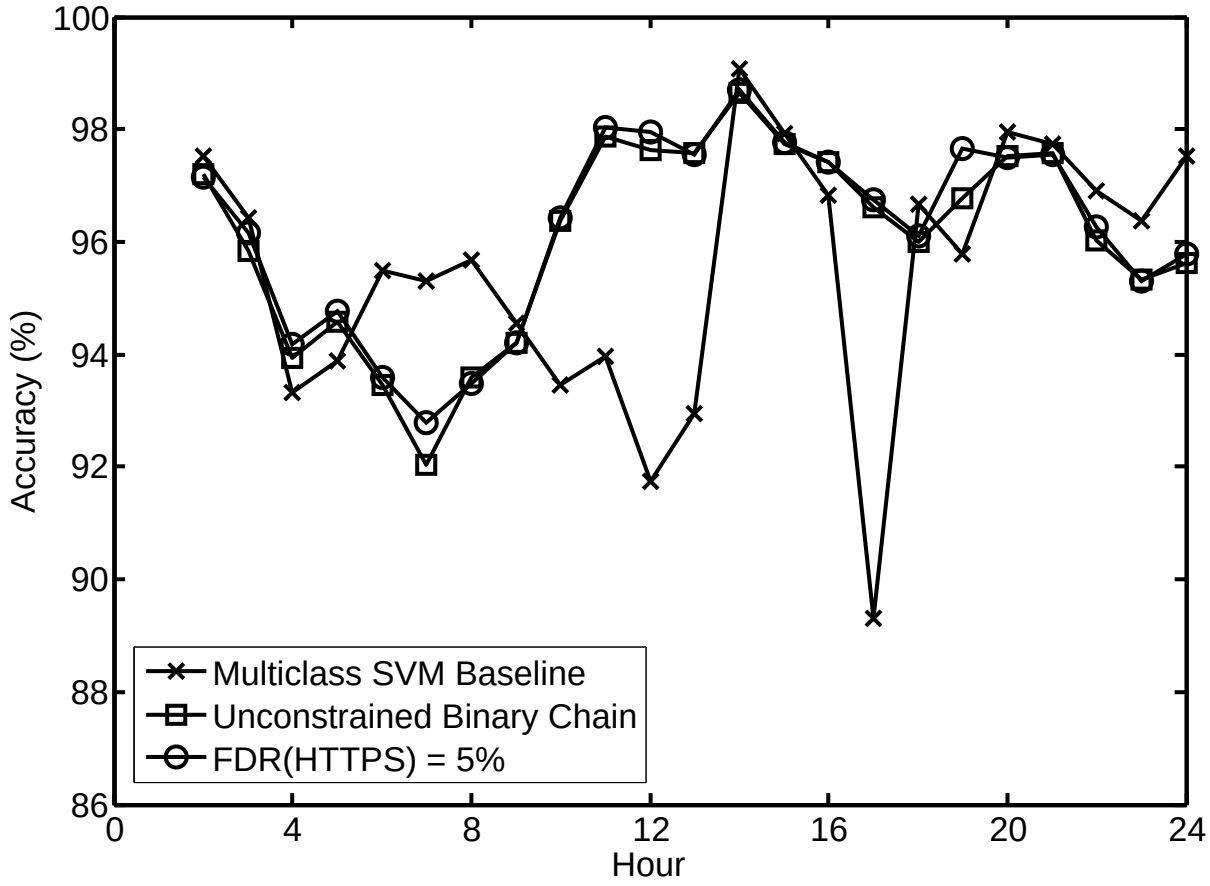

Fig. 5.7 The overall accuracy for hours 2-24 for the three classifiers discussed.

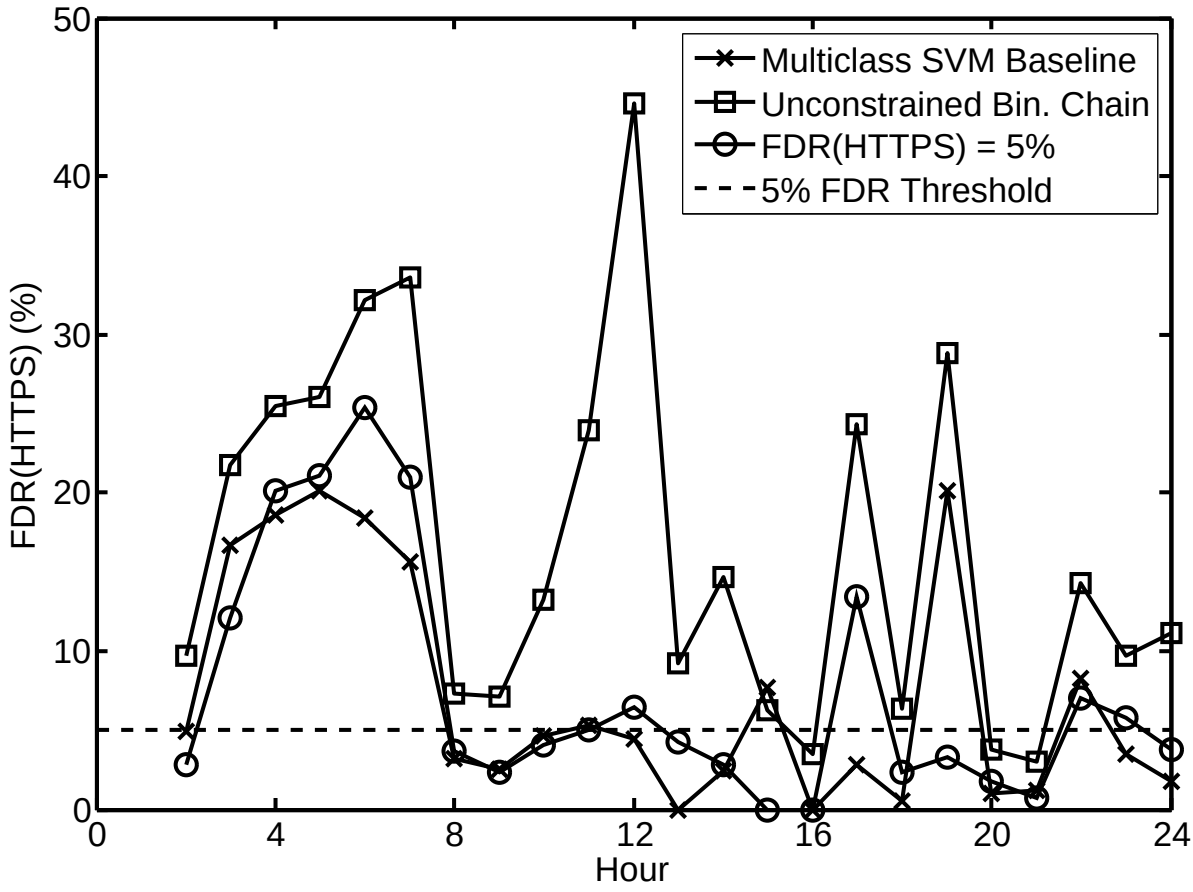

Fig. 5.8 The False Discovery Rate (FDR) of HTTPS for the baseline multiclass SVM classifier, unconstrained LSAT binary-chain classifier and for the FDR-constrained classifier where the FDR for HTTPS is set to 5%

kernel evaluations. In our simulation work, N_v was typically in the order of a few hundreds and $c = 4$, leading to the computation of a few thousands kernel functions — chosen here to be the exponential of the norm of a d-dimensional vector ($d = 5$). We observe that adding classes to the classifier only increases the computational complexity linearly. In our experiments, classifying a flow took around 1 ms on a modern general purpose computer. This indicates that the flow classification problem is well-suited to implementation in high-speed routers with dedicated hardware.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we provided a network operator valuable resources for network operations management. Specifically we proposed two Internet traffic classifiers with *performance guarantees* and a large flow detector. These applications will help the network operator better manage their network. By being able to assess what type of Internet traffic is passing through their network the operator can now perform actions on the traffic to improve the overall efficiency (or performance) of the network.

In the second chapter, we present a literature review of all the current Internet traffic techniques. The current techniques can be divided into three main categories — Port-based, Deep-Packet Inspection and Shallow-Packet Inspection. Port-based Internet traffic classification is the simplest type of classification as an application is mapped to the application associated to the port number it is using based on the known ports list from the Internet Assigned Numbers Authority. The drawback to this approach is that there is no one to enforce that an application uses the correct port number — an application just needs to use a different port number to avoid detection. Another problem is that some applications use a different application to operate (for example, MSN messenger can be run over the HTTP protocol and uses HTTP's assigned port number, 80) and port-based classification is not able to distinguish between the two applications. The next type of Internet traffic classifier, Deep-Packet Inspection, searches the payload of a packet for a pattern (or signature) to identify the type of application. This approach has shown to be effective — looking into the payload for application-specific messages generally produces a high accuracy in classifying

Internet traffic. Deep-Packet Inspection has drawbacks though, as if the payload of the packet is encrypted this process is rendered ineffective as there is not much DPI can do if it can not look inside the payload. Also, Deep-Packet Inspection is in a bit of a gray area at the moment because of legal and privacy concerns concerning who has the right to look inside someone's payload, especially if an Internet Service Provider uses this information to hinder a user on their network (for example, throttling the bandwidth of people using P2P). The final category of Internet traffic classification is Shallow-Packet Inspection where flow statistics or the behaviour of a flow are used for classification. This approach is less intrusive than the Deep-Packet Inspection because the only information from the packet header or how the flow behaves on the network is used. Currently there are many different techniques used to classify Internet traffic using Internet traffic. They include clustering, Support Vector Machines (SVMs) and various machine learning techniques such as C4.5 decision trees, Naïve Bayes and k-Nearest Neighbour (k-NN). For behavioural-based Internet classifiers, the classifiers looks at the behaviour of flows and hosts on the network. For example, if they find a web (HTTP) server then all flows that are associated with this web server would be classified as HTTP. While most of the Deep and Shallow-Packet Inspection are able to achieve a high overall accuracy, none are able to provide a hard performance guarantee on a class. This is important, as in practice, not all classes have the same level of importance for network operators. Therefore, while it is great to have a high overall accuracy, there are times where ensuring a particular class meet certain performance guarantees is more beneficial for the user. With that in mind, we provide two Internet traffic classifiers that provide false alarm rate and false discovery rate constraints.

In Chapter 3, we provide background information about the algorithms that are used for the Internet traffic classifiers and the large flow detector. The Internet traffic classifier with false alarm constraints is implemented by a 2ν-SVM while the Internet traffic classifier with false discovery constraints and the large flow detector are implemented by the Learning to Satisfy (LSAT) framework which can be extended from the 2ν-SVM. In order to better understand these two algorithms, we first explain what a Support Vector Machine is and how it is implemented. Then we show how a SVM can be extended to handle false alarm constraints (2ν-SVM). Finally, we show how the 2ν-SVM can be used for the Learning to Satisfy framework which allows the user to set false discovery constraints.

In Chapter 4, we first formulated mathematically the three problems we are trying to solve. For the two Internet traffic classifiers, the goal is to minimize the overall misclas-

sification while adhering to false alarm (or false discovery) rate constraints. For the large flow detector, the goal is to find the largest set of large flows that satisfies the constraints set. We showed how the algorithms we discussed in the previous chapter are implemented to fit our problem. For the Internet traffic classifiers, we discussed how we transformed the binary classifiers to become multi-class classifiers. We proposed a novel methodology where we used a chain of binary classifiers to classify the Internet traffic. Each binary classifier in the chain is responsible for identifying whether a flow belongs to a particular application class. The first classifier in the chain that identifies a flow as belonging to its class is what the flow is classified as. This approach was chosen over other approaches that transform binary classifiers to a multi-class classifiers as we found this is the best method to control our performance guarantees. We also discussed the risk functions used to assess both classifiers. Finally, we discussed how the large flow detector is implemented using the binary LSAT framework. This involves setting thresholds on what a large and small flow is and specifying the false discovery rate threshold.

In Chapter 5, we presented our results from experiments on a 24-hour trace from a Canadian ISP. We first discussed how we processed the data trace to extract the features we need. We then showed the application breakdown of our data trace which had four major classes: HTTP, HTTPS, MSN Messenger and POP3. In the next section, we evaluated our Internet traffic classifiers on this Internet traffic trace. We first used a feature selection tool so only relevant features are inputted to the classifiers. Then we trained the classifiers on the first hour of the trace and tested the classifiers on the remaining 23 hours. We also used a multi-class SVM and an unconstrained binary chain classifier (for LSAT only) as a baseline classifier for our classifiers. From our results, we found that our classifiers had a comparable accuracy to the baseline classifiers but generally was able to outperform them on the performance guarantees that we were measuring.

6.2 Discussion

From the results, we see that there were limitations with the data set that we used. Unfortunately, the data set that we processed had only four dominant application present so it was difficult to test our classifiers extensively. For instance, having more time sensitive traffic (i.e., VoIP or video streaming) inside the data set allows us to examine how our classifiers can perform as the first step in prioritizing Internet traffic for QoS. Another

traffic class that was missing was Peer to Peer (P2P) traffic which is of particular interest to Internet Service Providers due to the high volume of traffic this class occupies in most networks. Therefore, while our classifier was able to identify the applications in our data set well, further tests are required to see how our classifiers handle a broader range of applications.

Another issue with this data set was that we were unable to perform any large flow detector experiments with it. The reason is that the flows inside the data set are too similar to each other in size. For example, if we decided that the top 20% of the flows (in terms of size) inside this network were to be considered large then we are dealing with flows that are 7 kB and greater. With such a small threshold in terms of size for a flow to be considered large, it is difficult for a detector to distinguish between the two classes (only a couple of bytes distinguish between a small flow and a large flow). Generally, the flow characteristics of a 7 kB flow are not much different that the flow characteristics of a 5 kB flow. Also increasing the threshold of what constitutes a large flow (to say only the top 10% of flows in terms of size) does not help either because then there are too few flows inside the large class to train on. Just as we needed a data set with a diverse set of applications, we also need a data set that has a more broad distribution of flows in terms of size.

Overall, from our results we see that our classifier is aptly capable of distinguishing between different types of applications while being able to provide performance guarantees. If the training set properly represents all the network traffic our classifiers are able to provide performance guarantees on unknown data that enters our classifiers. This in contrast to the baseline classifiers that we used as they false alarm rate (or false discovery rate) fluctuated hour by hour depending on the traffic mix. Therefore our classifiers are then beneficial for network operators as by having these performance guarantees in place for applications they can more confidently perform actions on unknown traffic.

One final aspect of our classifiers that we wish to discuss is whether it is feasible to use our classifiers in a real-life setting. Our experiments consisted of capturing a network traffic trace and inputting our classifiers flows from this trace. There is a bigger time constraint if the flows need to be classified in real-time on a router. To provide some insight to this, refer to Figure 5.1 which shows that at most, there are 35 000 flows in a hour (or roughly 10 flows per second). Recall from Section 5.2.4, that online classification can be done in under 1 ms. Therefore, if we have dedicated hardware that is in the path (or parallel) of

the router our classifiers can handle 3.6 million flows per hour[1]. Therefore our classifiers are more than capable of handling the traffic load from the network where we collected our data. The network that we collected our data from is from a rural area in Canada though so the network load is not that high. For networks that have a higher capacity, there are ways to increase the number of flows that the classifiers can handle. For one, we can parallelize the classifiers in one of two ways. One way is to have every classifier in the chain run in parallel. The alternative is to have multiple classifiers running at the same time so that multiple flows can be handled at the same time. Either method increases the throughput of the classifiers and allows the classifiers to scale to larger networks. Also it is important to remember that the large flow detector can have a greater impact on larger network as the detector can reduce the load of the classifiers by filtering out the smaller flows of the networks.

6.3 Future Work

Future work with our classifiers is to create a hybrid classifier of both of our classifiers that were proposed. The hybrid classifier's chain of classifiers consists of classifiers with both false alarm and false discovery constraints. This allows the classifier to be more practical for a real-life setting as this classifier can handle applications that require false alarm constraints (i.e. applications that need to be prioritized) and applications that require false discovery constraints (i.e. harmful applications that need to be limited or blocked). Creating a hybrid classifier requires a new risk function to assess the performance of the classifier. Currently, the risk functions of the classifier with FAR constraints and FDR constraints measure two entirely different types of risk so a new risk function is required to incorporate the goals of both classifiers — minimizing the overall misclassification while adhering to the FAR constraints for the FAR-constrained classifier and creating the largest set of a given application while adhering to FDR constraints for the FDR-constrained classifier.

In this thesis we briefly examined the temporal stability of our classifiers but further work is required to examine the temporal and spatial stability of our classifiers. From our results we saw that training on one hour can provide accurate results for the next 23 hours. Future work can examine the temporal stability to see how far in the future our

[1] $1000 \frac{flows}{sec} \times 60 \frac{sec}{min} \times 60 \frac{min}{hour}$

classifier can still provide accurate results. Alternatively, future work can examine the spatial stability of our classifiers which is to see if our classifiers that are trained on one network can accurately classify traffic on another network. The more temporal and spatial stability that our classifiers exhibit means the less time the classifier needs to be re-trained which is advantageous characteristic for any classifier to have.

Future work can also try to implement our classifiers in a real-time setting rather than just collecting flows from a trace and simulating an online setting so as to see the new challenges this poses. Classifying traffic in an online setting brings up new problems. The first thing that needs to be decided is where to place the classifier in the network. For instance, it is logical to put the classifier on the router but then it needs to be verified if the router has the hardware capabilities to accommodate the classifier. If not, the classifier needs dedicated hardware placed before or after the router in the network path. Another solution is for the classifier to be in parallel with the router, where the traffic from the router is mirrored to the classifier. Then the classifier can send back its results to the router and the router can take the appropriate action. Another issue that arises in a real-time setting is dealing with live network traffic. In any network, packets can be dropped or received out of order. Future work then needs to decide how our classifiers handle these issues. As we stated earlier, the classifiers provide classification after the first p packets of a flow are received. If the p packets received are not actually the first p packets of a flow this can skew the statistics collected from the flow as the same sequence of packets are not being used every time. Therefore future work needs to investigate whether using out of order packets radically effects the classifier's performance.

Appendix A

Signatures Used for Bro

```
signature skype_bootstrap {
 ip-proto == udp
 src-ip == local_nets
 dst-ip != local_nets
 src-port >= 1024
 src-port <= 65535
 dst-port == 33033
}

signature udp1 {
    ip-proto == udp
    event "UDP" }


signature bittorrent_id {
    payload/.*(BitTorrent|BT_CHOKE|BT_UNCHOKE|BT_UNINTERESTED|BT_HAVE|BT_BITFIELD|
        BT_REQUEST|BT_PIECE|BT_CANCEL|BT_HAVE|BT_KEEP_ALIVE|AZ_PEER_EXCHANGE)/
    event "BitTorrent" }

signature http_id {
```

```
    ip-proto == tcp
    payload /.*(HTTP|GET.\/|POST |HEAD |HTTP\/1|GET )/
    event "HTTP" }

signature bb_id {
    dst-port = 1984
    payload /(server|ack|page)/
    event "BB" }

signature directconnect_id {
    payload /\$(Send|Get|Dir|ConnectT|Supports|Hello|MyINFO|Search|MyNick|Quit|
        Key|RevConn|Version |Lock|HubName)/
    event "DirectConnect" }
signature edoneky_id {
    ip-proto = tcp
    payload /(\xe3|\xc5)/
    event "eDonkey" }

signature ftp_id {
    ip-proto = tcp
    dst-port = 21
    payload /.*(FTP)/
    event "FTP" }

signature ftp2_id {
    dst-port = 21
    payload /.*(PASS|USER|CWD|PASV|PORT|250 OK|220)/
    event "FTP" }

signature gnutella_id {
    payload /GNUTELLA CONNECT/
    event "Gnutella" }
```

```
signature gotomypc_id {
    payload /GET \/jedi\?reques/
    event "GoToMyPC" }

signature kazaa_id {
    payload /.*KazaaClient/
    event "Kazaa" }

signature icq_id {
    dst-port = 5190
    payload /.*ICQ/
    event "ICQ" }

signature ident_id {
    dst-port = 113
    payload /[0-9]*,.*25/
    event "IDENT" }

signature imap_id {
    dst-port = 143
    payload /.*(CAPABILITY|LOGIN|login)/
    event "IMAP" }

signature jetdirect_id {
    dst-port = 9100
    payload /.*(PJL.SET.PAGEPROTECT=OFF|PJL.JOB)/
    event "JetDirectProtocol" }

signature msnmessenger_id {
    dst-port = 1863
    payload /.*(CAL|JOI|XFR|RINGING|USR|ANS|VER|MSG|QRY|CHL|NLN|ILN|CHG|LST|INF)/
    event "MSN" }
```

```
signature msnwebcam_id{
    payload /recipientid=[0-9]*&sessionid=[0-9]*/
    event "MSNWEBCAM" }

signature mssql_id {
    dst-port = 1433
    payload /.*(\0S\0E\0R\0V\0E\0R|\0S\0Q\0L)/
    event "MSSQL" }

signature mysql_id {
    payload /.*\x03(SELECT|select|INSERT|insert|SHOW|show|UPDATE|update)/
    event "MySQL" }

signature nntp_id {
    dst-port = 119
    payload /.*(mode.stream|MODE.STREAM|CHECK <|TAKETHIS <|check <|takethis
        <|LISGROUP|ARTICLE |\x0d\x0a=ybegin |mode.reader|MODE.READER)/
    event "NNTP" }

signature otherp2p_id {
    payload /.*(LimeWire|BearShare|Gnucleus|Morpheus|XoloX|gtk-gnutella|Mutella|
        MyNapster|Qtella|AquaLime |NapShare|Comback|PHEX|SwapNut|FreeWire|Openext|
        Toadnode|GnucDNA|morph500|morph460|Shareaza)/
    event "P2P" }

signature otherp2p2_id {
    payload /.*(CONNECT BACK)/
    event "P2P" }

signature otherp2p3_id {
    payload /.*GIV.*(mp3|avi|mpg|zip|iso|img|rar|file)/
    event "P2P-other" }
```

```
signature pop3_id {
    dst-port = 110
    payload /.*(POP3|Mail|mail|\+OK|ok|Ok|sender|recipient|RCPT TO|INBOX|DONE|\* OK|
        USER|PASS|APOP |AUTH|CAPA|STAT)/
    event "POP3" }

signature real_id {
    dst-port = 3077
    payload /.*GET/
    event "GETSon3077" }

signature rtsp_id {
    dst-port = 554
    payload /.*(rtsp)/
    event "RTSP" }

signature samba_id {
    dst-port = 873
    payload /.*RSYNCD/
    event "Samba" }

signature sip_id {
    payload /.*(REGISTER|INVITE).*SIP/
    event "SIP" }

signature smtp2_id {
    ip-proto = tcp
    payload /^(ELHO|elho|HELO|ELHO|EHLO|ehlo)/
    event "SMTP" }

signature spamassassin_id {
    dst-port = 2703
    payload /.*(cn=razor|a=(c|g)\x26|-nsl)/
```

```
    event "SpamAssassin" }

signature ssh_id {
    dst-port = 22
    payload /.*SSH/
    event "SSH" }

signature vnc_id {
    dst-port = 5900
    payload /.*RFB/
    event "VNC" }

signature z3950_id {
    payload /.*(Mike Taylor|Net::Z3950.pm|MetaStar Search SDK|BookWhere)/
    event "Z3950Client" }

signature afs3callback_id {
    dst-port = 7001
    event "AFS3" }

signature locsrv_id {
    dst-port = 135
    event "loc-srv" }

signature ymsg_id {
    payload /^(YMSG)|.*\<Ymsg/
    event "YMSG" }
```

References

[1] T. Karagiannis, A. Broido, and N. Brownlee, "Is P2P dying or just hiding?" in *Proc. IEEE GLOBECOM*, Dallas, TX, USA, Nov. 2004.

[2] "IANA port numbers." [Online]. Available: http://www.iana.org/assignments/port-numbers

[3] T. T. T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[4] C. D. Scott and R. D. Nowak, "A Neyman-Pearson approach to statistical learning," *IEEE Trans. Inform. Theory*, vol. 51, no. 11, pp. 3806–3819, 2005.

[5] D. Casasent and X. W. Chen, "Radial basis function neural networks for nonlinear Fisher discrimination and Neyman-Pearson classification," *Neural Networks*, vol. 16, no. 5, pp. 529–535, 2003.

[6] A. Cannon, J. Howse, D. Hush, and C. Scovel, "Learning with the Neyman-Pearson and min-max criteria," Los Alamos National Laboratory, Tech. Rep., Jun. 2002, lA-UR 02-2951.

[7] T. Landgrebe and R. Duin, "On Neyman-Pearson optimisation for multiclass classifiers," in *Proc. Pattern Recognition Assoc. of South Africa*, Langebaan, South Africa, Nov. 2005.

[8] M. A. Davenport, R. G. Baraniuk, and C. D. Scott, "Controlling false alarms with support vector machines," in *Proc. Int. Conf. Acoustics, Speech, and Signal Proc. (ICASSP)*, Toulouse, France, May 2006.

[9] F. Thouin, M. J. Coates, B. Erikkson, R. Nowak, and C. Scott, "Learning to Satisfy," in *Proc. Int. Conf. Acoustics, Speech, and Signal Proc. (ICASSP)*, Las Vegas, NV, USA, Apr. 2008.

[10] D. Sab, S. Hauger, and M. Köhn, "Architecture and scalability of a high-speed traffic measurement platform with a highly flexible packet classification," *Computer Networks*, vol. 53, no. 6, pp. 810–820, 2009.

[11] P.-C. Wang, "Scalable packet classification with controlled cross-producting," *Computer Networks*, vol. 53, no. 6, pp. 821–834, 2009.

[12] C. Kreibich and J. Crowcroft, "Honeycomb — creating intrusion detection signatures using honeypots," in *Proc. Hot Topics in Networks*, Boston, MA, USA, Nov. 2003.

[13] X. Li, B. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina, "Detection and identification of network anomalies using sketch subspaces," in *Proc. ACM Internet Measurement Conf.*, Rio de Janeiro, Brazil, Oct. 2006.

[14] M. Marsono, M. El-Kharashi, and F. Gebali, "Targeting spam control on middleboxes: spam detection based on layer-3 email content classification," *Computer Networks*, vol. 53, no. 6, pp. 864–881, 2009.

[15] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, "McPAD: A multiple classifier system for accurate payload-based anomaly detection," *Computer Networks*, vol. 53, no. 6, pp. 864–881, 2009.

[16] D. Moore, K. Keys, R. Koga, E. Lagache, and K. Claffy, "Coralreef software suite as a tool for system and network administrators," in *Proc. Usenix Sys. Admin.*, San Diego, CA, USA, Dec. 2001.

[17] E. P. Freire, A. Ziviani, and R. M. Salles, "On metrics to distinguish Skype flows from HTTP traffic," in *Proc. Latin American Network Operations and Management Symp.*, Petropolis, Brazil, Sep. 2007.

[18] J. Borland, "RIAA threat may be slowing file swapping." [Online]. Available: http://news.cnet.com/2100-1027-1025684.html

[19] "Pew internet & american life project. sharp decline in music file swappers: Data memo from pip and comscore media metrix," Jan. 2004. [Online]. Available: http://www.pewinternet.org/Reports/2004/Sharp-decline-in-music-file-swappers.aspx

[20] A. Madhukar and C. Williamson, "A longitudinal study of P2P traffic classification," in *Proc. IEEE/ACM Modeling, Analysis, and Simulation of Computer and Telecom Sys.*, Monterey, CA, USA, Sep. 2006.

[21] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of P2P traffic using application signatures," in *Proc. World Wide Web Conf.*, New York, NY, USA, May 2004.

[22] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proc. Passive and Active Measurement Workshop*, Boston, MA, USA, Apr. 2005.

[23] "QOSMOS - Deep Packet Inspection - Information Extraction." [Online]. Available: http://www.qosmos.com

[24] "Packeteer." [Online]. Available: http://www.packeteer.com

[25] "Arbor Networks." [Online]. Available: http://www.arbornetworks.com

[26] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Proc. USENIX Security Symp.*, San Antonio, TX, USA, Jan. 1998.

[27] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proc. Systems Administration Conf.*, Seattle, WA, USA, Nov. 1999.

[28] T. Karagiannis, A. Briodo, M. Faloutsos, and K. C. Claffy, "Transport layer identification of p2p traffic," in *Proc. ACM SIGCOMM Internet Measurement Conf.*, Sicily, Italy, Oct. 2004.

[29] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: Automated Construction of Application Signatures," in *Proc. ACM SIGCOMM Workshop on Mining Network Data*, Philadelphia, PA, USA, Aug. 2005.

[30] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, "Automatic protocol inference: unexpected means of identifying protocols," in *Proc. ACM SIGCOMM Conf. on Internet measurement*, Rio de Janeriro, Brazil, Oct. 2006.

[31] T. Choi, C. Kim, S. Yoon, J. Park, B. Lee, H. Kim, and H. Chung, "Content-aware Internet application traffic measurement and analysis," in *Proc. IEEE/IFIP Network Operations & Management Symp.*, Seoul, South Korea, Apr. 2004.

[32] Y. J. Won, B.-C. Park, H.-T. Ju, M.-S. Kim, and J. W. Hong, "A hybrid approach for accurate application traffic identification," in *Proc. IEEE/IFIP End-to-End Monitoring Tech and Services*, Vancouver, BC, Canada, Apr. 2006.

[33] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, vol. 31, no. 2, pp. 249–260, 1987.

[34] M. S. Kim, Y. J. Won, and J. W.-K. Hong, "Application-level traffic monitoring and analysis on IP networks," *Electronics and Telecom. Research Inst.*, vol. 27, no. 1, pp. 22–42, 2005.

[35] K. Claffy, "Internet traffic characterization," Ph.D. dissertation, University of California San Diego, 1994.

[36] V. Paxson, "Empirically derived analytic models of wide-area TCP connections," *IEEE/ACM Trans. on Networking*, vol. 2, no. 4, pp. 316–336, 1994.

[37] T. Lang, G. Armitage, P. Branch, and H.-Y. Choo, "A synthetic traffic model for half-life," in *Proc. Int. Australian Telecom. Networks and Applications Conf.*, Melbourne, Australia, Dec. 2003.

[38] T. Lang, P. Branch, and G. Armitage, "A synthetic traffic model for quake 3," in *Proc. ACM/SIGCHI Int. Conf. on Advances in Computer Entertainment Technology*, Singapore, Jun. 2004.

[39] C. Dewes, A. Wichmann, and A. Feldmann, "An analysis of Internet chat systems," in *Proc. ACM/SIGCOMM Internet Measurement Conf.*, Miami, FL, USA, Oct. 2003.

[40] A. W. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," Dept. of Comp. Sci., Queen Mary, University of London, Tech. Rep., Aug. 2005, tech. Report, RR-05-13.

[41] "Cisco IOS NetFlow." [Online]. Available: http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html

[42] "Waikato Environment for Knowledge Analysis (WEKA) 3.4." [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka

[43] M. Dash and H. Liu, "Consistency-based search in feature selection," *Artificial Intelligence*, vol. 151, no. 1–2, pp. 155–176, 2003.

[44] M. Hall, "Correlation-based feature selection in feature selection," Ph.D. dissertation, Waikato University, 1998.

[45] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 5, pp. 5–16, 2006.

[46] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/Realtime network traffic classification using semi-supervised learning," *Performance Evaluation*, vol. 64, pp. 1194–1213, 2007.

[47] F. Hernendez-Campos, A. Nobel, K. Jeffay, and F. D. Smith, "Statistical clustering of Internet communication patterns," in *Proc. Interface of Comp. Sci and Statistics*, Salt Lake City, UT, USA, Jul. 2003.

[48] J. Erman, M. Arlitt, and A. Mahanti, "Traffic clustering using clustering algorithms," in *Proc. ACM SIGCOMM MineNet Workshop*, Pisa, Italy, Sep. 2006.

[49] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman, "AutoClass: A Bayesian classification system," in *Proc. Int. Conf. on Machine Learning*, Ann Arbor, MI, USA, Jun. 1988.

[50] J. Micheel, I. Graham, and N. Brownlee, "The Auckland data set: an access link observed," in *Proc. Int. Teletraffic Cong.: Specialists Seminar on Access Networks and Systems*, Barcelona, Spain, Apr. 2001.

[51] J. Erman, A. Mahanti, and M. Arlitt, "Internet traffic identification using machine learning," in *Proc. IEEE GLOBECOM*, San Francisco, CA, USA, Nov. 2006.

[52] ——, "Byte me: a case for byte accuracy in traffic classification," in *Proc. ACM workshop on Mining Data*, San Diego, CA, USA, Jun. 2007.

[53] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and Peer-to-Peer traffic in the network core," in *Proc. World Wide Web Conference*, Banff, AB, Canada, May 2007.

[54] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Computer Communications Review*, vol. 36, no. 2, pp. 23–26, 2006.

[55] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proc. CoNEXT Int. Conf. Emerging Networking Experiments and Technologies*, Lisboa, Portugal, Dec. 2006.

[56] L. Bernaille and R. Teixeira, "Early recognition of encrypted applications," in *Proc. Passive and Active Measurement Conf.*, Louvain-la-neuve, Belgium, Apr. 2007.

[57] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," in *Proc. Passive and Active Measurement Workshop*, Antibes Juan-les-Pins, France, Apr. 2004.

[58] "NLANR Network traffic traces." [Online]. Available: http://pma.nlanr.net/Traces/

[59] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," in *Proc. ACM SIGMETRICS*, Banff, AB, Canada, Jun. 2005.

[60] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian Neural Networks for Internet traffic classification," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 223–239, 2007.

[61] H. Jiang, A. W. Moore, Z. Ge, S. Jin, and J. Wang, "Lightweight application classification for network management," in *Proc. SIGCOMM Workshop on Internet Network Management: The Five-Nines Workshop*, Kyoto, Japan, Aug. 2007.

[62] 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 SIGCOMM Internet Measurement Conf.*, Sicily, Italy, Oct. 2004.

[63] W. Li, M. Canini, A. W. Moore, and R. Bolla, "Efficient application identification and the temporal and spatial stability of classification schema," *Computer Networks*, vol. 53, no. 6, pp. 790–809, 2009.

[64] "Application Layer Packet Classifier for Linux." [Online]. Available: http://l7-filter.sourceforge.net

[65] B. Schölkopf and A. J. Smola, *Learning with kernels*. Cambridge, MA, USA: MIT Press, 2002.

[66] A. Madevska-Bogdanova, D. Nikolik, and L. Curfs, "Probabilistic SVM outputs for pattern recognition using analytical geometry," *Neurocomputing*, vol. 62, pp. 293–303, 2004.

[67] Y. Yang, Y. Liu, S. Li, and X. Zhou, "Solving P2P traffic identification problems using Support Vector Machines," in *IEEE/ACS Int. Conf. Comp. Syst. and Applications*, Amman, Jordan, May 2007.

[68] F. J. González-Castaño, P. S. Rodríguez-Hernández, R. P. Martínez-Álvarez, and A. Gómez-Tato, "Support vector machine detection of peer-to-peer traffic in high-performance routers with packet sampling," in *Int. Conf. Adaptive and Natural Computing Algorithms*, Warsaw, Poland, Apr. 2007.

[69] K. Crammer and Y. Singer, "On the learnability and design of output codes for multiclass problems," in *Proc. Computational learning theory*, Palo Alto, CA, USA, Jun. 2000.

[70] Z. Li, R. Yuan, and X. Guan, "Accurate classification of the Internet traffic based on the SVM method," in *Proc. IEEE Int. Conf. Comm.*, Glasgow, Scotland, Jun. 2007.

[71] Y. Liu, H. Liu, H. Zhang, and X. Luan, "The Internet traffic classification an online SVM approach," in *Proc. IEEE Int. Conf. Information Networking*, Busan, South Korea, Jan. 2008.

[72] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats and the best practices," in *Proc. ACM Int. Conf. on emerging Networking Experiments and Tech.*, Madrid, Spain, Dec. 2008.

[73] T. Karagiannis, D. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," in *Proc. ACM SIGCOMM*, Philadelphia, PA, USA, Aug. 2005.

[74] A. Este, F. Gringoli, and L. Salgarelli, "Support Vector machines for TCP traffic classification," *to appear, Computer Networks*, 2009.

[75] "Lawrence Berkeley's National Laboratory traces." [Online]. Available: http://ita.ee.lbl.gov/html/traces.html

[76] "Cooperative Association for Internet Data Analysis traces." [Online]. Available: http://www.caida.org/data/

[77] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese, "Network monitoring using traffic dispersion graphs," in *Proc. ACM Internet Measurement Conf.*, San Diego, CA, USA, Oct. 2007.

[78] H. G. Chew, R. E. Bogner, and C. C. Lim, "Dual-ν support vector machine with error rate and training size biasing," in *Proc. Int. Conf. Acoustics, Speech, and Signal Proc. (ICASSP)*, Salt Lake City, UT, USA, May 2001.

[79] C. Cortes and V. Vapnik, "Support-Vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[80] B. Scholkopf, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, no. 5, pp. 1083–1121, 2000.

[81] C. Scott, "Performance measures for Neyman-Pearson classification," *IEEE Transactions on Information Theory*, vol. 53, no. 8, pp. 2852–2863, 2007.

[82] M. J. Coates, B. Erikkson, R. Nowak, and C. Scott, "Learning to satisfy," Department of Electrical and Computer Engineering, McGill University, Tech. Rep., Nov. 2006.

[83] R. Rifkin and A. Klautau, "In defense of one-versus-all classification," *Machine Learning Research*, vol. 5, pp. 101–141, 2004.

[84] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 2005.

[85] "tcpdump." [Online]. Available: http://www.tcpdump.org

[86] "tcptrace - Official Homepage." [Online]. Available: http://www.tcptrace.org/

[87] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Comput. Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[88] J. Erman, "Offline/realtime network traffic classification using semi-supervised learning," Ph.D. dissertation, University of Calgary, Apr. 2007.

[89] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, "Characteristics of Internet background radiation," in *Proc. ACM SIGCOMM Internet Measurement Conference*, Sicily, Italy, Oct. 2006.

[90] M. A. Hall and L. A. Smith, "Feature subset selection: a correlation based filter approach," in *Proc. Int. Conf. on Neural Inf. Processing and Intelligent Inf. Sys.*, Dunedin, New Zealand, Nov. 1997.